Assignment 2: Selection and Iteration Algorithms

1. Objectives

The main objectives of this assignment are:

- To learn some basic string manipulation.
- To exercise with conditional algorithms, if-else, (switch statements) and iteration statements (for, do-while and while)
- 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 method. It can have parameters, if it needs more data from the caller method, data that is not available or accessible to the method.

```
private void PrintEvenNumbers(int number1, int number2)
{
    //Code
}
```

Returning-value methods work almost in the same way with the exception that they must always send a value back to the caller method through their return statements. The value must have the same type as defined in the method signature.

```
private int SumNumbers(int start, int end)
{
    int sum = 0;
     //Other code

    return sum;
```

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 in solving this assignment.

Important:

- Do not use recursive calls, i.e. a method that is calling itself!
- 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 the syntax, before going over to more advanced topics.

2. Description and requirements

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

Assignment 2A (Grade C), FunFeatures: Working with strings, 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, (**FunFeaturs.cs)**.

Assignment 2B (Grade C), MathWork: a class for performing some simple mathematical calculations. It contains some methods that use nested loops. Create a separate class for this part (**MathWork.cs**).

The following two sections are required only for a higher grade, (B and A), not required for the C grade.

Assignment 2C (Grade B), **TemperatureConverter:** In this part, you are requested to write a class that displays a list of temperatures in Celsius degrees converted to Fahrenheit and vice versa, a list of Fahrenheit degrees converted to Celsius. This part should be coded in a separate class (**TemperatureConverter.cs**).

Assignment 2D (Grade A), **WorkingSchedule:** Write a class that shows a working schedule according to some given assumptions (described later).

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, but if you are going for a higher grade, do the parts required for the grade B and A as specified in the table above.

Remember that the final grade for all the assignments will be based on the results of Assignments 1 to 6. For A and B grades, you must do the A and B parts of this and all the future assignments.

If your solution shows shortages, or does not meet 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 a grades C and B but required for grade A (described later).

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 including a class, **Program**.cs 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 (one class at a time). The best practice is to test the code every time you have a runnable part, for example when you write a method, run the program and test the functionality of the method. Then proceed with the next one.

To test methods of a class, you need to create an object of that class in the **Main** method and then call the method or methods of the class that you want to run. As a good recommendation, write a method (e.g. Start) 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 the 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.

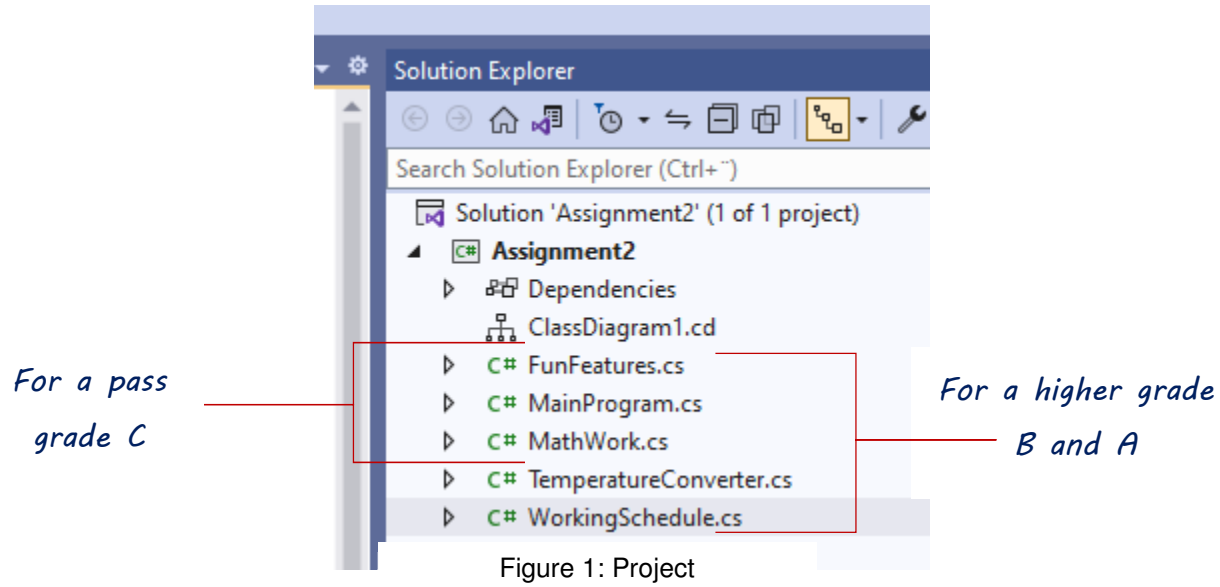


Figure 1: Project

Assignment 2A – Required for a C Grade

To solve this part of the assignment, create a class, **FunFeatures.cs** and write the methods described below. The idea is to exercise with simple string operations and conditional statements.

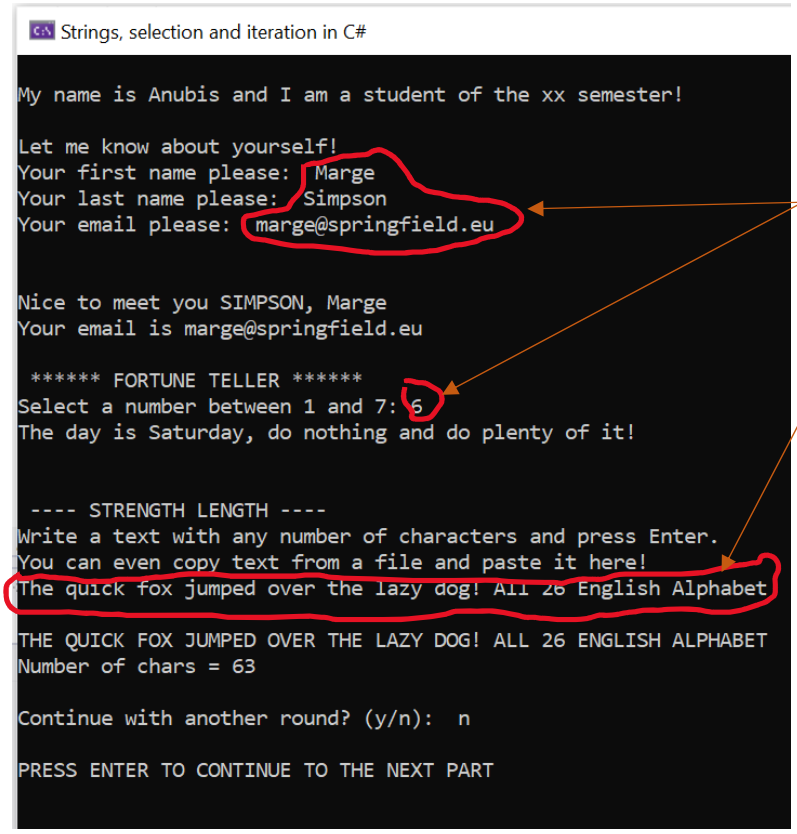


Figure 2 – A run time example

2.1. The class FunFeatures

In this class, you are going to write some methods to perform a number of tasks, which require some simple string manipulations. The above image shows how this part is expected to execute at run time.

Create a new class and name it **FunFeatures** and save it as **FunFeatures.cs.** Begin your coding with declaration of two private instance variables (fields), one for storing the name of a person and another for storing the Email of the same person.

The class diagram shows a general structure of the class with its two fields, and a list of the methods. The type of the fields and the return

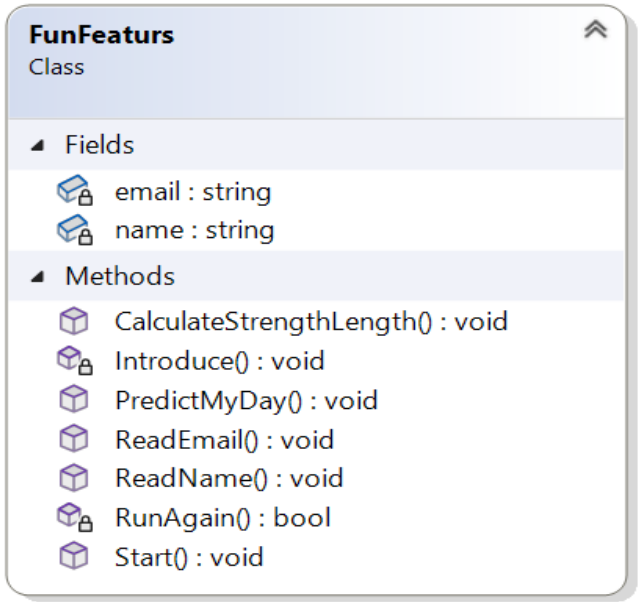


Figure 3: Class Diagram for **FunFeatures**

type of the methods are shown after the colon symbol, e.g., both name and email are string variables, and all methods are void. The methods **ReadName** and **ReadEmail** are called by the method **Introduce**.

Note: The class diagram lists the methods in an alphabetic order. Follow the order that is given in the **Start** method as shown in code clip given below.

```
class FunFeaturs
{
    private string name = "";
    private string email = "";

    1 reference
    public void Start()
    {
        //Get some user information
        Introduce();

        bool repeat = 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
            PredictMyDay();

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

            repeat = RunAgain();
        }while (repeat);
    }
```

Figure 4: -Start method, FunFeature class

Question: *Is the Start-method mandatory in every class? Can a class be tested without the Start method?*

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

***Note**: Questions are only for you to think about some important concepts. You don't have to include the answers in your submission, but make sure that you understand and have the answers in your mind. Use the forums to discuss ideas, if you not certain about concepts!*

The **Start** method as shown in the above code clip collects the calls to the methods of this class in an orderly manner. These are the methods that you will be writing to accomplish a functionality like the one shown in the above image. The Start methods serves as an interface to other classes. This way, the **Main** method would only need to call the Start method to run an object of this class. Only the Start method needs to be declared as public. All other methods in the class can remain private.

Note: The Start method is no way a part of the C# language. It is a method that we are using to organize our code.

To run the Start method, you need to create an object of the **FunFeature** class in the **MainProgram**'s **Main**-method, and then call the object's **Start** method.

Begin writing the methods in the order given in the Start method. As pointed out earlier, you can write one method at a time and test it before proceeding with the next one. To test, you need to create an instance of the class in the **Main** method and then call the method **Start**, as shown in the code clip in Figure 5.

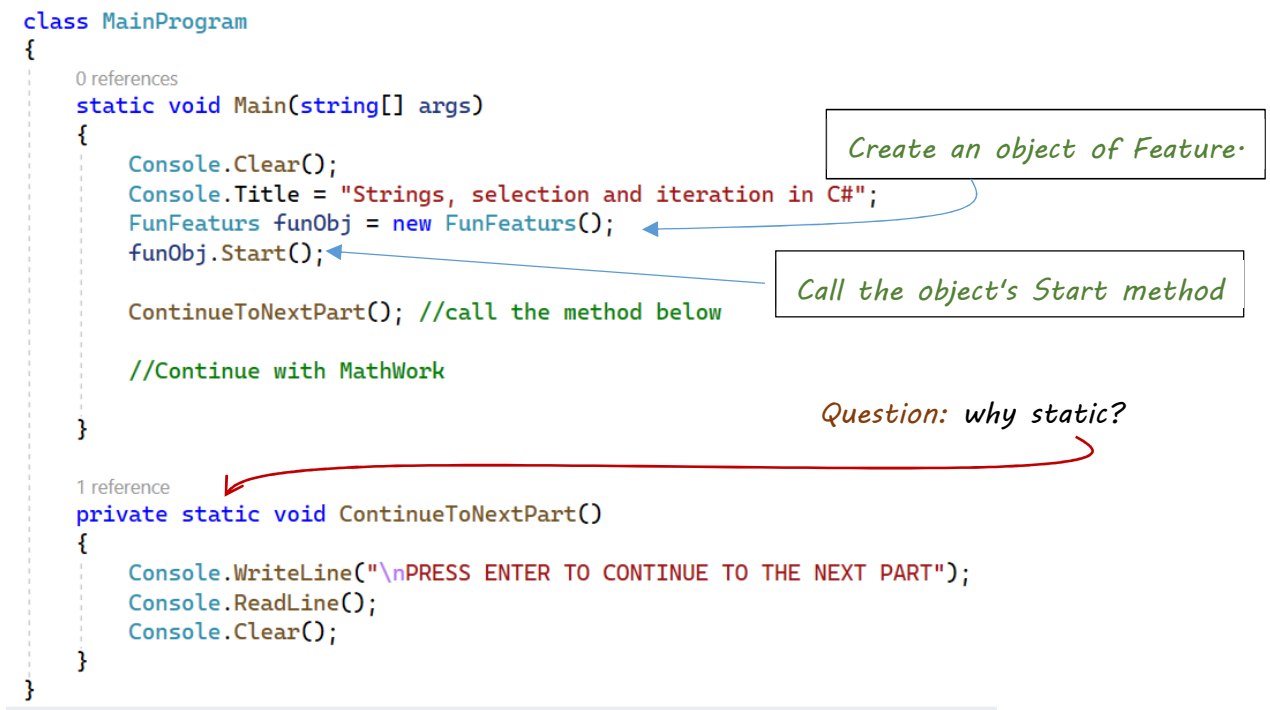


Figure 5: -Change the Main-method to run the Start method of the class FunFeatures

Back to the **FunFeatures** class and let's now write the methods. When you test one method at a time, you can comment out the call to the method which you have not yet completed. Begin with the Introduce method and comment out the whole do-while block using /* …*/ block commenting symbols.

2.1.1. 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 welcome text with the user's name and email to the Console. See the run-time example in the image above (Figure 2).

Try to encapsulate some of the functionalities of a method into separate methods. For instance, you can write a **ReadName** method to handle reading the first and last names from the user, formatting it as explained above, and saving it in the instance variable **name**. Call this method in the Introduce method. This way, you move a part of the Introduce-method's burden to a separate

method, making the code easier to read and easier to maintain. **Hint**: save the first and the last name each in a local variable of the type string.

Write another method **ReadEmail** in which you ask the user to input her/his email. Save the given string in the instance variable email. Call also this method from the **Introduce** method.

Remember that instance variables are accessible for all method of a class. Run and test this part before you move to the next task.

2.1.2. The **RunAgain** method

In this method, ask the user whether to continue or exit. If the answer is "yes" (or "y"), return true otherwise return false. Make sure that you use the return value (true/false) properly in the caller method. Notice how it is used in the **Start** method of this class; if the call to this method return true, the loop continues with another iteration and if it is false, the loop ends.

2.1.3. The **PredictMyDay** method:

The method prompts the user for a number 1 to 7 and the program displays some funny comment with the name of the day corresponding to the number (1 = Monday, 7 = Sunday). If the user provides a number that is out of the range (not between1 and 7), a message is to be given to the user. You may use the following phrases in your code as comments.

```
"Keep calm on Mondays! You can fall apart!"

"Tuesdays and Wednesdays break your heart!"

"Thursday is your lucky day, don't wait for Friday!"

"Friday, you are in love!"

"Saturday, do nothing and do plenty of it!"

"And Sunday always comes too soon!"

"No day? is a good day but it doesn't exist!"
```

Requirement: use a switch statement in this part, to determine which day corresponds to the user's given choice.

2.1.1. 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 (a string).
- Calculate the number of characters in the given text, using the Length property of the inputted string.
- Convert the string to upper case (capital letters)
- Display the string back to the user (Prompt Window)
- Display the number of characters in the string.

Assignment 2B:– Required for a C Grade

2.2. The class **MathWork**

In this class, we are going to write the methods included in the class diagram shown in Figure 6. The purpose of writing this class is to exercise with numbers as well as nested loops.

Note from the diagram that this class has **no instance variables**.

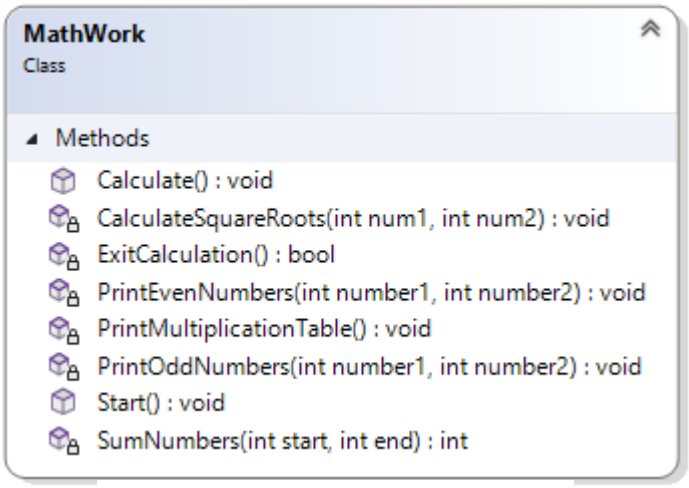


Figure 6: -Class Diagram for **MathWork**

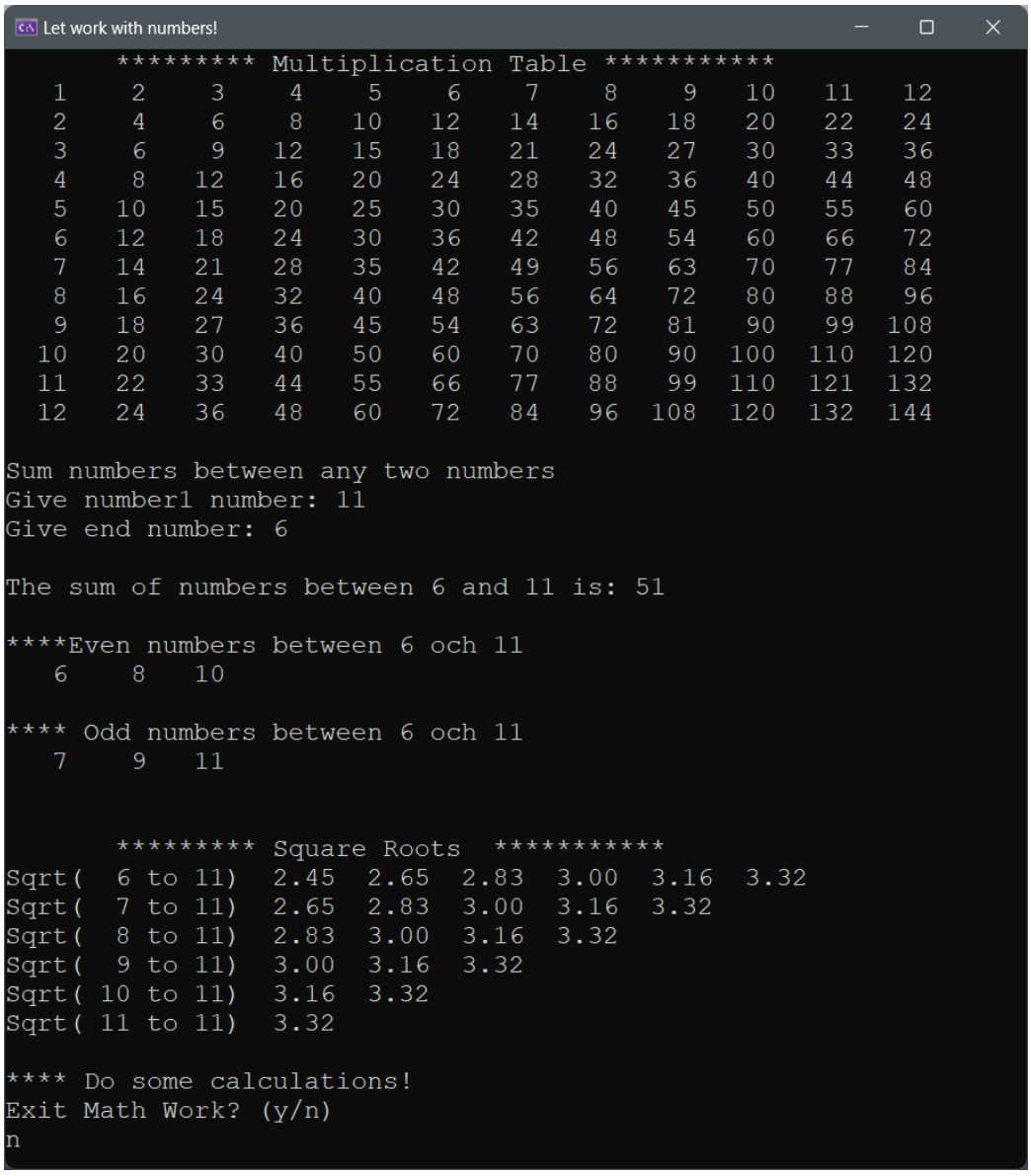


Figure 7: -A run-time execution of the Start method

As in the previous case, we use a **Start** method in which we call the methods in an orderly manner. We also change the Main-method to call the Start method. Copy the following code in the class.

```
class MathWork
{
    public void Start()
    {
       PrintMultiplicationTable();
       Calculate();
    }
```

Change the Main-method in the **MainProgram** as shown below.

```
static void Main(string[] args)
{
    Console.Clear();
    Console.Title = "Strings, selection and iteration in C#";
    FunFeaturs funObj = new FunFeaturs();
    funObj.Start();

    ContinueToNextPart(); //call the method below

    //Continue with MathWork
    Console.Title = "Let work with numbers!";
    MathWork calc = new MathWork();
    calc.Start();

}
```

2.2.1. The **PrintMultiplicationTable** method

Write a method that calculates and displays a multiplication table for numbers1 to 12 and present the results into a table-like chart using string.Format, as demonstrated in the run-time example above. This method does not need to have any parameter or a return type, i.e. you can use a void method.

```
private void PrintMultiplicationTable()
{

}
```

To accomplish this task, you need to use a nested **for**-loop. Run a loop for rows 1 to 12 and for columns 1 to 12. Calculate and display (using Console.Write) the product value = row * col (e.g. 12 *12 = 144).

2.2.2. The **Calculate** method

This method should do a few different calculations with two numbers given by the user. Write a method **Calculate ()** with no method parameters (void). The method should ask the user to give a start and an end integer numbers, and, to begin with, sum up them and display the sum, as demonstrated in the above run-time example. Proceed as described below.

a. Read a start and end number from the user. Save these in local variables. Let's refer to these variables as **startNum**, **endNum**.

b. Compare the numbers so **startNum** is less than or equal to **endNum**. If the check fails, swap the numbers such that **starNum** gets the value of **endNum** and **endNum** gets the value of the **startNum**. Look at the run-time example once again where the user gives 11 as the start number and 6 as the end number. The method has swapped the values before adding the numbers (6 to 11).

c. Call the methods as listed in the **Calculate** method below. Complete the implementation of the methods as described afterwards.

```
public void Calculate( )
{
     int startNum = 0;
     int endNum = 0;


     //Use a while-statement to repeat the following
     //call the method SumNumbers, get the sum and display it on
     //the Prompt Window
          int sum = SumNumbers(startNum, endNum);
     //call the method to determine and display the even numbers
     //in the range startNum to endNum
      PrintEvenNumbers(startNum, endNum);

     //Do the same for odd numbers in the range
      PrintOddNumbers(startNum, endNum);

     //Call a method to calculate and display the square root of
     //each number in the range startNum to endNum and in every
     //iteration, also calculate from the current number to endNum
     //This is explained later in the document.
      CalculateSquareRoots(startNum, endNum);
     //Ask the user whether to exit or continue
          done = ExitCalculation();
}
```

2.2.3. The **SumNumbers** method

This method should sum up the numbers from the startNum to endNum including these two limits and return the sum. For instance, if startNum = 2 and endNum = 5, the result should be 2+3+4+5 = 14.

Write a method **SumNumers** with two parameters of the type int and also a return value of int. The parameters can be called startNum and endNum or anything else like start and end as in the code clip below.

```
private int SumNumbers(int start, int end)
{
    //Code
}
```

Note: the actual values passed to method parameters are called method **arguments**.

Question: *What had happened if start and end had the names startNum and endNum, respectively? How do the parameters affect the value of the corresponding argument in the caller method?*

2.2.4. The **PrintEvenNumbers** method

Write a void method that finds out and prints all even numbers in the interval startNum and endNum. Let these two be method parameters. Use a **for**-loop in this method too.

2.2.5. The **PrintOddNumbers** method

In much the same way, write a void method that determines and displays all odd numbers in the interval.

2.2.6. The Calculate **SquareRoots** method

Write a method that calculates the square root of all the numbers between start and end numbers exclusive these two, and display the calculated values as shown in the run-time example in Figure 7. The start number and end numbers should come as argument to this method. Study how the results are presented. For each number in the range, the value of its square root is calculated and then also the square root of the subsequent numbers up the endNum are presented. For example, if the startNum is 6 and the endNum is 11, proceed as follows:

In iteration one, the number (counter) is 6:

Calculate and write the square root of: 6, 7, 8, 9, 10, 11

Then:

Calculate and write the square root of: 7, 8, 9, 10, 11 and so on.

To realize the above pattern, you should run a nested loop. In the first loop, iterate from the startNum to endNum, writing the value of the the counter. In the second loop, start from counter

and continue to endNum, calculate the square root of the counter of the second loop and display the value using Console.Write (to put values on the same line) .

Note: The square root of a number can be obtained by using **Math**.Sqrt(number). **Math** is a class in .NET and has many usefule methods.

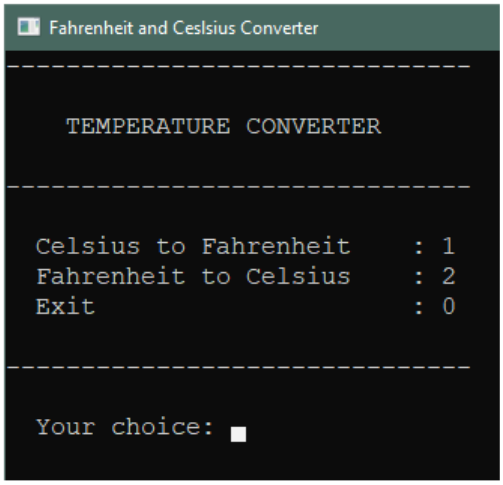
This is the end of the requirements for the grade C. **Note**: Your solution as well as the display of the results do not need to be extactly as descdribed here.

There is a help document related to the above two assignmentes. A good advice is, however, that you think about the solution by yourself, try to write your own code first, and then compare it with the suggestions provided in the help document.

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

3. 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) is to list 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 variable for each limit, 100 and 212, in the related method.

The following local constant can be placed in the method that calculates Celsius to Fahrenheit.

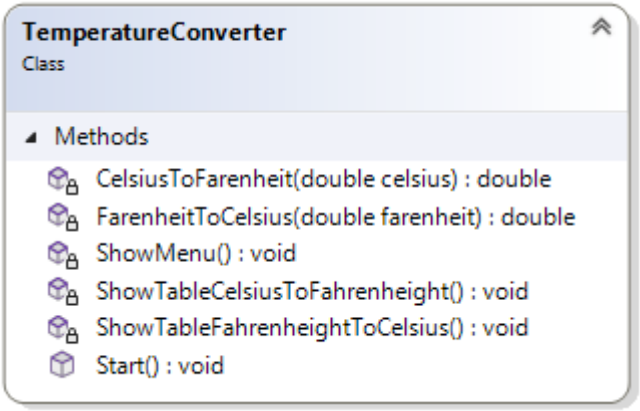
```
const int max = 100;
```

To limited the size of the displayed rows, the result can be shown 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.

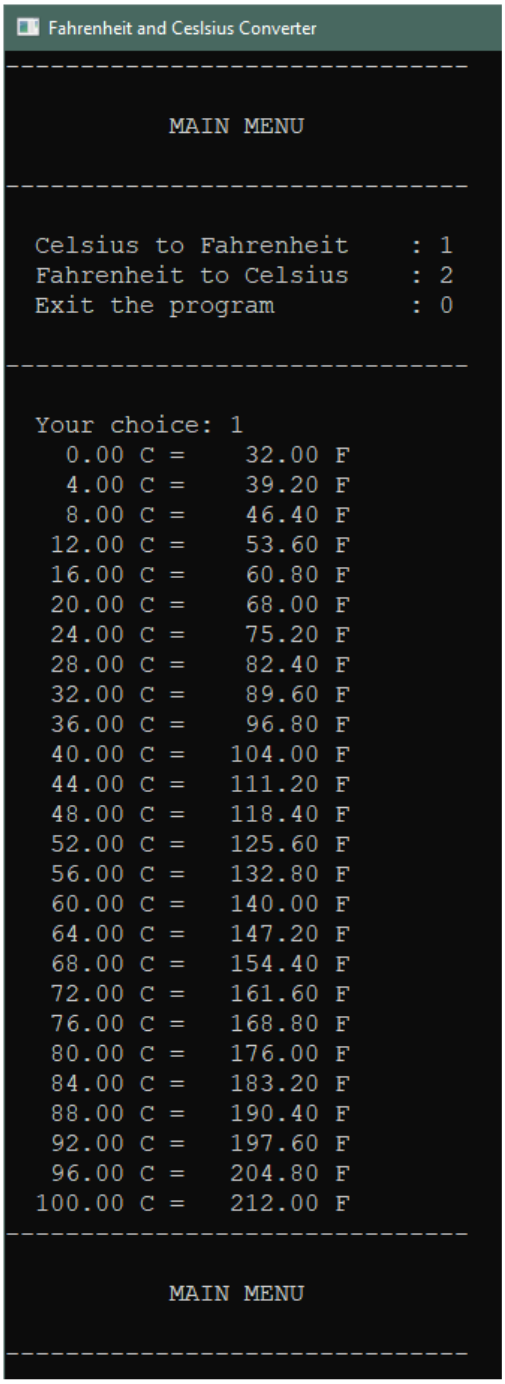
Use the conversion formulas below. Create a method for each formula.

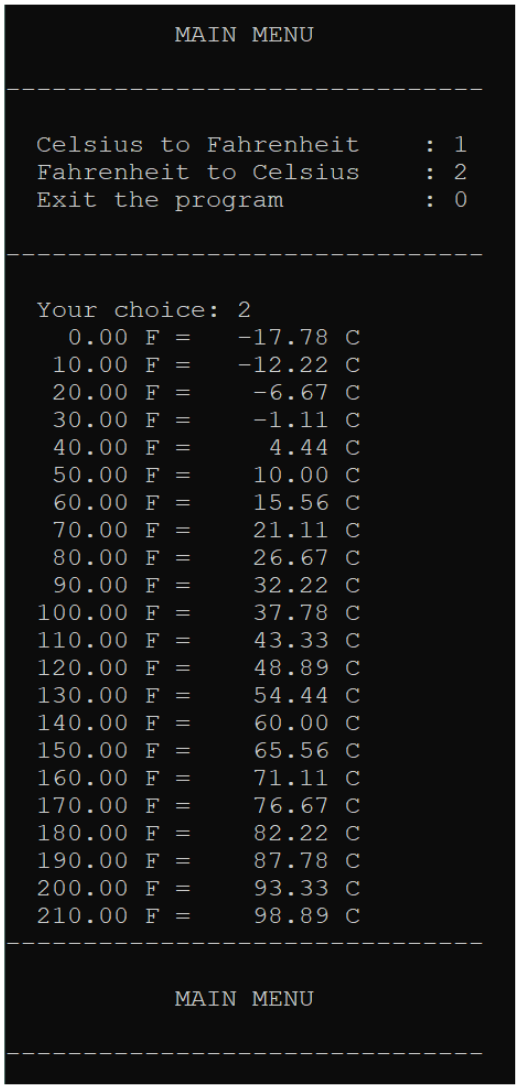
```
F = 9/5.0 * C + 32  //separate method
C = 5/9.0 *(F - 32) //separate method
```

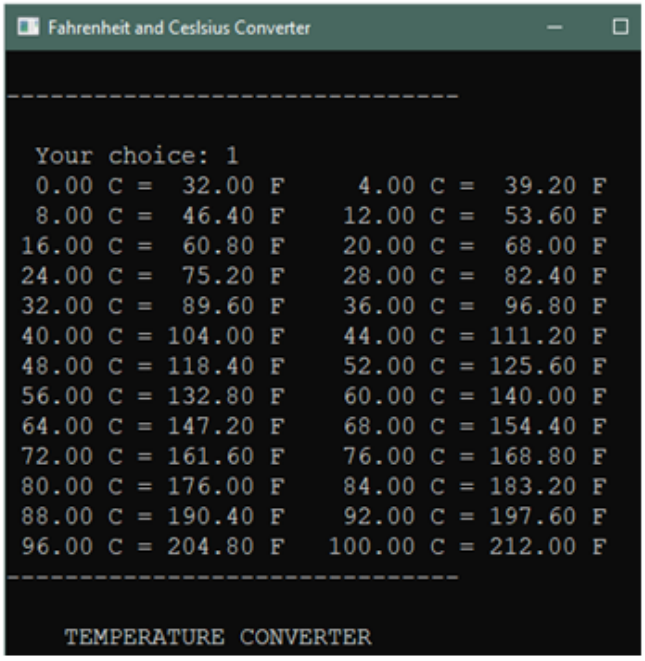
where F = Fahrenheit and C = Celsius. Do not use short variables like F and C in your code. Use words or combination of words in naming variables and methods. A class diagram showing methods of the TemperatureConverter class is presented below.



The images below 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.







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

4. 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.

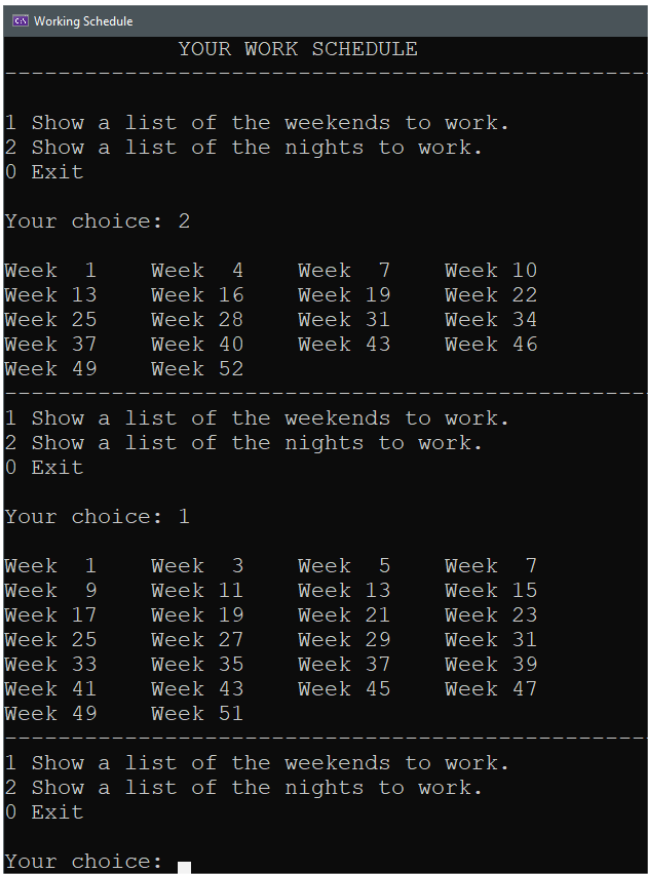
5.3.2 Display a list of weeks she/he scheduled for the selected option.

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.



5. 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