

Assignment 2: Selection and Iteration Algorithms

1. Objectives

The main objectives of this assignment are:

- To exercise with conditional, if-else, (switch statements) and iteration (for, do-while and while) algorithms
- To discover some of the method of the `string` object.
- To work with methods having parameters and return type.

A class consists mainly of *data* and *operations*. For data, we declare (define) and use variables. For operations, we write methods. Whatever task a class is expected to do must be coded inside a method.

Methods can be `void` or non-void, returning a value. Void methods perform tasks when they are called, but they do not send any value back to the caller. It can have parameters if it needs more data from the caller, data that is not available or accessible to the method. Returning-value methods work almost in the same way with the exception that they must always send a value back to the caller through their `return` statements.

This assignment is based on the numerous optional exercises that are available on the module. The suggested solutions to the exercises should be of big help.

Important:

- Do not use recursive calls, i.e. calling a method from inside the same method!
- The assignment requires a good amount of programming work but you will receive help for the first two parts in form of a help-document, videos and also live sessions (Code-Along). The contents are programming basics that are necessary to understand and learn to use.

2. Description and requirements

This assignment has four major parts. The first two are required to receiving a C grade. The last two are required for a B and C, respectively.

Assignment 2A (Grade C): Working with strings, and simple if-else and switch statements, loops with `for`, `while` and `do-while` statements. All operations in this part are to be coded in a separate class (**FunFeatures**).

Assignment 2B (Grade C): a class containing some methods that use nested for loops. Create a separate class for this part (**MathWork**).

Assignment 2C (Grade B): Temperature Converter, which gives training in using `for`-loops. In this assignment, we are going to write a program that displays a list of temperatures in Celsius after conversion to Fahrenheit and vice versa. This part should be coded in a separate class (**TemperatureConverter**). This part is only for those aiming for a higher Grade, B

Assignment 2D (Grade A): WorkingSchedule, a class handling a working schedule according to some assumptions. This part is only for those aiming for a higher Grade A.

Summary of the required parts to do for different grades:

Required	For Grade
Assignment 2A+ Assignment 2B	C
Assignment 2A + Assignment 2B+ Assignment 2C	B
Assignment 2A + Assignment 2B + Assignment 2C+ Assignment 2D	A

If your goal is a pass grade C, do only 2A and 2B. If your goal is higher, but if you are going for a higher grade, do the parts required for the grade B and A as specified in the table.

Remember the final grade will be partly based on grades of Assignments 1 to 6. For A and B grades, you must do the A and B part of all assignments.

If your solution shows shortages or not meeting the minimum quality standards, you will have the chance to improve the solution and resubmit. This applies to all grades!

Input Control: not required for grades C and B but required for grade A.

Getting Started:

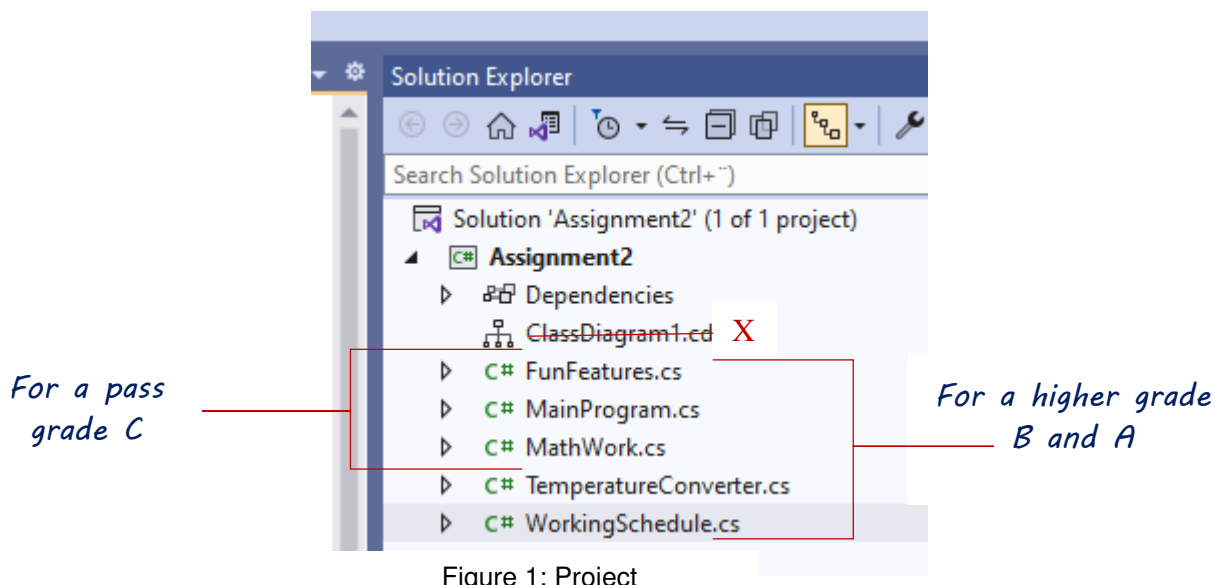
Create a new Desktop Console application in Visual Studio and save the project in a separate folder on your computer. VS creates a solution and a project with a class containing the **Main** method. You may rename the **Program** class to **MainProgram**. This class contains the Main method, which starts the program at run time.

Begin with creating and completing one part at a time (one class). The best practice is to test every time you have a runnable part, for example every time that you write a method, run and test the functionality of the method. Then proceed to the next one.

To test methods of a class you need to create an object of the class in the Main method and then call the method of the class that you want to run. As a good recommendation, write a method in the class that you are using in the Main-method and gather all the methods that are to be called in an orderly manner. Let us refer to this method as Start method. The Start method provides the possibility of controlling the order the class should execute its tasks.

The Project

Figure 1 presents the list of classes used in all the four parts. You can use your own class names if you wish and you can name your variables and methods by yourselves. Use appropriate names for all your identifiers.



3. Assignment 2A – Required for a C Grade

In this part of the assignment, we are going to create a class, **FunFeatures** containing a few methods that perform some tasks. The idea is to exercise with simple string operations and conditional statements. A sample run example of this part is shown below:

```

Strings, selection and iteration in C#

My name is: Anubis and I am a student of the xx semester!

Let me know about yourself!
Your first name please: Homer
Your last name please: Simpson
Nice to meet you Homer!
Give me your email please: sh@theworld.nu

Here is your full name and your email.
SIMPSON, Homer sh@theworld.nu

I am a fortune teller.
Select a number between 1 and 7: 6
Saturday, do nothing and do plenty of it!

Length of a text: Write a text with any number of characters and press Enter
I will then calculate the number of chars included in the text
Give me a text of any length, or press Enter to exit!

We cannot solve our problems with the same thinking we used when we created them.
WE CANNOT SOLVE OUR PROBLEMS WITH THE SAME THINKING WE USED WHEN WE CREATED THEM.
Number of chars = 81

Run again? (y/n)
n
Welcome back, SIMPSON, Homer

PRESS ENTER TO CONTINUE TO THE NEXT PART
    
```

User input, other texts come from the program

Note: The name, Anubus and xx in the above figure are to be replaced by your own name and the current semester. Do not forget this to avoid resubmission.

3.1. The class FunFeatures

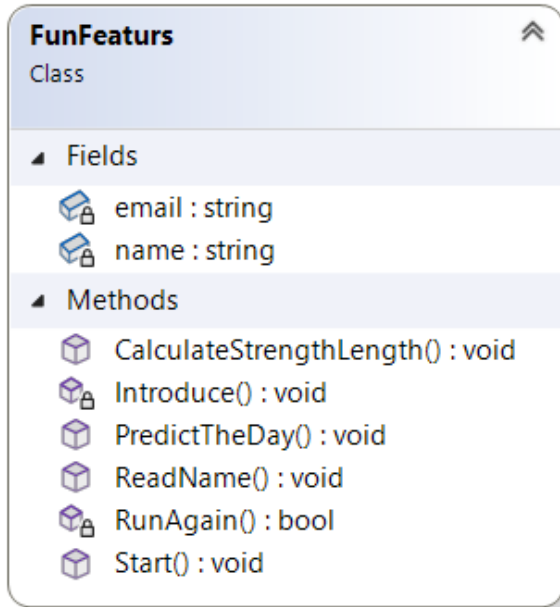
Create a class and name it **FunFeatures**. In this class, we are going to write some methods to perform a number of tasks, which require some simple string manipulation.

Begin with declaring two private instance variables (fields), one for storing the name of a person and another for the email of the same person.

The class diagram at the right show a general structure of the class and a list of the methods. The method **ReadName** is called by the method **Introduce**.

The **Start** method is intended to gather the calls to the methods of the class in an orderly manner, serving as an. interface to the **Main**-method so the **Main** method would only need to call the Start method to run an object of the class.

The code sample below shows calls to all the methods you will be writing in the class. but you can write one and test each method at a time. Write code in the Main method to test the class.



```

class FunFeatures
{
    private string name = "";
    private string email = "";

    1 reference
    public void Start()
    {
        Introduce();

        bool done = false;
        do
        {
            //Call method to read a number 1 to 7 and display the
            //name of the day (1 = Monday, 7 = Sunday) with a comment
            PredictTheDay();

            //Calculate the length of a given text
            CalculateStrengthLength();

            //Run again or exit
            done = RunAgain();
        } while (!done);

        Console.WriteLine("Welcome back, " + name);
    }
}
  
```

The loop (**do-while**) allows the user to run the methods repeatedly. The implementation of the **RunAgain** method is described below.

Question: Why **do-while** in the above code? Could a **while** or a **for** statement be used?

Note: Questions are only for you think about. You don't have to include the answers in your submission. Make sure you understand what happens in every line of the above code clip. Use the forums to ask questions or discuss ideas!


To run the Start method, you need to create an object of the calls and then call it in the Main method.

```
internal class MainProgram
{
    0 references
    static void Main(string[] args)
    {
        Console.Title = "Strings, selection and iteration in C#";
        FunFeatur funObj = new FunFeatur();
        funObj.Start();

        ContinueToNextPart(); //call the method below
    }

    1 reference
    private static void ContinueToNextPart()
    {
        Console.WriteLine("\nPRESS ENTER TO CONTINUE TO THE NEXT PART");
        Console.ReadLine();
        Console.Clear();
    }
}
```

Question: why static?



3.1.1. The **RunAgain** method

Back to the **FunFeatures** class. Define this method with a return type bool. Ask the user whether to run again or exit to go on with the next part. If the answer is "yes" or "y", return true otherwise return false, implying that the loop is to be stopped.

3.1.2. The **Introduce** method:

Write a method in which you begin to introduce yourself as the programmer of the application and then ask the user's first and last names as well as the user's email. It is not necessary to validate for email's format.

Convert the last name to upper case and combine it with the first name before saving it in the instance variable name. Print a string with the user's name and email to the Console. See the run-time example in the image above.

Try to isolate some of the functionalities of each method into a separate method. For instance, you can write a **Readname** method to handle getting the first and last names from the user. Call this method in the Introduce method. Both methods have access to the instance variable name.

Run and test this part before you move to the next task.

3.1.3. The **PredictTheDay** method:

The method prompts the user for a number 1 to 7 and the program displays the name of the day corresponding to the number (1 = Monday, 7 = Sunday) and some funny comment. If the user provides a number that is out of the range (not between 1 and 7), a message is to be given to the user. You may use the following phrases in your code as comments.

```
"Monday: Keep calm my friend! You can fall apart!"
"Tuesday and Wednesday break your heart!"
"Thursday, OMG, still one day to Friday!"
"It's Friday! You are in love!"
"Saturday, do nothing and do plenty of it!"
"And Sunday always comes too soon!"
"Not in a good mode? This is not a valid date!"
```

Requirement: use a `switch` statement in this part.

3.1.1. The **CalculateStringLength** method:

In this method, let the user input a string of any length and press Enter. The method should then read the input and calculate the number of characters, using the Length property of the inputted string.

Include a call to this method in the Start-method.

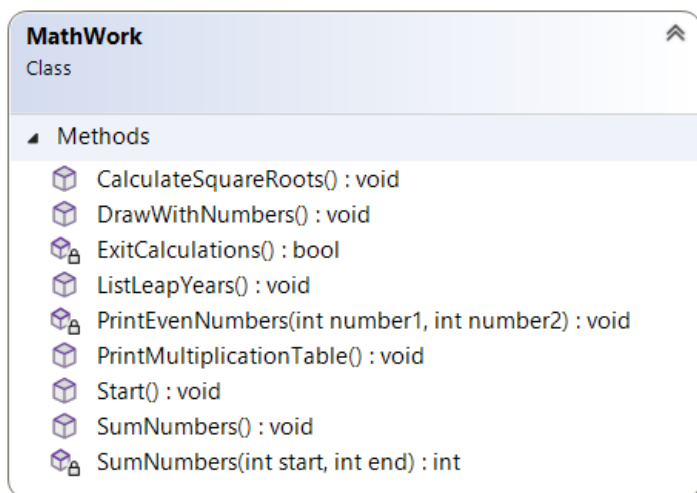
4. Assignment 2B:– Required for a C Grade

4.1. The class **MathWork**

In this class, we are going to write four methods to exercise with numbers using both single and nested loops. In the previous class, we used a loop in the Start method. In this part, we run a loop inside some of the methods.

4.1.1. The **SumNumbers** method

Write a method that takes two numbers of type `int` as method parameters, **startNum** and **endNum**. The method



should run a **for**-loop to add all numbers between and including these two limits. For instance, if `startNum = 2` and `endNum = 5`, the result should be $2+3+4+5 = 14$.

Write another method **SumNumbers** () with no method parameters. This method should ask the user to give a start and an end integer number. Call the method **SumNumbers** (as explain above), passing **startNum** and **endNum** as values to the parameters.

Requirements:

- Use a do-while loop (or a while-loop) for repeating the calculations, and use a for-statement to sum up numbers.
- If the start number is greater than the end number, swap the values so that the start number becomes the end number and the end number gets the value of the start number. Look at the run-time example below where the user gives 23 as the start number and 12 as the end number. The method has made the correction before adding the numbers from 12 to 23.

```

Let's do some mathematics!

Sum numbers between any two numbers
Give start number: 23
Give end number: 12
The sum of numbers between 12 and 23 is: 210

Exit calculations? (y/n)
n

Sum numbers between any two numbers
Give start number: 

```

4.1.2. The **WriteAllEvenNumbers** method

This method should be called in the above method, just before calling "Exit calculations". Its task is to find and print out all even numbers between the start number and end number inputted by the user in the above method.

Write a method that takes two parameters, number 1 and number2 of double type. Pass the values of the start and the end number in the code for the above method.

SumNumbers

PrintEvenNumbers

```

Sum numbers between any two numbers
Give start number: 12
Give end number: 23
The sum of numbers between 12 and 23 is: 210

Even numbers between 12 och 23
12  14  16  18  20  22

Exit calculations? (y/n)

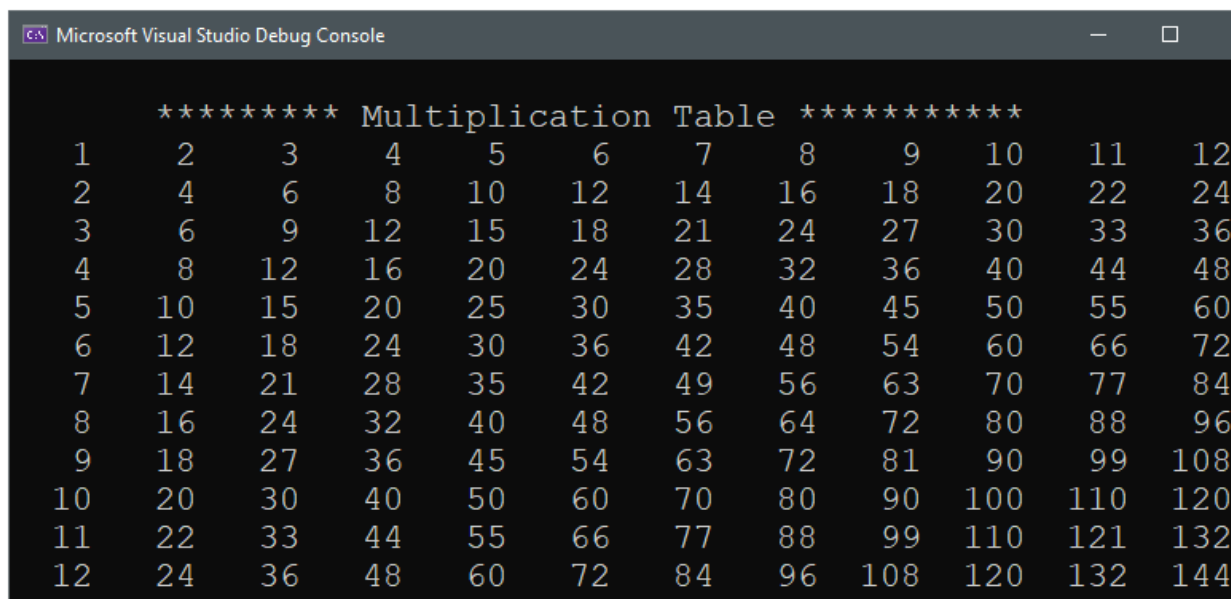
```

4.1.3. The **WriteMultiplicationTable** method

Write a method to print a multiplication table for numbers 1 to 12, and present the results into a table-like chart using `string.Format` as below:

Run a nested loop for rows 1 to 12 and columns 1 to 12: A product value = row * col (e.g. 12 * 12 = 144).

```
Console.WriteLine(string.Format("{0,4:d} ", row * col));
```



The screenshot shows the output of the `WriteMultiplicationTable` method in the Visual Studio Debug Console. The output is a 12x12 multiplication table with a title "***** Multiplication Table *****" and numbers 1 to 12 for both rows and columns. The numbers are formatted with a width of 4 characters.

1	2	3	4	5	6	7	8	9	10	11	12
2	4	6	8	10	12	14	16	18	20	22	24
3	6	9	12	15	18	21	24	27	30	33	36
4	8	12	16	20	24	28	32	36	40	44	48
5	10	15	20	25	30	35	40	45	50	55	60
6	12	18	24	30	36	42	48	54	60	66	72
7	14	21	28	35	42	49	56	63	70	77	84
8	16	24	32	40	48	56	64	72	80	88	96
9	18	27	36	45	54	63	72	81	90	99	108
10	20	30	40	50	60	70	80	90	100	110	120
11	22	33	44	55	66	77	88	99	110	121	132
12	24	36	48	60	72	84	96	108	120	132	144

End of parts for the grade C

If you are satisfied with a C grade, you can skip parts 2C and 2D. Go directly to "Help and Guidance" section. Otherwise if you wish to qualify for a higher grade, proceed with parts 2C and 2D as described below.

5. Assignment 2C: Temperature Converter – required for grade B

In this part, you are going to write the class **TemperatureConverter** in which you first show a menu to the user with the options shown in figure here.

When option 1 is selected by the user at run-time, the program (your class) lists values from **0** to **100** Celsius degrees converted to Fahrenheit degrees.

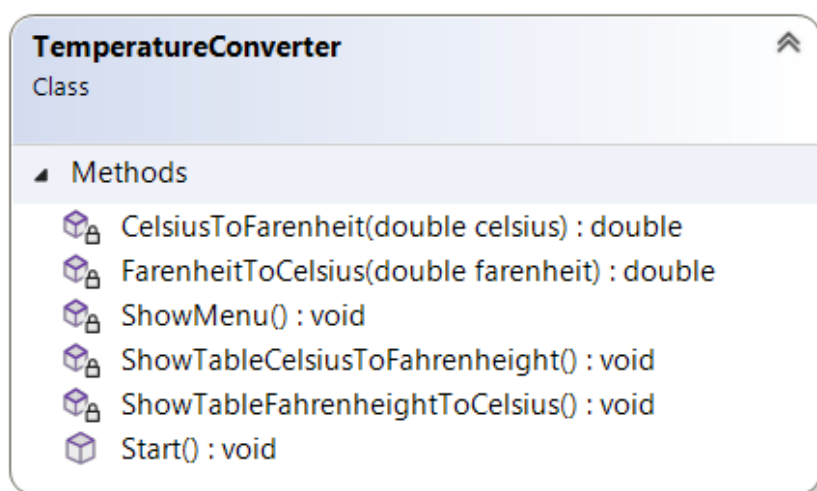
When option 2 is chosen, the program calculates and displays a list of values between **0** and **212** degrees in Fahrenheit converted to Celsius degrees. Use a constant for the values 100 and 212 in the related method. The following local constant can be placed in the method that calculates Celsius to Fahrenheit.

```
const int max = 100;
```

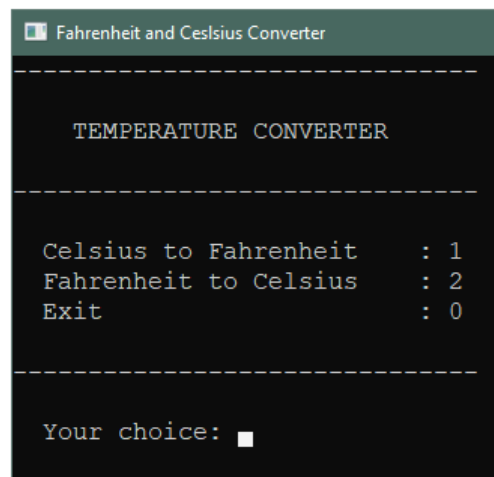
Display the result in intervals. You can try any intervals such as 1, 4 or 5. In the run example (next page) results are shown for every 4th value for the option C to F and every 10th for F to C. Use the conversion formulas below. Create a method for each formula.

```
F = 9/5.0 * C + 32 //separate method
C = 5/9.0 * (F - 32) //separate method
```

where F = Fahrenheit and C = Celsius. Do not use short variables like F and C in your code. Use words or combination of words in naming variables and methods. A class diagram showing methods of the TemperatureConverter class is presented below.



The images on the next page are screen shots of a running session, which are placed side by side to save space. When running the program, they are displayed one at a time on the Console window.



```

Fahrenheit and Celsius Converter

-----
MAIN MENU
-----

Celsius to Fahrenheit      : 1
Fahrenheit to Celsius     : 2
Exit the program          : 0
-----

Your choice: 1
 0.00 C =    32.00 F
 4.00 C =    39.20 F
 8.00 C =    46.40 F
12.00 C =    53.60 F
16.00 C =    60.80 F
20.00 C =    68.00 F
24.00 C =    75.20 F
28.00 C =    82.40 F
32.00 C =    89.60 F
36.00 C =    96.80 F
40.00 C =   104.00 F
44.00 C =   111.20 F
48.00 C =   118.40 F
52.00 C =   125.60 F
56.00 C =   132.80 F
60.00 C =   140.00 F
64.00 C =   147.20 F
68.00 C =   154.40 F
72.00 C =   161.60 F
76.00 C =   168.80 F
80.00 C =   176.00 F
84.00 C =   183.20 F
88.00 C =   190.40 F
92.00 C =   197.60 F
96.00 C =   204.80 F
100.00 C =  212.00 F
-----

MAIN MENU
-----

```

```

MAIN MENU
-----

Celsius to Fahrenheit      : 1
Fahrenheit to Celsius     : 2
Exit the program          : 0
-----

Your choice: 2
 0.00 F =   -17.78 C
10.00 F =   -12.22 C
20.00 F =    -6.67 C
30.00 F =    -1.11 C
40.00 F =     4.44 C
50.00 F =    10.00 C
60.00 F =    15.56 C
70.00 F =    21.11 C
80.00 F =    26.67 C
90.00 F =    32.22 C
100.00 F =   37.78 C
110.00 F =   43.33 C
120.00 F =   48.89 C
130.00 F =   54.44 C
140.00 F =   60.00 C
150.00 F =   65.56 C
160.00 F =   71.11 C
170.00 F =   76.67 C
180.00 F =   82.22 C
190.00 F =   87.78 C
200.00 F =   93.33 C
210.00 F =   98.89 C
-----

MAIN MENU
-----

```

You can display the results divided into columns as shown here (two columns).

```

Fahrenheit and Celsius Converter

-----

Your choice: 1
 0.00 C = 32.00 F    4.00 C = 39.20 F
 8.00 C = 46.40 F   12.00 C = 53.60 F
16.00 C = 60.80 F   20.00 C = 68.00 F
24.00 C = 75.20 F   28.00 C = 82.40 F
32.00 C = 89.60 F   36.00 C = 96.80 F
40.00 C = 104.00 F  44.00 C = 111.20 F
48.00 C = 118.40 F  52.00 C = 125.60 F
56.00 C = 132.80 F  60.00 C = 140.00 F
64.00 C = 147.20 F  68.00 C = 154.40 F
72.00 C = 161.60 F  76.00 C = 168.80 F
80.00 C = 176.00 F  84.00 C = 183.20 F
88.00 C = 190.40 F  92.00 C = 197.60 F
96.00 C = 204.80 F 100.00 C = 212.00 F
-----

TEMPERATURE CONVERTER

```

6. Assignment 2D - WorkingSchedule (WorkingSchedule.cs)

Skip this part if you are not going for a grade A.

In this part, you are given a chance to analyze the problem and decide what kind of a loop to implement.

Requirements for Grade A

In addition to Assignments B, you have to solve the following problem.

- 5.1 Control the user-input by using the **TryParse** method instead of **Parse**. It is acceptable if you implement this requirement only in this part. Use **TryParse** (for example **int.TryParse**, and **double.TryParse**) whenever you expect a number from the user. Refer to C# documentation on Internet to find out how TryParse works if you are not familiar with it.
- 5.2 Write a class **WorkingSchedule** to program a schedule for a worker who works some night shifts, and some weekends. The user of this option is an employee who has to work every **other (second)** weekend with start from week number **1**. She/he has also to work a night shift every **4th** week with start from week **1**. The schedule is to cover a **one-year** period. Assume that the last week of the year is **52**.

Your program should display a list of the weeks that the user has to work the weekends and the weeks when she/he should work nights

- 5.3 To interact with the user, you may follow the scenario below:

5.3.1 A menu is shown to the user for selecting the type of schedule, Weekends or Nights.

5.3.2 Display a list of weeks she/he scheduled for the selected option.

Hint: It is possible to write one method that can work for both options, if you think of startWeek, endWeek and interval as variables, when running a loop.

- 5.4 You have to explain your motivation briefly to that verifies your choice. You can do this by using comments in the code file above the class definition.
- 5.5 Write also a Start method to be called in the Main method.
- 5.6 The loop continues until the user enters a zero.

```

Working Schedule
YOUR WORK SCHEDULE
-----
1 Show a list of the weekends to work.
2 Show a list of the nights to work.
0 Exit
Your choice: 2
Week 1    Week 4    Week 7    Week 10
Week 13   Week 16   Week 19   Week 22
Week 25   Week 28   Week 31   Week 34
Week 37   Week 40   Week 43   Week 46
Week 49   Week 52
-----
1 Show a list of the weekends to work.
2 Show a list of the nights to work.
0 Exit
Your choice: 1
Week 1    Week 3    Week 5    Week 7
Week 9    Week 11   Week 13   Week 15
Week 17   Week 19   Week 21   Week 23
Week 25   Week 27   Week 29   Week 31
Week 33   Week 35   Week 37   Week 39
Week 41   Week 43   Week 45   Week 47
Week 49   Week 51
-----
1 Show a list of the weekends to work.
2 Show a list of the nights to work.
0 Exit
Your choice:

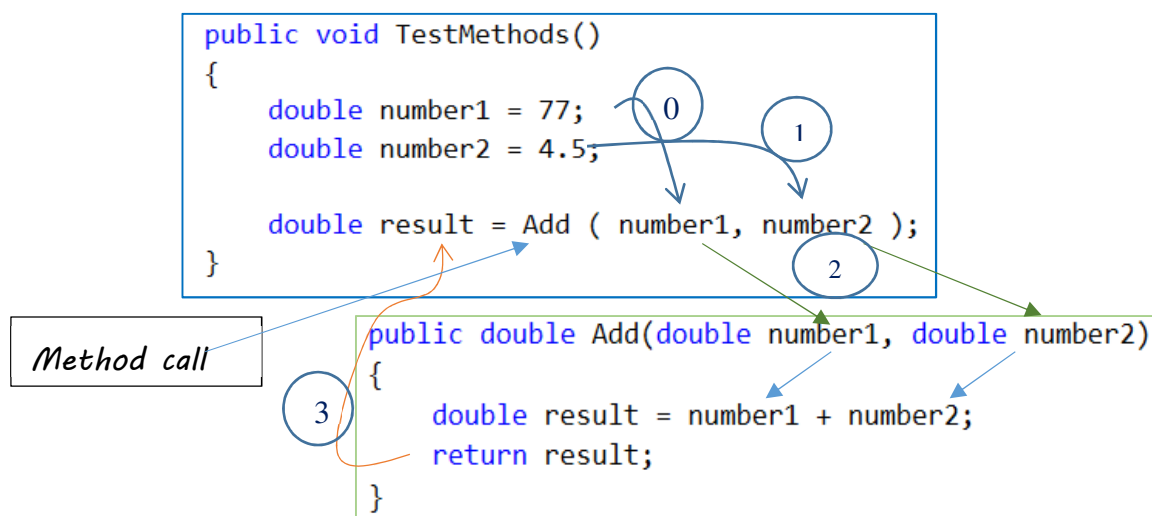
```

7. Help and Guidance:

See the document [string.Format](#), and other help material, available on the module. Do all the optional exercises as they contain code useful in this assignment.

7.1. Method with parameters and a return value

In code snippet that follows, you can see a method that has two parameters (arguments). The parameters are to receive input values from the caller method. The method then sends back a value, the result of the calculation, to the caller through the [return](#) statement.



Note that variables `number1` and `number2` in the method **Add** are not the same variables as in the caller method **TestMethods** although they have same names, but they will receive the values of `number1` and `number2` as they are passed to them. The parameters could have any other names.

7.2. Working with string

The class `string` (or `String`) has several useful methods and properties (special type of a method) that can be used to use with strings. Also string variables contain such features which make life a lot easier when working with strings.

`stringVariable.Length` (property): returns the number of chars in a given text.
`stringVariable.ToLower` (method): returns the string converted to its lowercase.
`stringVariable.ToUpper` (method): returns the string converted to its uppercase

```
string response = Console.ReadLine();
response = response.ToLower(); //change to lower case
```

Notice carefully how a string variable is uses its own features. `response = response.....`

`string.Format`:

```
string strOut = string.Format("{0} {1} {2,6:f2}", name, count, price);
```

The method `string.Format` has two parts: a formatting part enclosed within " ", followed by a comma and a variable list, counted from 0. In the example above, variable 0 is name, 1 is count and 2 is price. The expression `{0}` will be replaced by the value of the first variable, i.e. name. `{1}` will be replaced by value of count. `{2,6:2f}` will be replaced by variable 3, i.e. price, formatted within 6 characters, after rounding off to 2 decimals (the text 19.99 containing 4 chars will receive two blank spaces to its left).

```
public void FormatString()
{
    string name = "Egg";
    int count = 24;
    double price = 19.99;

    string strOut = string.Format("{0} {1} {2,6:f2}", name, count, price);
    Console.WriteLine(strOut);

    Console.WriteLine("Length: " + strOut.Length); //No parentheses after Length
    Console.WriteLine(strOut.ToUpper()); //Parentheses after ToUpper
}
```

Output:

```
Egg  24   19.99
Length:  13   //Call to the property (special method) Length
EGG 24   19.99 //Call to the method ToUpper()
```

8. Submission

Compress all the files, folders and subfolders into a **zip** or **rar** file, and then upload it via the Assignment page on Canvas

Good Luck!

Programming is fun. Never give up. Ask for help!

Farid Naisan,

Course Responsible and Instructor