# Assignment 1 – Data storage

a) Create a `class Person` (in a separate file) with the fields: `name` (`string`), `city` (`string`) and `age` (`int`).
Create a method:

```
Person ReadPerson()
```

This method asks for the data of a person, reads this data and returns a Person-object.
Also create a method:

```
void DisplayPerson(Person p)
```

This method displays the data of a person on screen.
→ Test both methods by calling them from the Start method.

b) We want to write the data of a person into a file so next time we don't have to enter this data again.
For this, create a method:

```
void WritePerson(Person p, string filename)
```

This method writes the (data of a) person to file, each field on a separate line.
→ Test the method in the Start method.
→ Check if the file is created and if it contains the data (of a person).

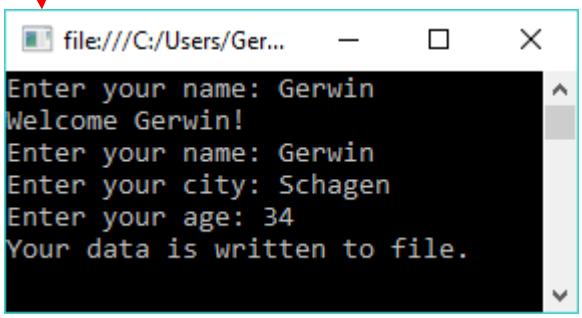
c) Create a method:

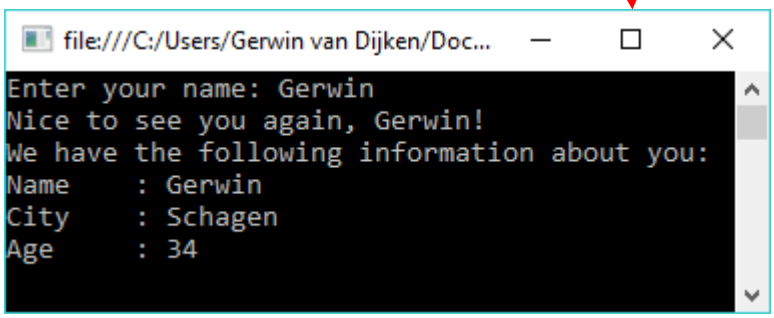
```
Person ReadPerson(string filename)
```

This method reads a person from a file with the given filename, and returns this person.
→ Test the method by calling it from the Start method.

d) Now use the implemented methods in order to create the Start method with the next functionality:

- The user is being asked to enter his/her name.
- [new user]: When there is no file with the name of the user ('<name>.txt'), then the message 'Welcome <name>!' is shown and the user must enter the information (name, city and age) and this information is written to file (with filename '<name>.txt').
- [existing user]: When a file exists with the name of the user (e.g. "Brian.txt"), then the message 'Nice to see you again, <name>!' is shown and next the information that is stored in the file.




# Assignment 2 – Hangman

The last version of Hangman was using a hardcoded list of words. Ofcourse it's more fun to pick a word from a long list of words (read from a textfile). You can't read a random word from file, so you first need to read all words from file and then pick one randomly (that one will be used as the secret word in the Hangman game).

a) Modify your Hangman program in order to read the words from a textfile. Use the file 'words.txt' from Blackboard. Make sure the program still works.
***Note***: The file on Blackboard also contains very short words. So, check if the randomly picked word contains at least 3 letters. If not, choose another word.

## Assignment 3 – Word finder

In this assignment the user can enter a word, and the program will try to find this word in a textfile. All lines containing the word will be displayed on screen. A file containing tweets of president Donald Trump can be found on Blackboard (but you can also use another text file).

a) Create a method with signature:

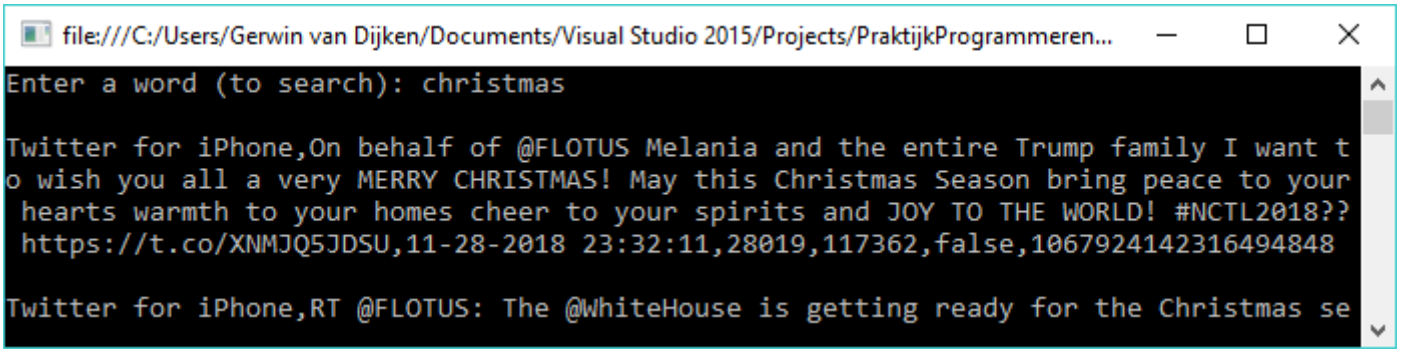
```
bool WordInLine(string line, string word)
```

This method returns true if the (given) word is present in the (given) line. The check has to be 'case insensitive'.

b) Create a method:

```
int SearchWordInFile(string filename, string word)
```

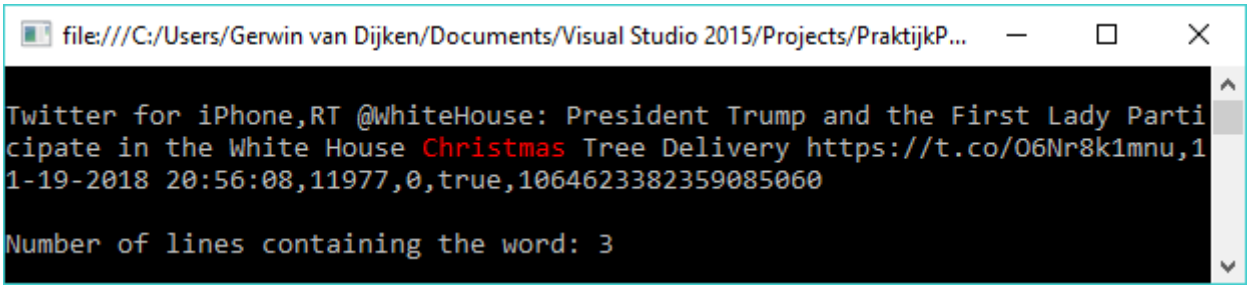
This method reads and processes each line in a file.
When a line contains the given word (use method `WordInLine`), then this line is printed on screen.
The method `SearchWordInFile` returns the number of lines containing the word.
→ Test this method by calling it from the Start method (with a word given by the user). The Start method displays the returned number on screen.

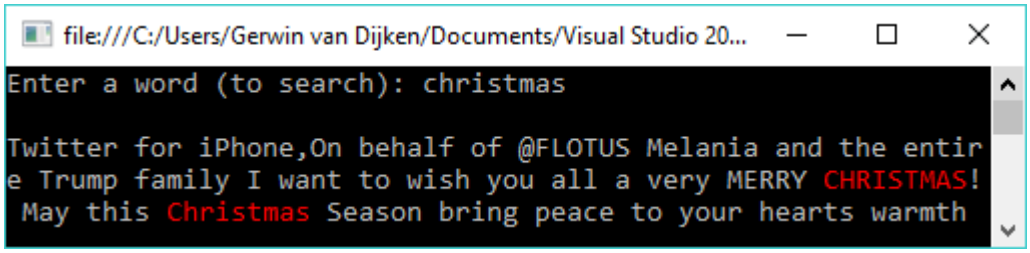


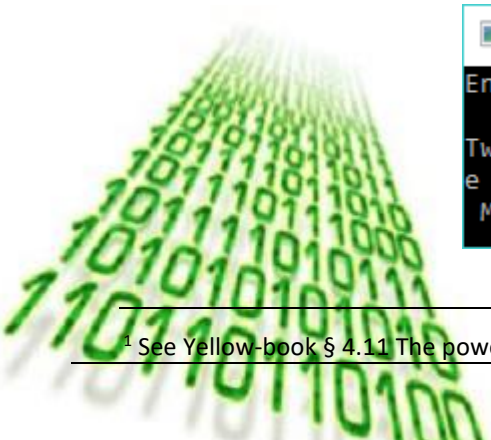

c) To indicate where the word is located in a line, we will print the word in a different color.
Create a method:

```
void DisplayWordInLine(string line, string word)
```

This method displays the given line to screen, with the word displayed in red (see screenshot below).
Use `line.IndexOf(word)` to get the starting position of the word in the line.
Use `line.Substring(start)` or `line.Substring(start, length)` to get a substring of the line.[1]
You only need to print the first occurence of the word in red.
→ Modify method `SearchWordInFile` in order to print the lines via method `DisplayWordInLine`.
→ [optional] Print all occurences of the word in red (see last screenshot).



[1] See Yellow-book § 4.11 The power of strings and chars.

## Assignment 4 – Candy Crush

We will modify the CandyCrush assignment of the 2nd week, so we will be able to save the playing field to a textfile, and to start the game with the playing field stored in a textfile (by reading it back).

a) Create a method:

```
void WritePlayingField(RegularCandies[,] playingField, string filename)
```

This method saves the playing field to a textfile. Use the `int`-values of the enumeration RegularCandies. Save the values row by row, so the playing field can be recognized in the textfile.

b) Create a method:

```
RegularCandies[,] ReadPlayingField(string filename)
```

This method reads a playing field from a file (with the given filename) and returns this playing field.
***Hint:*** To read multiple numbers from one line, read the whole line as a string. Then split this line into an array with string-items (with `Split`). The strings in the array can be parsed (one by one) to an `int`-value.
→ `string[] numberStrings = line.Split(' ');`

c) Now modify the Start method in order to read the playing field from a file (if the file exists) or to fill it with random candies (if the file does not exist). When a new playing field is created, then write it to a file.

d) Make sure the program does not crash if the file is not correct (maybe because there are not enough lines in the file, or a line does not contain enough numbers). In the Start-method catch the exception, and generate a new playing field with random candies (and save this to file).

e) You are now able to change manually the playing field in the file (with e.g. Notepad), in order to test the methods `ScoreRowPresent` and `ScoreColumnPresent` with specific values. There are certain 'boundary values' that will (most likely) not be present in a random generated playing field. What would be good 'boundary tests' for methods `ScoreRowPresent` and `ScoreColumnPresent`?
→ Execute (and check) these boundary tests.