

CHAPTER 15

Getting Started with Conditions

Up to now, a program's statements have always been executed from the beginning to the end regardless of anything else, simply when their turn came. In this chapter, the whole new world will start to unveil itself because you will learn about the *conditional execution* of program statements. This means you will work with statements that may or may not execute depending on whether some *condition* is fulfilled.

Password Input

Your first program with conditions will evaluate a password. The user may or may not be allowed to enter the system depending on whether they have entered the correct password.

Task

You will write a program that prompts the user to enter a password and then evaluates whether the entered password is correct. For the sake of simplicity, the correct password will be specified directly in the code (see Figure 15-1 and Figure 15-2).



Figure 15-1. Incorrect password



Figure 15-2. Correct password

Analysis

Let's look at this program in more detail.

The Program

In this program, some activity is performed when both passwords (the entered one and the stored one) agree, and a different activity is performed when they disagree. In this case, you either allow or refuse the user with an appropriate message (see Figure 15-3).

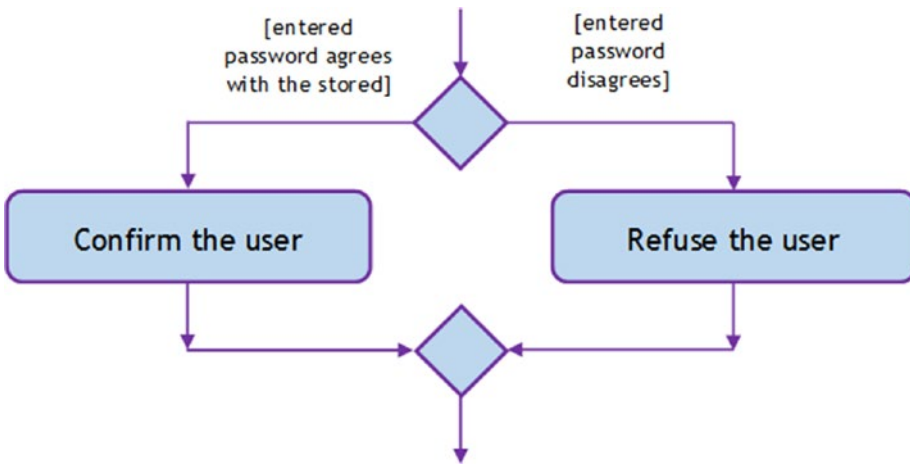


Figure 15-3. The program flow

Program Branching

Generally, *program branching* means taking different paths depending on the fulfillment of a condition (see Figure 15-4).

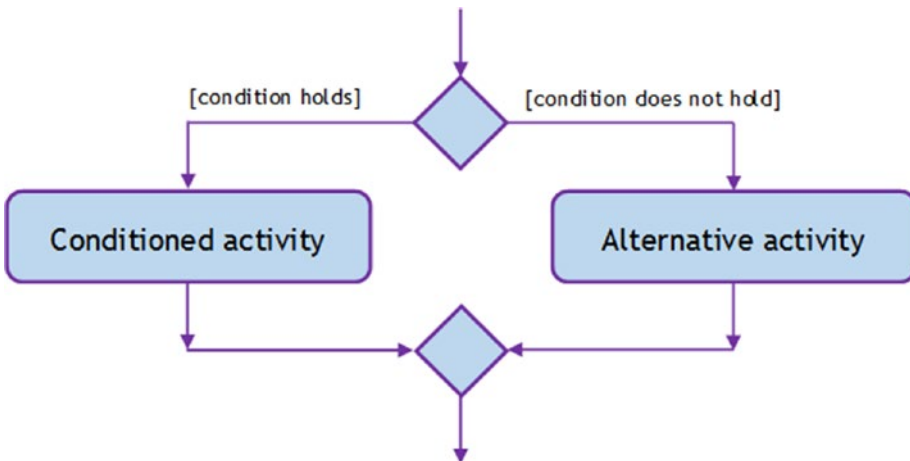


Figure 15-4. Branching

Syntax

For branching, C# uses the if-else construction shown here:

```
if (condition)
{
    Statements to perform when the condition holds
}
else
{
    Statements to perform otherwise
}
```

Solution

Here is the code:

```
static void Main(string[] args)
{
    // Input
    Console.Write("Enter password: ");
    string enteredPassword = Console.ReadLine();

    // Password check
    if (enteredPassword == "friend")
    {
        Console.WriteLine("Password is OK, please enter");
    }
    else
    {
        Console.WriteLine("Incorrect password");
    }

    // Waiting for Enter
    Console.ReadLine();
}
```

Discussion

To formulate the condition, I have used an equality test, which is entered using a couple of equal signs. If the compared values are the same, the test evaluates to `true`, the condition is considered fulfilled, and the statements in the `if` branch are executed. If the compared values are different, the test evaluates to `false`, the condition is considered not fulfilled, and the statements in the `else` branch are executed.

Test

Now you can check how the program executes! Besides doing an ordinary program run, you can also step through the code, as you learned in the previous chapter.

Reversed Condition

So that you get more familiar with conditions, it is useful to see them from different perspectives. Staying with the password issue, let's view it in another way.

Task

The task now is to solve the previous exercise alternatively, namely, with the condition reversed. In other words, you will test for inequality instead of equality.

Solution

Here is the code:

```
static void Main(string[] args)
{
    // Correct password
    string correctPassword = "friend";

    // Input
    Console.Write("Enter password: ");
    string enteredPassword = Console.ReadLine();
```

```
// Password check
if (enteredPassword != correctPassword)
{
    Console.WriteLine("Incorrect password");
}
else
{
    Console.WriteLine("Password is OK, please enter");
}

// Waiting for Enter
Console.ReadLine();
}
```

Discussion

In this exercise, I have used an inequality test, which is typed using an exclamation mark followed by an equal sign. The test returns true when the compared values **disagree**.

Length Check

While two pieces of text can only be compared to find out if they are the same or different, two numbers can also be compared to figure out which one is longer (or shorter). Let's take a look.

Task

In this section, you will study number comparisons in a program that evaluates whether the entered text is at most four characters long (see Figures 15-5 and 15-6).

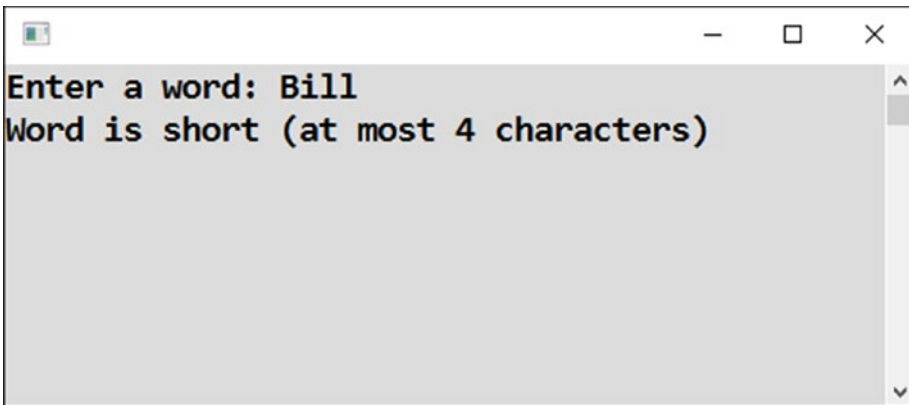


Figure 15-5. Short text

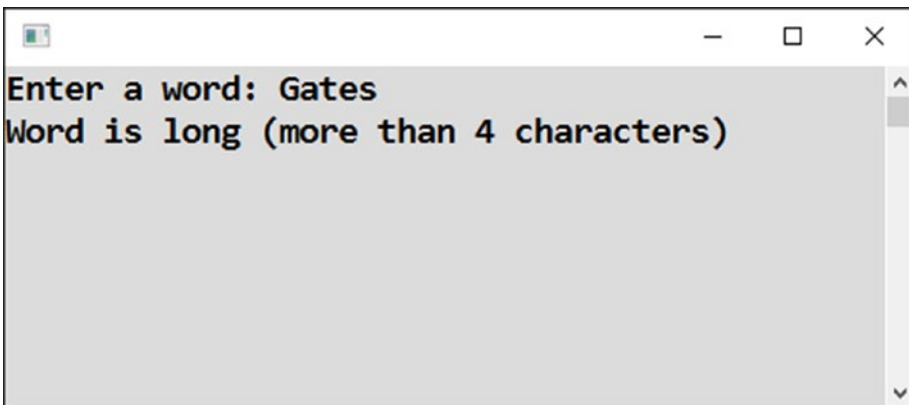


Figure 15-6. Long text

Solution

Presumably, you should determine the number of characters of the entered text and compare it to the number 4. You learned how to determine the number of characters of text—using its `Length` property—in Chapter 7 (the program was “Texts as objects”). Anyway, if you do not remember the name of the property, you can add a dot to the end of a text variable and browse through the IntelliSense possibilities to see what might be appropriate, as covered in the previous chapter.

Here is the code:

```
static void Main(string[] args)
{
    // Input
    Console.Write("Enter a word: ");
    string word = Console.ReadLine();

    // Determining length
    int wordLength = word.Length;

    // Checking length
    if (wordLength <= 4)
    {
        Console.WriteLine("Word is short (at most 4 characters)");
    }
    else
    {
        Console.WriteLine("Word is long (more than 4 characters)");
    }

    // Waiting for Enter
    Console.ReadLine();
}
```

Note

I have used a less-than-or-equal-to operator in this solution, which looks like this: <=.

Positive Numbers

In this section, you will get some more practice with number comparisons.

Task

You will write a program that evaluates whether the number entered by the user is positive or not (see Figures [15-7](#) and [15-8](#)).

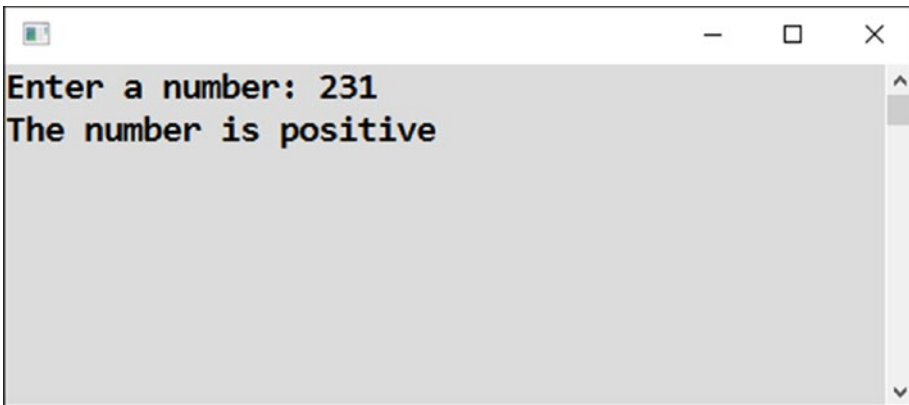


Figure 15-7. It's positive.

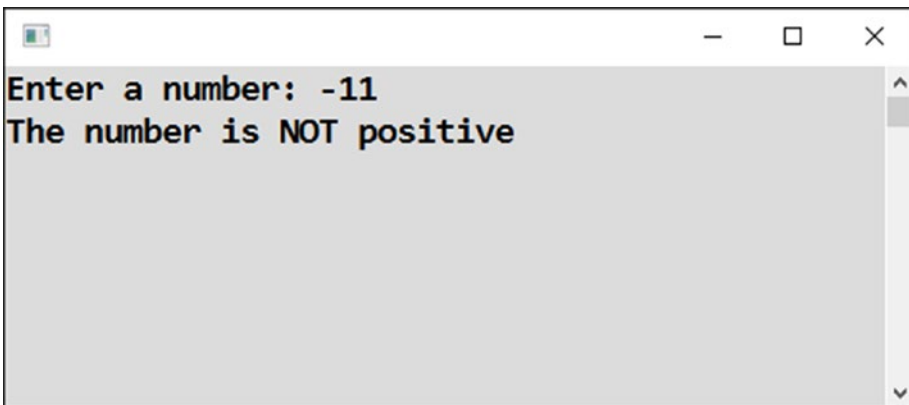


Figure 15-8. It's not positive.

Solution

Here is the code:

```
static void Main(string[] args)
{
    // Input
    Console.Write("Enter a number: ");
    string input = Console.ReadLine();
    int number = Convert.ToInt32(input);

    // Evaluation
    if (number > 0)
```

```
{
    Console.WriteLine("The number is positive");
}
else
{
    Console.WriteLine("The number is NOT positive");
}

// Waiting for Enter
Console.ReadLine();
}
```

Discussion

I have used a greater-than operator to compare the entered number to zero.

What do you think the program does when the user enters zero? It checks the condition $0 > 0$ and finds it is not fulfilled. Therefore, it displays that the number is not positive. This is the reason for the rather unusual message wording (“... NOT positive”), as shown in Figure 15-9. I have not used “...is negative”.

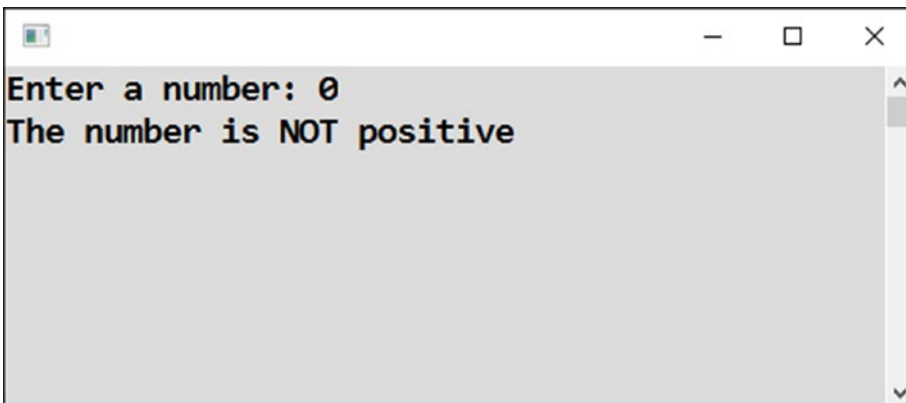


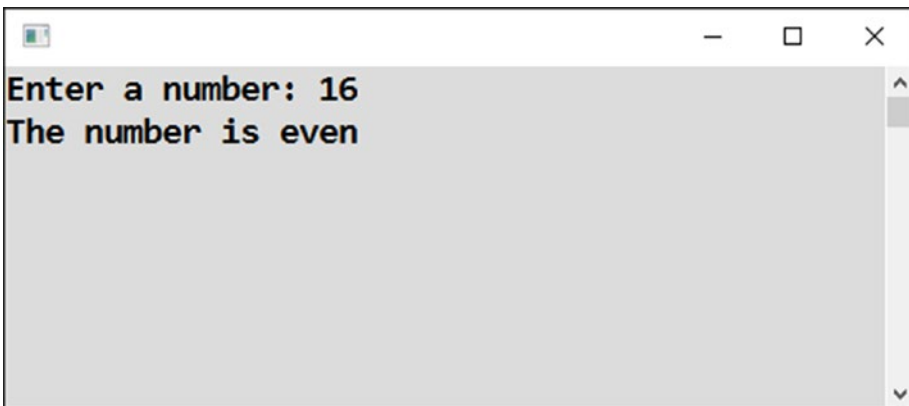
Figure 15-9. Results for zero

Odd and Even Numbers

Let's proceed to another number comparison.

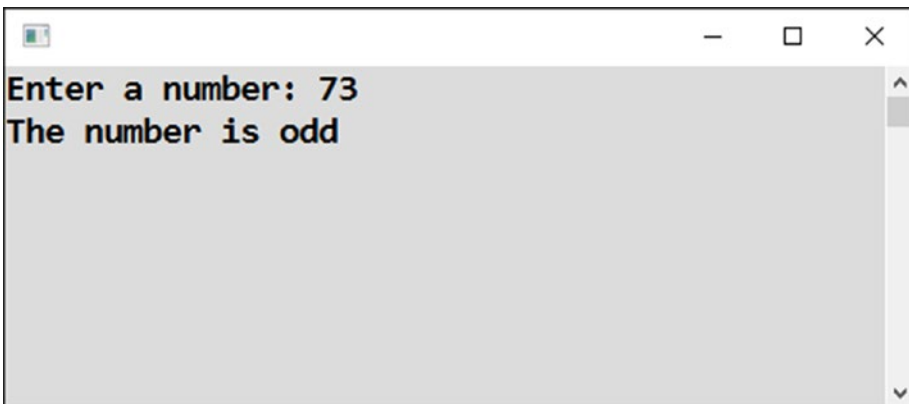
Task

Your task now is to write a program that evaluates whether the number entered by the user is odd or even (see Figure 15-10 and Figure 15-11).



```
Enter a number: 16
The number is even
```

Figure 15-10. Determining even



```
Enter a number: 73
The number is odd
```

Figure 15-11. Determining odd

Solution

The core of the solution is to determine the remainder of dividing the entered number by 2. If the remainder is zero, the number is even. If there is some remainder, the number is odd.

Here is the code:

```
static void Main(string[] args)
{
    // Input
    Console.Write("Enter a number: ");
    string input = Console.ReadLine();
    int number = Convert.ToInt32(input);

    // Remainder calculation
    int remainderAfterDivisionByTwo = number % 2;

    // Evaluation
    if (remainderAfterDivisionByTwo == 0)
    {
        Console.WriteLine("The number is even");
    }
    else
    {
        Console.WriteLine("The number is odd");
    }

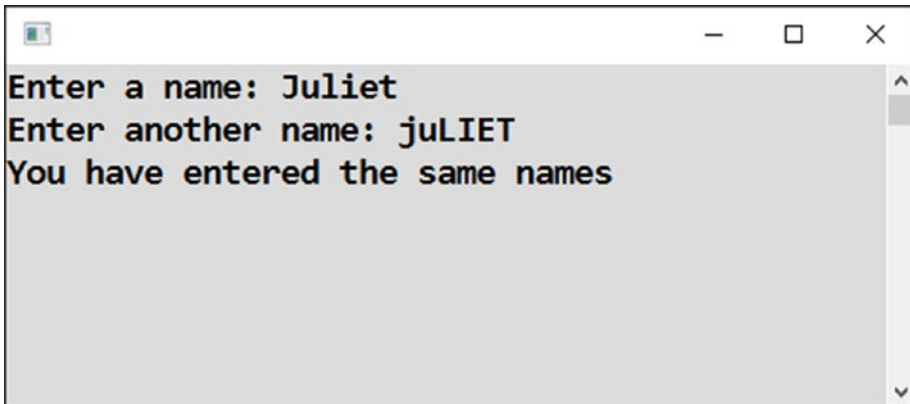
    // Waiting for Enter
    Console.ReadLine();
}
```

Case Indifference

You already know that two pieces of text can be compared to see if they are equal or unequal. This comparison is case-sensitive. In other words, *hobbit* and *Hobbit* are considered different words. Frequently, however, you need case-insensitive comparisons, which I will show you now.

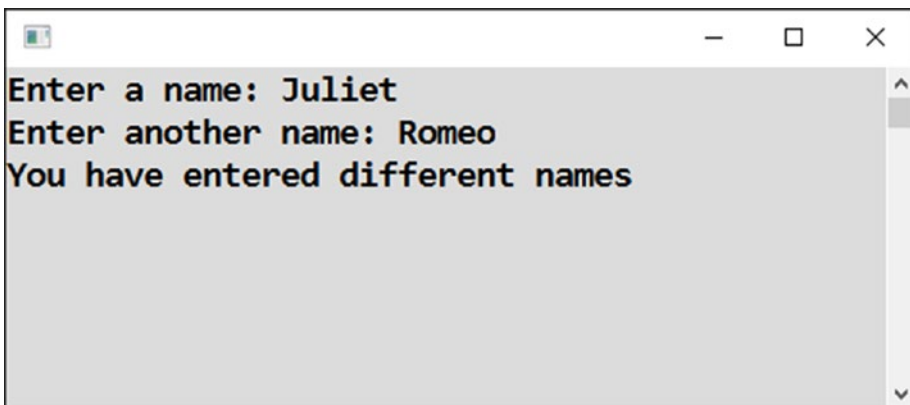
Task

In this program, the user will enter two names, and you will evaluate whether they are the same or different, disregarding the difference between lowercase and uppercase (see Figures 15-12 and 15-13).



```
Enter a name: Juliet
Enter another name: juLIET
You have entered the same names
```

Figure 15-12. The same names



```
Enter a name: Juliet
Enter another name: Romeo
You have entered different names
```

Figure 15-13. Different names

Solution

The core of the solution is to convert both pieces of text to lowercase prior to doing the comparison. You can use the `ToLower` method call for that purpose.

Here is the code:

```
static void Main(string[] args)
{
    // Inputs
    Console.Write("Enter a name: ");
    string name1 = Console.ReadLine();

    Console.Write("Enter another name: ");
    string name2 = Console.ReadLine();

    // Converting to small letters
    string name1inSmall = name1.ToLower();
    string name2inSmall = name2.ToLower();

    // Evaluating
    if (name1inSmall == name2inSmall)
    {
        Console.WriteLine("You have entered the same names");
    }
    else
    {
        Console.WriteLine("You have entered different names");
    }

    // Waiting for Enter
    Console.ReadLine();
}
```

Without Braces

C# allows you to omit the braces surrounding the `if` and `else` branches if the branch contains just a single statement. Generally, I do not recommend this practice because it can be misleading. I will show this to you now just so that you are aware of it.

Task

You will solve the previous exercise again, this time without braces.

Solution

Here is the code:

```
static void Main(string[] args)
{
    // Inputs
    Console.Write("Enter a name: ");
    string name1 = Console.ReadLine();

    Console.Write("Enter another name: ");
    string name2 = Console.ReadLine();

    // Converting to small letters
    string name1inSmall = name1.ToLower();
    string name2inSmall = name2.ToLower();

    // Evaluating
    // BRANCHES NOT DELIMITED BY BRACES (CURLY BRACKETS)
    if (name1inSmall == name2inSmall)
        Console.WriteLine("You have entered the same names");
    else
        Console.WriteLine("You have entered different names");

    // Waiting for Enter
    Console.ReadLine();
}
```

Greater of Two Numbers

A frequent task of a programmer is to find which of two numbers is greater (or smaller, analogously).

Task

Your task now is to write a program that asks the user for two numbers and then says which of the two numbers is greater (see Figure 15-14).

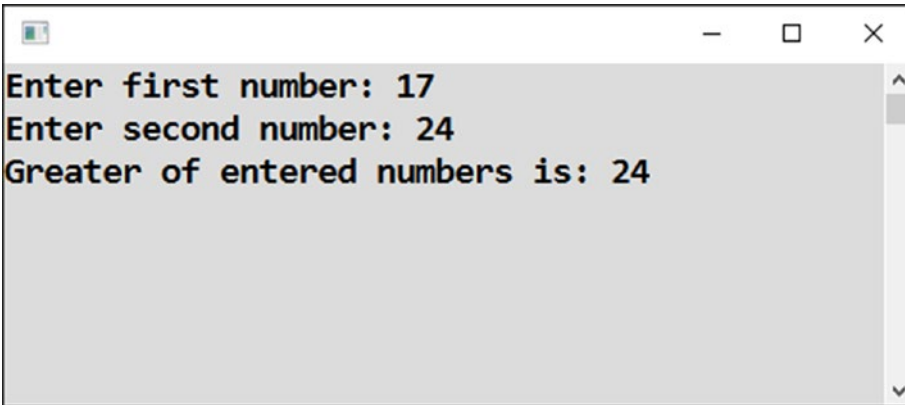


Figure 15-14. Determining which number is greater

Solution

Here is the code:

```
static void Main(string[] args)
{
    // Inputs
    Console.Write("Enter first number: ");
    string input1 = Console.ReadLine();
    int number1 = Convert.ToInt32(input1);

    Console.Write("Enter second number: ");
    string input2 = Console.ReadLine();
    int number2 = Convert.ToInt32(input2);

    // Evaluating
    int greater;
    if (number1 > number2)
    {
        greater = number1;
    }
}
```



```

else
{
    greater = number2;
}

// Output
Console.WriteLine("Greater of entered numbers is: " + greater);

// Waiting for Enter
Console.ReadLine();
}

```

Without the else Branch

In previous exercises, you always had two branches—the `if` branch and the `else` branch. In other words, you were always in an either-or situation. It is important to note, however, that the `else` branch can be omitted if you want. This means if a condition is fulfilled, you do something, and if it is not fulfilled, you simply do nothing. Take a look!

Task

In the previous exercise, you set the `greater` variable either to the first value or to the second value.

Now you will solve the same task in a different way. First you will set the `greater` variable directly to the first value, and then if the second one is greater, you will change the final result.

Solution

Here is the code:

```

static void Main(string[] args)
{
    // Inputs
    Console.Write("Enter first number: ");
    string input1 = Console.ReadLine();

```

```
int number1 = Convert.ToInt32(input1);

Console.Write("Enter second number: ");
string input2 = Console.ReadLine();
int number2 = Convert.ToInt32(input2);

// Evaluating
int greater = number1;
if (number2 > greater)
{
    greater = number2;
}

// Output
Console.WriteLine("Greater of entered numbers is: " + greater);

// Waiting for Enter
Console.ReadLine();
}
```

Using a Built-in Function

Frequently in this book, I show you things from different angles. I strongly believe this promotes your understanding. For the current problem of finding the greater value of two, I will show you a third way to solve it. The task is so frequent, in fact, that there is a convenient built-in function for it.

Task

You will solve the previous exercise using the built-in function `Math.Max`.

Solution

Here is the code:

```
static void Main(string[] args)
{
    // Inputs
    Console.Write("Enter first number: ");
    string input1 = Console.ReadLine();
    int number1 = Convert.ToInt32(input1);

    Console.Write("Enter second number: ");
    string input2 = Console.ReadLine();
    int number2 = Convert.ToInt32(input2);

    // Evaluating
    int greater = Math.Max(number1, number2);

    // Output
    Console.WriteLine("Greater of entered numbers is: " + greater);

    // Waiting for Enter
    Console.ReadLine();
}
```

Summary

In this chapter, you started studying the conditional execution of program statements, which means that the execution or nonexecution of one or more statements can be conditioned by some test. You saw the following examples of tests:

- Testing the equality of two pieces of text or two numbers with the `==` operator
- Testing the inequality of two pieces of text or two numbers with the `!=` operator
- Testing whether a number is greater (or less) than another number with the `>` (or `<`) operator

The last test can be extended to “greater than or equal to” (or “less than or equal to”) with the `>=` (or `<=`) operator.

To use conditional execution in your code, you learned about the `if-else` construct. This consists of a condition and two branches. If the condition is evaluated to be true (fulfilled), the statements in the `if` branch are executed. If the condition is evaluated to be false (not fulfilled), the statements in the `else` branch are executed.

You learned that if a branch consists of a single statement, C# syntax allows you to omit the braces surrounding the branch, though I discourage you from doing that because people frequently forget to include the braces later when they extend a branch to several statements.

More important, you learned that the `else` branch can be omitted if you want. This means there is no alternative action—nothing is done when the condition is not fulfilled.

As a bonus, you learned about the useful built-in function `Math.Max`. (You can probably guess that there is a similar function called `Math.Min`.)