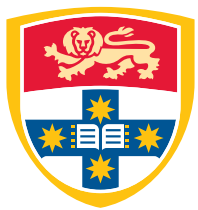
INFO1103: Introduction to Programming

School of Information Technologies, University of Sydney



Week 3: Control Flow 1: Branching and Looping

We will cover: Branching with `if` and `else`, Looping with `while` and more on handling numbers: `if`/`else`, `switch`

You should read: §§3.1 – 3.3, 4.1 of Savitch

Lecture 5: Control Flow 1: `if`

Handling simple choices

Evaluating multiple conditions

Boolean expressions

Boolean logic

Operation	*Meaning*
`x && y`	true if both `x` and `y` are true, false otherwise
`x || y`	true if `x` or `y` or both are true, false otherwise
`! x`	true if `x` is false, false otherwise
`x == y`	true if `x` and `y` are both true or both false, false otherwise
`x != y`	true if `x` is true and `y` is false, or `x` is false and `y` is true, false otherwise

Complex boolean expressions

Ask a simple question that has a yes/no answer.

Can depend on sub problems to be solve first.

Question: Ready to go out?

depends on: have keys? have phone? if it is raining, do I have umbrella?

```
1       boolean haveKeys = true;
2       boolean havePhone = true;
3       boolean isRaining = true;
4       boolean haveUmbrella = false;
5
6       boolean readyToLeave = ?
```

What are all the cases is readyToLeave `true` or `false`. Use your truth tables!

Complex boolean expressions

Question: Ready to go out?

depends on: have keys? have phone? if it is raining, do I have umbrella?

```
1       boolean readyToLeave =
2                   ( haveKeys && havePhone ) &&
3                   ( ( isRaining && haveUmbrella ) || ( !isRaining ) )
```

Complex boolean expressions

What is the boolean result

```
1 double flour = 0.25; // 1/4 cup
2 boolean canMakeCake = ( flour >= (1/2) ); // need at least 1/2 cup
3 System.out.println("canMakeCake = " + canMakeCake);
```

We need to be careful to evaluate the expressions as intended.

Casting

Treating variables as other types

Casting primitive

Operators used in an expression can be evaluated differently based on the data types involved.

What is the result?

```
1       int litres = 2;
2       int persons = 5;
3       double portion = litres / persons;
```

casting allows the programmer to explicitly tell the compiler to treat a variable, or expression, as another type.

Casting primitive types

Convert integer to floating point number

```
1       int litres = 2;
2       int persons = 5;
3       double portion = (double)litres / (double)persons;
```

The opposite is also possible. What is the result?

```
1       double portion = 0.330; // 330mL
2       int litres = 5;
3       int persons = litres / portion;
```

In this case how do we convert floating point number to integer?

if


(AN UNMATCHED LEFT PARENTHESIS
CREATES AN UNRESOLVED TENSION
THAT WILL STAY WITH YOU ALL DAY.

Source: xkcd

The `if` statement

- What's it for?
  - to have different behaviour in a program based on whether something is true or false (this is a kind of *control statement*).
- What does it look like?
  "if (*condition*) *statement*"

- *condition* is a boolean expression that evaluates to `true` or `false`

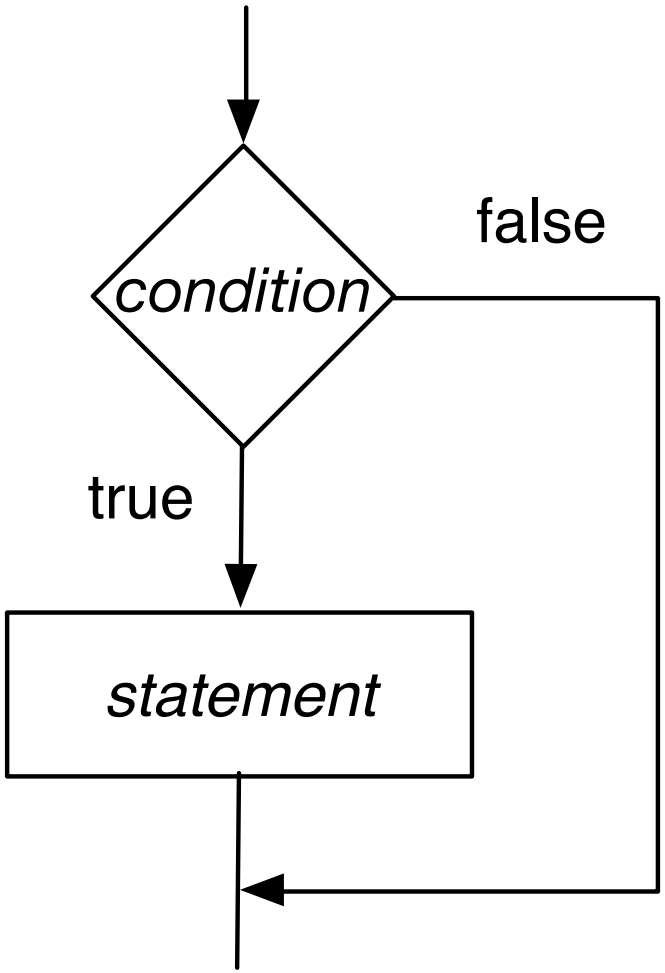
- *statement* is one computer instruction followed by a ; or a sequence of computer instructions within a block (with opening and closing { })

- In code this looks like

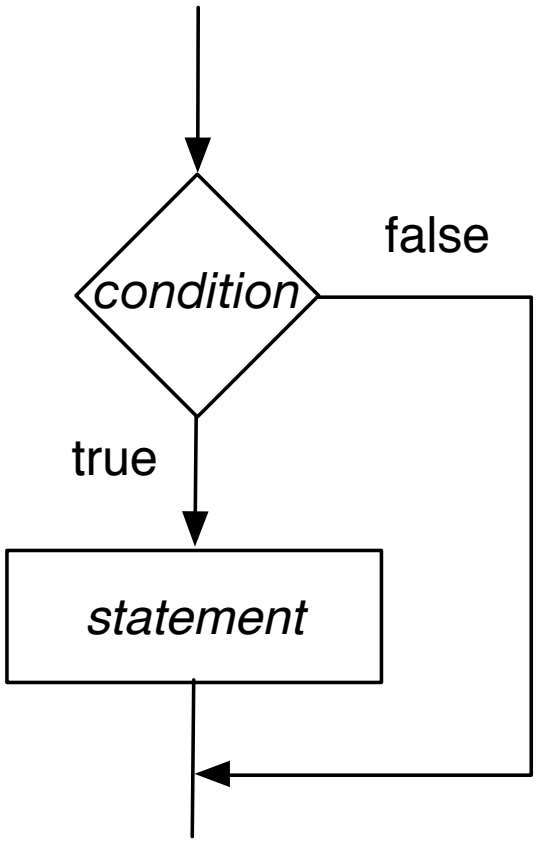
```
1    if (x > 0)
2    {
3        print("x is positive\n");
4    }
```

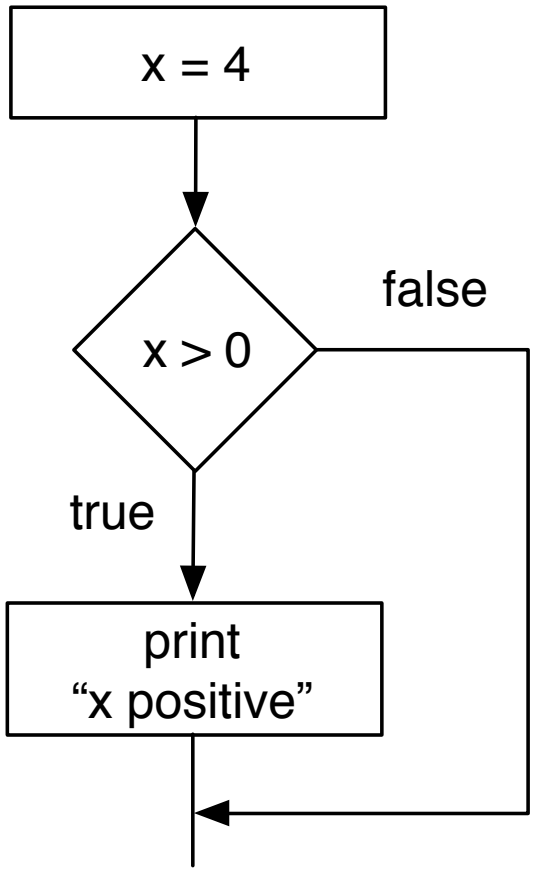
How `if` works

The “if” statement is a simple control flow structure: it is used to test the value of an expression, to see whether it’s true, and if true, then to do something else.



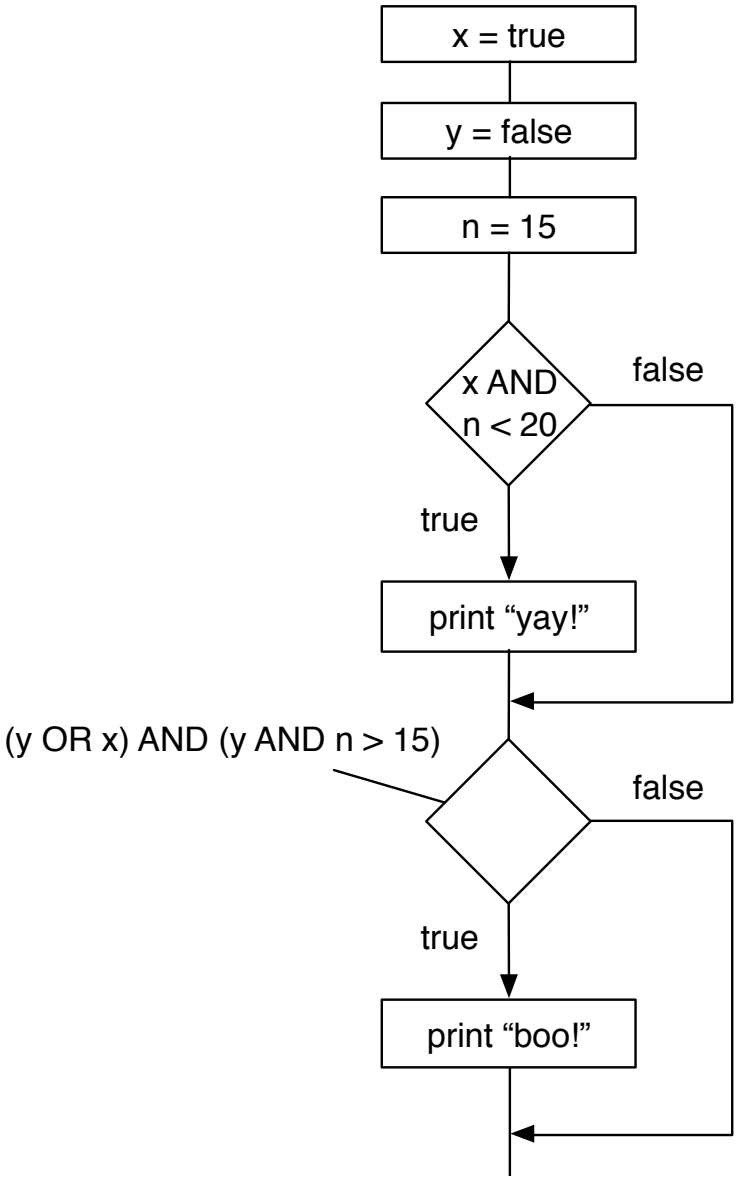
How `if` works


condition
false
true
statement

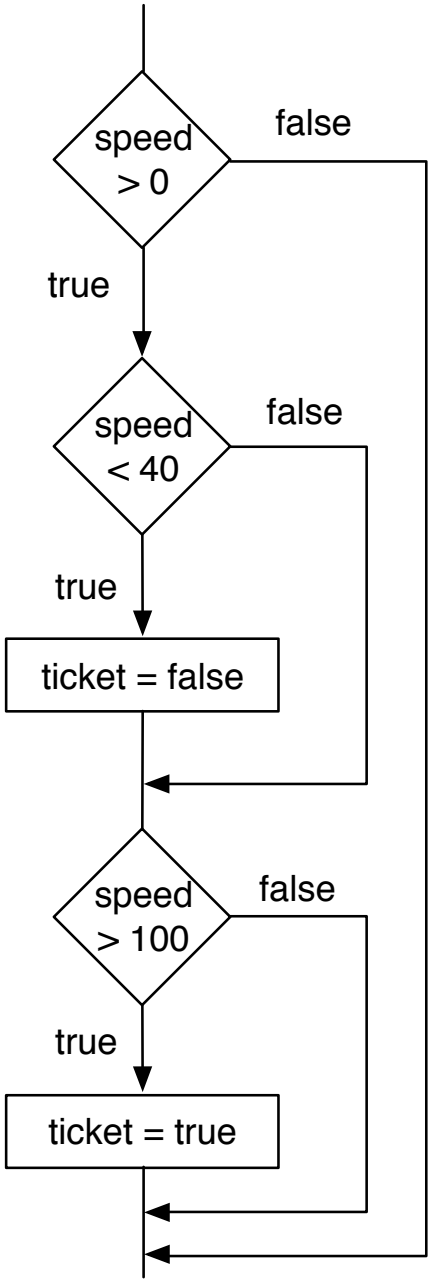

x = 4
x > 0
false
true
print
“x positive”

More logic with `if`

The *condition* can be a complex boolean expression. What is the code?



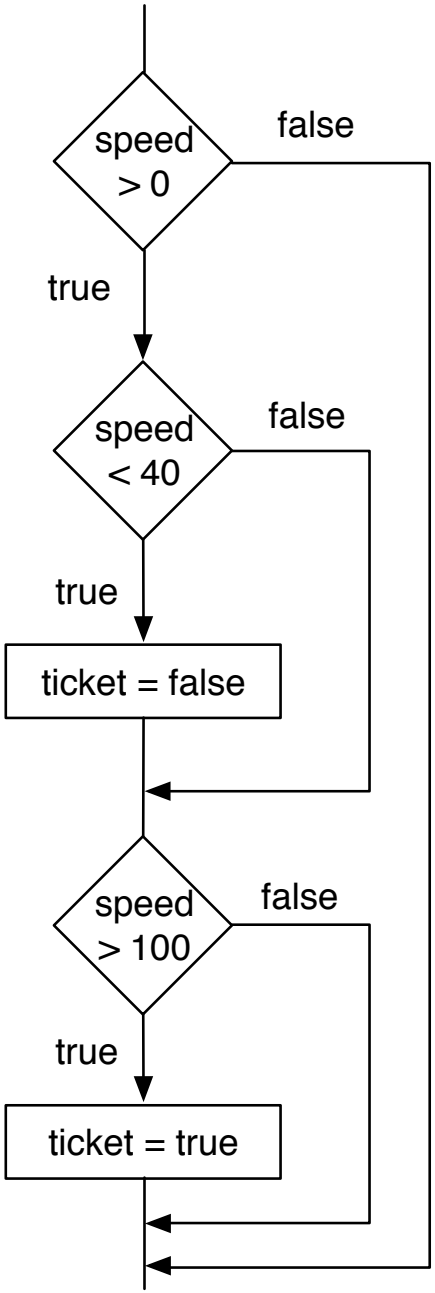
Nested `if`



You can put `if`s inside `if`s, like this:

```
1    boolean ticket = false;
2    if (speed > 0) {
3        if (speed < 40) {
4            ticket = false;
5        }
6        if (speed > 100) {
7            ticket = true;
8        }
9    }
```

Deskcheck



Control statements like `if` can lead to different statements being executed, code paths.

The variables will have different values depending on the current conditions

	speed	ticket	
	0	F	

Extending `if`

`if` you had to write an "`if`" for each possible outcome you can easily miss a case:

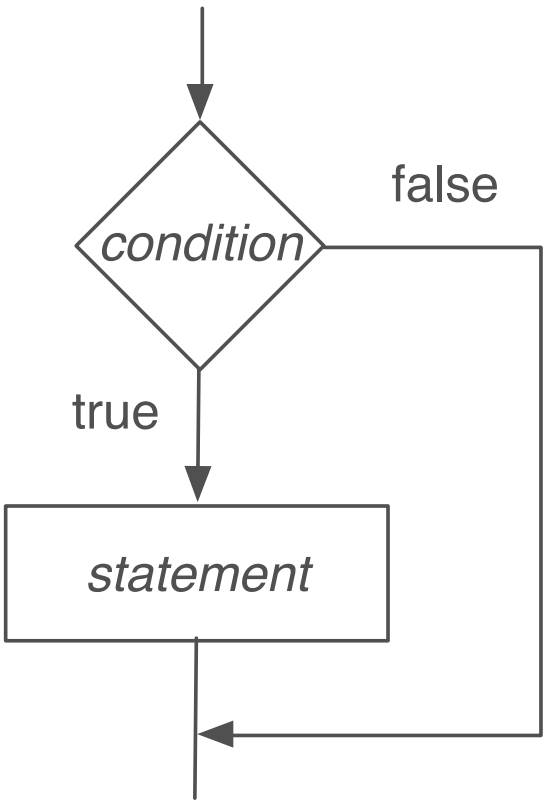
```
1       if (x > 0) {
2           // do something
3       }
4       if (x <= 0) {
5           // do something else
6       }
```

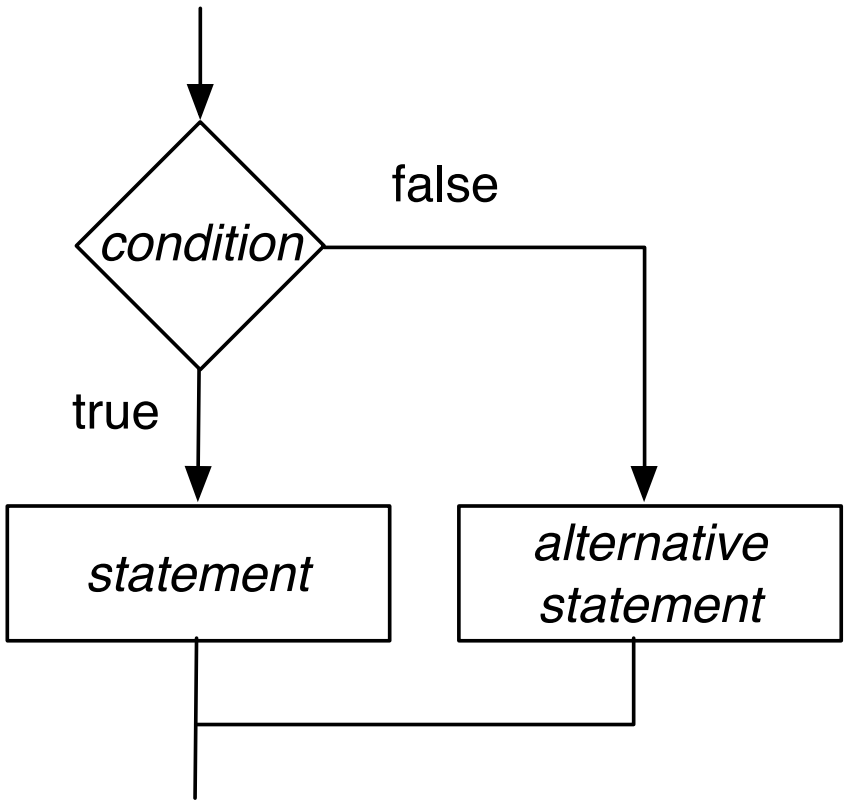
```
1       if (    (i == 7 && j < i) || ( !((j - i) < 5) && (j != 0) ) ) {
2           // do something
3       }
4       if ( !( (i == 7 && j < i) || ( !((j - i) < 5) && (j != 0) )) )
5           // do something else
6       }
```

if ... else

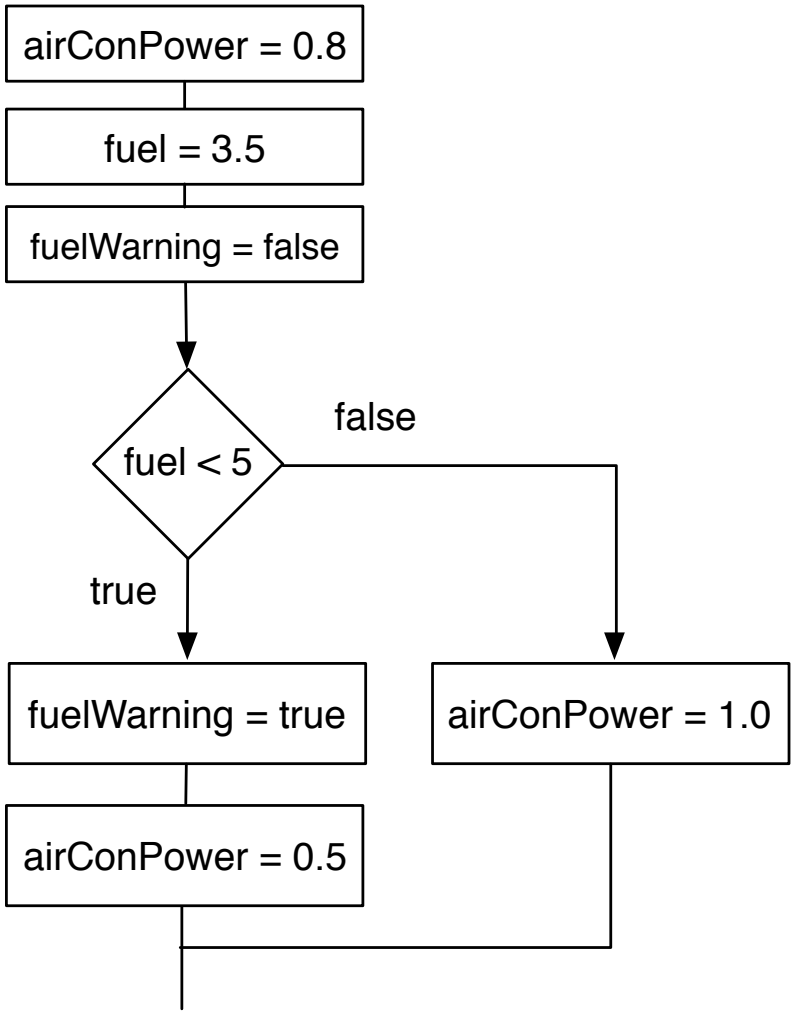
The alternative to writing both cases out explicitly is to use `else`.
Here's the syntax:

1 **if** (*condition*) **then**
2 · *statement*
3 **else**
4 · *alternative statement*

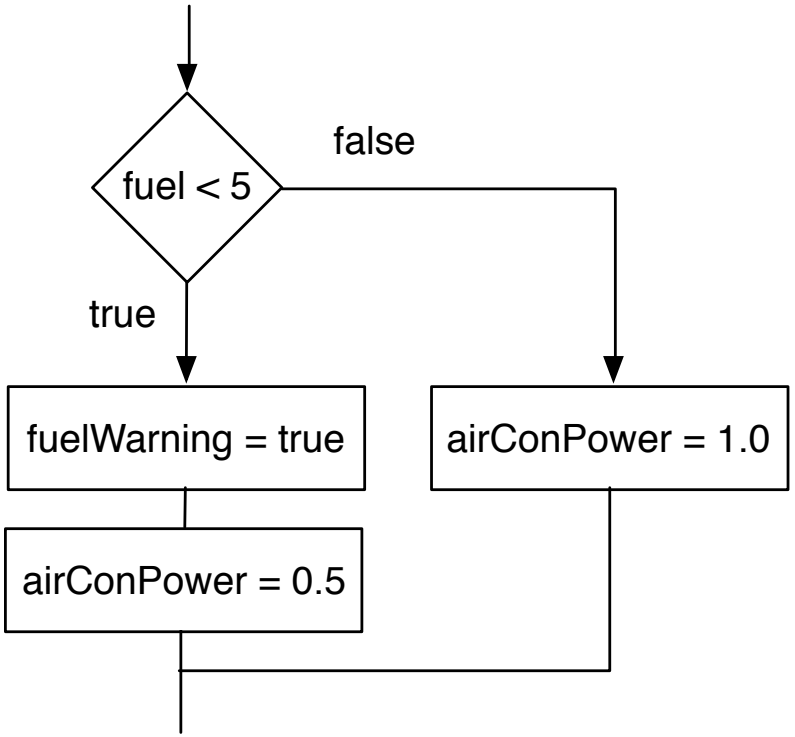




Example


airConPower = 0.8
fuel = 3.5
fuelWarning = false
fuel < 5
false
true
fuelWarning = true
airConPower = 1.0
airConPower = 0.5

Deskcheck 2



Don't always know values ahead of time

Use desk check to track each variable value and how it changes

As part of testing, you should pick values that are normal, abnormal, on the boundary etc.

fuel	airConPower	fuelWarning
0		
5		

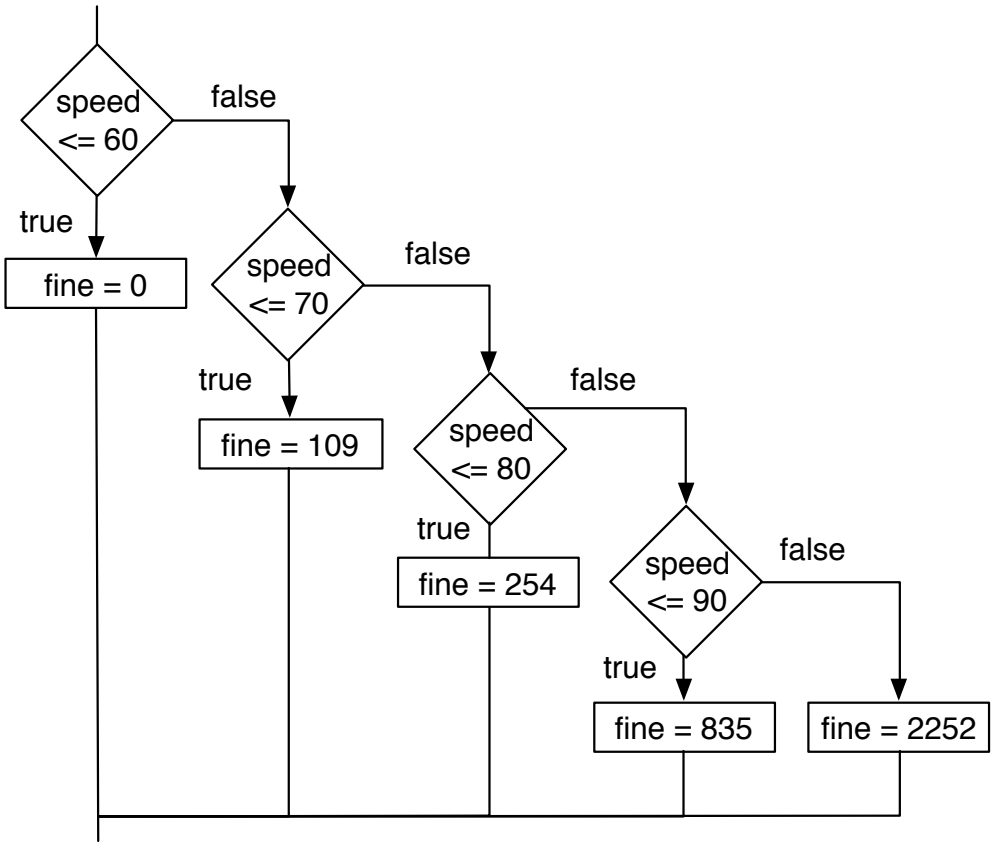
if ... else ... if

You can have separate conditions checked in sequence. `if` statements with the alternative statement as `else if`:

```
1    if (ch == 'a') {
2        // do thing 1
3    } else if (ch == 'b') {
4        // do thing 2
5    } else if (ch == 'c') {
6        // do thing 3
7    }
```

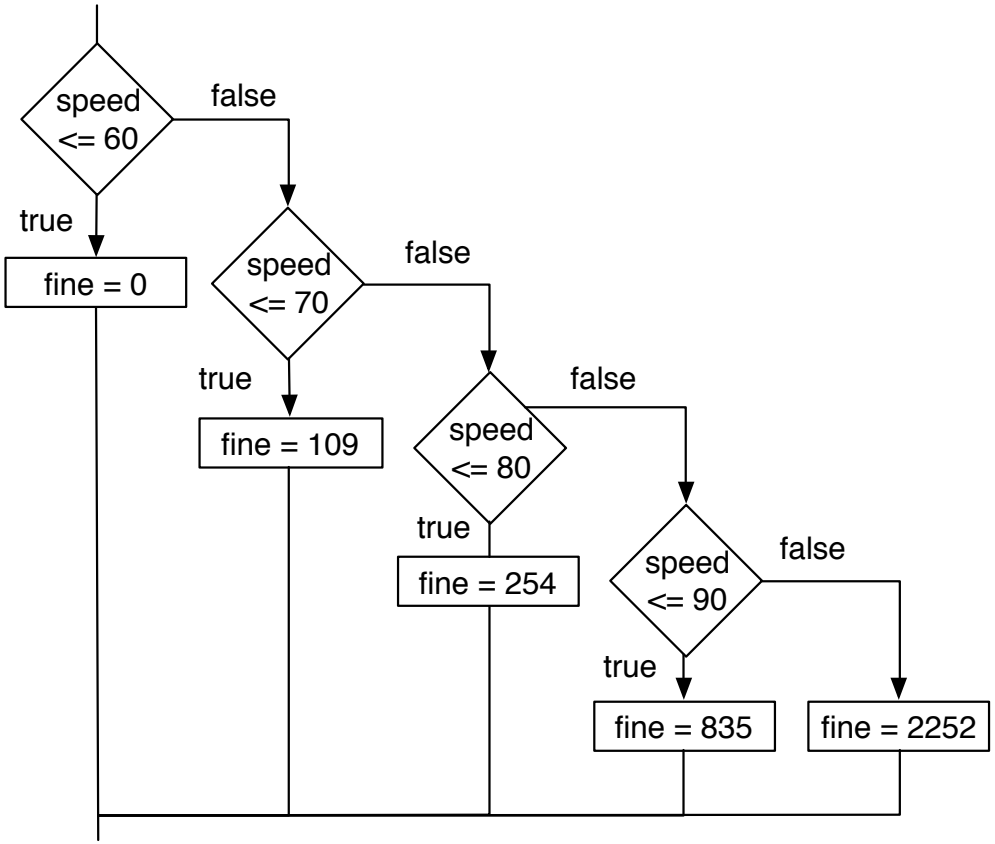
It is equivalent to putting the expressions after the `else`s into their own separate blocks.

An example of checking conditions in a specific order




```
1  if (speed <= 60) {
2    fine = 0;
3  } else if (speed <= 70) {
4    fine = 109;
5  } else if (speed <= 80) {
6    fine = 254;
7  } else if (speed <= 90) {
8    fine = 835;
9  } else {
10   fine = 2252;
11 }
```

Deskcheck 3



speed	fine
-789	