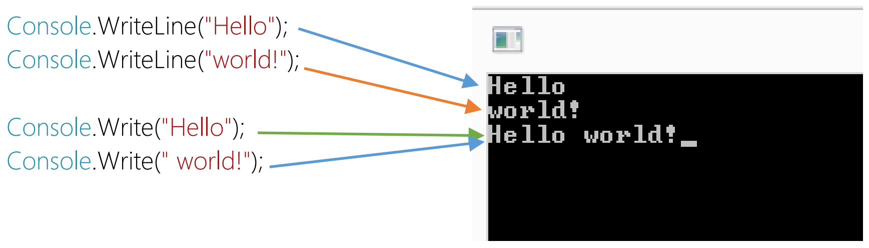
Hand-in 1

In this document, you will encounter 2 symbols. The question mark is intended to be a question to give some extra thought, and also being prepared to answer if asked. You do however not need to present it anywhere in a document or your code. If you want to make a comment in your code so you do not forget, feel free to do so. The symbol is an exercise which means that you have to write some code that solves the task given. Each keyboard symbol will have a number associated with them.

Good luck!

Writing to console

The code below uses Console.WriteLine and Console.Write to give the output seen in the picture. Apart from the output in the window, can you explain what set these two methods apart?



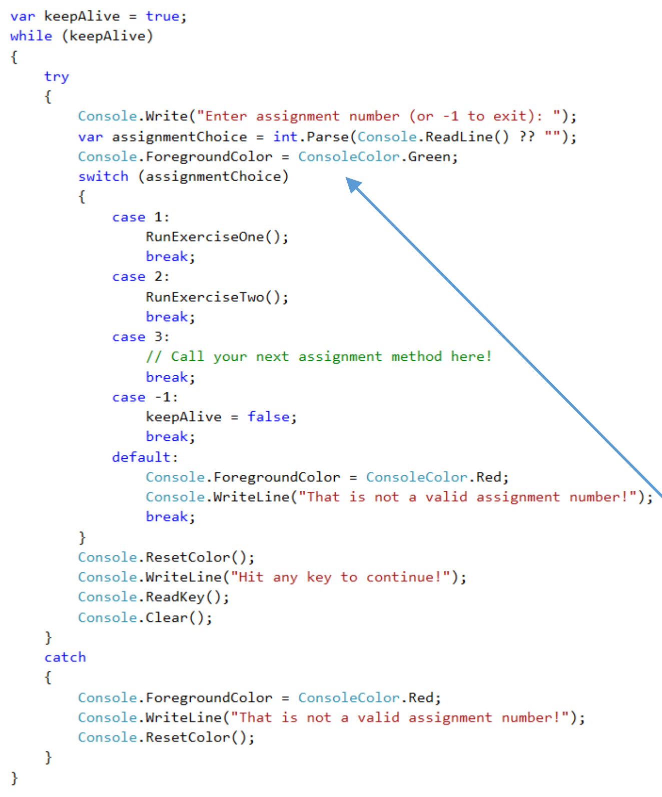
```
Console.WriteLine("Hello");
Console.WriteLine("world!");

Console.Write("Hello");
Console.Write(" world!");
```

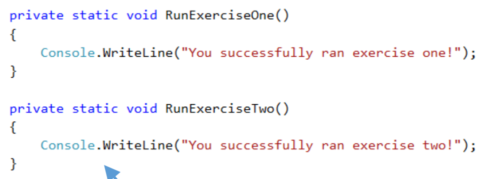
```
Hello
world!
Hello world!_
```

Exercise 0. Prerequisite

This first hand-in contains quite a lot exercises, so to present them in a good way, you are going to build a console application in which you can select and run the code for each task that you complete. Each exercise will be written inside a separate method.



```
var keepAlive = true;
while (keepAlive)
{
    try
    {
        Console.Write("Enter assignment number (or -1 to exit): ");
        var assignmentChoice = int.Parse(Console.ReadLine() ?? "");
        Console.ForegroundColor = ConsoleColor.Green;
        switch (assignmentChoice)
        {
            case 1:
                RunExerciseOne();
                break;
            case 2:
                RunExerciseTwo();
                break;
            case 3:
                // Call your next assignment method here!
                break;
            case -1:
                keepAlive = false;
                break;
            default:
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("That is not a valid assignment number!");
                break;
        }
        Console.ResetColor();
        Console.WriteLine("Hit any key to continue!");
        Console.ReadKey();
        Console.Clear();
    }
    catch
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("That is not a valid assignment number!");
        Console.ResetColor();
    }
}
```



```
private static void RunExerciseOne()
{
    Console.WriteLine("You successfully ran exercise one!");
}

private static void RunExerciseTwo()
{
    Console.WriteLine("You successfully ran exercise two!");
}
```

Figure 1

Figure 2

After creating a new Console Application project in Visual Studio, add the code above to the left (Figure 1) inside the ***main*** method. Then add the code above to the right (Figure 2) somewhere inside the ***Program*** class. Try run the program and verify that the console output correct messages when selecting exercise 1 & 2. Look through the code and see if you can get a general idea how it works, based on running it.

EXERCISE METHODS

To the right shows how to create a method for an exercise. Everything inside the curly brackets { } will execute when calling the method. You will place the code for your assignments in methods like this.

To the left shows the code needed for your menu. Depending on which number the user types, different things will happen.

If you feel that all additional methods takes up too much space in a single file, you can have multiple files for your code. What you can do is declare the Program class to *public partial class Program* and create an additional partial class with the same name in another file. Then the c# compiler will combine both files at compile time into one single class. You can have as many partial classes as you want.

Exercise 1.

Inside the *RunExerciseOne* method, add two new variables to your program that stores string values. One of them is going to store you first name and the other your last name, so give them informative names. Then let the program output:

"Hello <firstname> <lastname>! I'm glad to inform you that you are the test subject of my very first assignment!"

Exercise 2.

Inside the *RunExerciseTwo* method, create three new variables of type DateTime and let them store yesterdays, todays and tomorrows date. Remember to give them informative names. Then let your program output them to the screen like:

"Todays date is <todays date>"

"Tomorrows date is <tomorrows date>"

"Yesterdays date was <yesterdays date>"

DATETIME & TIMESPAN CLASS

Tips: You can get a DateTime-object with the current date using DateTime.Now.

There are plenty of methods on that can be used on a DateTime-object. For example you can get back a new date a number of years, months, weeks or days ahead of the given date using AddYears, AddMonths, AddWeeks and AddDays. You can also provide negative values to get dates back in time.

By subtracting two dates, you get back a TimeSpan-object which holds a time interval between the two dates. With the TimeSpan-object you can retrieve information like the number of hours, minutes or seconds passed between the two dates.

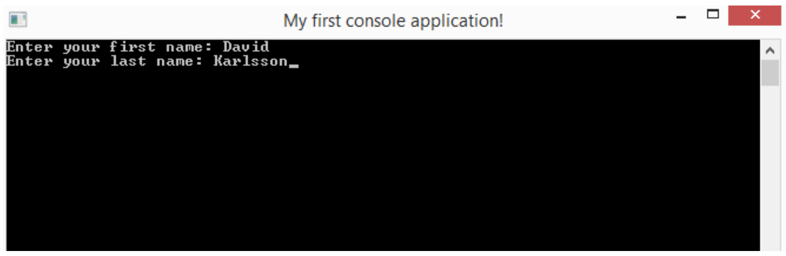
String manipulation

What do you think the following statements will yield for result to the console window?

```
String part = "if you're doing your";
part = part + " best, \tyou won't have any";
String quote = part + " time to worry about failure.";
Console.WriteLine(quote);
```

Exercise 3.

Create a new method named ***RunExerciseThree*** and write the program code inside that prompts the user to input his first and last name like shown below (with correct line breaks). Also, create two new variables that stores string values and save the input from the user into these (So that one variable stores the first name and the other stores the last name). Then output the full name to the screen on one line. Inside the switch-statement in the main method, call you new method (see the calls to the previous methods) after *case 3*.



```
Enter your first name: David
Enter your last name: Karlsson_
```

Exercise 4.

The code snippet below will be used for this exercise, so create a new method for exercise 4 and make sure to call in inside the switch statement after **case 4**.

Inside the method body, create a new empty string variable below the variable *str*. Use string manipulation methods like SubString, IndexOf, Remove, Replace, Insert to get the string "The brown fox jumped over the lazy dog", where all characters matches exactly and output these to the screen.

```
String str = "The quick fox Jumped Over the DOG";
 // code here
```

Exercise 5.

Below is a given string. Use string manipulations to get the [1,2,3,4,5] part from the string and store it into a new string variable. Then remove the values 2 and 3 and insert 6-10 into it in the end, comma separated so that the result is [1,4,5,6,7,8,9,10] and let your program output that to the console window.

```
String str = "Arrays are very common in programming, they look something like: [1,2,3,4,5]";
Console.ReadLine();
```

Mathematical operators and methods

In the following exercises you are going to use the common mathematical operators on doubles and integers, and the mathematical functions located in the System.Math-class.

Why does the following program not output the correct average of the two numbers given? And how can we alter the code to produce the correct result?

```
int a = 8;
int b = 1;
double average = a + b / 2;
Console.WriteLine(average);
```

Exercise 6.

Write the program code that lets the user input 2 integers from the console. Then let the program output the biggest, smallest, difference (-), sum (+), product (*) and ratio (/) between the two numbers.

Exercise 7.

Write the program code that lets the user input the radius (as a double). Then calculate the area and volume of a circle respective sphere with the given radius. (area = $2\ \pi r^{2}$, volume = $\frac{(4 \times \pi \times r^3)}{3}$). Output the result to the screen.

Exercise 8.

Let the user input a decimal number. Then output square root of the number and the number raised to the power of 2 and 10. That is $\sqrt{n}, n^2, n^{10}$.

MATHEMATICAL OPERATORS

C# uses the standard order of operations to do calculations so (), *, / and % (modulo) has priority over + and -.

When dividing integers in c#, this is done with integer division. This means 10 / 3 will yield the result 3, instead of 3.333… as you might expect. This is due to that 10 is divisible 3 whole times with 3 with the rest of 1 which will be thrown away.

To get the rest, use the modulo operator like

```
int rest = 10 % 3;
```

which yields back 1.

To get the floating point result back, you need to tell the c# compiler to temporary treat either the nominator or denominator as a floating point number. This can be done by type casting either like

```
double result = 10 / (double) 3;
```

or explicitly set one as a floating point literal by adding a decimal to the end

```
double result = 10.0 / 3;
```

Type casting will not affect the variable being type casted, and when type casting a decimal number to an integer, the number will be truncated.

```
int rest = (int) 3.9999;
```

The variable *rest* now store the value 3.

Selection

Exercise 9. Restaurant order

In this exercise, you are going to practice using selection. To start with, create a new exercise method and make a call to it after case 9.

Let your program ask the user for his name and save it into a variable.

Let your program greet the user by his name, and ask for his/her birth year and convert it into an integer. For example 1978 or 1983.

Calculate the age of the user by simple integer subtraction from the current year. (You can use *DateTime.Now.Year* to retrieve the current year). Depending in which month and day the user is born in, this calculation might not give back the correct whole number of years, but it will do for this example.

If you want to challenge yourself, you are free to input an entire date and calculate the age. (There is a DateTime.Parse method that allow you to parse a string value to a DateTime object, and subtracting dates will give you back a TimeSpan object that contains for example days passed between the two dates)

Then, using if-statements, construct the following program flow

- If the user is 18 or above
    - Ask if the user want to order a beer
    - If he/she want to order a beer
        - Display a message that the order has been done!
    - If he/she does not want to order a beer
        - Ask if he/she want to order a coke
        - If he/she want to order a coke
            - Then display a message that the coke has been served
        - If he/she does not want to order a coke
            - Then display a message that no order options are available
- Else (the user is below 18 years old)
    - Ask if the user want to order a coke
    - If he/she want to order a coke
        - Then display a message that the coke has been served
    - If he/she does not want to order a coke
        - Then display a message that no order options are available

Below (Figure 3) is a Flow chart diagram that shows the application flow.

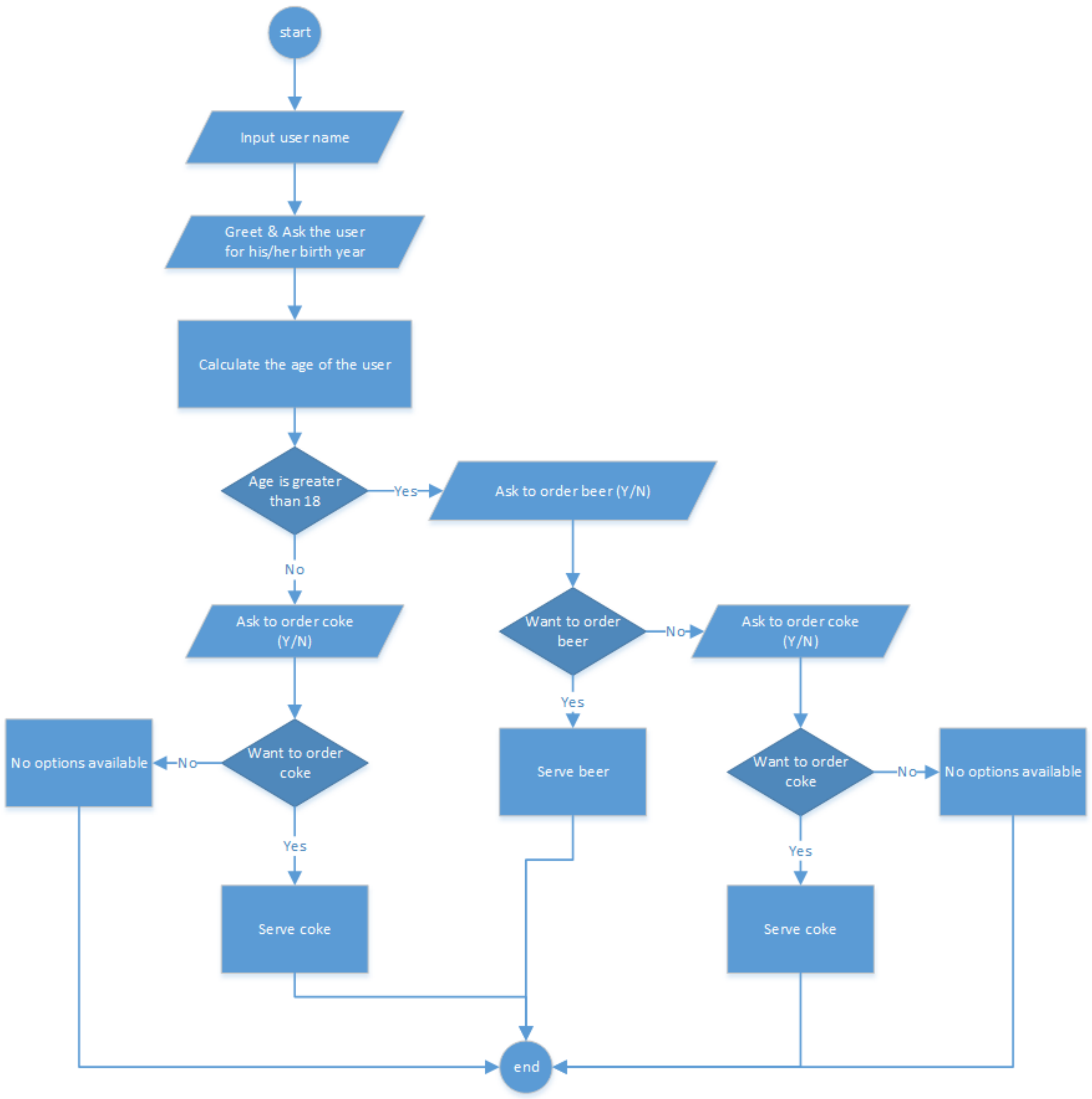


Figure 3

Exercise 10.

Write the program code that will ask to user to choose between 3 different options using a switch-statement.

1. Let the first option invoke a method that let the user input two numbers (*a* and *b)*. Check that *b* is not equal to zero and then output *a* divided by *b* to the screen. If *b* is equal to zero, display an error message to the user.
2. Let the second option invoke the method used in exercise 4 (The quick fox jumped over the lazy dog).
3. Let the third option toggle the foreground colour between two different colours each time you choose it. (Use an if-statement to check the current colour)

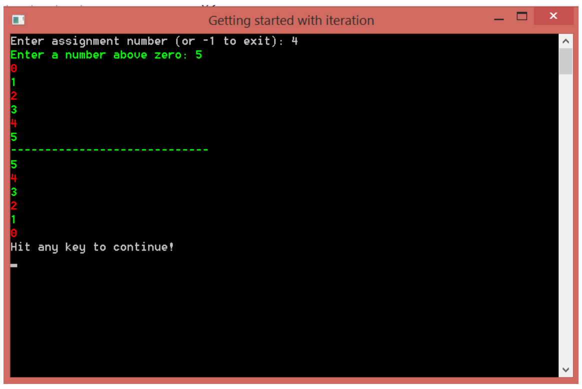
Iteration

Exercise 11. Get going with iterations

In the exercise method, write the code that lets the user input an integer number. Check that the given number is greater than 0. If not display an error message.

Then write two for-loops, where one start from zero and counts up to the given number (0,1,2,3,4..., n) and another for-loop that starts counting from the given number down to zero (n,...,3,2,1,0).

In both loops, check each number if it's evenly divisible by 2. If it is, change the console colour to red, else to green.



```
Enter assignment number (or -1 to exit): 4
Enter a number above zero: 5
0
1
2
3
4
5
-----------------------------
5
4
3
2
1
0
Hit any key to continue!
_
```

Exercise 12.

Write a program that output the multiplication table for 1 to 10 using nested for-loops. The format is not of great important but to make the code readably, add a tab after each number using the escape character \t.

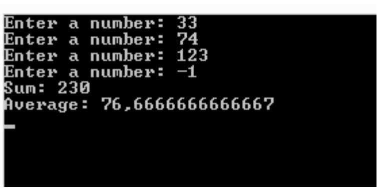
1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Exercise 13.

Write a program that first generates a random number between 1 and 500 and stores it into a variable (see the Random class). Then let the user make a guess for which number it is. If the user types the correct number, he should be presented with a message (including the number of guesses he has made). If he types a number that is greater or smaller than the given number, display either "Your guess was too small" or "Your guess was too big". The program should keep executing until the user input the correct guess.

Exercise 14.

Write a program that keeps asking the user for input numbers, until he enters -1. Store the amount of numbers the user have entered and the sum of the numbers added together. When the user types -1, the program should display the sum and the average of the numbers.



```
Enter a number: 33
Enter a number: 74
Enter a number: 123
Enter a number: -1
Sum: 230
Average: 76,6666666666667
```

Exercise 15. (Optional)

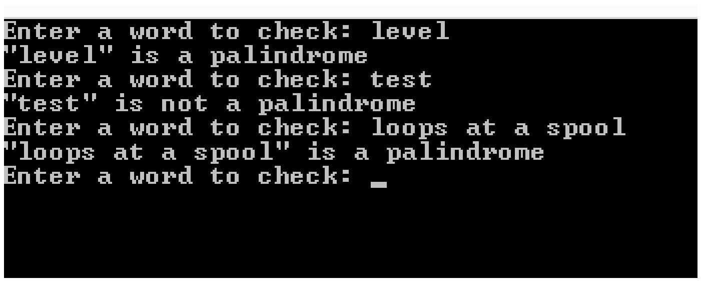
- Write the program code that asks the user for a number. Then display all numbers that the number is divisible by. Example entering 12, should output 6, 4, 3, 2 and 1. Tip: use modulo and a loop.
- Write the program code that outputs the 3 first perfect numbers. A perfect number is a number where all its positive divisors sums up to the actual number. The first number is 6, where 3 + 2 + 1 = 6 and the second is 28, where 14 + 7 + 4 + 2 + 1 = 28. Tip: look at the previous exercise and build on top of it.

Exercise 16. (Optional)

Write a program that asks the user for a number. Use this number to output the Fibonacci series up until that number. Entering 10 should then output: 0, 1, 1, 2, 3, 5, 8, 13, 21 and 34.

Exercise 17. (Optional)

Let the user input a string, then check if the string is a palindrome sentence. A palindrome is a word or sentence that reads the same in both directions. Example of palindrome sentences are Loops at a spool, wet stew and level. However, the spaces might look different depending on which direction you read it, so these should be excluded in your calculations, and a tip is to use some string manipulation to remove them.



```
Enter a word to check: level
"level" is a palindrome
Enter a word to check: test
"test" is not a palindrome
Enter a word to check: loops at a spool
"loops at a spool" is a palindrome
Enter a word to check: _
```

Arrays

Exercise 18. Get going with arrays

Create a new empty integer array of 10 elements. Loop through the array and assign each element to a new random value.

Create a new empty array of doubles, having the same size as the previous array. Loop through that array and assign the values to 1 divided by the value on the same position of the previous array. So if the first array has the value 42 on position 3, the second array should hold the double value 1 / 42.

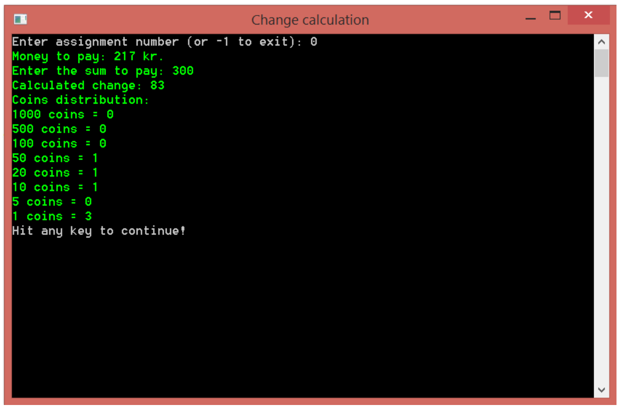
Finally, loop through both arrays and output the values to the screen using a foreach-loop.

Exercise 19. Calculate change

Create a program that outputs a price that the customer (user) needs to pay. This should be an integer value. Then let the user input the sum he hands the cashier.

Let your program then calculate the change that the customer should get back in different coin unit. For example, if the user hands the cashier 500 kr. and the price is 376 kr., the change will be 124. This can be divided up into 100x1 kr. + 20x1 kr. + 4x1 kr.

The goal here is to get as few coins as possible.



```
Enter assignment number (or -1 to exit): 0
Money to pay: 217 kr.
Enter the sum to pay: 300
Calculated change: 83
Coins distribution:
1000 coins = 0
500 coins = 0
100 coins = 0
50 coins = 1
20 coins = 1
10 coins = 1
5 coins = 0
1 coins = 3
Hit any key to continue!
```

Tip: Use an array to store the different coin units, like 100, 50, 20 etc. When you have calculated the change, go through the coin units from larges to smallest and calculate how many of each type the customer should get back. Here is a perfect scenario when integer division and modulus is viable to use.

Exercise 20. (Optional)

Create two arrays with arbitrary size and fill one with random numbers. Then copy over the numbers from the array with random numbers so that the even numbers are located in the rear (the right side) part of the array and the odd numbers are located in the front part (the left side).

Exercise 21.

Let the user input a string with numbers comma separated like "1,2,34,83,19,45". Create the code to separate the numbers in the string into an array and find the min, max and average value. Print these out to the screen.

Tip: use the Split-function on the String-object. Keep in mind that the Split-method returns an array of Strings, so you have to convert each value to an integer before you can do the calculations.

Exercise 22. (Optional)

The company See sharp AB have discovered that the users on their forums use a very harsh language when interacting with each other. So now they have asked you to write a swear word filter to censor these occurring words. Before implementing this filter on their website, they want a demonstration in form of a console program.

The program should ask the user for a textual input. Then it should check for all occurrences of swear words. Store the swear words in an array and check the textual input against the array and use string manipulation to replace all swear words with something more appropriate.

Exercise 23. (Optional, Hard)

Create a program that generates 7 unique numbers into an array. The numbers should be between 1 and 40. Each of the numbers may only appear once in the array! The numbers should be generated using the Random-class, and should be different each time you run the program.

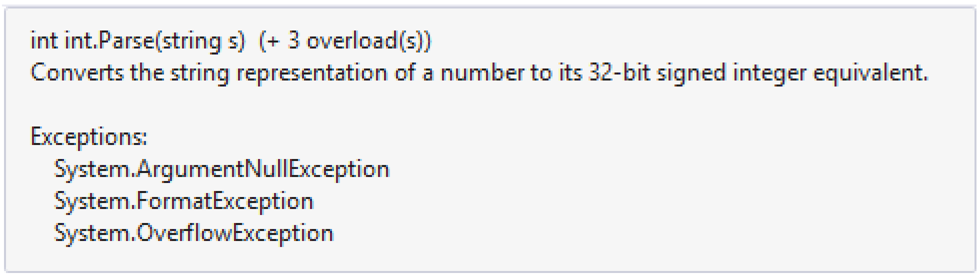
Exercise 24. (Optional, Hard)

Create a program that can shuffle a deck of cards. The cards can be represented as an array of integers, like [1,1,1,1,2,2,2,2,3,3,...n]. Then make it possible to draw 1 card from the deck and add to another array. (The card should then disappear from the card deck and appear in the array with the drawn cards. Tip: You can use Array.Resize(ref array, newSize) to change the size of an existing array, and Array.Copy. There is however not a requirement that the array needs to be in a specific order after a card has been drawn. Complete the functions below.

```
static int DrawCard(ref int[] deck)
{
    // code to draw a card here
}
```

```
static void ShuffleCards(ref int[] deck)
{
    // code to shuffle the deck here
}
```

Error and file handling



int int.Parse(string s) (+ 3 overload(s))
Converts the string representation of a number to its 32-bit signed integer equivalent.

Exceptions:
  System.ArgumentNullException
  System.FormatException
  System.OverflowException

Figure 4 Hover over a method call to see possible exceptions it throws

Exercise 25.

Create a separate function (from the exercise method) that asks the user to input a valid integer value. The function should keep executing until the user has inputted a valid integer value. Use a Try-catch combined with a while-loop. If the user inputs a none-valid number, display an error asking him/her to try again.

Then in the exercise method, call the method you just wrote twice to retrieve 2 integers from the user and store them in variables. Then divide one of the number with the other and use a try-catch to catch any potential division by zero. Display the result to the screen.

Note: Use the correct Exception type, example FormatException, DivideByZeroException and ArgumentNullException and multiple catch-statements to display a friendly error message depending on which error occurs.

Exercise 26.

In this exercise, you are going to print out the folder path to some different locations on your computer. These can be found in the Enviroment-class. To print one of the common folders on your computer, you can use the Enviroment.GetFolderPath, passing it one of the values from the Enviroment.SpecialFolder enumeration.

For example Enviroment.GetFolderPath(Enviroment.SpecialFolder.Desktop) will give back a string representing the folder path to the desktop.

Print out the following locations on your computer:

- My documents
- My Pictures
- Program files (x86)
- The folder containing information about cookies
- Current Directory (found inside in the Enviroment class)

In the same method, create a new file on the desktop, using File.Create(filepath), by retrieving the folder path to the desktop and append the filename at the end of the file path. When creating a new file using File.Create, a new FileStream will be created, and this must be closed using the Close method. Note: Depending on which computer you are running your application, some special folders might not exist.

Exercise 27. Reading a file

Inside your project in Visual Studio, right-click on your project and add a new text file. Open the file and add at least 10 names on one row each. Right click on the file and hit *Properties*. In the *Properties* window, make sure that *Copy to Output Directory* is set to *Copy Always*. This will make sure that the files are copied to the bin-folder when your program is compiled (see figure 5).

Now, use a StreamReader object (found inside System.IO) to read the names from the file and output them to the console window.

Note: There are a few possibilities when reading from a file, but one thing to be careful about, is that file might have a larger size than the available memory on your computer. Using ReadToEnd(), which reads every byte in the file would then cause a OutOfMemoryException. This is however not an issue with this exercise.

Exercise 28. Writing to a file

In the exercise method, create 2 new string arrays containing 5 names each.

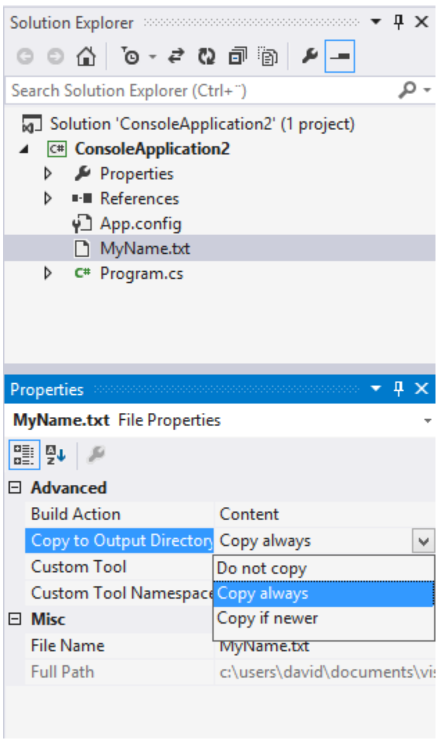
Create a new StreamWriter-object passing the file path to the file you created on the desktop in the previous exercise using the Enviroment-class. If you have not completed the previous task, manually create a new text file on your desktop.

Write the 5 names from one of the arrays to the file on one row each using a StreamWriter-object. Finally close the StreamWriter.

Create a new StreamWriter-object, passing the file path as previous step, but this time provide the StreamWriter-constructor with the second bool parameter set as true. If this is set to true, the writer will append any text written instead of overwriting all the content. Now, write the 5 names from the second array on one line each. Remember to close the StreamWriter.

Verify that all 10 names are located in the file. Try setting the second parameter to false in the second StreamWriter and verify that only the last 5 names are present.

Note: If you provide a filePath to the StreamWriter that does not exists, it will attempt to create a new file at that location. You can use the File.Exist method to make sure that the file already exists before attempting to write to it. File.Exists will return either true or false depending on if the file exists or not.



Build Action	Content
Copy to Output Directory	Copy always
Custom Tool	
Custom Tool Namespace	
File Name	MyName.txt
Full Path	c:\users\david\documents\vi

Figure 5

ADDITIONAL ON FILES

There are plenty of prebuilt functions when working with files in C# to allow your program to automate file and directory handling. A few of them are:

- File.Exists
- File.Delete
- File.Move
- File.Copy
- File.Replace

There are also similar functions when working with directories (folders)

- Directory.CreateDirectory
- Directory.Move
- Directory.Delete
- Directory.Exists

There are also the DirectoryInfo/FileInfo-classes that allow you to retrieve information about files and directories. For example the size of a file, or which files and subdirectories are present in a given folder.

We are not going to any deeper into these topics here, but if you are going to build an application that handles files and directories, this is a good way to start.