Deadline for evaluation: 7-11-2016

These exercises must be solved with a **test driven approach**: **one single** solution must contain one subproject for each of the exercises plus one unit testing project.

Exercise 01

Write a method that checks whether an element, from a certain position in an array is **greater than its two neighbors**. Test whether the method works correctly.

Just **perform a check**. The elements of the first and the last position in the array will be compared only with their left and right neighbor. Test examples like `GreaterThanNeighbours(new int[]{1,3,2}, 1)` → `true` and `GreaterThanNeighbours(new int[]{1}, 0)` → `true`.

By invoking the method written previously in a `for`-loop, write a second method that returns the position of **the first occurrence** of an element from an array, such that it is greater than its two neighbors simultaneously. Otherwise the result must be `-1`.

Exercise 02

Write a method that calculates the **sum of two very long positive integer numbers**. The numbers are represented as **array digits** and the last digit (the ones) is stored in the array at index 0. Make the method work for all numbers with length up to 10,000 digits.

The digits on position zero will keep the ones; the digit on the first position will keep the tenths and so on. When two very big numbers are about to be calculated, the ones of their sum will be equal to `(firstNumber[0] + secondNumber[0]) % 10`, the tenths on other side will be equal to `(firstNumber[1] + secondNumber[1]) % 10 + (firstNumber[0] + secondNumber[0])/10` and so on.

Exercise 03

Write a recursive method that **traverses recursively** a specific folder on the PC and returns all subfolders and files.
For each folder return an array of strings containing the name and the files from the current folder and **call a recursion** for each subfolder. Your program may crash with `UnauthorizedAccessException` in case you do not have access permissions for some folders on the hard disk. This is typical for some Windows installations so you could start the traversal from another directory or catch the exception.

Exercise 04

Write a program, which **calculates the count of workdays between the current date and another given date** after the current (inclusive). Consider that workdays are all days from Monday to Friday, which are not public holidays, except when Saturday is a working day. The program should keep a list of predefined public holidays, as well as a list of predefined working Saturdays.

Use the class `System.DateTime` and the methods in it. You can execute a loop from the current date (`DateTime.Now.Date`) to the end date, consecutively incrementing the day by the method `AddDays(1)` and count the working days according to your country (e.g. all days except Saturday and Sunday and a few fixed non-working official holidays).

Another approach that might work is to **subtract the dates** to find the `TimeSpan` between them (`DateTime` values can be subtracted, just like a numbers). This will give you the count of days between the dates. You will need to perform some additional calculations to find how much weekends are included in this count and discard them.

Deadline for evaluation: 7-11-2016

Exercise 05

Write a program, which **generates a random advertising message** for some product. The message has to consist of laudatory phrase, followed by a laudatory story, followed by author (first and last name) and city, which are selected from predefined lists. For example, let's have the following lists:

- **Laudatory phrases**: {"The product is excellent.", "This is a great product.", "I use this product constantly.", "This is the best product from this category."}.
- **Laudatory stories**: {"Now I feel better.", "I managed to change.", "It made some miracle.", "I can't believe it, but now I am feeling great.", "You should try it, too. I am very satisfied."}.
- **First name** of the author: {"Dayan", "Stella", "Hellen", "Kate"}.
- **Last name** of the author: {"Johnson", "Peterson", "Charles"}.
- **Cities**: {"London", "Paris", "Berlin", "New York", "Madrid"}.

Then the program would print randomly generated advertising message like the following:

```
I use this product constantly. You should try it, too. I am very satisfied.
-- Hellen Peterson, Berlin
```

Exercise 06

Search for information in Internet and define your own class for exception **`FileParseException`**. The exception has to contain the name of the processed file and the number of the row where the problem is occurred. Add appropriate constructors in the exception. Write a program that reads integers from a text file. If the during reading a row does not contain an integer throw **`FileParseException`** and pass it to the calling method.

Inherit from **`Exception`** class and add a constructor to it. For example **`FileParseException(string message, string filename, int line)`**. Use this exception the same way as using any other exception. The number can be read with **`StreamReader`** class.

Exercise 07

Write a program that **downloads a file from Internet** by given URL, e.g. select an image you like. Search for articles in Internet for "**downloading a file with C#**" or search for information and examples about using the **`WebClient`** class. Make sure you catch and process all **exceptions** that can be thrown.

Exercise 08

Write a program that **encrypts a text** by applying XOR (excluding or) operation between the given source characters and given cipher code. The encryption should be done by applying XOR between the first letter of the text and the first letter of the code, the second letter of the text and the second letter of the code, etc. until the last letter of the code, then goes back to the first letter of the code and the next letter of the text. Print the result as a series of Unicode escape characters **`\xxxx`**.
Sample source text: "**`Test`**". Sample cipher code: "**`ab`**". The result should be the following: "**`\u0035\u0007\u0012\u0016`**".
Let the cipher **`cipher`** consists of **`cipher.Length`** letters. Iterate through all letters in the text and encrypt the letter at position **`index`** in the text with **`cipher[index % cipher.Length]`**. If you have a letter from the text and letter from the cipher, we can perform **XOR** operation between them by transforming in advance the two letters into numbers of type **`ushort`**. We can print the result with "**`\u{0:x4}`**" format string.

Deadline for evaluation: 7-11-2016

Exercise 09

Write a program that extracts all e-mail addresses from a text. These are all substrings that are limited on both sides by text end or separator between words and match the shape <sender>@<host>…<domain>. Sample text:

```
Please contact us by phone (+001 222 222 222) or by email at example@gmail.com or at test.user@yahoo.co.uk. This is not email: test@test. This also: @gmail.com. Neither this: a@a.b.
```

Extracted e-mail addresses from the sample text:

```
example@gmail.com
```

```
test.user@yahoo.co.uk
```

Use **`RegEx.Match(…)`** with an appropriate **regular expression**.
If you want to solve the task without regular expressions, you will need to process the text letter by letter from start to finish and process the next character, depending on the current mode, which can be one of **`OutsideOfEmail`**, **`ProcessingSender`** or **`ProcessingHostOrDomain`**. If a separator or the end of the text is reached and host or domain is processed (mode **`ProcessingHostOrDomain`**), then you have found an e-mail, otherwise potentially a new e-mail is starting and mode must be changed to`ProcessingSender`. If **`@`** character is reached in **`ProcessingSender`** mode, **`ProcessingSender`** is switched on. When meeting letters or dot in **`ProcessingSender`** or **`ProcessingHostOrDomain`** mode, they ate accumulated in a buffer. You can look at all possible groups of characters encountered respectively in each of the three modes and process them appropriately. We come to something like a final automaton (state machine), which detects e-mail addresses. All found e-mail addresses must be checked whether they have nonempty recipient, host, and domain with a length between 2 and 4 letters, as well as not beginning or ending with a dot.
Another easier approach to this problem is to split the text by all characters that are not letters and dots and to verify that the extracted "words" are valid e-mail addresses. Check can be done through an attempt to split them to nonempty parts: **`<sender>`**, **`<host>`**, **`<domain>`**, meeting the listed conditions.

Exercise 10

Write a program that reads a string from the console and prints in alphabetical order **all words from the input string** and **how many times each one of them occurs** in the string.
Use a hash table (**`Dictionary<string, int>`**) which keeps how many times each word occurs in the input string. Read on the Internet for class **`System.Collections.Generic.Dictionary<K,T>`**. With iteration through words you can accumulate information for each word occurrences in the hash table and with hash table iteration you can print the result.