Deadline for evaluation: 9-10-2016

All the following exercises must be added to **one single** solution as sub-projects!

Exercise 01

You should have Visual Studio open.

1. Create a new Project either from the shortcut on the Start page or through the file menu,
2. Select the type to be a C# Console Project
3. In the Location textbox, enter your desired hard drive location
4. Select OK.
5. The Program.cs file should have been created and opened automatically.
6. Take some time to familiarize yourself with the IDE. I usually work with the Solution Explorer view (rather than the Class View) to the right, with the Properties View underneath it.
7. Clean-up some of the automated code:
   a. Remove all of the using statements from the top of Program.cs
   b. In the Solution Explorer (View->Solution Explorer), expand the Properties and the References tree views.
   c. In the References, select everything but System and System.Core. Right-click and select Remove.
8. Configure the IDE (optional):
   a. Open the Options dialogue (Tools->Options).
   b. Under Fonts and Colors, change the Text Editor Font Size to your liking (I'm old so I use a bigger font).
   c. Scroll down to Text Editor and expand it to find the C# options. Under General, check the box to show line numbers.
   d. Explore the other options available and click OK when done.
9. In Program.cs, position your cursor after the { in Main and hit Enter. Note that things are automatically indented for you.
10. Make Hello World:
    a. Type a capital S. Note the intellisense that is highlighting System. Now type a period ('.').
    b. Type Co and then use the arrow keys to scroll down to Console. Hit return and then a period (or simply hit a period again).
    c. Type WriteLine(. Note that intellisense now gives you 19 method signatures to follow.
    d. Click somewhere else and then click back to after the (. You can get intellisense back by deleting the parenthesis and retyping it or, by hitting Ctrl->Shift->space. Use the arrows (either arrow keys or click on the arrow icons with the mouse) to reveal the syntax for a string.
    e. Type in "Hello C# World");
    f. Hit enter and type in garbage++. Note that the Error List window (View Error List) is displaying an error indicating a missing semicolon. Go ahead and add a semicolon to the end of the previous line.
11. Build your application
    a. Select Build->Build Solution
    b. The Error list pane has colored buttons at the top of it. These are toggle buttons. You can turn on the display of Warnings, etc. Here we have one Error. Toggling the error button hides it (not something we want to do).
    c. Delete the line of text garbage++;
    d. Also note the color coding on the side of the text window. Yellow indicates changed since last build, green indicates changed during this session. Deletions are not shown.

   e. Rebuild the solution.

12. Debug and Run your Hello World:
   a. Click Debug->Start Debugging. Your program just ran, finished and cleaned up. How rude, it did not even say hello! You also saw a bunch of windows flash.
   b. At line 8 (if you followed along exactly, this should be the line under the Console.WriteLine statement), click in the grey area to the left of the 8. This will place a breakpoint at this location.
   c. Re-run your program again with the debugger. Note that it now stops at line 8 (it is highlighted). You also have many debugger windows and your IDE layout has changed.
   d. Move your mouse over the args parameter to the Main function. Intellisense will indicate that it is an array of strings of size zero. Occasionally, you will see a magnifying glass here. These pull up data specific visualizers (e.g., text will open a text window).
   e. Under the Debug menu the first item is Windows. This is a little confusing, but these are all the windows you open when debugging an application. Look at the Locals window. The only variable is args. This is an alternative (and the main way) to examine the variables.
   f. You should also see a new process is running in the taskbar. This is the Console window. The application is now stopped and you can look at the Console window and see your message.
   g. Make sure the Debug toolbar is visible. View->Toolbars->Debug. Click the Step Over tool button. This will normally execute the current line and stop at the next, but since this is the last line of our simple program, it runs the application until completion.

13. Final Thoughts
   a. If you look at the files created, the key ones are the Program.cs, Properties/AssemblyInfo.cs, your solution file (.sln) and your project file (.csproj).
   b. When you created this project, it asked if you wanted to put the solution into a separate directory. Visual Studio can have many projects within a solution, including several dll projects and even several executable projects. With the Pro version of Visual Studio, these can be C++, C#, Visual Basic and any combination of such projects. The .sln file is just a container that points to the various .proj files.
   c. Both the .sln and the .csproj files are XML files. Open them in notepad to get a feel for what goes in there, but in general, do not change them.
   d. The .suo file contains Solution User Options. This includes which files were open last time you closed it, windows opened and other IDE specific settings. It also indicates which project is the “StartUp Project”.
   e. You will notice in the Solution Explorer that your project ConsoleApplication1 is in bold. This indicates that it is the StartUp project. This is the project that Visual Studio will try to execute when you run the debugger or start without debugging. If you right click on this you will see options for the project, including to set this as the StartUp project. Since it is the only one in this simple example it selected it automatically.
   f. Select Debug->Start Without Debugging. Note that the Console window displays and prompts you to hit a key to continue.
   g. Finally, you can have several builds for the same project, including a Release version that optimizes the code more and strips the debugging information. You can also build other versions as your needs dictate.

Deadline for evaluation: 9-10-2016

Exercise 02

Declare several variables by selecting for each one of them the most appropriate of the types **sbyte**, **byte**, **short**, **ushort**, **int**, **uint**, **long** and **ulong** in order to assign them the following values: 52,130; -115; 4825932; 97; -10000; 20000; 224; 970,700,000; 112; -44; -1,000,000; 1990; 123456789123456789.

Look at the ranges of the numerical types in C# described in class.

Exercise 03

Declare two variables of type **string** with values "Hello" and "World". Declare a variable of type **object**. Assign to this variable the value obtained of concatenation of the two string variables (add space if necessary). Print the variable of type **object**.

Look at the sections about Strings and Object Data Type.

Exercise 04

A company dealing with marketing wants to keep a data record of its **employees**. Each record should have the following characteristic – first name, last name, age, gender ('m' or 'f') and unique employee number (27560000 to 27569999). **Declare appropriate variables** needed to maintain the information for an employee by using the appropriate data types and attribute names.

For the names use type **string**, for the gender use type **char** (only one char **m/f**), and for the unique number and age use some integer type.

Exercise 05

Declare two variables of type **int**. Assign to them values 5 and 10 respectively. **Exchange (swap) their values** and print them.

Use **third temporary variable** for exchanging the variables:

```
int a = 5;
int b = 10;

int oldA = a;
a = b;
b = oldA;
```

To swap integer variables **other solutions** exist which do not use a third variable. For example, if we have two integer variables **a** and **b**:

```
int a = 5;
int b = 10;

a = a + b;
b = a - b;
a = a - b;
```

Deadline for evaluation: 9-10-2016

Exercise 06

Write an expression that checks whether an integer is **odd or even**.

Take the **remainder of dividing the number by 2** and check if it is **0** or **1** (respectively the number is even or odd). **Use % operator** to calculate the remainder of integer division.

Exercise 07

Write an expression that checks whether the **third bit** in a given integer is 1 or 0.

Use **bitwise "AND"** on the current number and the number that has 1 only in the third bit (i.e. number 8, if bits start counting from 0). If the returned result is different from 0 the third bit is 1:

```
int num = 25;
bool bit3 = (num & 8) != 0;
```

Exercise 08

Write a program that takes as input a **four-digit number** in format **abcd** (e.g. 2011) and performs the following actions:

- Calculates the sum of the digits (in our example 2+0+1+1 = 4).
- Prints on the console the number in reversed order: **dcba** (in our example 1102).
- Puts the last digit in the first position: **dabc** (in our example 1201).
- Exchanges the second and the third digits: **acbd** (in our example 2101).

To get the individual **digits of the number** you can divide by 10 and take the remainder of the division by 10:

```
int a = num % 10;
int b = (num / 10) % 10;
int c = (num / 100) % 10;
int d = (num / 1000) % 10;
```

Exercise 09

We are given number **n** and position **p**. Write a sequence of operations that prints the value of **the bit on the position p** in the number (0 or 1). Example: **n**=35, **p**=5 -> 1. Another example: n=35, **p**=6 -> 0.

Use **bitwise operations**:

```
int n = 35; // 00100011
int p = 6;
int i = 1; // 00000001
int mask = i << p; // Move the 1-st bit left by p positions

// If i & mask are positive then the p-th bit of n is 1
Console.WriteLine((n & mask) != 0 ? 1 : 0);
```

Deadline for evaluation: 9-10-2016

Exercise 10

Write a program that checks if a given number **n** (1 < **n** < 100) is a **prime number** (i.e. it is divisible without remainder only to itself and 1).

Read about **loops** in the Internet. Use a loop and check the number for divisibility by all integers from 1 to the square root of the number. Since **n < 100**, you can find in advance all prime numbers from 1 to 100 and checks the input over them. The **prime numbers** in the range [1…100] are: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89 and 97.