C# Iterations

C# provides the following four iteration constructs:

- for loop
- foreach/in loop
- while loop
- do/while loop

for loop

When you need to iterate over a block of code a fixed number of times, the for statement provides a good deal of flexibility. In essence, you are able to specify how many times a block of code repeats itself, as well as the terminating condition.

```
for(int i = 0; i < 4; i++)
{
	Console.WriteLine($"Number is: {i}");
}
```

foreach loop

The C# foreach keyword allows you to iterate over all items in a container without the need to test for an upper limit. So, when you simply need to walk a collection item by item, the foreach loop is the perfect choice.

```
string[] carTypes = {"Ford", "BMW", "Yugo", "Honda" };
foreach (string c in carTypes)
{
	Console.WriteLine(c);
}
```

The item after the in keyword can be a simple array (seen here) or, more specifically, any class implementing the IEnumerable interface.

It is also possible to use implicit typing within a foreach looping construct. As you would expect, the compiler will correctly infer the correct “type of type.”

while loop

The while looping construct is useful should you want to execute a block of statements until some terminating condition has been reached. Within the scope of a while loop, you will need to ensure this terminating event is indeed established; otherwise, you will be stuck in an endless loop. In the following example, the message "In while loop" will be continuously printed until the user terminates the loop by entering yes at the command prompt:

```
string userIsDone = "";
while(userIsDone.ToLower() != "yes")
{
	Console.WriteLine("In while loop");
	Console.Write("Are you done? [yes] [no]: ");
	userIsDone = Console.ReadLine();
}
```

do/while loop

Closely related to the while loop is the do/while statement. Like a simple while loop, do/while is used when you need to perform some action an undetermined number of times. The difference is that do/while loops are guaranteed to execute the corresponding block of code at least once. In contrast, it is possible that a simple while loop may never execute if the terminating condition is false from the onset.

```
string userIsDone = "";
 do
 {
         Console.WriteLine("In do/while loop");
         Console.Write("Are you done? [yes] [no]: ");
         userIsDone = Console.ReadLine();
} while(userIsDone.ToLower() != "yes");     // Note the semicolon!
```

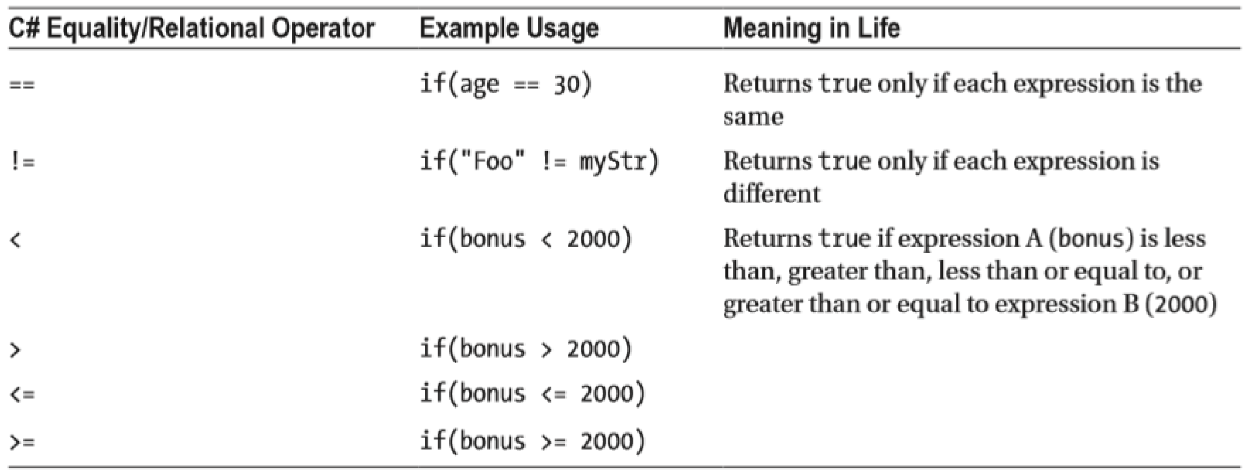
if/else Statement

Unlike in C and C++, ***the if/else statement in C# operates only on Boolean expressions***, not ad hoc values such as –1 or 0.

```
int number = 100;
if (number == 100)
{
    number /= 5;
}
else
{
    Console.WriteLine(number);
}
```

C# if/else statements typically involve the use of the C# operators shown below to obtain a literal Boolean value.

Equality and Relational Operators



C# Equality/Relational Operator	Example Usage	Meaning in Life
`==`	`if(age == 30)`	Returns `true` only if each expression is the same
`!=`	`if("Foo" != myStr)`	Returns `true` only if each expression is different
`<`	`if(bonus < 2000)`	Returns `true` if expression A (bonus) is less than, greater than, less than or equal to, or greater than or equal to expression B (2000)
`>`	`if(bonus > 2000)`	
`<=`	`if(bonus <= 2000)`	
`>=`	`if(bonus >= 2000)`	

Conditional Operator

The conditional operator (?:) is a shorthand method of writing a simple if-else statement. The syntax works like this:

condition ? first_expression : second_expression;

The condition is the conditional test (the if part of the if-else statement). If the test passes, then the code immediately after the question mark (?) is executed. If the test does not evaluate to true, the code after the colon (the else part of the if-else statement) is executed.

There are some *restrictions to the conditional operator*.

- both types of first_expression and second_expression must be the same.
- the conditional operator can be used only in assignment statements (Only assignment, call, increment, decrement, and new object expressions can be used as a statement).

Logical Operators

Operator	Example	Meaning in Life
&&	`if(age == 30 && name == "Fred")`	AND operator. Returns `true` if all expressions are true.
||	`if(age == 30 || name == "Fred")`	OR operator. Returns `true` if at least one expression is true.
!	`if(!myBool)`	NOT operator. Returns `true` if false, or `false` if true.

switch Statement

As in other C-based languages, the switch statement allows you to handle program flow based on a predefined set of choices. In fact, all version of C# can evaluate char, string, bool, int, long, and enum data types.

Falling through from one case statement to another case statement is not allowed, but what if multiple case statements should produce the same result? Fortunately, they can be combined

```
switch (n)
{
        case "1":
                // do something
                break;
        case "2":
        case "3":
                //do something else
                break;
        case "4":
                goto case "1";
        default:
                // default action
                break;
}
```

If any code was included between the case statements, the compiler would throw an error.