

CHAPTER 16

Practical Conditions

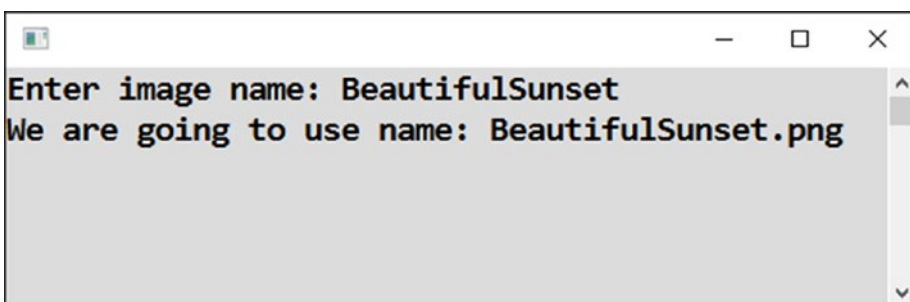
In the previous chapter, you learned about the conditional execution of a program's statements. In this chapter, you will deepen your knowledge of this topic. I will show you how to use conditions and branching on several simple tasks that you will encounter sooner or later in your programming career.

Appending Extension

Sometimes you want to ask the user about a file name, but you do not know whether the user will enter it with or without an extension.

Task

You will write a program that appends the `.png` extension to the entered file name unless the extension is already part of the input (see Figures 16-1 and 16-2).



```
Enter image name: BeautifulSunset
We are going to use name: BeautifulSunset.png
```

Figure 16-1. Appending the `.png` extension

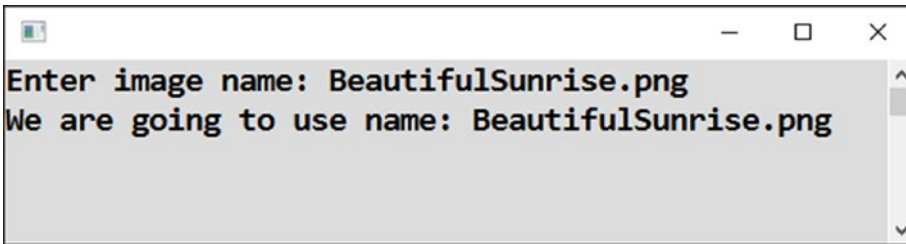


Figure 16-2. Not appending the .png extension

Solution

Here is the code:

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

    // Appending extension (ONLY IN CASE OF NEED)
    if (!fileName.ToLower().EndsWith(".png"))
    {
        fileName += ".png";
    }

    // Output
    Console.WriteLine("We are going to use name: " + fileName);

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

Discussion

Let's discuss this program a bit.

Extension Detection

The most interesting point of the current exercise is finding out whether the entered file name ends with a particular extension.

- First, you convert the file name to lowercase so you do not have to distinguish between `.png` and `.PNG`.
- You use the method `EndsWith` to find whether the text ends or does not end with something specific. In this case, the method call returns `true` if the text ends with `.png`. Otherwise, it returns `false`.
- You negate the result returned by the `EndsWith` method using the `!` operator. The exclamation mark changes `true` to `false`, and vice versa. This means you actually ask “Does the text *not* end with `.png`?” instead of “Does it end with `.png`?”

Entering a Condition

Note that you do not always have to enter a comparison when specifying a condition. You do not always have to use “less than,” for example. It is enough if the condition evaluates to a Boolean value, such as `true` or `false`.

If the condition evaluates to `true`, it is considered fulfilled, and the statements in the `if` branch are executed.

If the condition evaluates to `false`, it is considered not fulfilled, and the statements in the `else` branch are executed (or nothing is executed in the case of a missing `else` branch).

Missing else Branch

The example program has a missing `else` branch. If the entered name ends with `.png`, the `EndsWith` method will find it and will return `true`. If you get `false`, the condition is considered not fulfilled, so the statement appending the extension will not be executed, and the entered name will remain unchanged.

Chaining

Note the *chaining* of the `ToLower` and `EndsWith` methods. The output of the lowercase conversion is not stored in any variable. Instead, it serves as input for the next method in the chain, in other words, `EndsWith`.

Head and Tail

Let's do some more exercises concerning conditions.

Task

You will write a program that throws a coin once (see Figure 16-3).



Figure 16-3. Throwing a coin

Solution

The core of the solution is to generate a random number—zero or one—and convert it to heads or tails subsequently.

Here is the code:

```
static void Main(string[] args)
{
    // Random number generator
    Random randomNumbers = new Random();

    // Random number 0/1 and its transformation
    int randomNumber = randomNumbers.Next(0, 1 + 1);
    if (randomNumber == 0)
    {
        Console.WriteLine("Head tossed");
    }
    else
    {
```

```

        Console.WriteLine("Tail tossed");
    }

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

```

Discussion

I just want to remind you that the `Next` method requires the upper bound of a random number range to be specified already augmented by 1. That is why you wrote `1+1` in the previous program. Of course, you could also have written `2` directly, but `1+1` seems to me more logical, stating `1` as the upper bound and adding the (strangely) required `1`.

Deadline Check

“Never trust the user,” as the old saying goes. This means you as a programmer always have to check user-entered data in production software.

You need to check the user data usually not because of malicious use because 99.9 percent of your users do not have any intention to abuse your software. Users simply make mistakes. That is why you should check their input and prompt them to correct it.

So, now you will learn how to implement some input checking.

Task

You will write a program that prompts the user to enter an order deadline and presents a warning if the user enters a date in the past (see Figure 16-4).

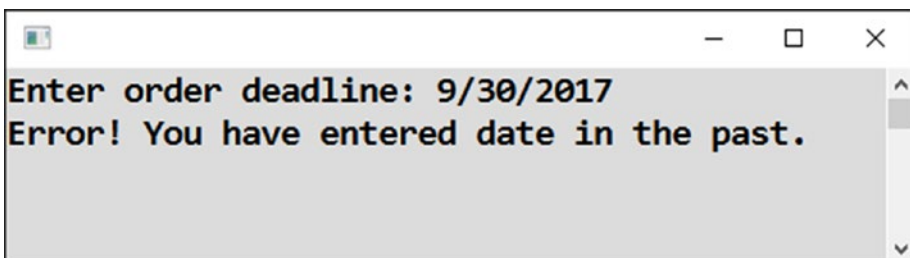


Figure 16-4. Checking a date

Solution

Here is the code:

```
static void Main(string[] args)
{
    // Input
    Console.Write("Enter order deadline: ");
    string input = Console.ReadLine();
    DateTime enteredDeadline = Convert.ToDateTime(input);

    // Checking entered value
    DateTime today = DateTime.Today;
    if (enteredDeadline < today)
    {
        Console.WriteLine("Error! You have entered date in the past.");
    }
    else
    {
        Console.WriteLine("Deadline accepted.");
    }

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

Discussion

Note the following:

- To convert a date entered in text form into the `DateTime` object, you use the `Convert.ToDateTime` method call.
- Conversion fails if a nonexistent date is entered. You can handle this using try-catch.
- Similar to number conversions, `Convert.ToDateTime` can accept a second parameter specifying the language to be used for the conversion.

Invoice Date Check

Let's do one more exercise for checking user-entered data.

Task

Value-added tax (VAT) regulations in my country require that the date an invoice is issued cannot precede the date of payment, and at the same time, it cannot be later than 15 days after the payment.

The current task is to perform both checks (see Figure 16-5, Figure 16-6, and Figure 16-7).

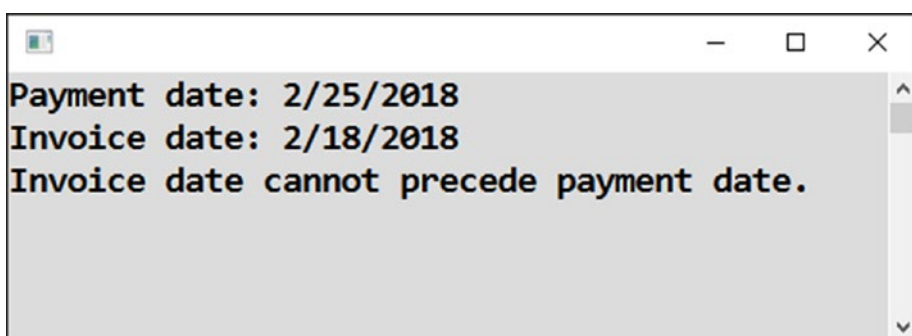


Figure 16-5. Date too early

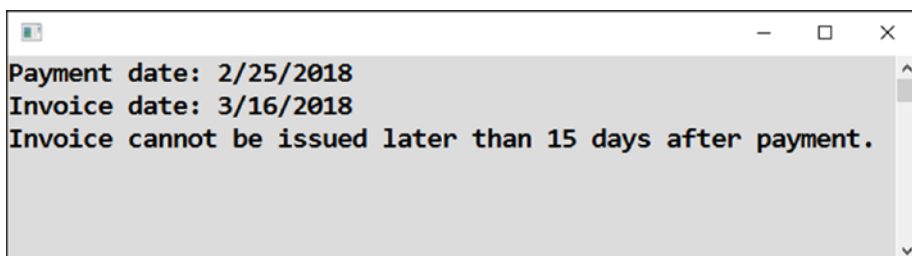


Figure 16-6. Date too late

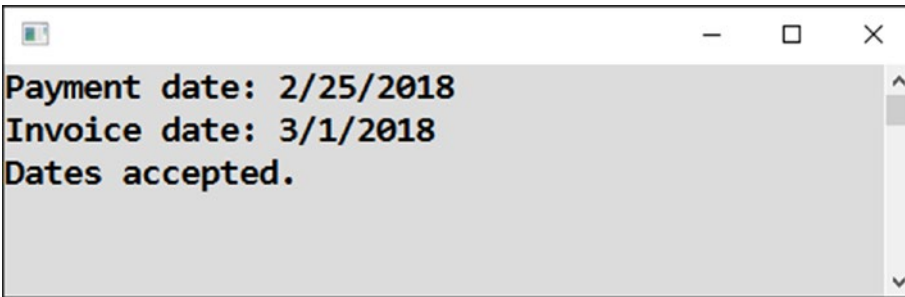


Figure 16-7. Dates accepted

Solution

Here is the solution:

```

static void Main(string[] args)
{
    // Inputs
    Console.Write("Payment date: ");
    string inputPayment = Console.ReadLine();
    DateTime paymentDate = Convert.ToDateTime(inputPayment);

    Console.Write("Invoice date: ");
    string inputInvoice = Console.ReadLine();
    DateTime invoiceDate = Convert.ToDateTime(inputInvoice);

    // Checking
    bool ok = true;
    if (invoiceDate < paymentDate)
    {
        Console.WriteLine("Invoice date cannot precede payment date.");
        ok = false;
    }
    if (invoiceDate > paymentDate.AddDays(15))
    {
        Console.WriteLine("Invoice cannot be issued later than 15 days
        after payment.");
        ok = false;
    }
}
  
```



```
if (ok)
{
    Console.WriteLine("Dates accepted.");
}

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

Discussion

You are using a helper variable called `ok` in this solution. The variable monitors whether everything is OK. At first, you set it to `true`. If any of the performed checks fail, you toggle the value to `false`. If the variable stays `true` after both checks, you know everything is OK, and a confirming message is displayed to the user.

Spanish Day of Week

Now you will learn how to split the code's execution into multiple branches.

Task

You will write a program that displays the Spanish version of the day of week (lunes, martes, miércoles, and so on) for a date entered by the user (see Figure 16-8).

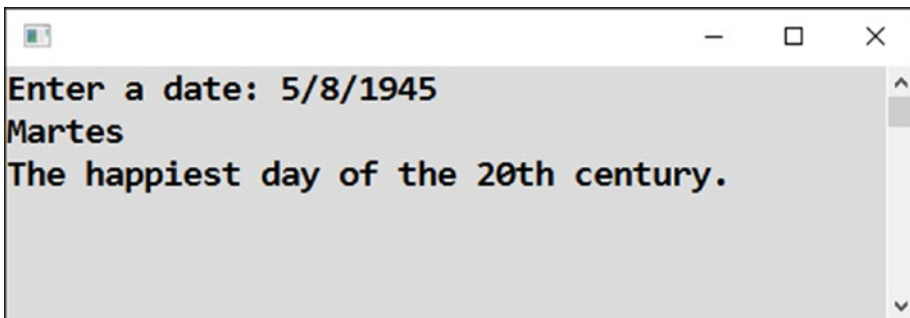


Figure 16-8. *Displaying days in Spanish*

Solution

You can find the day of the week using the `DayOfWeek` property of the `DateTime` object. The conversion to Spanish can be made using a series of conditions.

Here is the code:

```
static void Main(string[] args)
{
    // Input
    Console.Write("Enter a date: ");
    string input = Console.ReadLine();
    DateTime enteredDate = Convert.ToDateTime(input);

    // Finding day of week (in enumeration)
    DayOfWeek dayOfWeek = enteredDate.DayOfWeek;

    // Spanish texts
    string spanish = "";
    if (dayOfWeek == DayOfWeek.Monday)
        spanish = "Lunes";
    if (dayOfWeek == DayOfWeek.Tuesday)
        spanish = "Martes";
    if (dayOfWeek == DayOfWeek.Wednesday)
        spanish = "Miercoles";
    if (dayOfWeek == DayOfWeek.Thursday)
        spanish = "Jueves";
    if (dayOfWeek == DayOfWeek.Friday)
        spanish = "Viernes";
    if (dayOfWeek == DayOfWeek.Saturday)
        spanish = "Sábado";
    if (dayOfWeek == DayOfWeek.Sunday)
        spanish = "Domingo";

    // Output
    Console.WriteLine(spanish);
    if (enteredDate == new DateTime(1945, 5, 8))
        Console.WriteLine("The happiest day of the 20th century.");
}
```

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

Discussion

Note the following:

- You have omitted braces surrounding individual `if` branches. You can do that because there is only a single statement in every branch. I normally do not do this, but in this case of many simple `ifs`, it seemed to me that it would make the code neater.
- Individual days of the week are members of the `DayOfWeek` enumeration. Visual Studio offers you the enumeration as soon as you hit the spacebar on your keyboard after entering two equal signs (see Figure 16-9). Use what Visual Studio offers!

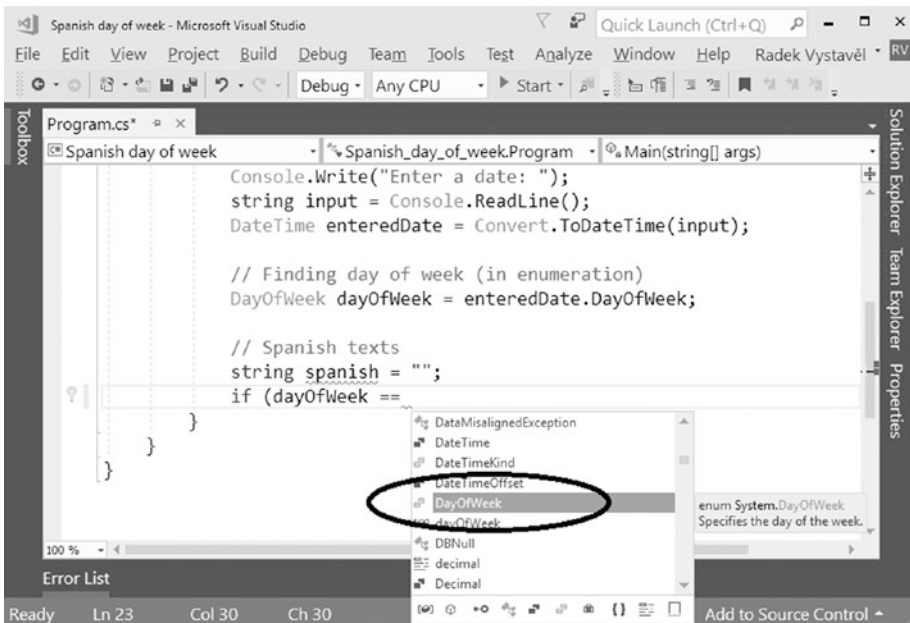


Figure 16-9. Using `DayOfWeek` enumeration

Switch Statement

For certain cases of multiple branching, there also exists a switch statement in C#. Now you will learn how to work with it.

Task

You will solve the last task using a switch statement.

Solution

Here is the code:

```
static void Main(string[] args)
{
    // Input
    Console.Write("Enter a date: ");
    string input = Console.ReadLine();
    DateTime enteredDate = Convert.ToDateTime(input);

    // Finding day of week (in enumeration)
    DayOfWeek dayOfWeek = enteredDate.DayOfWeek;

    // Spanish texts
    string spanish = "";
    switch (dayOfWeek)
    {
        case DayOfWeek.Monday:
            spanish = "Lunes";
            break;
        case DayOfWeek.Tuesday:
            spanish = "Martes";
            break;
        case DayOfWeek.Wednesday:
            spanish = "Miercoles";
            break;
        case DayOfWeek.Thursday:
```

```

        spanish = "Jueves";
        break;
    case DayOfWeek.Friday:
        spanish = "Viernes";
        break;
    case DayOfWeek.Saturday:
        spanish = "Sábado";
        break;
    case DayOfWeek.Sunday:
        spanish = "Domingo";
        break;
}

// Output
Console.WriteLine(spanish);
if (enteredDate == new DateTime(1945, 5, 8))
    Console.WriteLine("The happiest day of the 20th century.");

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

```

Discussion

You can use the `switch` statement as an `if`-series replacement if the repeated branching is always based on the same value. This is the `dayOfWeek` variable's value in this case.

As to the syntax, the `switch` keyword is followed (in parentheses) by a variable (or expression) whose value determines which branch the execution will take. The individual branches start with the `case` keyword followed by a specific value of the control variable and a colon. You should terminate each branch with the `break` keyword.

Summary

In this chapter, you wrote programs with conditional execution for a variety of practical tasks. Specifically, you learned the following:

- To enter conditions without any of relational operators such as `<`, `==`, and so on. The condition simply has to evaluate to the `bool` type. It is considered fulfilled when it evaluates to `true`.
- To negate the condition using the `!` operator.
- To transform random numbers into another kind of data, such as a heads/tails pair.
- To perform various checks of the user input, especially for dates.
- To branch your program into several alternative execution paths, either by using a series of `if` statements or by using `switch` statement.