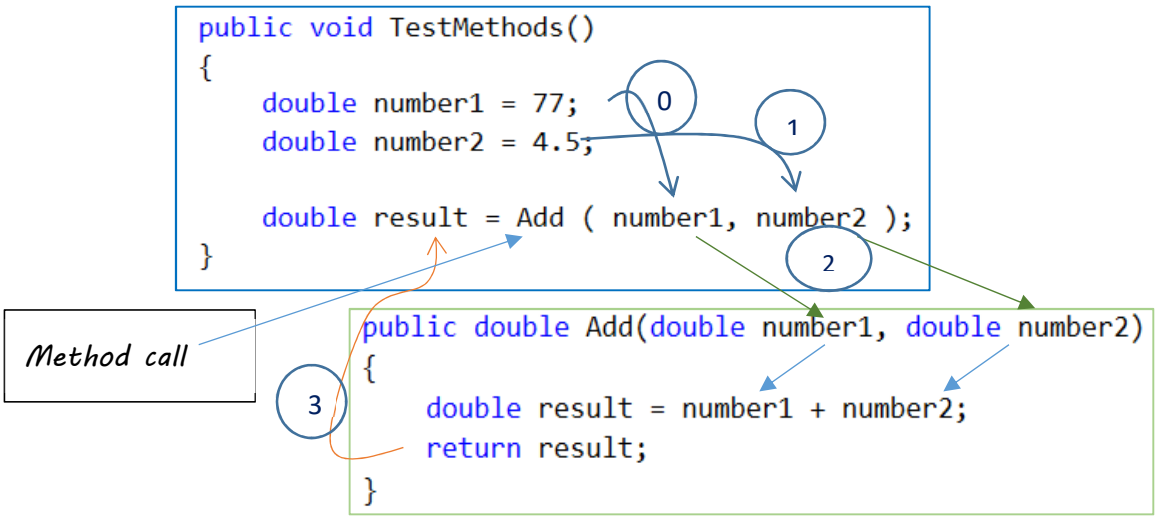
1. Some general help and guidance:

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. We begin with some general topics and then provide help on the first two parts of the assignment, i.e. Assignment 2A and Assignment 2B, as these are required for a C grade.

1.1. Method with parameters and a return value

In the code snippet that follows, you can see the method, **Add**, that has two parameters of the double type. The parameters are to receive input values (arguments) 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; they will receive the values of number1 and number2 in the **TestMethods**, as they are passed to them. The parameters could have any other names.

1.2. Working with strings

The class string (or String) has several useful methods and properties (property is a special type of a method) that can be used in working with text (string). Also, a string variable (e.g. string **name**) contains features which make life a lot easier when working with strings.

Using one of the string's methods:

```
string name = "No name";
if (!string.IsNullOrEmpty())
{
    //Code . . .
}
```

Using the string-variable's (**name**'s) features:

name.Length:	returns the number of chars in a given text.
name.ToLower:	returns the string converted to its lowercase.
name.ToUpper	returns the string converted to its uppercase

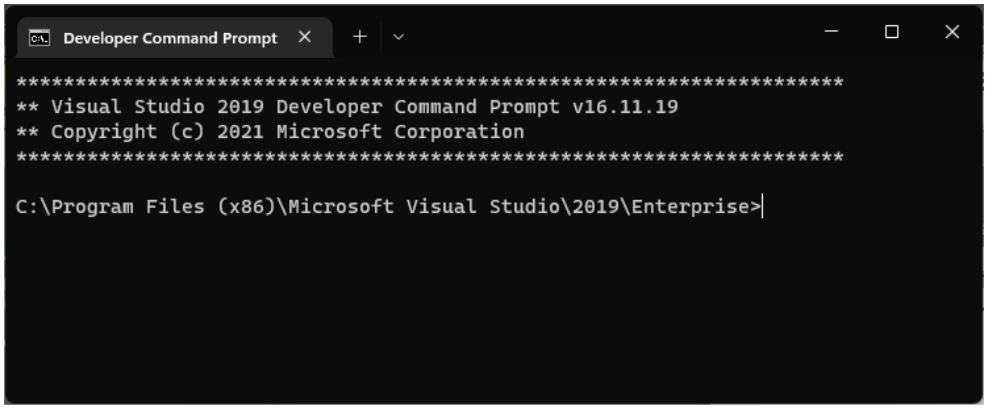
```
string response = Console.ReadLine(); //#1
response = response.ToLower();  //#2 change to lower case
```

Notice carefully how a string variable uses its own features, in the last statement above. Assume, as an example, that the user inputs the text "No", which is returned by the Console.ReadLine() and saved in the string variable, **response** when the code line #1 is executed.

Then when the next line, #2, is executed, i.e. **response = response.ToLower();** , the current content of the string **response** ("No") is converted to it lower case, ("no"). Finally, the converted value ("no") is saved back in the variable **response**, overwriting the previous value ("No");

1.3. Input and output:

- A Command Prompt is the window that runs the console application such as the figure below.



- Use **Console.WriteLine (text)** to write the text to the prompt window (i.le. to the user). The cursor will then move to the next line on the command prompt window.
- Use **Console.Write(text)** to write the text on the same line on the command prompt window where the cursor is. The cursor will remain at the end of the text after displaying the text.
- Use text = **Console.ReadLine()** to read (fetch) a line of text that the user inputs on the prompt window from the cursor location.
- Do not use **Console.ReadKey** and **Console.Read()** as these do not return a string.

Conversion to string and string-concatenation:

Putting together two or more stings to one is known as concatenation. In C#, the operator symbol '+' can be used to combine strings. The operator += also works well to add new strings to a string.

```
string  fullName = lastName.ToUpper() + ", " + firstName;
```

or:

```
string  fullName = lastName.ToUpper();
full += ", " + firstName;
```

Writing output to the user with the help of Console.Write() and Console.WriteLine() is done by sending a string as a parameter of the methods. When you want to print a numerical value, you need to convert the value to its textual form, as a series of characters. However, Console's methods do the conversion if you do not explicitly do it. Console.WriteLine(56.99) works fine because the WriteLine method is made so intelligent that it can accept a value that is not a text. There is also another way to combine text, and numerical values into a string using the sring.Format method, as shown in the code line below:

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

The string.Format method can in fact be used for formatting currency, date and time as well. Refer to C# documentation for learning more about it.

The method `string`.Format has two parts: a formatting part enclosed within the quotation marks, " ", 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 (name). The expression {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 textual representation of the value 19.99 containing 5 chars (counting the decimal sign) will receive one blank space 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
}
```

Remember also that any text and the spaces that are included in the formatting part text will also be included in the formatted text. There is a space after each { } expression in the above code example. The output from above code is as shown below:

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

2. Assignment 2A - The **FunFeatures** class

The table below includes the methods of this class and summarizes the purpose of each method while suggesting an algorithm for its implementation.

Description of some frequently used terms:

- Implementation: writing code for the body of the method.
- Methods that begin with the word **Read** are for getting input from the user.
- By "user" it is meant the end user of your application or yourself when testing your program.
- By the words: display, print, write or show, it is meant to write information to the user with help of Console.Write or WriteLine methods.
- Writing to the user means also to write a text to the Prompt Window.

Before each reading, write a friendly and informative text to the user with information on what type of input you are expecting and sometimes in which form it should be given.

Use Console.WriteLine() with no text within the parentheses to insert a blank line on the Prompt Window. To include a blank-line in text, use Environment.NewLine which is a better alternative '\n'.

No.	**Method Name** **Purpose and pseudo code (algorithm)**

1	`private void Introduce()`
	
	Begin your interaction with the user by first introducing yourself as a brief text.
	- Write your name and which semester of the year you are attending this course.
	- Read the user's first name and last name. Use a separate method **ReadName**.
	- Call the method **ReadName** to read the user's name and save it in the instance variable **name**
	- Call the method **ReadEmail** to read the user's email. You don't have to validate how the user provides the email, just read a string.
	- Write a welcome phrase to the user
	  - Nice to meet you, SIMPSON, Marge!
	  - Your Email is: marge@springfield.net.

2	`private void ReadName()` Read the user's first name and last name. Convert the last name to upper case. Combine the user's last name in capital letters and the first name and save it in the instance variable **name**. Proceed as follows: - Read the user's first name in a local variable. - Read the user's last name in another local variable. - Convert the last name to uppercase (capital letters), - Combine (concatenate) the two names into one string, using lastName.ToUpper(). - Save the final text in the variable **name**.

3	`private void ReadEmail()` - Read the user's email and save it in the instance variable **email**.

4	`private bool RunAgain()`
	
	- Declare a variable **again** = false;
	- Ask the user to run again or exit.
	- If the user answers yes (or y),
	  - again = true
	- else
	  - again = false
	- Return again.
	-

5	`private void PredictMyDay()`
	
	Ask the user to input a number between 1 and 7 and using a switch-statement, check with day of the week corresponds to that (1 = Monday). Then, pick-up a comment for that day and write the text to the prompt window. The comments are given in the assignment description, but you can use your own.
	
	- Read the user's option (in a local string variable), using Console.ReadLine().
	- Convert the return string of the ReadLine method to its corresponding numerical representation ("6" → 6).
	- Declare a local variable strComment = string.Empty;
	
	- Use switch (the number):
	  case 1
	  strComment = "Keep calm on Mondays! You can fall apart!"
	  break;
	
	- Continue with the rest!
	- For cases 2 and 3, you have the same action (same comment). You can combine the cases like this:
	case 2:
	case 3:
	  strComment = ….
	- After case 7, use `default` to cover all other values in case the inputs a value outside the range.
	
	default:
	  strComment = "No day is a good day but it doesn't exist!"
	  break;

6	`private void CalculateStrengthLength()` Read a text from the user and save it in a local variable

	- Calculate the number of characters in the given text (using the string variable's Length property), save it in a local variable - Change the text to its uppercase. - Write back the converted text to the prompt window. - Write also the length of the text, i.e. the number of characters.

Some code help provided in the next sections.

2.1. The **RunAgain** method

This is, as it sounds, a method that determines whether the loop in the **Start** method should be repeated or not. You could, in the same way, write a method that asks the user whether to exit or continue. In this case, you would need to adjust the loop condition in the Start method accordingly.

```
34 private bool RunAgain()
35 {
36     Console.Write("Continue with another round? (y/n):  ");
37     string response = Console.ReadLine();
38
39     response = response.ToLower();  //change to lower case
40
41     bool again = false; //initialize, answer = no
42
43     if ( (response == "y")|| (response == "yes") )
44        again  = true;  //repeat = true
45
46     return again;
47 }
```

The code statements between lines 41 and 44 can be simplified as follows:

```
bool  again = (response == “y”) || (response=="yes);
```

If the user writes anything else than a "y" or "yes" (not case sensitive) the variable **again** will be equal to false. . Notice also that you could have an else statement after Line 44 to set *again = false*, but this is not necessary as **again** will have a false value if the response is not equal to “y” or "yes", because it is initialized to false on Line 41.

2.2. The **ReadName** method

```
public void ReadName()
{
    Console.Write("Your first name please:  ");
    string firstName = Console.ReadLine();

  // read last name in the same way before combining the
  // names and saving the result in the instance variable name

    name = lastName.ToUpper() + ", " + firstName;
}
```

2.3. The **ReadEmail** method

Write a message using Console.Write() and then read the user input with Console.ReadLine().

```
public void ReadEmail()
{
  //Message the user
  email = . . .

  //move the cursor to beginning of the next line, for next time
  Console.WriteLine();
}
```

Question: *What happens if you by mistake write string email = ….?*

2.4. The **PredictMyDay** Method

The purpose of this method is to prompt the user for a number between 1 and 7. As the input from **Console.ReadLine()** has the string type (e.g. "6"), you need to convert it to its correspondent numerical value (e.g. 6). Save the converted value in a local variable. Then, use a switch-block to compare the input to numbers 1 to 7, and depending the match, write out a comment corresponding to that day.

```
public void PredictMyDay()
{
    Console.WriteLine();
    Console.WriteLine(" ****** FORTUNE TELLER ******");
    Console.Write("Select a number between 1 and 7: ");
    int day = int.Parse(Console.ReadLine());

    string fortune = string.Empty;

    switch (day)
    {
        case 1:
            fortune = "Keep calm on Mondays! You can fall apart!";
            break;
        case 2:
        case 3:
            fortune = "Tuesdays and Wednesdays break your heart!";
            break;



        default:
            fortune = "No day? is a good day but it doesn't exist!";
            break;
    }
}
```

Question: *Could an if-else if-else be used instead of switch?*

2.5. The **CalculateStrengthLength** method

```
public void CalculateStrengthLength()
{
    //Write info messages to the user
    Console.WriteLine();
    Console.WriteLine(" ---- STRENGTH LENGTH ---- ");
    //Continue with more text (see the runtime image in the assignment description)


    //Read the user's text (input)
    string text = Console.ReadLine();

    //Text can never be a null because Enter is a char that will be
    //returned but it will not be counted as a written char.
    //text will therefore be an empty string.

    //Calculate the number of characters
    int length = text.Length;

    //Change the text to upper case (capital letters)


    //Write the string back to the user and also print the number of chars.

    Console.WriteLine();
}
```

3. Assignment 2B – The **MathWork** class

The code for the Start-method is given in the assignment description. The table below includes the other methods of this class and summarizes the purpose each method and suggests an algorithm for its implementation. A run-time example of the features of this class is given in the image after the table.

No.	**Method Name** **Purpose and pseudo code (algorithm)**

1	`private void PrintMultiplicationTable()`
	
	Display a multiplication table for numbers 1 to 12..
	- Use a for-loop
	- for row = 1 to row <= 12
	  - for col =1 to col <= 12
	  Write row*col + some spaces
	  - Go to next line after the above loop (Console.WriteLine())

2	`private void Calculate()`
	
	This method reads two integer numbers and then calls different methods to perform different calculations with the given numbers.
	The body of this method is commented in the assignment description. Here is a more detailed algorithm that you can use.
	
	Note: The methods called in the loop below are described in the rows down in the table.
	
	Let done = false
	while (!done)
	- Read an integer and save it in a local variable as startNum
	- Read another in and save it in a local variable as endNum
	- Remember to use int.Parse or preferably int.TryParse to convert the text from ReadLine() to an integer.
	- If (startNum > endNum)
	  - Swap the numbers so that startNum < endNum
	  - See code-help below (Swap numbers)
	- Call the method **SumNumbers**, get the sum of all numbers from startNum to endNum.
	
	`int sum = SumNumbers(startNum, endNum);`
	
	- Display sum on the Prompt Window
	- Call the method below 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 the following method to calculate and display the square root of each number in the range startNum to endNum
	
	`CalculateSquareRoots(startNum, endNum);`
	
	- For every number from startNum to endNum (inclusive these)
	  - Calculate and display the square root of the numbers from that number to endNum (explained later)
	
	- Here, ends the while loop, Ask the user whether to exit or continue
	`done = ExitCalculation();`

3	`private int SumNumbers(int start, int end)` - The arguments start and end come from the Calculate method where the corresponding variables are startNum and end Num inclusive these two - Declare a local variable **sum** - Run a loop with counter from **start** to **end** and add the counter to the sum - Return **sum** after the loop is ended.

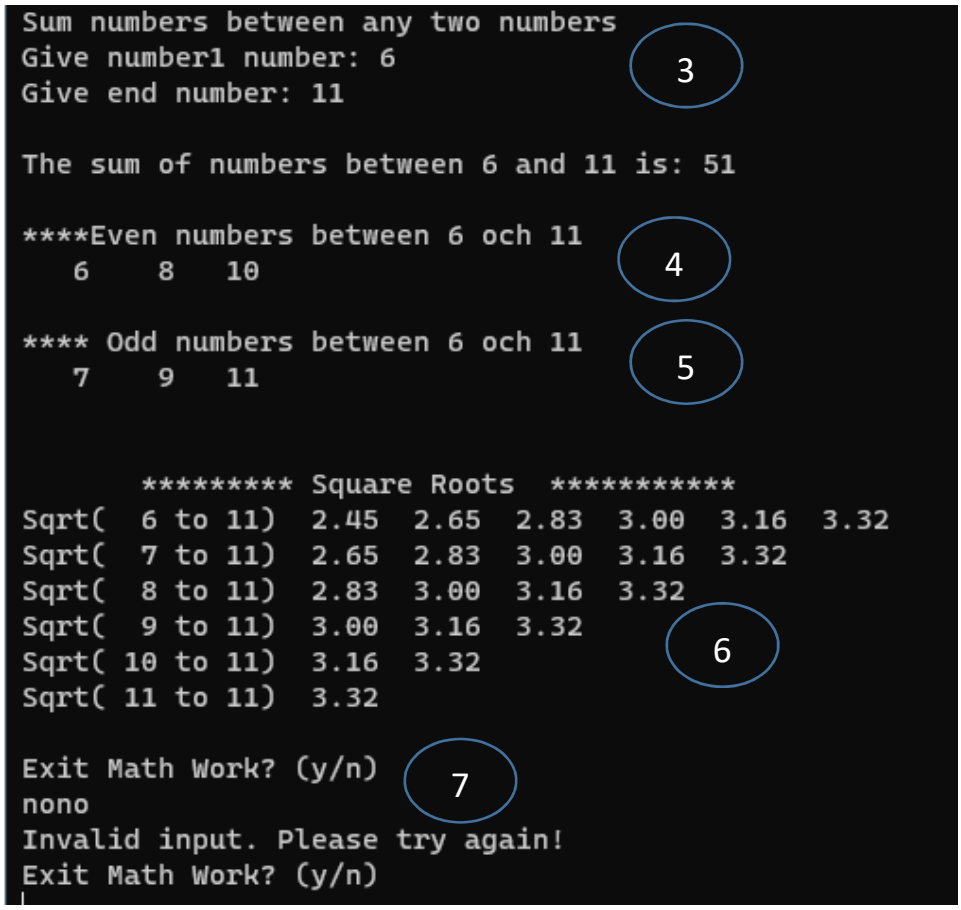
4	`private void PrintEvenNumbers(int number1, int number2)`
	
	- The arguments number1 and number2 come from the Calculate method where the corresponding variables are startNum and end Num.
	- Run a loop with counter from number1 to number2 inclusive both.
	- If the counter is divisible by 2 (use %) , it is an even number.
	- To check if the counter (i) is even:
	`if (i % 2 == 0)`
	It is an even number, print it out (use Console.Write, adding spaces) so the output comes on the same line.

5	`void PrintOddNumbers(int number1, int number2)`
	
	- Proceed as for the even number but check if the counter is not divisible by 2 which then the counter is an odd number
	  - To check if the counter (i) is odd:
	`if (i % 2 != 0)`

6	`void CalculateSquareRoots(int num1, int num2)` This method should calculate and display the square root of each number in the range num1 to num2 and in every iteration, also calculate and write out the square root of number from the current number to the end number num2. This way you get a triangular output as shown in the run-time example below. Follow this algorithm: - Run a loop with counter (for example **i**) from num1 to num2 inclusive these. - Write the value of the counter - Run another loop from counter (for example j) from i to and inclusive num2 - Calculate and print the square root of j

7	`private bool ExitCalculation()`
	
	- Before start the loop, use the following local variables:
	`bool done = false;  //initialization`
	`bool responseOK = true; //initialization`
	
	- Repeat the following until the user inputs a "y" or a "n", although ignoring case sensitivity.
	  - Ask the user whether to exit or continue.
	  - Convert the user's answer to lower case to make comparison easier and safer.
	  - if the user's answer is equal to "y"
	  - done = true
	  - else if the answer is equal to "n"
	  - done = false
	  - else
	  responseOK = false
	  Inform the user that the input is invalid and to try again.
	
	- Return the value of done (meaning to exit).

The following image shows a run-time example of the **MathWork** with references to above table:



Some code-help is provided in the following pages

1. The Multiplication Table

```
private void PrintMultiplicationTable()
{
    Console.WriteLine();
    Console.WriteLine("       ********* Multiplication Table **********");
    for (int row = 1; row <= 12; row++)
    {
        for (int col = 1; col <= 12; col++)
        {
            Console.Write(string.Format("{0,4:d} ", row * col));
        }
        Console.WriteLine();
    }
}
```

2. The ExitCalculation method

Note that you can write this method to function as the RunAgain method in the previous part. It is only another way of handling a loop condition.

```
private bool ExitCalculation()
{
    Console.WriteLine();
     string response = String.Empty;

    bool done = false;  //initialization
    bool responseOK = true; //initialization
    do
    {
        Console.WriteLine("Exit Math Work? (y/n)");
        response = Console.ReadLine();
        response = response.ToLower();  //change to lower case

        responseOK = true;  //initialize every iteration
        if (response == "y")
        {
            done = true;
        }
        else if (response == "n")
        {
            done = false;
        }
        else
        {
            responseOK = false;
            Console.WriteLine("Invalid input. Please try again!");
        }
    } while (!responseOK);
    return done;
}
```

3. The Calculate method

a. How to swap numbers:

```
//swap numbers if given in the wrong order
if (startNum > endNum)
{
    int tempNum = endNum;

    endNum = startNum;
    startNum = tempNum;
}
```

Question: Why is tempNum necessary?

b. The Square root calculation

```
private void CalculateSquareRoots(int num1, int num2)
{
    Console.WriteLine();
    Console.WriteLine("       ********* Square Roots  ***********");

    for (int i = num1; i <= num2; i++)
    {
        Console.Write("Sqrt({0,3} to{1,3})", i, num2);
        for (int j = i; j <= num2; j++)
        {
            double value = Math.Sqrt(j);
            Console.Write(string.Format("{0,6:f2}", value));
        }
        Console.WriteLine();
    }
}
```

The reason you are receiving so much code for this assignment is for you to think and learn how to use and work with basic operations in a proper way in order to be able to be well-prepared for the coming assignments. This assignment includes syntax some of the most fundamental programming concepts such as conditional and iterative statements and operators Therefore, it is essential that you learn and work with these in a proper way already from the beginning.

It is important that you understand what happens in every single statement in the code clips that are given in this document. Use the forums to discuss things that you are not sure about.

Good Luck!

Farid Naisan,

Course Responsible and Instructor