Contents

The Conditional Operator

- There are many situations where we need to assign a variable to a different value depending on the result of a condition
- For example, the `if-else-if` and `switch` statements in the previous section were used to decide which value to assign to the variable `monthName`
- A simpler example: Imagine your program needs to tell the user whether a number is even or odd. You need to initialize a `string` variable to either "Even" or "Odd" depending on whether `myInt % 2` is equal to 0. We could write an `if` statement to do this:

```
string output;
if(myInt % 2 == 0)
{
    output = "Even";
}
else
{
    output = "Odd";
}
```

Assignment with the conditional operator

- If the only thing an `if` statement does is assign a value to a variable, there is a much shorter way to write it
- The **conditional operator** `?:` tests a condition, and then outputs one of two values based on the result
- Continuing the "even or odd" example, the conditional operator is used like this:

```
string output = (myInt % 2 == 0) ? "Even" : "Odd";
```

When this line of code is executed:

- The condition `(myInt % 2 == 0)` is evaluated, and the result is either true or false

- If the condition is true, the conditional operator returns (outputs) the value `"Even"` (the left side of the `:`)
- If the condition is false, the operator returns the value `"Odd"` (the right side of the `:`)
- This value, either "Even" or "Odd", is used in the initialization statement for `string output`
- Thus, `output` gets assigned the value `"Even"` if `(myInt % 2 == 0)` is true, or `"Odd"` if `(myInt % 2 == 0)` is false
- In general, the syntax for the conditional operator is:

```
condition ? true_expression : false_expression;
```

- The "condition" can be any expression that produces a `bool` when evaluated, just like in an `if` statement
- `true_expression` and `false_expression` can be variables, values, or more complex expressions, but they must both produce the same *type* of data when evaluated
- For example, if `true_expression` is `myInt * 1.5`, then `false_expression` must also produce a `double`
- When the conditional operator is evaluated, it returns either the value of `true_expression` or the value of `false_expression` (depending on the condition) and this value can then be used in other operations such as assignment

Conditional operator examples

- The `true_expression` and `false_expression` can both be mathematical expressions, and only one of them will get computed. For example:

```
int answer = (myInt % 2 == 0) ? myInt / 2 : myInt + 1;
```

If `myInt` is even, the computer will evaluate `myInt / 2` and assign the result to `answer`. If it is odd, the computer will evaluate `myInt + 1` and assign the result to `answer`.

- Conditional operators can be used with user input to quickly provide a "default value" if the user's input is invalid. For example, we can write a program that asks the user their height, but uses a default value of 0 if the user enters a negative height:

```
Console.WriteLine("What is your height in meters?");
double userHeight = double.Parse(Console.ReadLine());
double height = (userHeight >= 0.0) ? userHeight : 0.0;
```

- The condition can be a Boolean variable by itself, just like in an if statement. This allows you to write code that looks kind of like English, due to the question mark in the conditional operator. For example,

```
bool isAdult = age >= 18;
decimal price = isAdult ? 5.0m : 2.5m;
string closingTime = isAdult ? "10:00 pm" : "8:00 pm";
```