Contents

While Loops

Introduction to `while` loops

- There are two basic types of decision structures in all programming languages. We've just learned about the first, which is the "selection structure," or **`if`** statement. This allows the program to choose whether or not to execute a block of code, based on a condition.
- The second basic decision structure is the loop, which allows the program to execute the same block of code repeatedly, and choose when to stop based on a condition.
- The **while statement** executes a block of code repeatedly, *as long as a condition is true*. You can also think of it as executing the code repeatedly *until a condition is false*

Example code with a `while` loop

```
int counter = 0;
while(counter <= 3)
{
    Console.WriteLine("Hello again!");
    Console.WriteLine(counter);
    counter++;
}
Console.WriteLine("Done");
```

- After the keyword **`while`** is a condition, in parentheses: `counter <= 3`
- On the next line after the **`while`** statement, the curly brace begins a code block. The code in this block is "controlled" by the **`while`** statement.
- The computer will repeatedly execute that block of code as long as the condition `counter <= 3` is true

- Note that inside this block of code is the statement `counter++`, which increments `counter` by 1. So eventually, `counter` will be greater than 3, and the loop will stop because the condition is false.
- This program produces the following output:

```
Hello again!
0
Hello again!
1
Hello again!
2
Hello again!
3
Done
```

Syntax and rules for `while` loops

- Formally, the syntax for a **`while`** loop is this:

```
while(<condition>)
{
    <statements>
}
```

- Just like with an **`if`** statement, the condition is any expression that produces a `bool` value (including a `bool` variable by itself)
- When the computer encounters a **`while`** loop, it first evaluates the condition
- If the condition is false, the loop body (code block) is skipped, just like with an **`if`** statement
- If the condition is true, the loop body is executed
- After executing the loop body, the computer goes back to the **`while`** statement and evaluates the condition again to decide whether to execute the loop again
- Just like with an **`if`** statement, the curly braces can be omitted if the loop body is just one statement:

```
while(<condition>)
    <statement>
```

- Examining the example in detail
- When our example program executes, it initializes `counter` to 0, then it encounters the loop

- It evaluates the condition `counter <= 0`, which is true, so it executes the loop's body. The program displays "Hello again!" and "0" on the screen.
- At the end of the code block (after `counter++`) the program returns to the **`while`** statement and evaluates the condition again. 1 is less than 3, so it executes the loop's body again.
- This process repeats two more times, and the program displays "Hello again!" with "2" and "3"
- After displaying "3", `counter++` increments `counter` to 4. Then the program returns to the **`while`** statement and evaluates the condition, but `counter <= 3` is false, so it skips the loop body and executes the last line of code (displaying "Done")

While loops may execute zero times

- You might think that a "loop" always repeats code, but nothing requires a while loop to execute at least once
- If the condition is false when the computer first encounters the loop, the loop body is skipped
- For example, if we initialize `counter` to 5 with our previous loop:

  ```
  int counter = 5;
  while(counter <= 3)
  {
      Console.WriteLine("Hello again!");
      Console.WriteLine(counter);
      counter++;
  }
  Console.WriteLine("Done");
  ```

  The program will only display "Done," because the body of the loop never executes. `counter <= 3` is false the first time it is evaluated, so the program skips the code block and continues on the next line.

Ensuring the loop ends

- If the loop condition is always true, the loop will never end, and your program will execute "forever" (until you forcibly stop it, or the computer shuts down)
- Obviously, if you use the value **`true`** for the condition, the loop will execute forever, like in this example:

```
int number = 1;
while (true)
    Console.WriteLine(number++);
```

- If you do not intend your loop to execute forever, you must ensure the statements in the loop's body do something to *change a variable* in the loop condition, otherwise the condition will stay true
- For example, this loop will execute forever because the loop condition uses the variable `counter`, but the loop body does not change the value of `counter`:

```
int counter = 0;
while(counter <= 3)
{
    Console.WriteLine("Hello again!");
    Console.WriteLine(counter);
}
Console.WriteLine("Done");
```

- This loop will also execute forever because the loop condition uses the variable `num1`, but the loop body changes the variable `num2`:

```
int num1 = 0, num2 = 0;
while(num1 <= 5)
{
    Console.WriteLine("Hello again!");
    Console.WriteLine(num1);
    num2++;
}
Console.WriteLine("Done");
```

- It's not enough for the loop body to simply change the variable; it must change the variable in a way that will eventually *make the condition false*
  - For example, if the loop condition is `counter <= 5`, then the loop body must increase the value of `counter` so that it is eventually greater than 5
  - This loop will execute forever, even though it changes the right variable, because it changes the value in the wrong "direction":

```
int number = 10;
while(number >= 0)
{
    Console.WriteLine("Hello again!");
    Console.WriteLine(number);
    number++;
}
```

The loop condition checks to see whether `number` is ≥ 0, and `number` starts out at the value 10. But the loop body increments `number`, which only moves it further away from 0 in the positive direction. In order for this loop to work correctly, we need to *decrement* `number` in the loop body, so that eventually it will be less than 0.

– This loop will execute forever, even though it uses the right variable in the loop body, because it multiplies the variable by 0:

```
int number = 0;
while (number <= 64)
{
    Console.WriteLine(number);
    number *= 2;
}
```

Since `number` was initialized to 0, `number *= 2` does not actually change the value of `number`: $2 \times 0 = 0$. So the loop body will never make the condition `number <= 64` false.

Principles of writing a `while` loop

- When writing a **`while`** loop, ask yourself these questions about your program:
  1. When (under what conditions) do I want the loop to continue?
  2. When (under what conditions) do I want the loop to stop?
  3. How will the body of the loop bring it closer to its ending condition?
- This will help you think clearly about how to write your loop condition. You should write a condition (Boolean expression) that will be **`true`** in the circumstances described by (1), and **`false`** in the circumstances described by (2)
- Keep your answer to (3) in mind as you write the body of the loop, and make sure the actions in your loop's body match the condition you wrote.

While Loop With Complex Conditions

In the following example, a complex boolean expression is used in the *while* statement. The program gets a value and tries to parse it as an integer. If the value can not be converted to an integer, the program tries again, but not more than three times.

```
int c;
string message;
int count;
bool res;

Console.WriteLine("Please enter an integer.");
message = Console.ReadLine();
res = int.TryParse(message, out c);
count = 0; // The user has 3 tries: count will be 0, 1, 2,
 ↪  and then we default.
while (!res && count < 3)
{
    count++;
    if (count == 3)
    {
        c = 1;
        Console.WriteLine("I'm using the default value
 ↪  1.");
    }
    else
    {
        Console.WriteLine("The value entered was not an
 ↪  integer.");
        Console.WriteLine("Please enter an integer.");
        message = Console.ReadLine();
        res = int.TryParse(message, out c);
    }
}
Console.WriteLine("The value is: " + c);
```