SCDT41 Programming and Software Fundamentals

Mikhail Nazarov

Task 1

First solution

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

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

1. **using** System;
3. **namespace** Task1-1
4. {
5. **class** Program
6. {
7. **static** **void** Main(**string**[] args)
8. {
9. // set variables
10. **int**[] A = { 3, 5, 2, 1, 7, 4 };
11. **int** T = 14;
12. **int** total = 0;
13. **int** answer = 0;
15. // for loop to go through all numbers in array
16. **for** (**int** i = 0; i < A.Length; i++)
17. {
19. // adds numbers until is less than 14
20. **if** ((total += A[i]) <= T)
21. {
22. answer += A[i];
24. }
25. **else**
26. {
27. Console.WriteLine("Total Items added: " + i);
28. Console.WriteLine("TOTAL VALUE: " + answer);
29. **break**;
30. }

33. }
34. }
35. }
36. }

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, by using a for loop to calculate the total amount of values in the array. However, instead of one if statement with a formula from previous method, it uses a larger amount 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.

1. **using** System;
3. **namespace** Task11
4. {
5. **class** Program
6. {
7. **static** **void** Main(**string**[] args)
8. {
9. // set variables
10. **int**[] A = { 3, 5, 2, 1, 7, 4 };
11. **int** T = 14;
12. **int** total = 0;
13. **int** answer = 0;
15. // for loop to check total + next number
16. **for** (**int** i = 0; i < A.Length; i++)
17. {
19. **if** (total <= T)
20. {
21. total += A[0];
22. i++;
23. }
24. **if** (total <= T)
25. {
26. total += A[1];
27. i++;
28. }
29. **if** (total <= T)
30. {
31. total += A[2];
32. i++;
33. }
34. **if** (total <= T)
35. {
36. total += A[3];
37. i++;
38. }
39. **if** (total <= T)
40. {
41. total += A[4];
42. i++;
43. }
44. **if** (total <= T)
45. {
46. total += A[5];
47. i++;
48. }
50. **else**
51. {
52. // output the answer
53. answer += A[i];
54. Console.WriteLine("Total Items added: " + answer);
55. **break**;
56. }

59. }
60. }
61. }
62. }

Comparison

The first solution compares to second one uses the mathematical formula, instead of going through comparison of each value. The main advantage of the first solution, over the second is compatibility with array that includes more values in them. In addition, the constant use of if statements will affect badly on performance of the program (Sharma, 2019). Besides 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 factor. However, the disadvantage of both solutions, is their algorithm of going from first value, while If numbers were different, a higher amount could have been achieved.

Task 2

First Solution

For 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 to use Enumerable.Any Method.

As the first step, the variables have been set in 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 contains special characters. The purpose of integer criteria is to add 1, every time when one requirement is met. Then, the program checks if first password is equal or has more than 7 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 special character. The final check 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.

1. **using** System;
2. **using** System.Linq; // Using System.Linq
4. **namespace** Task3\_1
5. {
6. **class** Program
7. {
8. **static** **void** Main(**string**[] args)
9. {
10. // set variables
11. **string** P = Console.ReadLine();
12. **bool** valid;
13. **string** special = @"|!#$%&/()=?@~£{}.-;'<>\_,";
14. **int** criteria = 0; // criteria starts with 0 and adds 1 when it meets each criteria

17. // checks if password is at least 7 characters long
18. **if** (P.Length >= 7)
19. {
20. criteria++;
22. }
23. **else** //(P.Length <= 6)
24. {
25. Console.WriteLine("Error: your password contains less than seven characters!");
26. }
28. // checks if pasword contains letter of numbers and if there are at least one upper and lower
29. **if** (P.Any(**char**.IsUpper) && P.Any(**char**.IsLower) && P.Any(**char**.IsLetterOrDigit))
30. {
31. criteria++;
32. }
33. **else**
34. {
35. Console.WriteLine("Error: your password should contain at least one uppercase, lowercase and a number!");
36. }
38. // checks is password contains at least one special character
39. **foreach** (var item **in** special)
40. {
41. **if** (P.Contains(item))
42. {
43. criteria++;
44. }
45. }
46. **if** (criteria != 3)
47. {
48. Console.WriteLine("Error: your password should contain at least one special character!");
49. }

52. // checks if there are more than three repeating characters in a row
53. **for** (**int** i = 0; i < P.Length - 2; i++)
54. {
56. **if** (P[i] == P[i + 1] && P[i] == P[i + 2] && P[i] == P[i + 3])
57. {
58. criteria = 0;
59. }
61. }
62. **if** (criteria != 3)
63. {
64. Console.WriteLine("Error: There were more than three repreating characters in a row!");
65. }

68. // checks if password matched all criteria
69. **if** (criteria == 3)
70. {
72. valid = **true**;
73. Console.WriteLine("Successfull: your password match all requirements");
74. }
75. **else**
76. {
77. valid = **false**;
78. Console.WriteLine("Unsuccessful: your password did not meet one or more requirements");
79. }


83. }
84. }
85. }

Second solution

The second solution does not uses 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 of requirements. For example, hasUpperChar will check if string contains uppercase characters from A to Z. The second solution uses same for loop to identify if not 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.

1. **using** System;
2. **using** System.Text.RegularExpressions; // using System.Text.RegularExpressions namespace for Regex method
3. **namespace** Task3\_2
4. {
5. **class** Program
6. {
7. **static** **void** Main(**string**[] args)
8. {
10. **string** P = Console.ReadLine();
11. // set variables
12. **bool** valid;
13. **int** maxRepeat = 0;
14. // Regex has been used to identify if password matches required criterias
15. var hasNumber = **new** Regex(@"[0-9]+"); // For example, this regex contains numbers from 0-9
16. var hasUpperChar = **new** Regex(@"[A-Z]+");
17. var hasLowerChar = **new** Regex(@"[a-z]+");
18. var hasMiniMaxChars = **new** Regex(@".{7,}");
19. var hasSymbols = **new** Regex(@"[!@#$%^&\*()\_+=\[{\]};:<>|./?,-]");

22. // checks if there are more than three repeating characters in a row
23. **for** (**int** i = 0; i < P.Length- 2; i++)
24. {
26. **if** (P[i] == P[i + 1] && P[i] == P[i + 2] && P[i] == P[i + 3])
27. {
29. maxRepeat = 1;
31. }
32. }
34. // checks if password matched all criterias
35. **if** (!hasLowerChar.IsMatch(P))
36. {
37. Console.WriteLine("Error: your password should contain at least one lowercase character!");
38. valid = **false**;
39. }
40. **else** **if** (maxRepeat == 1)
41. {
42. Console.WriteLine("Error: There were more than three repreating characters in a row!");
43. valid = **false**;
44. }
45. **else** **if** (!hasUpperChar.IsMatch(P))
46. {
47. Console.WriteLine("Error: your password should contain at least one uppercase character!");
48. valid = **false**;
49. }
50. **else** **if** (!hasMiniMaxChars.IsMatch(P))
51. {
52. Console.WriteLine("Error: your password contains less than seven characters!");
53. valid = **false**;
54. }
55. **else** **if** (!hasNumber.IsMatch(P))
56. {
57. Console.WriteLine("Error: your password should contain at least one number!");
58. valid = **false**;
59. }
60. **else** **if** (!hasSymbols.IsMatch(P))
61. {
62. Console.WriteLine("Error: your password should contain at least one special character!");
63. valid = **false**;
64. }
65. **else**
66. {
67. valid = **true**;
68. Console.WriteLine("Successfull: your password match all requirements");
69. }
70. }
71. }
72. }

Comparison

Each solution uses an addition namespace, which restrict the using of these techniques in other programming languages. When comparing their performance, both solutions could be implemented in shorter way to increase the performance. According to the tests (donnetperls, n.d), the method with a switch case performance better than an if else statements, when applying to similar program. Another disadvantage of both solutions, that 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 cleared and well organiser structure, compare to first solution.

Task 3

First solution

The first solution for the task 3 is using two additional namespaces: System.Collections.Generic and System.Linq.

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**namespace** Assignment1

{

**class** Task3a

{

**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);

}

}

}

}

References:

<https://stackoverflow.com/questions/32432281/find-the-longest-repetition-character-in-string>

<https://stackoverflow.com/questions/27402126/check-if-string-have-uppercase-lowercase-and-number/27402159>

<https://stackoverflow.com/questions/20032450/detect-if-a-string-contains-uppercase-characters>

<https://stackoverflow.com/questions/4503542/check-for-special-characters-in-a-string>

<https://stackoverflow.com/questions/14634477/how-to-check-if-string-has-more-than-two-repeating-characters>

<https://stackoverflow.com/questions/45957599/regex-to-include-one-lowercase-uppercase-number-given-special-characters-in>

<https://stackoverflow.com/questions/5859632/regular-expression-for-password-validation>

<https://www.c-sharpcorner.com/article/performance-consideration-for-c-sharp-conditional-statements/>

<https://www.geeksforgeeks.org/7-common-programming-principles-that-every-developer-must-follow/>

<https://www.dotnetperls.com/if-switch-performance>