Mr. Bellavia

# LESSON 01 - WHILE-LOOPS

In this lesson we will examine the use of while-loops and discuss the difference between definite and indefinite loops. We will also look at the use of counter & accumulator variables. We will also briefly examine the do-while-loop.

**Sections:**

## I. REPETITION STATEMENTS (USING THE WHILE-STATEMENT):

You may have noticed that some of our programs that were written would sometimes contain **repetitive** code. For example:

```
double mark1 = 0, mark2 = 0, mark3 = 0, mark4 = 0;

Console.Write("Enter course mark: ");
mark1 = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter course mark: ");
mark2 = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter course mark: ");
mark3 = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter course mark: ");
mark4 = Convert.ToDouble(Console.ReadLine());

double total = mark1 + mark2 + mark3 + mark4;
double avg = total / 4.0;
Console.WriteLine("Your average is: " + avg);
```

Doing this is fine, but our code can be shortened using a **loop.** Consider the following code:

```
double mark = 0, total = 0;

while(true)
{
  Console.Write("Enter course mark: ");
  mark = Convert.ToDouble(Console.ReadLine());
  total = total + mark;
}

double avg = total / 4.0;
Console.WriteLine("Your average is: " + avg);
```

Sample **Input** & **Output**:

```
Enter a course mark: 75
Enter a course mark: 87
Enter a course mark: 34
Enter a course mark: 65
Enter a course mark: 65
```
(… keeps repeating…)

The above code demonstrates a **while-loop**. All the lines of code inside the curly braces **{ }** will keep repeating. The problem with the above code is that this loop will never stop! When a loop does not stop, it is called an **indefinite-loop** (we will look more at these in the next section). This is due to the condition of our **while-loop** in the brackets ( ). The condition here is always going to be true.

Let's modify the code so that we only ask for 4 marks:

```
double mark = 0, total = 0;
int x = 1;

while(x < 5)
{
  Console.Write("Enter course mark " + x + ": ");
  mark = Convert.ToDouble(Console.ReadLine());
  total = total + mark;
  x++;
}

double avg = total / 4.0;
Console.WriteLine("Your average is: " + avg);
```

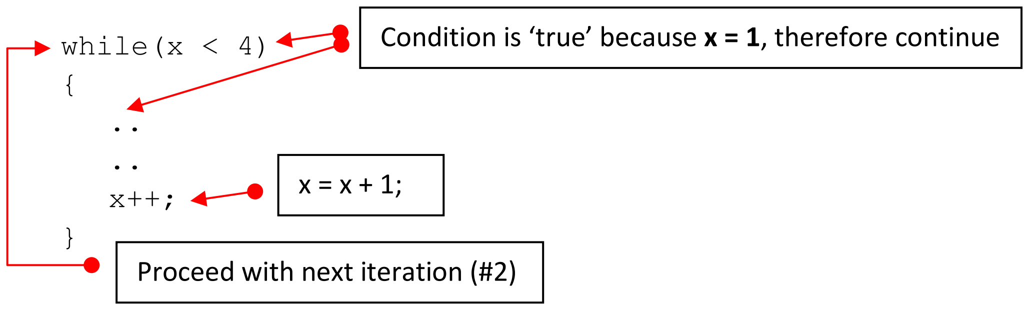
Sample **Input** & **Output**:

```
Enter course mark 1: 79
Enter course mark 2: 86
Enter course mark 3: 81
Enter course mark 4: 85
Your average is: 82.75
```

Notice that our **while-loop** now has a condition that says **while(x < 5).** This means that the loop will continue repeating until x is equal to 5. Initially we set **x** equal to **'1'**, then the last line in our **while-loop** increments x by 1 **(x++).**

Let's break this down. Every time the while-loop cycles **(iterates)** it checks if the condition is true and continues appropriately:

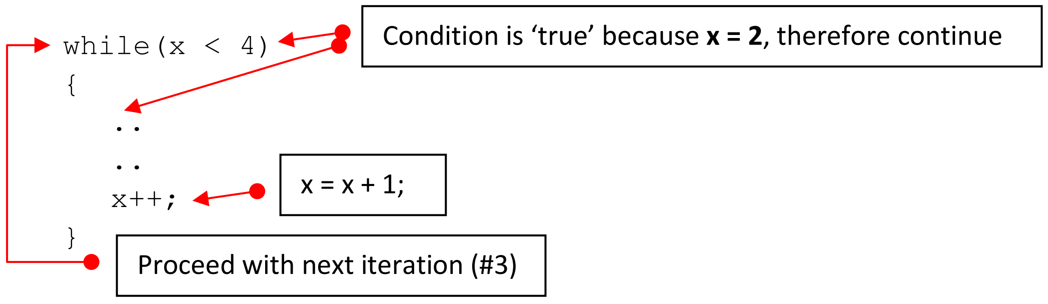
**Iteration #1:**



```
while(x < 4)
{
    ..
    ..
    x++;
}
```

Condition is 'true' because **x = 1**, therefore continue
x = x + 1;
Proceed with next iteration (#2)

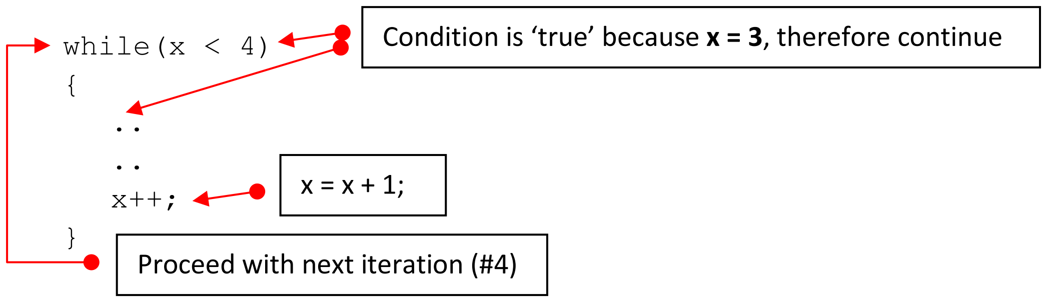
**Iteration #2:**



```
while(x < 4)
{
    ..
    ..
    x++;
}
```

Condition is 'true' because **x = 2**, therefore continue
x = x + 1;
Proceed with next iteration (#3)

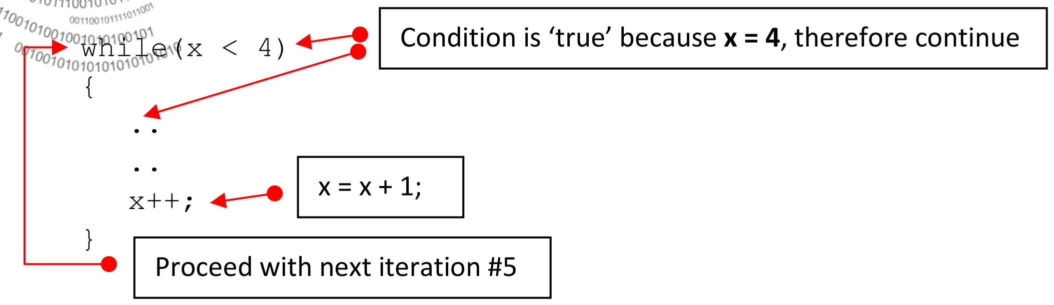
**Iteration #3:**



```
while(x < 4)
{
    ..
    ..
    x++;
}
```

Condition is 'true' because **x = 3**, therefore continue
x = x + 1;
Proceed with next iteration (#4)

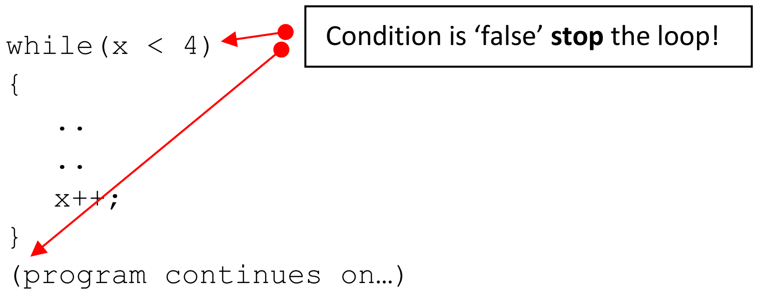
**Iteration #4:**



```
while(x < 4)
{
    ..
    ..
    x++;
}
```

Condition is 'true' because **x = 4**, therefore continue
x = x + 1;
Proceed with next iteration #5

**Iteration #5:**



```
while(x < 4)
{
    ..
    ..
    x++;
}
(program continues on…)
```

As you can see the loop keeps repeating until x is equal to 5. We say that this while-loop has 4 **iterations** (i.e., cycles 4 times). On the 5th iteration, the loop **breaks** (i.e., stops executing).

Let's look again inside our **while-loop:**

```
while(x < 5)
{
  Console.Write("Enter course mark " + x + ": ");
  mark = Convert.ToDouble(Console.ReadLine());
  total = total + mark;
  x++;
}
```

Notice how we use the value of **'x'** in our output statement so that the user knows what mark they are inputting:

```
Console.Write("Enter course mark " + x + ": ");
```

Since 'x' increments by 1 on every iteration, then our output will show the current mark that is being expected from the user.

Also, notice how we increase our **‘total’** and **‘x’** variables:

```
total = total + mark;
x++;
```

On each **iteration:**

- **total** will equal itself plus **‘mark’**
- **x** will equal itself plus **‘1’.**

Therefore:

**Iteration #1:**

| | | | |
|---|---|---|---|
| total | = total + mark | x | = x + 1 |
| | = 0 + 79 | x | = 0 + 1 |
| | = 79 | x | = 1 |

**Iteration #2:**

| | | | |
|---|---|---|---|
| total | = total + mark | x | = x + 1 |
| | = 79 + 86 | x | = 1 + 1 |
| | = 165 | x | = 2 |

**Iteration #3:**

| | | | |
|---|---|---|---|
| total | = total + mark | x | = x + 1 |
| | = 165 + 81 | x | = 2 + 1 |
| | = 246 | x | = 3 |

**Iteration #4:**

| | | | |
|---|---|---|---|
| total | = total + mark | x | = x + 1 |
| | = 246 + 85 | x | = 3 + 1 |
| | = 331 | x | = 4 |

When we add a variable to itself plus another value the variable is referred to as an **accumulator**. Therefore, our **‘total’** variable above is an **accumulator.**

When we add a variable to itself plus ‘1’ then the variable is referred to as a **counter**. Therefore, our **‘x’** variable above is a **counter.**

## II. INDEFINITE LOOPS:

In the previous example our **while-loop** was **definite** (i.e., we knew the number of times that we wanted to loop which was 4 times). Sometimes we will not know how many times we need to run a loop. Consider the following example where we ask the user to guess a number between 1 and 20. We will keep asking until the user guesses correctly:

```
int num = 12;
int guess = 0;

while(guess != num)
{
  Console.Write("Please guess a number from 1 to 20: ");
  guess = Convert.ToInt32(Console.ReadLine());
}

Console.WriteLine("Your guess is correct!");
```

Sample **Input** & **Output**:

```
Please guess a number from 1 to 20: 2
Please guess a number from 1 to 20: 4
Please guess a number from 1 to 20: 10
Please guess a number from 1 to 20: 6
Please guess a number from 1 to 20: 12
```

Notice our while-loop condition reads: **while(guess != num).** If the **guess** inputted by the user is **not** equal to **num** then the condition will be **true** and the loop will keep repeating (i.e., we will keep asking the user for a number). Once the user guesses correctly the while-loop stops executing and the user has guessed correctly.

This is an example of an **indefinite (infinite)** loop because we do not know how many times the loop my iterate. **Note:** You must be careful to not crash your program with an indefinite-loop. In such a case, the loop runs for ever and your program becomes unresponsive (like the initial example of this lesson). Indefinite-loops can be useful for example: when you play/design a video game, you are essentially in an infinite-loop.

## III. DO-WHILE-LOOPS:

Sometimes it is necessary to have the **condition** of our **while-loop** occur at the **end of the loop** instead of at the beginning. We can do this with a **do-while-loop**. Using the example from before we can construct a **do-while-loop** like this:

```
int num = 12;
int guess = 0;

do
{
  Console.Write("Please guess a number from 1 to 20: ");
  guess = Convert.ToInt32(Console.ReadLine());
}while(guess != num);

Console.WriteLine("Your guess is correct!");
```

The program will run the exact same. Most of the time you can stick with using a regular **while-loop**, but you may see sample code that uses the **do-while-loop.** The main difference is that if you have your condition at the beginning of a loop (like a while-loop) it is a **pre-condition**, and if you have your condition at the end of a loop (like a do-while-loop) it is a **post-condition**.