10:04 – 11:34

1. Palindrome method:

public static bool Palindrome(string myString)

{

int length = myString.Length;

for (int i = 0; i < length / 2; i++)

{

if (myString[i] != myString[length - i - 1])

return false;

}

return true;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a method that prints the numbers from 1 to 100, but for multiples of 3 print Foo, for multiples of 5 print Bar and for numbers that are multiples of both 3 and 5 print FooBar.

public static void FooMethod()

{

for (int counter = 1; counter <= 100; counter++)

{

if ((counter % 3 == 0) && (counter % 5 == 0))

Console.WriteLine("FooBar");

else if(counter % 3 == 0)

Console.WriteLine("Foo");

else if(counter % 5 == 0)

Console.WriteLine("Bar");

else

Console.WriteLine(counter);

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a method that can find and replace valid email adresses in a string.

For example, find and replace all valid email adresses in the following text:

Christian has the email address christian+123@gmail.com.

Christian's friend, John Cave-Brown, has the email address john.cave-brown@gmail.com.

John's daughter Kira studies at Oxford University and has the email adress Kira123@oxford.co.uk.

Her Twitter handle is @kira.cavebrown.

public static void ReplaceEmails(string data)

{

//Example data

data = "Christian has the email address christian+123@gmail.com."

+ "\nChristian's friend, John Cave-Brown, has the email address john.cave-brown@gmail.com."

+ "\nJohn's daughter Kira studies at Oxford University and has the email adress Kira123@oxford.co.uk."

+ "\nHer Twitter handle is @kira.cavebrown.";

//Print original text

Console.WriteLine("Original text: " + data + "\n");

//Instantiate with this pattern

Regex emailRegex = new Regex(@"\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*",

RegexOptions.IgnoreCase);

//Print new text, putting "newValue" instead of email

Console.WriteLine("New text: " + emailRegex.Replace(data, "newValue"));

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a method that can generate a list of words based on input word and alphabet.

In spell checking it is assumed that all words are wrong and alternative words are proposed if they fit better in the context. One way to generate alternative words is taking the original word and applying certain operations:

Deleting a letter.

Inserting a letter.

Replacing a letter.

Swapping two adjacent letters.

If only one operation is performed on the original word the Damerau-Levenshtein distance between the original word and the new alternative word is 1, for example, herroes [DELETE OPERATION] => heroes. If two or more operations are performed on the new alternative word(s) the Damerau-Levenshtein distance increases with the amount of operations, for example, herros [DELETE OPERATION] => heros [INSERT OPERATION] => heroes yields a Damerau-Levenshtein distance of 2.

The method should generate all possible alternative words based on the 4 operations listed above and maximum Damerau-Levenshtein distance = 1.

4a. Generate a list of alternative words using the word test, alphabet a-z (26 letters) and maximum Damerau-Levenshtein distance = 1?

public static void AltWords(string inputWord)

{

//Given input word as 'test'

inputWord = "test";

//Total alphabets

string alphabets = "abcdefghijklmnopqrstuvwxyz";

//Converting to char array

char[] charAlphabets = alphabets.ToCharArray();

Console.WriteLine("Alternative words:");

/\*Delete a letter\*/

Console.WriteLine("After Deleting a letter:");

for (int counter = inputWord.Length - 1; counter >= 0; counter--) // Decreasing maximum Damerau-Levenshtein distance = 1

{

string tempWord = inputWord;

//Deleting 1 char at a time, in counter's position

string afterCharDeleted = tempWord.Remove(counter, 1);

Console.Write(afterCharDeleted + "\t");

}

Console.WriteLine("\n");

//Console.WriteLine("\n");

/\*Insert a letter\*/

Console.WriteLine("After Inserting a letter:");

for (int counter = inputWord.Length - 1; counter >= 0; counter--)

{

foreach (char ch in charAlphabets)

{

//Console.Write("After Inserting a letter '" + ch.ToString() + "':");

// A copy of inputWord as a temp variable

string tempforInsert = inputWord;

// Inserting at counter's position, a to z

string afterCharInserted = tempforInsert.Insert(counter, ch.ToString());

Console.Write(afterCharInserted + "\t");

}

Console.WriteLine("\n");

}

//Console.WriteLine("\n");

/\*Replace a letter\*/

//Expected: test

//aesa besb...

//tast tbst...

//teat tebt...

//aesa besb...

Console.WriteLine("After Replacing a letter:");

// A copy of inputWord as a temp variable

string tempReplace = inputWord;

// Converting it to char array

char[] charTempReplace = tempReplace.ToCharArray();

foreach (char ch1 in charTempReplace)

{

foreach (char ch in charAlphabets)

{

//Console.Write("After Replacing a letter with '" + ch.ToString() + "':");

// Replacing TempReplace's first to last letter, from a - z

string afterReplacedWithAChar = tempReplace.Replace(ch1.ToString(), ch.ToString());

Console.Write(afterReplacedWithAChar + "\t");

}

Console.WriteLine("\n");

}

//Console.WriteLine("\n");

/\*Swap 2 adjacent letters\*/

Console.WriteLine("After Swapping letters:");

// A copy of inputWord as a temp variable

string tempSwap = inputWord;

// Converting it to char array

char[] charTempSwap = tempSwap.ToCharArray();

// Swapping 2 adjacent letters

//Expected: test

//etst tset tets

for (int i = 0; i < charTempSwap.Length - 1; i++)

{

charTempSwap = tempSwap.ToCharArray();

char temp = charTempSwap[i];

charTempSwap[i] = charTempSwap[i + 1];

charTempSwap[i + 1] = temp;

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

{

Console.Write(charTempSwap[j]);

}

Console.Write("\t");

}

Console.WriteLine("\n");

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4b. Write a method that can calculate the number of non-unique alternative words based on input word length and alphabet length (assuming maximum Damerau-Levenshtein distance = 1)\*

\* Without generating the word list, i.e. write a formula based on input word length and alphabet word length.

  public static void AltWordsNumber(string inputText)

{

//Total alphabets

string alphabets = "abcdefghijklmnopqrstuvwxyz";

//Converting to char array

char[] charAlphabets = alphabets.ToCharArray();

/\*Delete a letter\*/

int numOfWordsAfterDeleting = (inputText.Length);

Console.WriteLine("After Deleting a letter:" + numOfWordsAfterDeleting);

Console.WriteLine("\n");

/\*Insert a letter\*/

int numOfWordsAfterInserting = (inputText.Length) \* charAlphabets.Length;

Console.WriteLine("After Inserting a letter:" + numOfWordsAfterInserting);

Console.WriteLine("\n");

/\*Replace a letter\*/

int numOfWordsAfterReplacing = inputText.Length \* charAlphabets.Length;

Console.WriteLine("After Replacing a letter:" + numOfWordsAfterReplacing);

Console.WriteLine("\n");

/\*Swap 2 adjacent letters\*/

int numOfWordsAfterSwapping = inputText.Length - 1;

Console.WriteLine("After Swapping a letter:" + numOfWordsAfterSwapping);

Console.WriteLine("\n");

/\*Total number of alternate words after maximum Damerau-Levenshtein distance = 1\*/

int totNumOfAltWords = numOfWordsAfterDeleting + numOfWordsAfterInserting + numOfWordsAfterReplacing + numOfWordsAfterSwapping;

Console.Write("Total Number of Non-unique Alternate words: " + totNumOfAltWords);

Console.WriteLine("\n");

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_