Contents

Switch

Switch Statements

Multiple equality comparisons

- In some situations, your program will need to test if a variable is equal to one of several values, and perform a different action based on which value the variable matches
- For example, you have an `int` variable named `month` containing a month number, and want to convert it to a `string` with the name of the month. This means your program needs to take a different action depending on whether `month` is equal to 1, 2, 3, ... or 12:
- One way to do this is with a series of `if-else-if` statements, one for each possible value, like this:

```
Console.WriteLine("Enter the month as a number between 1 and 12.");
int month = int.Parse(Console.ReadLine());
string monthName;
if(month == 1)
{
    monthName = "January";
}
else if(month == 2)
{
    monthName = "February";
}
else if(month == 3)
{
    monthName = "March";
}
else if(month == 4)
{
```

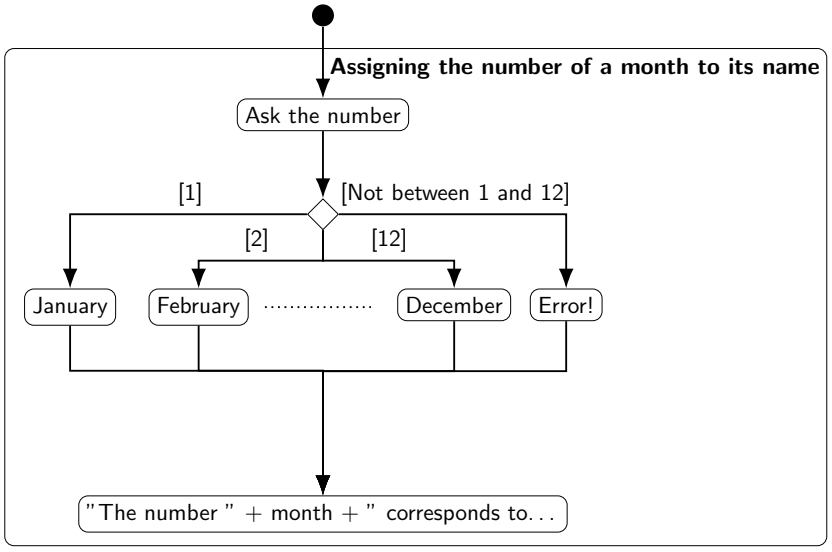

Figure 1: "A flowchart representation of the mapping between month number and name"

```
    monthName = "April";
}
else if(month == 5)
{
    monthName = "May";
}
else if(month == 6)
{
    monthName = "June";
}
else if(month == 7)
{
    monthName = "July";
}
else if(month == 8)
{
    monthName = "August";
}
else if(month == 9)
{
    monthName = "September";
```

```
}
else if(month == 10)
{
    monthName = "October";
}
else if(month == 11)
{
    monthName = "November";
}
else if(month == 12)
{
    monthName = "December";
}
else
{
    monthName = "Error!"; // Invalid month
}
Console.WriteLine("The number " + month + " corresponds to the month " + monthNa
```

- This code is very repetitive, though: every `else if` statement is almost the same, with only the number changing. The text "`if(month ==`" is copied over and over again.

Syntax for `switch` statements

- A `switch` statement is a simpler, easier way to compare a single variable against multiple possible values
- It is written like this:

```
switch (<variable name>)
{
    case <value 1>:
        <statement block 1>
        break;
    case <value 2>:
        <statement block 2>
        break;
    …
    default:
        <statement block n>
        break;
}
```

- First, the "header" of the `switch` statement names the variable that will be compared

- The "body" of the switch statement is enclosed in curly braces, and contains multiple `case` statements
- Each `case` statement gives a possible value the variable could have, and a block of statements to execute if the variable equals that value. Statement block 1 is executed if the variable is equal to value 1, statement block 2 is executed if the variable is equal to value 2, etc.
- The statement "block" within each `case` is **not** enclosed in curly braces, unlike `if` and `else if` blocks. Instead, it begins on the line after the `case` statement, and ends with the keyword `break`.
- The `default` statement is like the `else` statement: It defines code that gets executed if the variable does not match any of the values in the `case` statements.
- The values in each `case` statement must be **literals**, not variables, and they must be **unique** (you cannot write two `case` statements with the same value)

Example `switch` statement

- This program has the same behavior as our previous example, but uses a `switch` statement instaed of an `if-else-if` statement:

```
Console.WriteLine("Enter the month as a number between 1 and 12.");
int month = int.Parse(Console.ReadLine());
string monthName;
switch(month)
{
    case 1:
        monthName = "January";
        break;
    case 2:
        monthName = "February";
        break;
    case 3:
        monthName = "March";
        break;
    case 4:
        monthName = "April";
        break;
    case 5:
        monthName = "May";
        break;
    case 6:
        monthName = "June";
```

```
            break;
        case 7:
            monthName = "July";
            break;
        case 8:
            monthName = "August";
            break;
        case 9:
            monthName = "September";
            break;
        case 10:
            monthName = "October";
            break;
        case 11:
            monthName = "November";
            break;
        case 12:
            monthName = "December";
            break;
        default:
            monthName = "Error!"; // Invalid month
            break;
    }
    Console.WriteLine("The number " + month + " corresponds to the month " + monthNa
```

- Since the variable in the `switch` statement is `month`, each `case` statement means, effectively, `if (month == <value>)`. For example, `case 1:` has the same effect as `if (month == 1)`
- The values in each `case` statement must be `int` literals, since `month` is an `int`
- The `default` statement has the same effect as the final `else` in the `if-else-if` statement: it contains code that will be executed if `month` did not match any of the values

`switch` with multiple statements

- So far, our examples have used only one line of code in each `case`
- Unlike `if-else`, you do not need curly braces to put multiple lines of code in a `case`
- For example, imagine our "months" program needed to convert a month number to both a month name and a three-letter abbreviation. The `switch` would look like this:

```
string monthName;
```

```
string monthAbbrev;
switch(month)
{
    case 1:
        monthName = "January";
        monthAbbrev = "Jan";
        break;
    case 2:
        monthName = "February";
        monthAbbrev = "Feb";
        break;
    // and so on, with all the other months...
}
```

- The computer knows which statements are included in each case because of the `break` keyword. For the "`1`" case, the block of statements starts after `case 1:` and ends with the `break;` after `monthAbbrev = "Jan";`

Intentionally omitting `break`

- Each block of code that starts with a `case` statement must end with a `break` statement; it will not automatically end at the next `case` statement
  - The `case` statement only defines where code execution *starts* when the variable matches a value (like an open `{`). The `break` statement defines where it *ends* (like a close `}`).
- However, there is one exception: A `case` statement with *no body* (code block) after it does not need a matching `break`
- If there is more than one value that should have the same behavior, you can write `case` statements for both values above a single block of code, with no `break` between them. If *either one* matches, the computer will execute that block of code, and then stop at the `break` statement.
- In a switch statement with this structure:

```
switch(<variable>)
{
    case <value 1>:
    case <value 2>:
        <statement block 1>
        break;
    case <value 3>:
    case <value 4>:
```

```
        <statement block 2>
        break;
    default:
        <statement block 3>
        break;
}
```

The statements in block 1 will execute if the variable matches value 1 *or* value 2, and the statements in block 2 will execute if the variable matches value 3 *or* value 4.

- For example, imagine our program needs to tell the user which season the month is in. If the month number is 1, 2, or 3, the season is the same (winter), so we can combine these 3 cases. This code will correctly initialize the string `season`:

```
switch(month)
{
    case 1:
    case 2:
    case 3:
        season = "Winter";
        break;
    case 4:
    case 5:
    case 6:
        season = "Spring";
        break;
    case 7:
    case 8:
    case 9:
        season = "Summer";
        break;
    case 10:
    case 11:
    case 12:
        season = "Fall";
        break;
    default:
        season = "Error!"
        break;
}
```

If `month` is equal to 1, execution will start at `case 1:`, but the computer will continue past `case 2` and `case 3` and execute `season = "Winter"`. It will then stop when it reaches the `break`, so `season` gets the value "Winter". Similarly, if `month` is equal to 2, execution will start at `case 2:`, and continue until the `break` statement, so

`season` will also get the value "Winter".

- This syntax allows `switch` statements to have conditions with a logical OR, equivalent to an `if` condition with an `||`, like `if(x == 1 || x == 2)`
- For example, the "seasons" statement could also be written as an `if-else-if` with `||` operators, like this:

```
if(month == 1 || month == 2 || month == 3)
{
    season = "Winter";
}
else if(month == 4 || month == 5 || month == 6)
{
    season = "Spring";
}
else if(month == 7 || month == 8 || month == 9)
{
    season = "Summer";
}
else if(month == 10 || month == 11 || month == 12)
{
    season = "Fall"
}
else
{
    season = "Error!"
}
```

Scope and `switch`

- In C#, the scope of a variable is defined by curly braces (recall that local variables defined in a method have a scope that ends with the `}` at the end of the method)
- Since the `case` statements in a `switch` do not have curly braces, they are all in the same scope: the one defined by the `switch` statement's curly braces
- This means you cannot declare a "local" variable within a `case` statement – it will be in scope (visible) to all the other `case` statements
- For example, imagine you wanted to use a local variable named `nextMonth` to do some local computation within each case in the "months" program. This code will not work:

```
switch(month)
```

```
{
    case 1:
        int nextMonth = 2;
        monthName = "January";
        // do something with nextMonth...
        break;
    case 2:
        int nextMonth = 3;
        monthName = "February";
        // do something with nextMonth...
        break;
    //...
}
```

The line `int nextMonth = 3` would cause a compile error because a variable named `nextMonth` already exists – the one declared within `case 1`.

Limitations of `switch`

- Not all `if-else-if` statements can be rewritten as `switch` statements
- `switch` can only test equality, so in general, only `if` statements whose condition uses == can be converted to `switch`
- For example, imagine we have a program that determines how much of a fee to charge a rental car customer based on the number of miles the car was driven. A variable named `mileage` contains the number of miles driven, and it is used in this `if-else-if` statement:

```
decimal fee = 0;
if(mileage > 1000)
{
    fee = 50.0M;
}
else if(mileage > 500)
{
    fee = 25.0M;
}
```

- This `if-else-if` statement could not be converted to `switch(mileage)` because of the condition `mileage > 1000`. The `switch` statement would need to have a `case` for each number greater than 1000, which is infinitely many.