University of Anbar
College of Computer Science
and Information Technology
Computer Network Systems Department

Visual Programming I

Lecture Three

Third Stage

First Course 2021 - 2022

Seddiq Qais Abd Al-Rahman

MSc Computer Science

co.sedeikaldossary@uoanbar.edu.iq

Visual Programming

Visual Studio

In programming, it is often desired to execute certain block of statements for a specified number of times. A possible solution will be to type those statements for the required number of times. However, the number of repetition may not be known in advance (during compile time) or maybe large enough (say 10000).

The best solution to such problem is loop. Loops are used in programming to repeatedly execute a certain block of statements until some condition is met.

In this article, we'll learn to use while loops in C#.

C# while loop

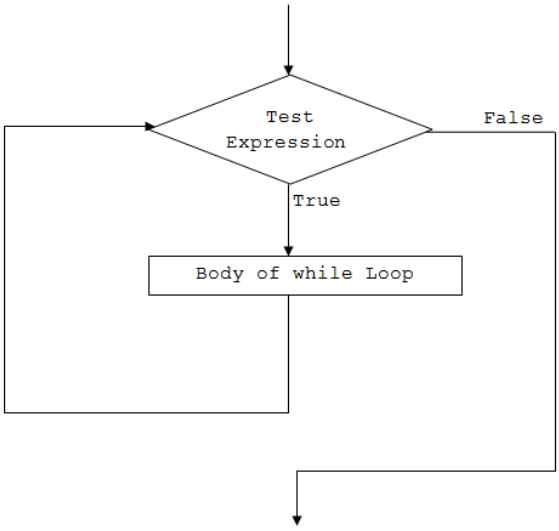
The **while** keyword is used to create while loop in C#. The syntax for while loop is:

```
while (test-expression)
{
        // body of while
}
```

How while loop works?

- C# while loop consists of a test-expression.
- If the test-expression is evaluated to true,
- statements inside the while loop are executed.
- after execution, the test-expression is evaluated again.
- If the test-expression is evaluated to false, the while loop terminates.

while loop Flowchart



Example: while Loop

```
using System;

namespace Loop
{
        class WhileLoop
        {
                public static void Main(string[] args)
                {
                        int i=1;
                        while (i<=5)
                        {
                                Console.WriteLine("C# For Loop: Iteration
{0}", i);
                                i++;
                        }
                }
        }
}
```

When we run the program, the output will be:

```
C# For Loop: Iteration 1
C# For Loop: Iteration 2
C# For Loop: Iteration 3
C# For Loop: Iteration 4
C# For Loop: Iteration 5
```

Initially the value of i is 1.

When the program reaches the while loop statement,

- the test expression i <=5 is evaluated. Since i is 1 and 1 <= 5 is true, it executes the body of the while loop. Here, the line is printed on the screen with Iteration 1, and the value of i is increased by 1 to become 2.
- Now, the test expression (i <=5) is evaluated again. This time too, the expression returns true (2 <= 5), so the line is printed on the screen and the value of i is now incremented to 3..
- This goes and the while loop executes until i becomes 6. At this point, the test-expression will evaluate to false and hence the loop terminates.

Example: while loop to compute sum of first 5 natural numbers

```
using System;

namespace Loop
{
```

```
class WhileLoop
{
        public static void Main(string[] args)
        {
                int i=1, sum=0;

                while (i<=5)
                {
                        sum += i;
                        i++;
                }
                Console.WriteLine("Sum = {0}", sum);
        }
}
}
```

When we run the program, the output will be:

Sum = 15

This program computes the sum of first 5 natural numbers.

- Initially the value of sum is initialized to 0.
- On each iteration, the value of sum is updated to sum+i and the value of i is incremented by 1.
- When the value of i reaches 6, the test expression i<=5 will return false and the loop terminates.

Let's see what happens in the given program on each iteration.

Initially, i = 1, sum = 0

Iteration	Value of i	i<=5	Value of sum
1	1	true	0+1 = 1
2	2	true	1+2 = 3
3	3	true	3+3 = 6
4	4	true	6+4 = 10
5	5	true	10+5 = 15
6	6	false	Loop terminates

So, the final value of sum will be 15.

C# do...while loop

The **do** and **while** keyword is used to create a do...while loop. It is similar to a while loop, however there is a major difference between them. In while loop, the condition is checked before the body is executed. It is the exact opposite in do...while loop, i.e. condition is checked after the body is executed. This is why, the body of do...while loop will execute at least once irrespective to the test-expression.

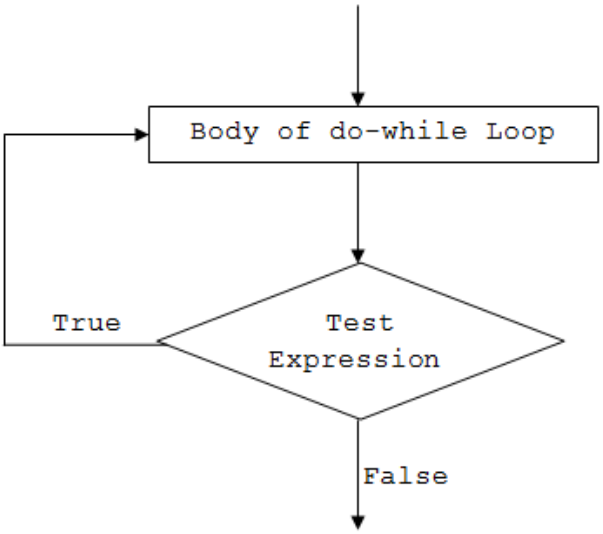
The syntax for do...while loop is:

```
do
{
        // body of do while loop
} while (test-expression);
```

How do...while loop works?

1. The body of do...while loop is executed at first.
2. Then the test-expression is evaluated.
3. If the test-expression is true, the body of loop is executed.
4. When the test-expression is false, do...while loop terminates.

do...while loop Flowchart



Example: do...while loop

```
using System;
namespace Loop
{
        class DoWhileLoop
        {
                public static void Main(string[] args)
                {
                        int i = 1, n = 5, product;
                        do
                        {
                                product = n * i;
                                Console.WriteLine("{0} * {1} = {2}", n, i,
product);
                                i++;
                        } while (i <= 10);
```

```
            }
        }
    }
```

When we run the program, the output will be:

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

As we can see, the above program prints the multiplication table of a number (5).

- Initially, the value of i is 1. The program, then enters the body of do..while loop without checking any condition (as opposed to while loop).
- Inside the body, product is calculated and printed on the screen. The value of i is then incremented to 2.
- After the execution of the loop's body, the test expression i <= 10 is evaluated. In total, the do...while loop will run for 10 times.
- Finally, when the value of i is 11, the test-expression evaluates to false and hence terminates the loop.

Infinite while and do...while loop

If the test expression in the while and do...while loop never evaluates to false, the body of loop will run forever. Such loops are called infinite loop. For example:

Infinite while loop

```
while (true)
{
        // body of while loop
}
```

Infinite do...while loop

```
do
{
        // body of while loop
} while (true);
```

The infinite loop is useful when we need a loop to run as long as our program runs. For example, if your program is an animation, you will need to constantly run it until it is

stopped. In such cases, an infinite loop is necessary to keep running the animation repeatedly.

Example: while loop with if statement

```
using System;
class Kodify_Example
{
   static void Main()
   {
      string[] websites = { "Google", "Facebook", "Twitter", "Instagram", "YouTube"
};
      int i = 0;
      while (i < websites.Length)
      {
         // Only output those sites with an 'o' in their name
         if (websites[i].Contains("o"))
         {
            Console.WriteLine(websites[i]);
         }
         i++;
      }
   }
}
```

Output:

```
Google
Facebook
YouTube
```

C# for loop

The **for** keyword is used to create for loop in C#. The syntax for **for loop** is:

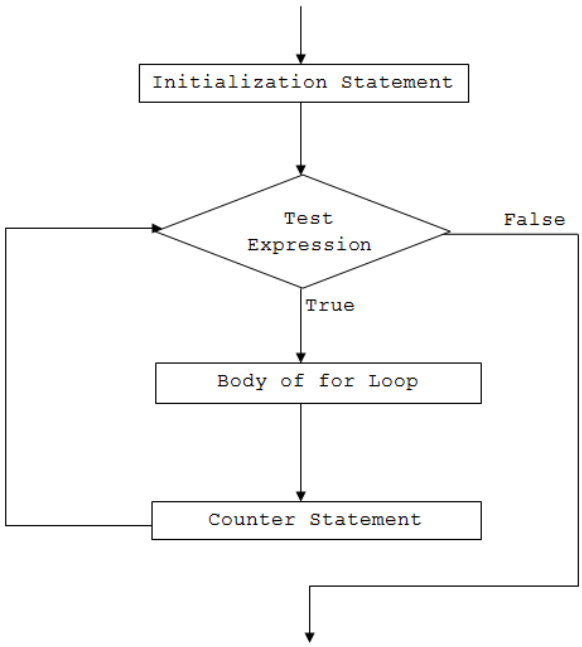
```
for (initialization; condition; iterator)
{
        // body of for loop
}
```

How for loop works?

1. C# for loop has three statements: initialization, condition and iterator.
2. The initialization statement is executed at first and only once. Here, the variable is usually declared and initialized.

3. Then, the condition is evaluated. The condition is a boolean expression, i.e. it returns either true or false.
4. If the condition is evaluated to true:
   A. The statements inside the for loop are executed.
   B. Then, the iterator statement is executed which usually changes the value of the initialized variable.
   C. Again the condition is evaluated.
   D. The process continues until the condition is evaluated to false.
5. If the condition is evaluated to false, the for loop terminates.

for Loop Flowchart



Example: C# for Loop

```
using System;
namespace Loop
{
        class ForLoop
        {
                public static void Main(string[] args)
                {
                        for (int i=1; i<=5; i++)
                        {
                        Console.WriteLine("C# For Loop: Iteration {0}", i);
                        }
                }
        }
}
```

When we run the program, the output will be:

```
C# For Loop: Iteration 1
```

```
C# For Loop: Iteration 2
C# For Loop: Iteration 3
C# For Loop: Iteration 4
C# For Loop: Iteration 5
```

In this program,

- initialization statement is int i=1
- condition statement is i<=5
- iterator statement is i++

When the program runs,

- First, the variable i is declared and initialized to 1.
- Then, the condition (i<=5) is evaluated.
- Since, the condition returns true, the program then executes the body of the for loop. It prints the given line with Iteration 1 (Iteration simply means repetition).
- Now, the iterator (i++) is evaluated. This increments the value of i to 2.
- The condition (i<=5) is evaluated again and at the end, the value of i is incremented by 1. The condition will evaluate to true for the first 5 times.
- When the value of i will be 6 and the condition will be false, hence the loop will terminate.

Example: for loop to compute sum of first n natural numbers

```
using System;
namespace Loop
{
        class ForLoop
        {
                public static void Main(string[] args)
                {
                        int n = 5,sum = 0;
                        for (int i=1; i<=n; i++)
                        {
                                // sum = sum + i;
                                sum += i;
                        }
                        Console.WriteLine("Sum of first {0} natural numbers =
{1}", n, sum);
                }
        }
}
```

When we run the program, the output will be:

```
Sum of first 5 natural numbers = 15
```

Here, the value of sum and n are initialized to 0 and 5 respectively. The iteration variable i is initialized to 1 and incremented on each iteration. Inside the for loop, value of sum is

incremented by i i.e. sum = sum + i. The for loop continues until i is less than or equal to n (user's input). Let's see what happens in the given program on each iteration.

Initially, i = *1*, sum = *0 and* n = *3*

Iteration	Value of i	i<=5	Value of sum
1	1	true	0+1 = 1
2	2	true	1+2 = 3
3	3	true	3+3 = 6
4	4	true	6+4 = 10
5	5	true	10+5 = 15
6	6	false	Loop terminates

So, the final value of sum will be 15 when n = 5.

Multiple expressions inside a for loop

We can also use multiple expressions inside a for loop. It means we can have more than one initialization and/or iterator statements within a for loop. Let's see the example below.

Example: for loop with multiple initialization and iterator expressions

```
using System;
namespace Loop
{
    class ForLoop
    {
        public static void Main(string[] args)
        {
            for (int i=0, j=0; i+j<=5; i++, j++)
            {
                Console.WriteLine("i = {0} and j = {1}", i,j);
            }
        }
    }
}
```

When we run the program, the output will be:

```
i = 0 and j = 0
i = 1 and j = 1
i = 2 and j = 2
```

In this program, we have declared and initialized two variables: i and j in the initialization statement. Also, we have two expressions in the iterator part. That means both i and j are incremented by 1 on each iteration.

For loop without initialization and iterator statements

The initialization, condition and the iterator statement are optional in a for loop. It means we can run a for loop without these statements as well.
In such cases, for loop acts as a while loop. Let's see the example below.

Example: for loop without initialization and iterator statement

```
using System;

namespace Loop
{
        class ForLoop
        {
                public static void Main(string[] args)
                {
                        int i = 1;
                        for ( ; i<=5; )
                        {
                                Console.WriteLine("C# For Loop: Iteration
{0}", i);
                                i++;
                        }
                }
        }
}
```

When we run the program, the output will be:

```
C# For Loop: Iteration 1
C# For Loop: Iteration 2
C# For Loop: Iteration 3
C# For Loop: Iteration 4
C# For Loop: Iteration 5
```

In this example, we haven't used the initialization and iterator statement.

The variable i is initialized above the for loop and its value is incremented inside the body of loop.

Similarly, the condition is also an optional statement. However if we don't use test expression, the for loop won't test any condition and will run forever (infinite loop).

Infinite for loop

If the condition in a for loop is always true, for loop will run forever. This is called infinite for loop.

Example: Infinite for loop

```
using System;
namespace Loop
{
        class ForLoop
        {
```

```
                public static void Main(string[] args)
                {
                        for (int i=1 ; i>0; i++)
                        {
                        Console.WriteLine("C# For Loop: Iteration {0}", i);
                        }
                }
        }
}
```

Here, i is initialized to 1 and the condition is i>0. On each iteration we are incrementing the value of i by 1, so the condition will never be false. This will cause the loop to execute infinitely. We can also create an infinite loop by replacing the condition with a blank. For example,

```
for ( ; ; )
{
        // body of for loop
}
```

or

```
for (initialization ; ; iterator)
{
        // body of for loop
}
```

Random Number And Random String Generator

There are scenarios where we are required to generate random numbers, alphabets, characters, etc. For achieving this we have Random class available in the System namespace. The random class allows you to randomly generate an integer value. Using this random class one can generate a different set of numbers/characters. The **Next()** Method of **System.Random** class in C# is used to get a random integer number. This method can be overloaded by passing different parameters to it as follows:

- Next()
- Next(Int32)
- Next(Int32, Int32)
- Next() Method

Next() Without Argument:

This method is used to returns a non-negative random integer.

Syntax: public virtual int Next ();

Return Value: This method returns the 32-bit signed integer which is greater than or equal to 0 and less than *MaxValue*.

Example:

```
using System;
class GFG {
   public static void Main()
   {
Random ran = new Random();
int a = ran.Next();
Console.WriteLine("The random number generated is: {0}", a);
Console.ReadKey();
   }
}
```

Output:

```
The random number generated is: 157909285
```

Next() With One Argument:

Next overload for the Random.Next() accepts one argument. The argument provided specifies the maximum value that can be generated by the method. The max value should be either greater than or equal to zero. It returns a non-negative integer with max value as the argument provided by the user.

Next(Int32) Method: This method is used to get a non-negative random integer which is less than the specified maximum.

Syntax: public virtual int Next (int maxValue);

Here, *maxValue* is the upper boundary of the random number to be generated. It must be greater than or equal to 0.

Return Value: The function returns a 32-bit signed integer which is greater than or equal to 0, and less than *maxValue*. However, if *maxValue* equals 0, *maxValue* is returned.

Exception: This method will give *ArgumentOutOfRangeException* if the *maxValue* is less than 0.

Example:

```
using System;
 class GFG {
   public static void Main()
```

```
    {
        Random rand = new Random();
        Console.WriteLine("Printing 10 random numbers less than 100");
        for (int i = 1; i <= 10; i++)
            Console.WriteLine("{0} -> {1}", i, rand.Next(100));
    }
}
```

Output:

```
Printing 10 random numbers less than 100
1 -> 19
2 -> 94
3 -> 14
4 -> 54
5 -> 94
6 -> 73
7 -> 39
8 -> 42
9 -> 18
10 -> 77
```

Next() With Two Arguments:

Random class is used to simulate a random event. To generate a random character, we use Next(). The Next() accepts two arguments, the first one is the minimum and inclusive value allowed for the random generator. The second argument accepts the maximum exclusive value. A maximum exclusive value means that the value passed in the second argument will never be generated. The generated value will always be less than the max value.

Next(Int32, Int32) Method: This method is used to get the random integer that is within a specified range.

Syntax: public virtual int Next (int minValue, int maxValue);

Parameters:

maxValue: It is the exclusive upper boundary of the random number generated. It must be greater than or equal to minValue.

minValue: It is the inclusive lower bound of the random number returned.

Return Value: The function returns a 32-bit signed integer greater than or equal to *minValue* and less than *maxValue*; that is, the range of return values includes *minValue* but not *maxValue*. If *minValue* equals *maxValue*, *minValue* is returned.

Exception: This method will give the *ArgumentOutOfRangeException* if the *minValue* is greater than *maxValue*.

Example:

```
using System;
 class GFG {
   public static void Main()
   {
      Random rand = new Random();
      Console.WriteLine("Printing 10 random numbers"+
                        " between 50 and 100");
      for (int i = 1; i <= 10; i++)
         Console.WriteLine("{0} -> {1}", i, rand.Next(50, 100));
   }
}
```

Output:

```
Printing 10 random numbers between 50 and 100
1 -> 91
2 -> 85
3 -> 93
4 -> 74
5 -> 88
6 -> 77
7 -> 92
8 -> 76
9 -> 77
10 -> 52
```

Generate Random Alphabets

We can generate a random alphabet by using the random class. Although Random class only returns an integer, we can use that to generate random alphabets. The easiest way to do that is to combine the "ElementAt" method with the Random.Next() to point out the position of a random alphabet from the series of alphabets.

Example:

```
class Program
    {
public static void Main(string[] args)
        {
            Random ran = new Random();
            String b = "abcdefghijklmnopqrstuvwxyz";
            int length = 6;
```

```
            String random = "";
            for(int i =0; i<length; i++)
            {
                int a = ran.Next(26);
                random = random + b.ElementAt(a);
            }
        Console.WriteLine("The random alphabet generated is: {0}", random);
        Console.ReadLine();
        }
    }
```

The output of the above program will be:

The random alphabet generated is: icysjd

Code Explanation: Similar to our previous examples, here we created a Random object. Then we stored all the alphabets in a string i.e. String b. We defined a variable called "length" of integer type which will denote the number of characters required in a randomly generated string. We initialized empty string random, where we will store our alphabets. Then we wrote a for loop. Inside the for loop we used Random.Next() to generate a random number less than 26 because the number of alphabets we stored in the String b is 26. You can also other numbers depending on the requirement. Hence, the int a will have a random number generated during each loop cycle, then that number will be used as a position indicator to get the character that position using ElementAt(). This will give a random character every time when the loop runs. Then we will append the characters together on each loop cycle and we will get the required string with the given length.

The following program will generate an 8-digit random alphanumeric output with the last two digits as special characters.

```
class Program
    {
public static void Main(string[] args)
        {
            Random ran = new Random();
            String b = "abcdefghijklmnopqrstuvwxyz0123456789";
            String sc = "!@#$%^&*~";
            int length = 6;
            String random = "";
for(int i =0; i<length; i++)
            {
                int a = ran.Next(b.Length); //string.Lenght gets the size of string
                random = random + b.ElementAt(a);
            }
for(int j =0; j<2; j++)
            {
```

```
                    int sz = ran.Next(sc.Length);
                    random = random + sc.ElementAt(sz);
                }
        Console.WriteLine("The random alphabet generated is: {0}", random);
        Console.ReadLine();
            }
        }
```

The output of the above program will be:

*The random alphabet generated is: 718mzl~^*

Exercises

1. Write a program in C# Sharp to display the first 10 natural numbers.
2. Write a C# Sharp program to find the sum of first 10 natural numbers.
3. Write a program in C# Sharp to display n terms of natural number and their sum.
4. Write a program in C# Sharp to read 10 numbers from keyboard and find their sum and average.
5. Write a program in C# Sharp to display the cube of the number upto given an integer.
6. Write a program in C# Sharp to display the multiplication table of a given integer.
7. Write a program in C# Sharp to display the multiplication table vertically from 1 to n.
8. Write a program in C# Sharp to display the n terms of odd natural number and their sum.
9. Write a program in C# Sharp to display the pattern like right angle triangle using an asterisk. The pattern like :

```
*
**
***
****
```

10. Write a C#Sharp program to display alphabet pattern like S with an asterisk.

```
 ****
*
*
 ***
    *
    *
****
```

References:

- Tony Gaddis, “Starting out with Visual C#.”, Fourth edition, Boston, Pearson Inc., 2017.
- Salvatore A. Buono, “C# and Game Programming: A Beginner's Guide.” Second Edition, Boca Raton, CRC Press Inc., 2019.
- Eric Butow and Tommy Ryan, "C#: Your Visual Blueprint for Building .NET Applications.", Hungry Minds Inc., New York, 2002.
- Faraz Rasheed, “Programmer's Heaven: C# School.”, First Edition, Fuengirola, Synchron Data, 2006.