



MALMÖ HÖGSKOLA

**Malmö University
Center for Technical Studies**

Programmering med C#, grundkurs
Programming Using C#, Basic Course

Trash Manager

Assignment 2 – Selection and Iteration Algorithms

Mandatory

[Farid Naisan](#)

University Lecturer
Malmö University, Malmö Sweden



Assignment 2

1. Objectives

The main objective is to provide training in iteration and selection algorithms, things that are a part of a programmer's everyday life.

Several code excerpts are provided in this assignment to prepare you for your future assignments. If you wonder why the code is given as images, it is for the purpose of preventing the copy-paste technique! I believe that by rewriting, you will notice details and you will get a time to wonder why things are done in a certain way.

It is expected that you follow guidelines given in the document "General Quality Standards and Guidelines" before beginning with this assignment.

2. Description

This assignment consists of a number of sections. In the first two sections, you will be writing a program that sums up numbers. Then you will write a class for converting the local currency to a foreign currency. In the last section, you are asked to write a program for determining the weeks that the trash transport truck collects household trashes in a community in Lund, a city in southern Sweden, where your instructor lives.

The input and output operations are performed from and to a console window. A different iteration algorithm is used in every part, except in the last section where you are given the freedom of selecting the algorithm by yourself.

The assignment will also use both forms of selection algorithm, i.e. `if-else` and `switch` statements. A good deal of hints and guidance is provided in the first parts to prepare you gradually to get on your own.

It is not meant to use arrays in this assignment, but if you are an experienced array user, you may go ahead and do that. We will be working with arrays in a later assignment.

When the program starts, a menu is presented to the user. The user selects a menu item and the program performs the job using an object of the class related to the corresponding task.

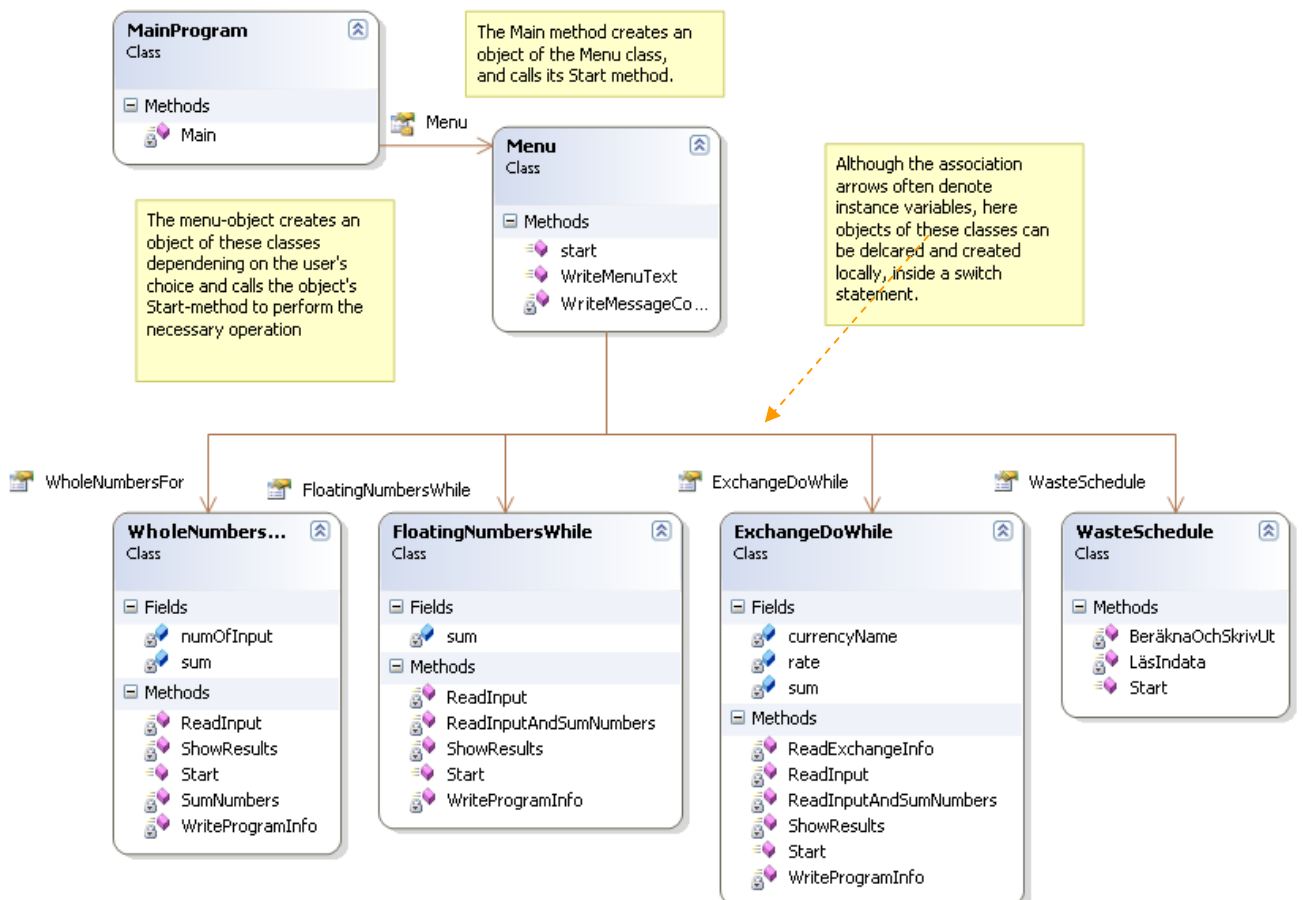
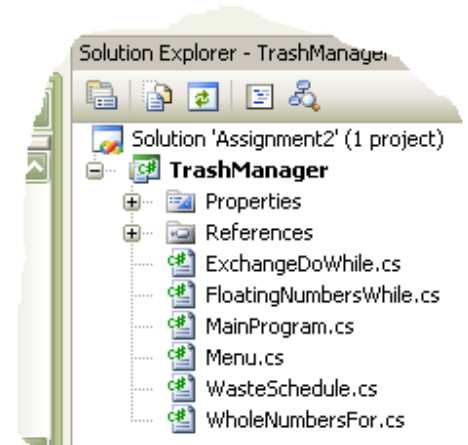
```
File: \\77D:\\D0\\NET\\DATA\\U7a\\Module 2\\Assignment 2\\bin\\Release\\Assignment2.exe
PROGRAM MENU
-----
Whole Numbers with For           : 1
Floating Point Numbers with While : 2
Currency Converter with Do While-loop: 3
WasteSchedule                   : 4
Exit the program                 : 0
-----
Your choice: _
```

3. The Project

The figure shows a project created in the Visual Studio, containing the classes that you are expected to write. Even if you don't use any IDE and write your code in a simple editor, make sure that you gather your source code files under a separate directory for this assignment. It is this folder that will be referred to as "the project" in this document.

The classes are also illustrated graphically in the diagram that follows. The diagram is called a "Class Diagram" in object-modeling languages like UML (Unified Modeling Language). The boxes represent classes and the arrows show the association between them. Each box has three compartments, for class-name, for attributes (fields) and for methods of the class. The arrows go from the class that uses, i.e. creates an instance of, to the class that it points to. This sort of association is known to have a so called "Has-a" relation, meaning that one object has another object as its component. In other words, one object is built up of (contains) other objects. A Car object has for example a number of Wheel objects, a CDPlayer object, etc.

An object of the **MainProgram** class "has an" object of the **Menu** class which in turn has objects of the classes **FloatingNumberWhile**, **ExchangeDoWhile**, etc. Notice that the arrow used for this purpose is drawn with open head.



An arrow, as drawn in the figure, shows that a class uses an instance of the object it is pointing to in the diagram.

4. A Work Plan

Let's be effective and make a work plan.

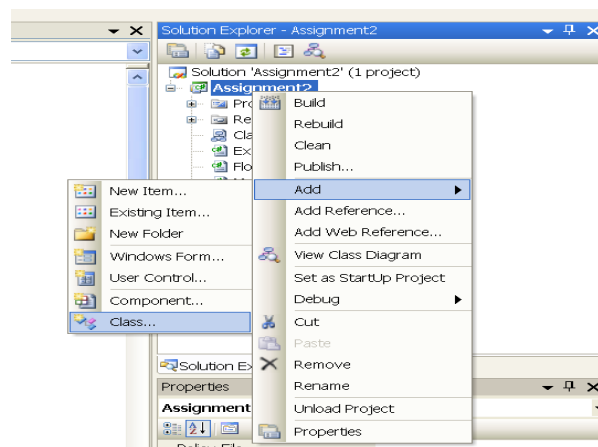
- 4.1 Create a C# Console Application project using the Visual C# or Visual Studio. The IDE will create a solution for you and put the project into this solution. In this document we refer to this project as Assignment2.
- 4.2 The IDE has also prepared some default classes for you; one such class is program.cs in which the IDE has written a Main method.
- 4.3 Rename the program.cs to **MainProgram.cs** in the Solution Explorer. It will be inside the Main method where we will then create an instance of the Menu class to start the program, but we will wait until we have some code ready in the Menu class to test. So, set aside this file for a while and proceed with the next part. Don't forget to save your project often.
- 4.4 We begin with writing the Menu class. Write the skeleton for the **Menu** class, and save it in the **Menu.cs** (see next section). At this stage, the **Menu** class should
 - 4.4.1 show the menu text,
 - 4.4.2 read the user's choice,
 - 4.4.3 control the input and,
 - 4.4.4 then show another message to confirm the choice of the user, like "You chose no 2, but it is not ready yet".

5. The Menu class

Add a new class, (right-click on the project name, Assignment2, and choose Add, Class, see figure below), rename the file to **Menu.cs** and Visual Studio will change the class name for you.

The purpose of this class is to show a menu with four alternatives, numbered 0 to 4, and ask the user to make a choice. The Menu class then reads the value from the console window and creates an object of the related class to perform the task related to the menu choice. You may certainly use other menu system if you like to.

Add a class to the project in the IDE, by right-clicking on the project-name





This class should perform the following:

- 5.1 Show a menu with four choices, numbered 0-4 (or in your own way) and wait for the user to input a choice.
- 5.2 Read the user's choice and if a valid one,
 - Declare and create an object of the chosen class.
 - Call the object's "start" method to.
 - o read input,
 - o calculate (perform its operations), and
 - o print out results.
- 5.3 If the user's input is not valid, i.e. the user does not write a number 0 to 4 (for example), give an error message and repeat from 5.1.
- 5.4 Write a private method, **WriteMenuText**, that shows the menu text to the user.
- 5.5 Write a public method, **start** to program the above loop in (should call 5.4).

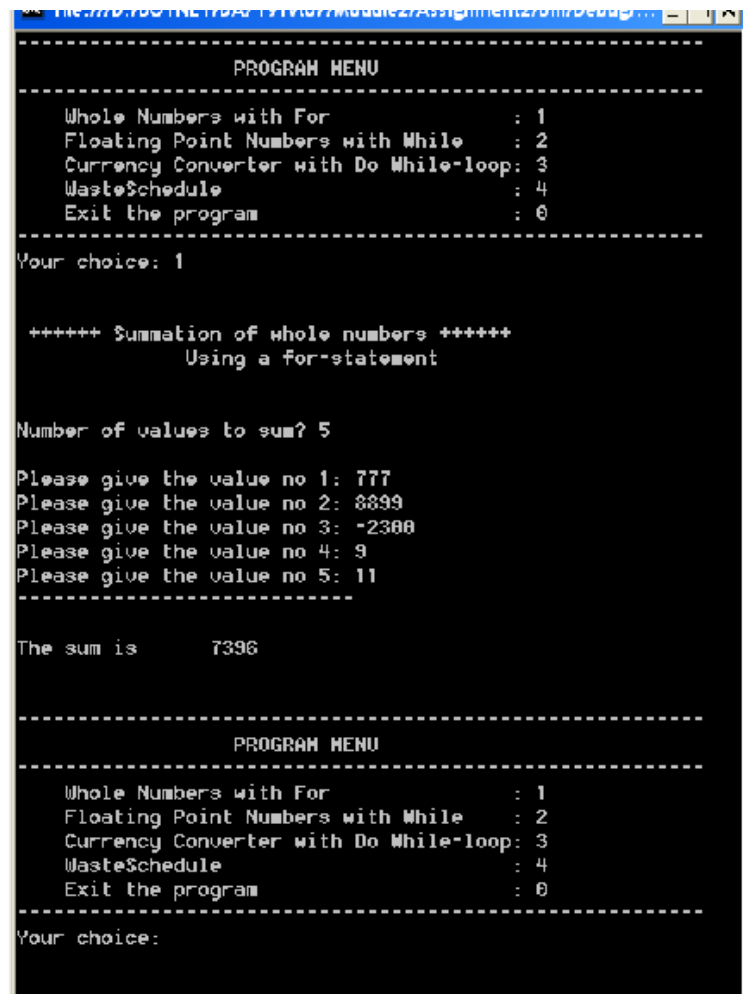
According to our work plan we proceed with 5.1 and 5.2 but wait with 5.2.1 to 5.3 until we have implemented the functionality for the menu options; be sure to do 5.4 and 5.5 though.

6. The MainProgram class

- 6.1 Go to the **MainProgram** class and create an object of the **Menu** class and call the object's Start-method. Need help? See the code at the end of this document:
- 6.2 Build the solution (), run the program ( or F5) after a successful compilation and test your program so the code written so far works well, i.e. the menu is shown and it repeats itself until the user chooses to exit the program by feeding a zero value.
- 6.3 When you are satisfied with the performance of this part of your program, proceed to the next step. We will come back to the Menu class later on.

7. Summation of Whole Numbers (for-Statement)

In this part we make the menu item 1 work completely, but if the user chooses another number the program will give a message saying that the program is not yet complete, just as before.



```

PROGRAM MENU
-----
Whole Numbers with For           : 1
Floating Point Numbers with While : 2
Currency Converter with Do While-loop: 3
WasteSchedule                   : 4
Exit the program                 : 5
-----
Your choice: 1

++++++ Summation of whole numbers ++++++
          Using a for-statement

Number of values to sum? 5
Please give the value no 1: 777
Please give the value no 2: 8899
Please give the value no 3: -2388
Please give the value no 4: 9
Please give the value no 5: 11
-----
The sum is      7396

PROGRAM MENU
-----
Whole Numbers with For           : 1
Floating Point Numbers with While : 2
Currency Converter with Do While-loop: 3
WasteSchedule                   : 4
Exit the program                 : 5
-----
Your choice:
  
```

7.1 Add a new class, **WholeNumbersFor** and save it. This class should perform the following tasks:

- 7.1.1 Read input from the console window (user's input), i.e. how many numbers to sum and then the numbers themselves..
- 7.1.2 Calculate the sum of the given numbers.
- 7.1.3 Show the results back to the user.

7.2 The code given here can be used as a template. In order to fit the code into one page, some blank lines and comments have been omitted. Most of the code is given but the following tasks are left for you to do.

7.2.1 Complete the method **SumNumbers**, as marked in the code below.

7.2.2 Write the method **ShowResults** to print out the results.

7.3 It is a requirement to use a **for** statement to carry out the summation in this class. In view of the fact that the number of iterations is already known, a **for**-statement is most convenient here.

7.4 Go to the Menu class and complete the code as described in 5.2.1 to 5.3, to get the program working.

7.5 Compile, run and test the program and when you are satisfied with the performance, move on to the next section.

```

/// <summary>
/// This class takes care of the whole process of
/// (1) reading nput from the console window,
/// (2) performing the calculation and
/// (3) printing the results.
/// Let objects take care of everything that belongs to the object!!
/// </summary>
public class WholeNumbersFor
{
    //Declare a variable (aka field, instance variable, or attribute)
    private int numOfInput; //num of values to be added
    private int sum; //result of the summation

    //-----
    //public void-method that performs the whole process
    public void Start()
    {
        //Call the method which writes the program info, title, etc.
        WriteProgramInfo();
        ReadInput();
        SumNumbers();
        ShowResults();
    }

    //-----
    //void-metoden läser indata, antalet tal
    private void ReadInput()
    {
        // Determine how many numbers there are to be added
        Console.WriteLine("Number of values to sum? ");
        numOfInput = int.Parse(Console.ReadLine());
        Console.WriteLine(); //blank line
    }

    //-----
    private void WriteProgramInfo()
    {
        //you can use \n to put a blank line or use Console.WriteLine();
        Console.WriteLine("\n\n +++++ Summation of whole numbers +++++");
        Console.WriteLine("          Using a for-statement\n");
        Console.WriteLine(); //blank line
    }

    //void-method that sums upp the numbers as they are read
    //and the results are stored in the instance variable sum.
    private void SumNumbers()
    {
        //Local variables
        int index; //counter variable
        int num = 0; //stores the value that the user gives

        // A for-statement that iterates
        for (index = 0; index < numOfInput; index++)
        {
            //TO BE COMPLETED

        }
    }

    private void ShowResults()
    {
        //TO BE COMPLETED
    }
}
}

```



8. Summation of Floating Point Numbers (while-Statement)

In this section, the program lets the user feed in values that can be real number or integers. The program does not require the user to specify the number of values to be read. It stops reading when the user writes a zero value as shown in the sample program below. A while statement is the best choice when the number of iterations is not known.

8.1 There are two important requirements in this part of the assignment:

8.1.1 Use **double** data type for the numeric values.

8.1.2 Use a **while** statement to perform the iteration.

8.2 Create a new class **FloatNumbersWhile.cs** as in the previous part.

Hint: When you compare two floating-point numbers, it would be a good idea to compare the round-off values. In the example comparison is done to a 7 decimal positions. number of decimal points.

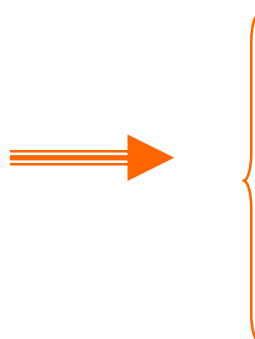
```
num = ReadInput();  
if ((Math.Round( num, 7) == 0.0))  
{  
    // code  
}
```

Math is a class from the System namespace. In the example above, it is assumed that a number that is like 0.0000004 is practically zero

8.3 Go back to the Menu class and now make the choice number two functional.

```
/// <summary>  
/// The program calculates sum of real numbers that are given by the user  
/// through a console window.  
/// </summary>  
public class FloatingNumbersWhile  
{  
    //Declare a variable (aka field, instansvariable, or attribute)  
    private double sum; //result of the sammation  
  
    public void Start()  
    {  
        //Call the method which writes the program info, title, etc.  
        WriteProgramInfo();  
        ReadInputAndSumNumbers();  
        ShowResults();  
    }  
  
    private void ReadInputAndSumNumbers()  
    {  
        double num = 0.0; //initiate to start value 0  
  
        //Read a number. If the value is given as 0, end the iteration  
        //otherwise accumulate the results in the insans-variable sum  
        bool done = false;  
  
        while (!done)  
        {  
            //TO BE COMPLETED  
            //Read a number  
            //check the value if zero, terminate the iteration (done = true)  
            //otherwise add the value to the variable sum (don't overwrite)  
  
        }  
    }  
}
```





```
private void WriteProgramInfo()
{
    //TO BE COMPLETED
}

private double ReadInput()
{
    //TO BE COMPLETED
}

private void ShowResults()
{
    //TO BE COMPLETED
}

} //class
```

9. Currency Converter (do while - Statement)

In this part we first sum up a number of values as in the previous part. The difference is that we now deal with values representing a currency. We also introduce a new facility, that is to convert the accumulated amount in the local currency to a given foreign currency. In order to do a currency conversion, the user of your program, must specify the name of the currency and the exchange rate.

10.1 There are two important requirements in this part of the assignment:

Use a **decimal** data type for currency values.

Use a **do-while** statement for the iteration.

10.2 Add a new class to your project and save it as **ExchangeDoWhile.cs**. You can now apply the copy-paste method by copying code from the previous part and paste it to this class. Change the code such that you use a do-while loop instead of while. A do-while statement is very appropriate for loops where at least one iteration is necessary.

10.3 Change the code such that you now use a **do-while** loop instead of just while. A do-while statement is very appropriate for loops where at least one iteration is necessary.

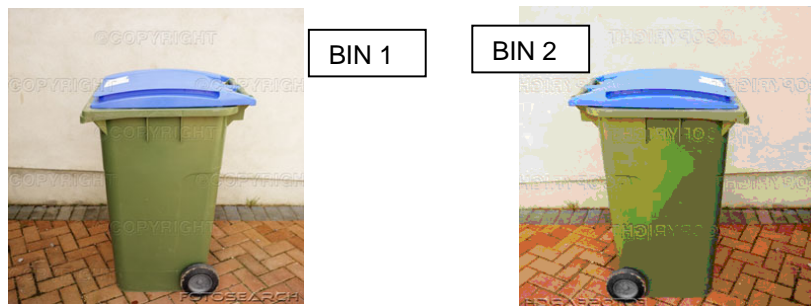
10.4 Go to the **Menu** class, create an instance of this class and call the start methods as before, build, run and test the program before going to the last section of this assignment. No more code is given in this part.

10. Waste Collection Scheduler – a real case study

Using the skills that you have now built through the previous sections, you should be able to help the residents in a community in Lund, by writing a program that lists the weeks when their household waste collection truck empties the trash bins this year.

10.5 In the so called "Krokarna", a little community in the northern Lund, the city's Department of Sanitary Services has provided two garbage bins to every house in the area.

- 10.5.1 One of these bins (Bin 1) is for household garbage (food stuff), yard wastes and clear glass.
- 10.5.2 The other one (Bin 2) contains compartments for metal wastes, colored glass, paper and carton packages
- 10.5.3 Bin 1 is emptied every other week while Bin 2 is emptied every 6th week. The first emptying took place Week No 1 for Bin 1 and Week 5 for Bin 2 this year.



10.6 The program must run iteratively until the user chooses to exit as shown in the sample program running below.

10.7 You can choose your iteration algorithm by yourself but make sure that you have a good reason for your selection. Write a little comment in your code that motivates your choice.

Remember:

A **for**-statement is used when the number of iterations is known in advance.

A **while** statement is used when the number of iterations is not known and the loop continues as long as one or more conditions prevail.

A **do-while** statement is used for similar situation as a **while**-loop, but with the difference that the iteration must loop at least once.

```

0 to exit the program.

Your choice: 1

Bin number 1 is emptied the following weeks:

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

-----

Choose type of trash bin:

1 Bin No 1. kitchen trash, household wastes (every other week)
2 Bin No 2: paper, carton packages, yard wastes, etc (every 6th
week)
0 to exit the program.

Your choice: 2

Bin number 2 is emptied the following weeks:

Week    5      Week   11      Week   17
Week   23      Week   29      Week   35
Week   41      Week   47

-----

Choose type of trash bin:

1 Bin No 1. kitchen trash, household wastes (every other week)
2 Bin No 2: paper, carton packages, yard wastes, etc (every 6th
week)
0 to exit the program.

Your choice:

```

11. Hints and guidance

The tabulation shown in the figure is programmed using the following formatting form:

```
Console.WriteLine("{0,15} {1,2}", "Week", i);
```

The method **Write** uses the format given inside the double quotes. The braces { } says to the compiler to replace them with values that come after closing quote. The value can come from variable (ex i) or a constant (ex "Week"). The numbers 0 and 1 inside the braces refer to the first and second value source in the list that comes after the format expression. The second value inside the braces, for example 15 in {0, 15} directs the compiler to right-align the value in a 15-character long string. The compiler will then add blank spaces at the left of the value giving the string a width of 15 characters. For left-adjustment, you can write {0, -15}.

A more detailed example:

```
Console.WriteLine("{0:C} is converted to {1:f2}{2} at the rate of {3:C}/{4}.",  
    sum, foreingAmount, currencyName, rate, currencyName);
```

With floating point values (values with a fractional part), the number of decimal positions can also be specified. The format {1:f2} in the code example means “show value from pos 1 (counted from 0) with 2 decimal positions; for example if the value is 2.667881, it will be rounded to 2.67. We could also specify the alignment and a width in which the value should be formatted, for example {1, 12:f2}, value no 1 (counted from 0) is to be right-aligned within a width of 10 characters.

The letter ‘C’ uses the currency format according to the regional settings in your Windows, through the Control Panel.

To format the output into columns, after every third row (for example), you can use the following algorithm:

Before the loop begins:

```
int p= 0;  
const int cols = 3;
```

Inside the loop when writing out the weeks:

```
p++;  
if ( (p >= cols) && (p % cols == 0))  
    Console.WriteLine();
```

If you would like to have 5 columns, you can simply change the value of cols from 3 to 5.

Notes:

p++ is equivalent to **p = p+1**

&& is the logical operator AND meaning that if both of the logical conditions are **true**, the result is **true** otherwise **false**. Thus if

If a = true, b = true

(a && b) is **true**, otherwise **false**.

% is the modulus operator that gives the remainder after an integer division.

(12 % 5) results in the value 2 (remainder after division = 2)

(6 % 2) gives 0 (remainder = 0).

And now comes code excerpts from the **Menu** class, as promised earlier in this document.

```
class Menu
{
    public void start()
    {
        int choice = -1;

        while (choice != 0)
        {
            WriteMenuText(); //Show the menu
            //Read user's choice
            choice = int.Parse(Console.ReadLine());
            //Take the necessary action
            switch (choice)
            {
                case 1:
                {
                    // Declare a local reference variable and
                    // create an instance of WholeNumbersFor
                    WholeNumbersFor sum = new WholeNumbersFor();

                    //call the objects start method
                    sum.Start();
                    break;
                }
                // Continue...
            }
        }

        public void WriteMenuText()
        {
            Console.WriteLine("-----");
            Console.WriteLine("                PROGRAM MENU");
            Console.WriteLine("-----");
            Console.WriteLine("    Whole Numbers with For           : 1");
            Console.WriteLine("    Floating Point Numbers with While : 2");
            Console.WriteLine("    Currency Converter with Do While-loop: 3");
            Console.WriteLine("    WasteSchedule                     : 4");
            Console.WriteLine("    Exit the program                  : 0");
            Console.WriteLine("-----");
            Console.Write("Your choice: ");
        }

        //Writes a message during about the program being under development
        private void WriteMessageCodeNotImplemented(int choice)
        {
            //Temporary message
            string str = "Your Choice is " + choice + " but the program is not complete yet.";
            str += Environment.NewLine + "Please come again!";
            Console.WriteLine(str);
        }
    }
}
```

Good Luck!

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

Farid Naisan
Instructor