Objective

To practice and learn the following coding skills:

- define methods with parameters
- use predefined methods (methods in Console class, and Math class)
- using Parse() method

Instructions

Please read the entire assignment to the last page and then start the coding.

We would like to refactor[1] the program in assignment2 and implement the same program but using methods.
You are provided with the Main() method source code (open *WAC_to be completed.cs* file) and you are requested to implement the definition of methods to complete the program.
Except three lines of code in Main() (Line 33, 64 and 74) that you must change as requested, YOU MUST NOT CHANGE ANY OTHER LINE of the code in Main() method! That is a strict requirement of the assignment. No change means you do not delete any line or add or modify any line (except Line 33, 64 and 74).
You must keep the main method as it is and just add methods definition to the code to complete this assignment.
You must write a definition for the following 5 methods:

1. `GetUserInput`
2. `DisplayBanner`
3. `DisplayTableRow (overloaded: note the method has 3 parameters, all strings. Check lines 52 and 53)`
4. `DisplayTableRow (overloaded: note the method has 3 parameters, 1 string and 2 floats. Check lines 55, 56, 57, 58, and 59. The data types are different than lines 52 and 53. Search google for` **“Method Overloading”**`)`
5. `WeightedGrade`

You are provided with an executable file for your test. Make sure your program behaves just like this executable file of the assignment.

Requirements

1. On very top of your program, before using directives, update the multiline comment with your name, date and purpose.
2. Make sure your code looks neat and tidy. Press Ctrl+KD in Visual Studio to format your code based on coding styles.
3. Respect Naming Conventions for constants (ALL_CAPS) , variables (camelCase), Method names (PascalCase).
4. To write a definition for any of 5 methods, follow steps a to e:
   a. What name do you choose for the method?
      i. The name must follow identifier name rule (the same rule as you respected for variables: only a-z and A-Z and underline is allowed, no number at the beginning, etc.) name of methods must be in PascalCase style.
      ii. The name of method must be descriptive. For example, *GetUserInput* is descriptive and describes the method is used for getting the input from the user.
   b. Does the method have any input parameter? How many? If any, what are their data types?
      i. For example, *GetUserInput* method has 1 input parameter and its data type is string.
   c. Does the method return any value? What is the data type of return value?
      i. For example, *GetUserInput* method has 1 return value and its type is float. we can find input parameters and return value of *GetUserInput* method by looking at

```
assignments = GetUserInput("Assignments");
```

“assignments” is a string and it is the only input parameter of the method. The return value is stored to *assignments* variable that is of float type.

d. When we found the answer to the questions in a, b, and c then we can write a method with those characteristics:
   i. For example, we write the method for GetUserInput as:

```
static float GetUserInput(string textMessage)
{
// Here is the body of GetUserInput method and some statements must be written here
}
```

e. Next, we must write some statements in the body of the method. We must know what we expect from the method. Also, we must ask ourselves, given the input parameters to the method, how the method must produce the return value?
   For example, what do we expect from *GetUserInput* to do for us? Answer: we expect from GetUserInput to:
   1. display on Console a message like “Enter a value for X ” (X is replaced by the input parameter textMessage)
   2. get the input from user (Console.ReadLine())
   3. convert the input that is string to a numeric value (use Parse method)
   4. return the numeric value.

5. Because you want to define methods one by one, you may want to use comments to disable some part of the code to be able to compile and test your methods. For example, after defining method GetUserInput, if you want to test it, you may want to comment the lines 43 to 75. When you are sure the GetUserInput method works properly, you uncomment lines 43 to 75 and comment lines 51 to 75 to be able to test WeightedGrade method. You repeat this commenting and uncommenting until you define all methods and your code is complete.

Checklist Before Submission

- All variables and consts are defined together on top of the methods (in this assignment, Main method)
- Your program works correctly, and the results are the same as numbers in 5 screenshots provided to you in assignment 2. **Note**, the values for Weighted Total Average and Weighted Exam Average Totals in assignment2 and assignment3 are different because in this assignment you use Ceiling and Floor. For example, in Assignment2, you get 70.79 but in Assignment3 the value would be 70.00. or the value for Weighted Average of Exams in assignment2 is 63.49 but in Assignment3 would be 64.00.
- The name of methods that you define must be in PascalCase Style.

Submission

Please submit the following items to the corresponding assignment folder on BlackBoard prior to the deadline:

- A zip/archive folder containing the project **folder** specially source code files (.cs file). Rename the zip file name as a#_xy.zip where # is assignment number, x is your first name and y is your last name. For example, when I submit my work for assignment 2, I will rename the zip file as a2_SaeedMirjalili.zip

Appendix 1: How to define a method

I am going to use an example to explain how you can define a method.

Example 1: define a method that receives two numbers and returns sum of them.

To define this method, you must follow these steps:
step1. Choose a descriptive name for the method. A name that describes the function of the method. It is recommended to use an imperative form of a verb. For this example, we can choose a name like Sum, or CalculateSum. The name style should be in PascalCase.

step2. You must know how many inputs the method has. In this example, there are two inputs. We choose a name for them value1 and value2. The names are written in camelCase.

step3. Then, we must decide on their data type. Because they are numbers, we must choose a proper numeric data type. There are two types of numeric types: integral (byte, short, int, long…) and floating point (float, double, decimal). I choose double data type because I can use my method for addition of all types of numbers (integer and floating point) and double can hold big numbers. So far, the header of the method looks like below:

```
CalculateSum(double value1, double value2)
```

step4. Now, we must think about the return value of the method. Does method return any value? In this example, CalculateSum method returns the sum of two values. If the method returns a value, what is the data type of the method? In this example, we decide that the return data type to be double. Now the method looks like below:

```
double CalculateSum(double value1, double value2)
```

if the method does not return any value, we must put void before the name of the method, that is:

```
void CalculateSum(double value1, double value2)
```

In this example, the method must return a value, as we need to get sum of two numbers out of the method. Therefore, we do not put void before the name of the method.

step5. C# is an Object-Oriented Language, and many concepts of Object-Oriented Programming are implemented in C#. One concept in OOP, is static methods. You must decide to make your method static or non-static. If you make your method static, you do not need to create an object before calling a method. If you do not make your method static, you must create a new object before calling the method. See below the difference:

Static Method

```
class Program
{
  static void Main()
  {
    double result;

    result = CalculateSum(3, 4);
  }

  // CalculateSum Method Definition
  static double CalculateSum(double value1, double value2)
  {
    return (value1 + value2);
  }
}
```

Non-static Method

```
class Program
{
  static void Main()
  {
    double result;
    Program myObject = new Program();
    result = myObject.CalculateSum(3, 4);
  }

  // CalculateSum Method Definition
  double CalculateSum(double value1, double value2)
  {
     return (value1 + value2);
  }
}
```

step6. So far, we designed the header of the method (static, return type, Method Name, Input parameters), now, we must write the statements in the body of the method. In this example, the body of the method is very simple. Just gets two inputs and add them together and returns the result. **Meanwhile**, where should we put the method? You can put methods before or after Main method. It does not matter where to put, for example, CalculateSum method, can be before Main method or after. But, make sure CalculateSum method is not inside the Main method and it **must be inside** a class, e.g. Program class.

step7. Then, we start write the code inside the body of the method. The method is for calculating sum of two values. We must write statements that gets input parameters (value1 and value2), process them (add them together) and return the sum (result). Make sure that input parameters are used inside the body of the method. Otherwise, there is no point to have input parameters.
In fact, when we finished defining a method, the body of the method does not change, and we change only the input parameters.
This is the body of CalculateSum method.

```
static double CalculateSum(double value1, double value2)
{
  double result = 0;
  result = value1 + value2;
  return result;
}
```

We can write the method even shorter. We do not need to store the result of addition and then return the result. We can return simply value1 + value2 without storing it in a variable.

```
static double CalculateSum(double value1, double value2)
{
  return (value1 + value2);
}
```

Step8. We are not done yet. We have defined a method but if we do not call (invoke) the method, there is no benefit of defining a method. If you define a method, but you do not call it, it is like, you write a book, but nobody reads it. What was the benefit of writing this book?

To call (invoke) a method, we must write the name which we chose when we defined the method, and we must pass some values to the parameters, and if the method returns a value, we must store it in a variable or use it.
In this example:

- The name of the method is CalculateSum
- There are two input parameters: value1 and value2, both are of type double
- CalculateSum returns a value of type double

Considering these three points, this is the way we call CalculateSum in our Main method

```
static void Main()
{
   double result;
   result = CalculateSum(3.5, 4.7);
}
```

When we call the method, the parameters, can be fixed values (3.5 and 4.7) or they can be variables like below:

```
static void Main()
{
   double result;
   double val1 = 7.5;
   double val2 = 6.8;
   result = CalculateSum(val1, val2);
}
```

Or, parameters can be a mix of variables and fixed valued like:

```
static void Main()
{
   double result;
   double val1 = 7.5;
   double val2 = 6.8;
   result = CalculateSum(val1, 5.8);
}
```

Or even expressions like (`val1*val2 and valu2/2.0`):

```
static void Main()
{
   double result;
   double val1 = 7.5;
   double val2 = 6.8;
   result = CalculateSum(val1*val2, valu2/2.0);
}
```

Note that if you use variables while you call the method, the name of the variables may not match with the variable names you used when you defined the method. For example, when I call the method, the names of variables are val1 and val2
result = CalculateSum(val1, val2);

but when I defined the method, the names of the variables are value1 and value2:

```
static double CalculateSum(double value1, double value2)
```

when you call CalculateSum method, the values of val1 and val2 are copied to value1 and value2. If you call the method and pass fixed values, e.g. CalculateSum(3.5, 4.7); the first value (3.5) is copied to value1 and the second value (4.7) is copied to value2.

[1] **Code refactoring** is the process of **restructuring** existing computer code **without** changing its external behavior. Refactoring is intended to improve nonfunctional attributes of the software. Advantages include improved code readability and reduced complexity. (Wikipedia)
In this assignment we refactor assignment2. This means that we do not change the behavior of the program created in assignment2, but we change the way we write our code using method to make our code more readable and less complex. The executable file that you create works exactly like assignment2, but the codes of assignment 2 and 3 are going to be different.