Contents

if **1**
if Statements . . . 1
Introduction . . . 1
Example code with an `if` statement . . . 1
Syntax and rules for if statements . . . 3
if-else Statements . . . 3
Syntax and comparison . . . 4
Nested if-else Statements . . . 5
Using nested if statements . . . 6
if-else-if Statements . . . 8
If-else-if syntax . . . 8
Using if-else-if to solve the "floors problem" . . . 9
if-else-if with different conditions . . . 10
if-else-if vs. nested if . . . 11

if

if Statements

Introduction

- Recall from a previous lecture (Booleans and Comparisons) that decision structures change the flow of code execution based on conditions
- Now that we know how to write conditions in C#, we can write decision structures
- Our first decision structure is the **if statement**, which executes a block of code *only if a condition is true*

Example code with an `if` statement

```
Console.WriteLine("Enter your age");
int age = int.Parse(Console.ReadLine());
if (age >= 18)
{
    Console.WriteLine("You can vote!");
}
Console.WriteLine("Goodbye");
```

- After the keyword **`if`** is a condition, in parentheses: `age >= 18`
- On the next line after the **`if`** statement, the curly brace begins a code block. The code in this block is "controlled" by the **`if`** statement.

- If the condition `age >= 18` is true, the code in the block (the WriteLine statement with the text "You can vote!") gets executed, then execution proceeds to the next line (the WriteLine statement that prints "Goodbye")
- If the condition `age >= 18` is false, the code in the block gets *skipped*, and execution proceeds directly to the line that prints "Goodbye"
- The behavior of this program can be represented by this flowchart:

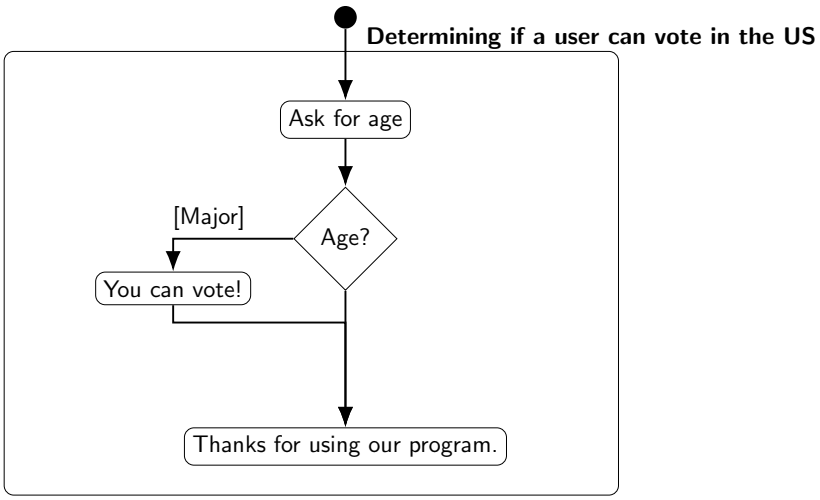

Figure 1: "A flowchart representation of an if statement"

- Example interaction 1:

```
Enter your age
20
You can vote!
Goodbye
```

When the user enters "20", the value 20 is assigned to the `age` variable, so the condition `age >= 18` is true. This means the code inside the **`if`** statement's block gets executed.

- Example interaction 2:

```
Enter your age
17
Goodbye
```

When the user enters "17", the value 17 is assigned to the `age` variable, so the condition `age >= 18` is false, and the **`if`** statement's code block gets skipped.

Syntax and rules for if statements

- Formally, the syntax for an **`if`** statement is this:

```
if (<condition>)
{
    <statements>
}
```

- The "condition" in parentheses can be any expression that produces a `bool` value, including all of the combinations of conditions we saw in the previous lecture (Booleans and Comparisons). It can even be a `bool` variable, since a `bool` variable "contains" a `bool` value.
- Note that there is no semicolon after the **`if`** `(<condition>)`. It's a kind of "header" for the following block of code, like a method header.
- The statements in the code block will be executed if the condition evaluates to **`true`**, or skipped if it evaluates to **`false`**
- If the code block contains only *one* statement, the curly braces can be omitted, producing the following syntax:

```
if(<condition>)
    <statement>
```

  For example, the **`if`** statement in our previous example could be written like this, since there was only one statement in the code block:

```
if(age >= 18)
    Console.WriteLine("You can vote!");
Console.WriteLine("Goodbye");
```

- Omitting the curly braces is slightly dangerous, though, because it makes it less obvious which line of code is controlled by the **`if`** statement. It is up to you, the programmer, to remember to indent the line after the **`if`** statement, and then de-indent the line after that; indentation is just a convention. Curly braces make it easier to see where the **`if`** statement starts and ends.

if-else Statements

Example:

```
if(age >= 18)
{
    Console.WriteLine("You can vote!");
}
else
{
    Console.WriteLine("You are too young to vote");
}
Console.WriteLine("Goodbye");
```

- The **if-else statement** is a decision structure that chooses *which* block of code to execute, based on whether a condition is true or false
- In this example, the condition is `age >= 18` again
- The first block of code (underneath the `if`) will be executed if the statement is true – the console will display "You can vote!"
- The *second* block of code, which comes after the keyword `else`, will be executed if the statement is *false* – so if the user's age is less than 18, the console will display "You are too young to vote"
- Only one of these blocks of code will be executed; the other will be skipped
- After executing one of the two code blocks, execution continues at the next line after the `else` block, so in either case the console will next display "Goodbye"
- The behavior of this program can be represented by this flowchart:

Syntax and comparison

- Formally, the syntax for an `if-else` statement is this:

```
if (<condition>)
{
    <statement block 1>
}
else
{
    <statement block 2>
}
```

- As with the `if` statement, the condition can be anything that produces a `bool` value
- Note that there is no semicolon after the `else` keyword

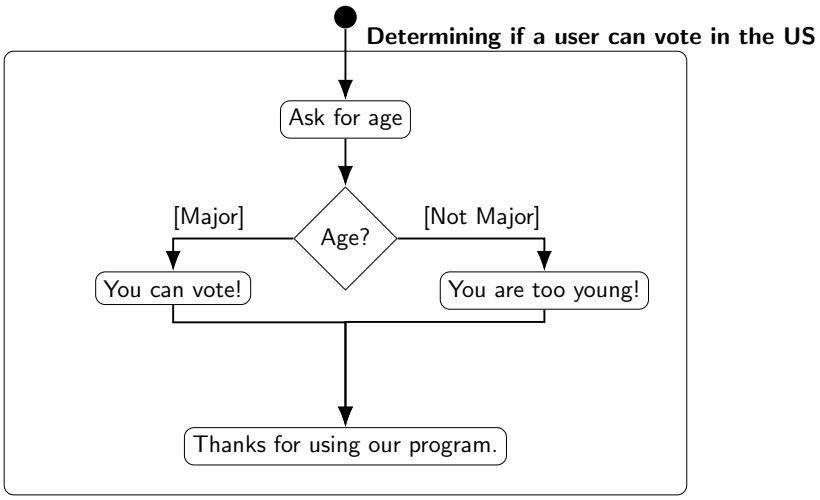

Figure 2: "A flowchart representation of an if-else statement"

- If the condition is true, the code in statement block 1 is executed (this is sometimes called the "if block"), and statement block 2 is skipped
- If the condition is false, the code in statement block 2 is executed (this is sometimes called the "else block"), and statement block 1 is skipped
- This is very similar to an if statement; the difference is what happens if the condition is false
  - With an `if` statement, the "if block" is executed if the condition is true, but *nothing happens* if the condition is false.
  - With an `if-else` statement, the code in the "else block" is executed if the condition is false, so something always happens - one of the two code blocks will get executed

Nested if-else Statements

- If-else statements are used to change program flow based on a condition; they represent making a decision
- Sometimes decisions are more complex than a single yes/no question: once you know whether a certain condition is true or false, you then need to ask another question (check another condition)

based on the outcome

- For example, we could improve our voting program to ask the user whether he/she is a US citizen, as well as his/her age. This means there are two conditions to evaluate, as shown in this flowchart:

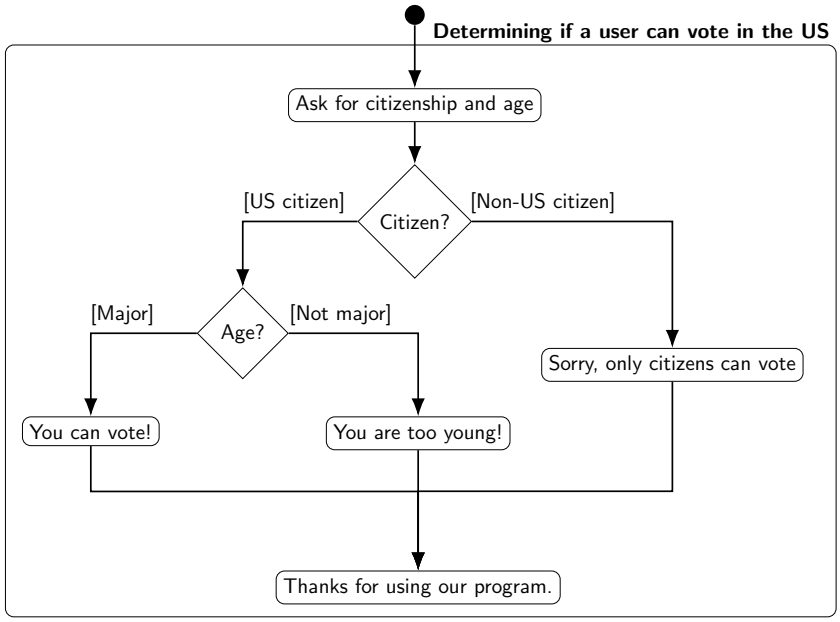

Figure 3: "A flowchart representation of the nested if-else statement"

  - First, the program should test whether the user is a citizen. If not, there is no need to check the user's age, since he/she cannot vote anyway
  - If the user is a citizen, the program should then test whether the user is over 18 to determine if he/she is old enough to vote.

Using nested if statements

- An `if` statement's code block can contain any kind of statements, including another `if` statement
- Putting an `if` statement inside an if block represents making a sequence of decisions - once execution has reached the inside of an if block, your program "knows" that the `if` condition is true, so it can proceed to make the next decision

- For the voting example, we can implement the decision structure from the flowchart above with this code, assuming `age` is an `int` and `usCitizen` is a `bool`:

```
if(usCitizen == true)
{
    if(age >= 18)
    {
        Console.WriteLine("You can vote!");
    }
    else
    {
        Console.WriteLine("You are too young to
↪   vote");
    }
}
else
{
    Console.WriteLine("Sorry, only citizens can
↪   vote");
}
Console.WriteLine("Goodbye");
```

  - First, the program tests the condition `usCitizen == true`, and if it is true, the code in the first "if block" is executed
  - Within this if block is another `if` statement that tests the condition `age >= 18`. This represents checking the user's age after determining that he/she is a US citizen - execution only reaches this second `if` statement if the first one evaluated to true. So "You can vote" is printed if both `usCitizen == true` and `age >= 18`
  - If the condition `usCitizen == true` is false, the if block is skipped and the else block is executed instead, so the entire inner `if` statement is never executed – the user's age does not matter if he/she isn't a citizen
  - Note that the condition `usCitizen == true` could also be expressed by just writing the name of the variable `usCitizen` (i.e., the if statement would be `if(usCitizen)`), because `usCitizen` is a `bool` variable. We do not need the equality comparison operator to test if it is `true`, because an `if` statement already tests whether its condition is `true` (and a `bool` variable by itself is a valid condition)
  - Note that indentation helps you match up an `else` block to its corresponding `if` block. The meaning of `else` depends on which `if` statement it goes with: the "outer" `else` will be executed if the condition `usCitizen == true` is false, while the "inner" `else` will be executed if the condition `age >= 18`

is false.

- Nested **if** statements do not need to be the *only* code in the if block; you can still write other statements before or after the nested **if**
- For example, we could change our voting program so that it only asks for the user's age if he/she is a citizen:

```
if(usCitizen == true)
{
    Console.WriteLine("Enter your age");
    int age = int.Parse(Console.ReadLine());
    if(age >= 18)
    {
        Console.WriteLine("You can vote!");
    }
    else
    {
        Console.WriteLine("You are too young to
↪   vote");
    }
}
else
{
    Console.WriteLine("Sorry, only citizens can
↪   vote");
}
Console.WriteLine("Goodbye");
```

if-else-if Statements

- Sometimes your program needs to test multiple conditions at once, and take different actions depending on which one is true
- Example: We want to write a program that tells the user which floor a `ClassRoom` object is on, based on its room number
  - If the room number is between 100 and 200 it is on the first floor; if it is between 200 and 300 it is on the second floor; if it is greater than 300 it is on the third floor
- There are 3 ranges of numbers to test, and 3 possible results, so we cannot do it with a single if-else statement

If-else-if syntax

- An if-else-if statement looks like this:

```
if(<condition 1>)
{
    <statement block 1>
}
else if(<condition 2>)
{
    <statement block 2>
}
else if(<condition 3>)
{
    <statement block 3>
}
else
{
    <statement block 4>
}
```

- Unlike an **`if`** statement, there are multiple conditions
- They are evaluated *in order*, top to bottom
- Just like with **`if`**-**`else`**, exactly one block of code will get executed
- If condition 1 is true, statement block 1 is executed, and everything else is skipped
- If condition 1 is false, statement block 1 is skipped, and execution proceeds to the first **`else if`** line; condition 2 is then evaluated
- If condition 2 is true, statement block 2 is executed, and everything else is skipped
  - Thus, statement block 2 is only executed if condition 1 is false *and* condition 2 is true
- Same process repeats for condition 3: If condition 2 is false, condition 3 is evaluated, and statement block 3 is either executed or skipped
- If *all* the conditions are false, the final else block (statement block 4) is executed

Using if-else-if to solve the "floors problem"

- Assuming `myRoom` is a `ClassRoom` object, this code will display which floor it is on:

```
if(myRoom.GetNumber() >= 300)
{
    Console.WriteLine("Third floor");
```

```
}
else if(myRoom.GetNumber() >= 200)
{
    Console.WriteLine("Second floor");
}
else if(myRoom.GetNumber() >= 100)
{
    Console.WriteLine("First floor");
}
else
{
    Console.WriteLine("Invalid room number");
}
```

- If the room number 300 or greater (e.g. 365), the first "if" block is executed, and the rest are skipped. The program prints "Third floor"
- If the room number is less than 300, the program continues to the line `else if(myRoom.GetNumber() >= 200)` and evaluates the condition
- If `myRoom.GetNumber() >= 200` is true, it means the room number is between 200 and 299, and the program will print "Second floor." Even though the condition only tests whether the room number is >= 200, this condition is only evaluated if the first one was false, so we know the room number must be < 300.
- If the second condition is false, the program continues to the line `else if(myRoom.GetNumber() >= 100)`, evaluates the condition, and prints "First floor" if it is true.
- Again, the condition `myRoom.GetNumber() >= 100` is only evaluated if the first two conditions have already been tested and turned out false, so we know the room number is less than 300 and less than 200.
- In the final `else` block, the program prints "Invalid room number" because this block is only executed if the room number is less than 100 (all three conditions were false).

if-else-if with different conditions

- We often use if-else-if statements to test the same variable multiple times, but there is no requirement for the conditions to use the same variable
- An if-else-if statement can use several different variables, and its conditions can be completely unrelated, like this:

```
int x;
if(myIntVar > 12)
{
    x = 10;
}
else if(myStringVar == "Yes")
{
    x = 20;
}
else if(myBoolVar)
{
    x = 30;
}
else
{
    x = 40;
}
```

- Note that the order of the else-if statements still matters, because they are evaluated top-to-bottom. If `myIntVar` is 15, it does not matter what values `myStringVar` or `myBoolVar` have, because the first if block (setting x to 10) will get executed.
- Example outcomes of executing this code (which value x is assigned) based on the values of `myIntVar`, `myStringVar`, and `myBoolVar`:

myIntVar	myStringVar	myBoolVar	x
12	"Yes"	**true**	20
15	"Yes"	**false**	10
-15	"yes"	**true**	30
10	"yes"	**false**	40

if-else-if vs. nested if

- Sometimes a nested **if** statement can be rewritten as an **if-else-if** statement
- This reduces the amount of indentation in your code, which makes it easier to read
- To convert a nested **if** statement to **if-else-if**, you'll need to combine the conditions of the "outer" and "inner" **if** statements, using the logical operators
- A nested **if** statement inside an **if** block is testing whether the

outer **if**'s condition is true *and* its own condition is true, so combine them with the && operator

- The **else** block of the inner **if** statement can be rewritten as an **else if** by combining the outer **if**'s condition with the *opposite* of the inner **if**'s condition, since "else" means "the condition is false." We need to explicitly write down the "false condition" that is normally implied by **else**.
- For example, we can rewrite this nested **if** statement:

```
if(usCitizen == true)
{
    if(age >= 18)
    {
        Console.WriteLine("You can vote!");
    }
    else
    {
        Console.WriteLine("You are too young to
↪   vote");
    }
}
else
{
    Console.WriteLine("Sorry, only citizens can
↪   vote");
}
```

as this **if-else-if** statement:

```
if(usCitizen == true && age >= 18)
{
    Console.WriteLine("You can vote!");
}
else if(usCitizen == true && age < 18)
{
    Console.WriteLine("You are too young to vote");
}
else
{
    Console.WriteLine("Sorry, only citizens can
↪   vote");
}
```

- Note that the **else** from the inner if statement becomes **else if**(usCitizen == true && age < 18) because we combined the outer if condition (usCitizen == true) with the opposite of the inner if condition (age >= 18).

- Not all nested **if** statements can be rewritten this way. If there is additional code in a block, other than the nested **if** statement, it is harder to convert it to an if-else-if
- For example, in this nested **if** statement:

```
if(usCitizen == true)
{
    Console.WriteLine("Enter your age");
    int age = int.Parse(Console.ReadLine());
    if(age >= 18)
    {
        Console.WriteLine("You can vote!");
    }
    else
    {
        Console.WriteLine("You are too young to
 ↪  vote");
    }
}
else
{
    Console.WriteLine("Sorry, only citizens can
 ↪  vote");
}
Console.WriteLine("Goodbye");
```

  the code that asks for the user's age executes after the outer **if** condition is determined to be true, but before the inner **if** condition is tested. There would be nowhere to put this code if we tried to convert it to an if-else-if statement, since both conditions must be tested at the same time (in `if(usCitizen == true && age >= 18)`).
- On the other hand, any if-else-if statement can be rewritten as a nested **if** statement
- To convert an if-else-if statement to a nested **if** statement, rewrite each **else if** as an **else** block with a nested **if** statement inside it – like you're splitting the "if" from the "else"
- This results in a lot of indenting if there are many **else if** lines, since each one becomes another nested **if** inside an **else** block
- For example, the "floors problem" could be rewritten like this:

```
if(myRoom.GetNumber() >= 300)
{
    Console.WriteLine("Third floor");
}
```

```
else
{
    if(myRoom.GetNumber() >= 200)
    {
        Console.WriteLine("Second floor");
    }
    else
    {
        if(myRoom.GetNumber() >= 100)
        {
            Console.WriteLine("First floor");
        }
        else
        {
            Console.WriteLine("Invalid room number");
        }
    }
}
```