

## Week 5 assignments

When creating the program code, you must apply the following basic principles:

- create a separate project for each assignment;
- use name 'assignment1', 'assignment2', etcetera for the projects;
- create one solution for each week containing the projects for that week;
- make sure the output of your programs are the same as the given screenshots;

Note: for assignment 3 and 4, you need to use command line arguments to pass the filename to be used. Use the following code in your Main method:

```
static void Main(string[] args)
{
    if (args.Length != 1)
    {
        Console.WriteLine("invalid number of arguments!");
        Console.WriteLine("usage: assignment[3-4] <filename>");
        return;
    }

    string filename = args[0];

    Program myProgram = new Program();
    myProgram.Start(filename);
}

void Start(string filename)
{
    // your code here...
}
```

## CodeGrade auto checks

Make sure all CodeGrade auto checks pass (10/10) for your assignments. The manual check will be done by the practical teacher.

Auto checks assignment 1-<sup>10</sup>/<sub>10</sub> **AT**
Auto checks assignment 2-<sup>10</sup>/<sub>10</sub> **AT**
Auto checks assignment 3-<sup>10</sup>/<sub>10</sub> **AT**
Auto checks assignment 4-<sup>10</sup>/<sub>10</sub> **AT**

Manual check

Automatic checks for assignment 1

0

10

10

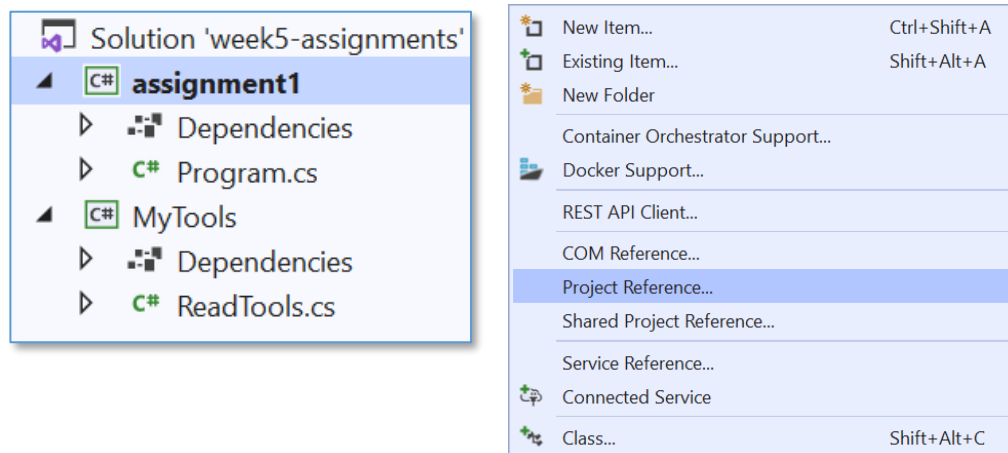
100 %

Submit
8.00
40 / 50

## Assignment 1 – A library with some useful methods

We will refactor assignment 0 of the 1<sup>st</sup> lesson by introducing a Class Library. Later we can use this Class Library in other projects.

- Create a new project of type 'Class Library' and give it the name "MyTools". In this project create a new class ReadTools.
- Copy the 2 ReadInt methods and the ReadString method of assignment 0 of the 1<sup>st</sup> lesson to class ReadTools. Make sure these methods are public and static.
- Create a new Project of type 'Console App' and give it the name "assignment1". Copy the Start method of assignment 0 of the 1<sup>st</sup> week to this new Console App.  
→ Add a reference in the Console App to Class Library "MyTools".



→ Add at the top of your Program the following statement:

```
using MyTools;
```

→ Adjust the method calls (in method Start) in order to use the ReadTools-methods of Class Library MyTools (see code below).

→ Test your program.

You can add multiple classes to this MyTools Library (for all kinds of useful methods).

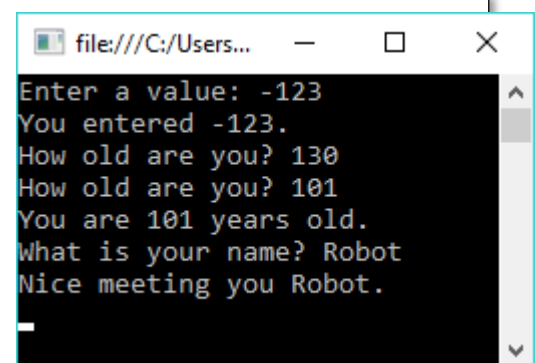
```
static void Main()
{
    Program myProgram = new Program();
    myProgram.Start();
}

void Start()
{
    int value = ReadTools.ReadInt("Enter a value: ");
    Console.WriteLine("You entered {0}.", value);

    int age = ReadTools.ReadInt("How old are you? ", 0, 120);
    Console.WriteLine("You are {0} years old.", age);

    string name = ReadTools.ReadString("What is your name? ");
    Console.WriteLine("Nice meeting you {0}.", name);

    Console.ReadKey();
}
```



## Assignment 2 – CandyCrush with logic layer

We can distinguish userinteraction and game-logic in the CandyCrushGame. So it makes sence to create a separate layer for the CandyCrush-logic.

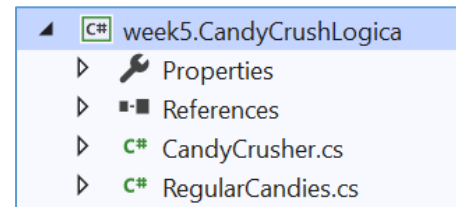
- a) Create a new CandyCrush-application “assignment2” (Console App) and copy the code of the CandyCrush-application of week 2 into this new application.  
→ Check if this new application is working.

- b) Add (to your solution) a new ‘Class Library’ project with the name CandyCrushLogic.

Add a `class CandyCrusher` to this Class Library.

Move the 2 ‘Present’ methods to this CandyCrusher class:

- `ScoreRowPresent(...)`
- `ScoreColumnPresent(...)`



Also move the file with `enum RegularCandies` to the logic layer (change the namespace, this has to be the same as the namespace of class CandyCrusher).

→ build this Class Library, there should be no compile errors.

- c) Now the CandyCrush Console Appl (created in ‘a’) has to use the logic layer:

- In the CandyCrush application add a reference to the logic layer.
- In Program.cs add a `using` statement for the namespace of the logic layer.

→ Make sure to use (call) the ‘Present’ methods defined in the logic layer, and test your application again.

## Assignment 3 – Translation

In this assignment we will read a file with Dutch words and the corresponding English translation. With this program the user can enter a Dutch word and the program will display the English translation. You can use the file “dictionary.csv” that can be found on Moodle. Use a command line parameter for the file to use.

- a) Create a new Console project (“assignment3”) and add the following method:

```
Dictionary<string, string> ReadWords(string filename)
```

This method reads all words from a textfile and stores these words in a Dictionary. Each line contains a Dutch word and the English translation, separated with a semicolon (e.g.: “slecht;bad”). Add each line in the dictionary with the Dutch word as key, and the English translation as the value. You can split each read line via `line.Split(';')`, in order to get an array with (in this case 2) separated fields.

Return the dictionary after all lines have been processed.

→ Call method `ReadWords` from the Start method.

- b) Add the following method:

```
void TranslateWords(Dictionary<string, string> words)
```

This method reads continuously a Dutch word from the user until the word ‘stop’ has been entered. For all other words the translation is displayed via the words-dictionary. If the dictionary does not contain the (Dutch) word, then a message is displayed (“word not found”, see below), otherwise the English translation is displayed. A few examples can be seen in the screenshot below, to the left (*the use of different colors is optional*).

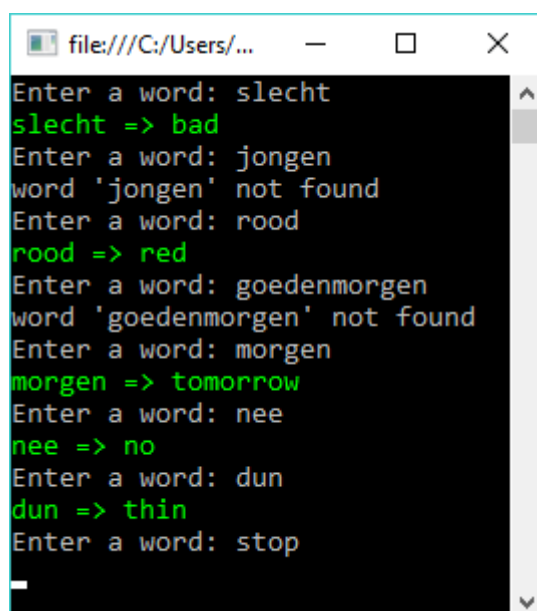
→ call method `TranslateWords` from the Start method.

→ Test different words to test the program.

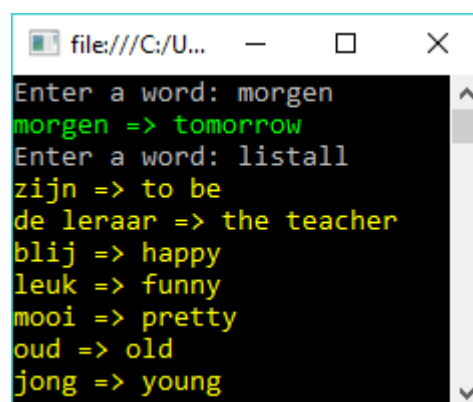
- c) Add the following method:

```
void ListAllWords(Dictionary<string, string> words)
```

This method displays all Dutch words in the dictionary, together with the corresponding translations. Adjust method `TranslateWords` in order to display all words (by calling `ListAllWords`) when the user enters “listall”. An example can be seen below, to the right (*again, the use of different colors is optional*).



```
file:///C:/Users/...
Enter a word: slecht
slecht => bad
Enter a word: jongen
word 'jongen' not found
Enter a word: rood
rood => red
Enter a word: goedenmorgen
word 'goedenmorgen' not found
Enter a word: morgen
morgen => tomorrow
Enter a word: nee
nee => no
Enter a word: dun
dun => thin
Enter a word: stop
```

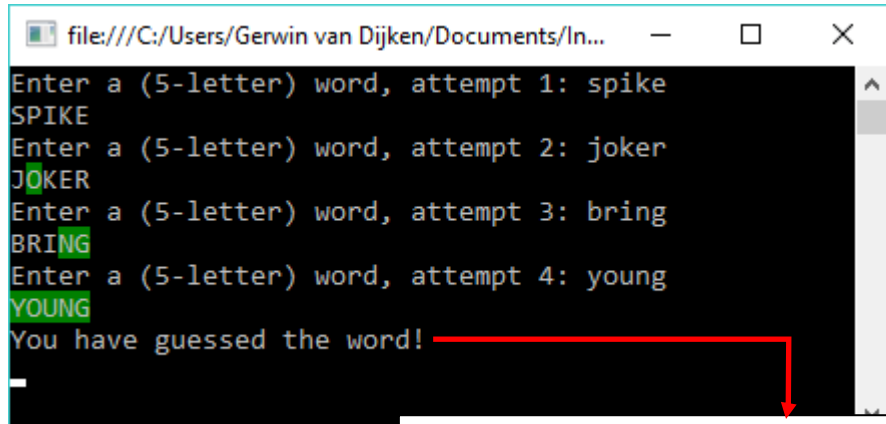


```
file:///C:/U...
Enter a word: morgen
morgen => tomorrow
Enter a word: listall
zijn => to be
de leraar => the teacher
blij => happy
leuk => funny
mooi => pretty
oud => old
jong => young
```

## Assignment 4 - Lingo

In the Programming 2 theory class the Lingo game has been discussed: the main task and the subtasks of the game. Create this application by implementing the mentioned tasks (methods). Of course you can use the pseudocode from the slides.

You can use the file “lingo-words.txt” that can be found on Moodle. Use a command line parameter for the file to use.



```
file:///C:/Users/Gerwin van Dijken/Documents/ln...
Enter a (5-letter) word, attempt 1: spike
SPIKE
Enter a (5-letter) word, attempt 2: joker
JOKER
Enter a (5-letter) word, attempt 3: bring
BRING
Enter a (5-letter) word, attempt 4: young
YOUNG
You have guessed the word!
_
```

Other option is:  
Too bad, you did not guess the word (YOUNG)