CHAPTER 7

Conditionals

Conditional statements are used to execute different code blocks based on different conditions.

If Statement

The `if` statement will execute only if the condition inside the parentheses is evaluated to `true`. The condition can include any of the comparison and logical operators.

```
// Get a random integer (0, 1 or 2)
int x = new System.Random().Next(3);

if (x < 1) {
  System.Console.Write(x + " < 1");
}
```

To test for other conditions, the `if` statement can be extended by any number of `else if` clauses. Each additional condition will be tested only if all previous conditions are false.

```
else if (x > 1) {
  System.Console.Write(x + " > 1");
}
```

The if statement can have one else clause at the end, which will execute if all previous conditions are false.

```
else {
  System.Console.Write(x + " == 1");
}
```

As for the curly brackets, they can be left out if only a single statement needs to be executed conditionally. However, it is considered good practice to include them since they improve readability.

```
if (x < 1)
  System.Console.Write(x + " < 1");
else if (x > 1)
  System.Console.Write(x + " > 1");
else
  System.Console.Write(x + " == 1");
```

Switch Statement

The switch statement checks for equality between a value and a series of case labels and then passes execution to the matching case. The statement can contain any number of case clauses and may end with a default label for handling all other cases.

```
int x = new System.Random().Next(4);
switch (x)
{
  case 0: System.Console.Write(x + " is 0"); break;
  case 1: System.Console.Write(x + " is 1"); break;
  default:System.Console.Write(x + " is >1"); break;
}
```

Note that the statements after each case label are not surrounded by curly brackets. Instead, the statements end with the break keyword to break out of the switch. Case clauses in C# must end with a jump statement, such as break, because unintentional fall-throughs are a common programming error. An exception to this is if the case clause is completely empty, in which case execution is allowed to fall through to the next label.

```
switch (x)
{
  case 0:
  case 1: System.Console.Write("x is 0 or 1"); break;
}
```

Goto Statement

To cause a fall-through to occur for a non-empty case clause, this behavior has to be explicitly specified using the goto jump statement followed by a case label. This will cause the execution to jump to that label.

```
case 0: goto case 1;
```

Goto may also be used outside of switches to jump to a label in the same method's scope. Control may then be transferred out of a nested scope, but not into a nested scope. However, using goto in this manner is discouraged since it makes it more difficult to follow the flow of execution.

```
myLabel:
// ...
goto myLabel; // jump to label
```

Switch Expression

C# 8.0 introduced the switch expression which is more concise than the regular switch statement. It can be used when each case is an assignment expression instead of a statement, as seen in the following example.

```
int x = new System.Random().Next(4);
string result = x switch {
  0 => "zero",
  1 => "one",
  _ => "more than one"
};
System.Console.WriteLine("x is " + result);
```

The switch expression returns the expression to the right of the arrow (=>) if the expression tested matches the pattern to the left of the arrow. Note that there are no `case` or `break` keywords in the switch expression and that the default case is represented with an underscore (_).

Ternary Operator

In addition to the `if` and `switch` statements, there is the ternary operator (`?:`). This operator takes three expressions. If the first one is evaluated to `true`, then the second expression is returned, and if it is `false`, the third one is returned.

```
// Get a number between 0.0 and 1.0
double x = new System.Random().NextDouble();
x = (x < 0.5) ? 0 : 1; // ternary operator (?:)
```