

Exercise 2b: The do-while statement

1. Objectives

This exercise gives you practice with do-while loops. This type of loops is used when you don't know the number of iterations beforehand, and the iterations are to be continued until some conditions are met. Examples are: when you repeat a task "as long as" or "until" something changes, e.g. I am going to continue working until I get tired.

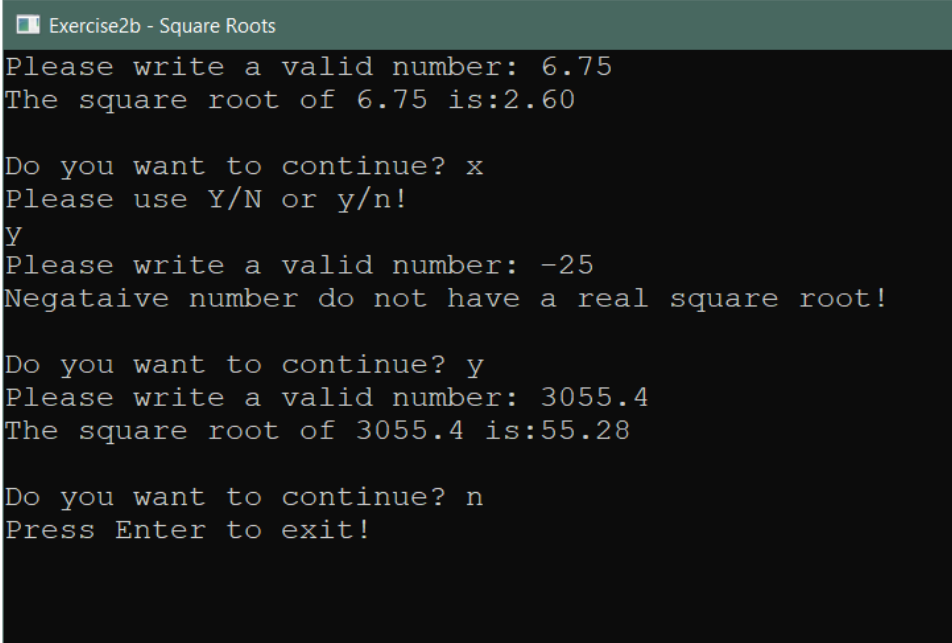
2. To Do

Create an application that calculates the root of a number. The number is to be provided by the user. Assume that the user gives a valid number each time, and uses the correct number format. The program should continue as long as the user wishes. Use the following scenario:

- a. Ask user for input (a number)
- b. Calculate the square root of the number
- c. Ask the user whether to continue
- d. If the user answers y, Y, or any word starting with any of these:
 Continue the loop from a
- e. else if the answer is n, N or any word starting with one of these
 Exit the program
- f. else
 Repeat from d.

The figure is a run example illustrating the above

The solution is given on the next page. You may try to solve the exercise before looking at the solution, but you can of course refer to the solution code whenever you feel to get an idea or if you are stuck at any point.



```
Exercise2b - Square Roots
Please write a valid number: 6.75
The square root of 6.75 is:2.60

Do you want to continue? x
Please use Y/N or y/n!
Y
Please write a valid number: -25
Negataive number do not have a real square root!

Do you want to continue? y
Please write a valid number: 3055.4
The square root of 3055.4 is:55.28

Do you want to continue? n
Press Enter to exit!
```

3. Solution:

- 3.1. Create a new project in VS, save it as Exercise2b. VS creates a start class, **Program.cs** and prepares the body of the **Main**-method for you. You can wait with this class for now, and proceed with the next step.
- 3.2. Create a new class in your project, and name it **Assignment2b**. Write the following code in this new class and save the file:

```
class WhileStatements
{
    //Input: Number whose square root is to be calculated
    private double number;

    public void Start()
    {
        bool calcAgain = true;

        //Calculate the root of a number given by the user
        //continue until the user decides to exit
        do
        {
            ReadNumber();
            CalculateRoot();

            calcAgain = CalculateAgain();

        } while (calcAgain);
    } //end of Start
} //end of class
```

Comments: The user's input should be saved as an instance variable to make it available to other methods of the class. The do while loop is used because we must run the steps at least once. The loop contains these steps: (1) Read user input, (2) Calculate and (3) ask the user whether to calculate again or exit. The first two methods are void methods, the last one (**CalculateAgain**) returns a true or false value. This value is stored in the bool variable **calcAgain**.

The loop continues as long as **calcAgain** is true. Note that all the three methods are a part of my solution; I have given them these names and now I am going to write code for these methods. You may change the names of these methods and the names of the variables.

Programming note:

```

while (calcAgain) is a shorter way of writing while (calcAgain == true)
if (condition) is the same as if (condition == true)
if (!condition) is the same as if (condition == false)
while (!condition) is the same as while (condition == false)
    
```

Console.ReadLine() reads and returns a line of text (sequence of characters) from the cursor position to the end of that line on the Command Prompt window. The end of line is marked when the Enter key is pressed by the user. It returns an empty string ("") when the Enter key is pressed without writing any character. A character is marked in code by a pair of "while a string is marked by "".

3.3. Write the method **ReadNumber()**:

```

//Read user's input, a number to calculate the square root of
private void ReadNumber()
{
    Console.Write("Please write a valid number: ");

    //Read user input, from the cursor position to end of line
    //Save the line (text) in a temporary string variable
    string str = Console.ReadLine();

    //Convert str to a number of double type.
    //The user is assumed to give the text in correct number format.
    number = double.Parse(str);
}
    
```

Comments: All input from the user through the Console window is a series of characters, put together as a string by Console.ReadLine(). So, if the user writes something like "65.5", "6", "6E4", or "2o", these become texts. We have to

convert these to numbers, in order to be able to work with them as numbers. Of these examples, the last one "2o" cannot be converted as it contains an "o" (which is not a zero). This puts the application into an abnormal situation. However, in this exercise, we assume that the user give valid characters (in a correct number format), and we leave the input control for the future modules.

3.4. Write the method **CalculateRoot** ():

C# provides thousands of classes that we can use in our programming. One of these is the class **Math**. This class has many useful methods for performing arithmetical, mathematical, and trigonometrical calculations such as, Round, Sqrt (Square Root), Sin, Cos, as well as constants like PI. In addition, the methods and constants are defined as "static" which means that we can call the methods as well as use the constant values by the class name, without the need for creating an object (instance) of the **Math** class. Examples:

Math.PI	gives the value of PI (3.14...)
Math.Pow(x, y)	returns the value x raised to the power of y.

Note: for numbers to the power of small whole numbers like 2 or 3, use multiplication (x *x, or x *x*s) as this is faster and gives better precision.

```
//Calculate the square root of the value saved in the variable, number
//Display the result
private void CalculateRoot()
{
    if (number >= 0)
    {
        double root = Math.Sqrt(number);
        Console.WriteLine("The square root of " + number + " is:" + root.ToString("0.00"));
    }
    else //negative number
        Console.WriteLine("Negataive number do not have a real square root!");
}
```

Comments: Math.Sqrt calculates the square root of a double number.

3.5. Write the method **CalculateAgain**:

We write this method to ask the user, after each calculation, whether to continue calculating, or exit the application. The user's response (y or n) is saved in a Boolean type, which in C# is called **bool**. The difference between this method and those we wrote earlier is that this method is not **void** but rather a **bool**.

Moreover, the user is required to give a valid answer (Y, y, n, N) for which we use a loop and continue until the answer is correct.

```
//Ask user to whether to continue,
//return true if the user answers Y, y or any word beginning
//with these, or N, n or any word beginning with these.
//Otherwise, return false.
private bool CalculateAgain()
{
    bool answer = false; //true = y, false = n

    Console.WriteLine(); //blankline
    Console.Write("Do you want to continue? ");

    bool goodAnswer = false; //make user answer correctly

    do
    {
        string str = Console.ReadLine();
        //change str to capital letters to make comparison
        //easier
        str = str.ToUpper();

        if (str[0] == 'Y')    //str[0]is the first letter in the string
        {
            answer = true; // continue wit
            goodAnswer = true;
        }
        else if (str[0] == 'N')
        {
            answer = false; // do not continue with calculation
        }
    }
}
```

```
        goodAnswer = true;
    }
    else //anything else
    {
        Console.WriteLine("Please use Y/N or y/n!");
        goodAnswer = false;
    }

    } while (!goodAnswer); //same as: if (goodAnswer == false)

    return answer;
} //end of method
} //end of class
} //end of namespace
```

Comments: In this method, two Boolean variables are used, one (**answer**) to store user's answer to be returned to the caller, provided the answer begins with a valid letter, i.e Y, y, N, n. The other one, (**goodAnswer**) is used inside the do-while loop to control the loop. If the user's answer is correct, the loop does not repeat, otherwise, the user is prompted again, and a new iteration is performed.

4. Further development

4.1. Use a while-statement in all the do-while loops used in this exercise. Here is an example:

```
public void Start()
{
    bool calcAgain = true; //1

    //Calculate the root of a number given by the user
    //continue until the user decides to exit
    while (calcAgain)
    {
        ReadNumber();
        CalculateRoot();

        calcAgain = CalculateAgain();
    }
}
```

As you can see, the condition is to be met already at the beginning of the loop. If we had initiated **calcAgain** to false (line marked 1), then the loop would never be executed.

The difference is twofold:

- a. The **do-while** statement is iterated at least one time whereas the **while** statement may not be iterated at all.
- b. To continue the iterations, the condition is checked at the end of the loop in case of **do-while** loop whereas the condition is checked at the beginning of the loop when using the **while** statement.

Note that there is a semicolon after the condition expression in case of **do-while**.

```
do
{
    //code
} while (condition);
```

```
while (condition)
{
    //code
}
```

4.2. Question: Can **do-while** and **while** statements be rewritten using for-statements? Try one of the loops and use a **for**-statement. For-statements are not appropriate to use in cases where you cannot determine the number of iterations for the loop, but it is possible to use a **while** statement (not always a do-while) instead of a **for** statement and vice versa.

Good Luck!