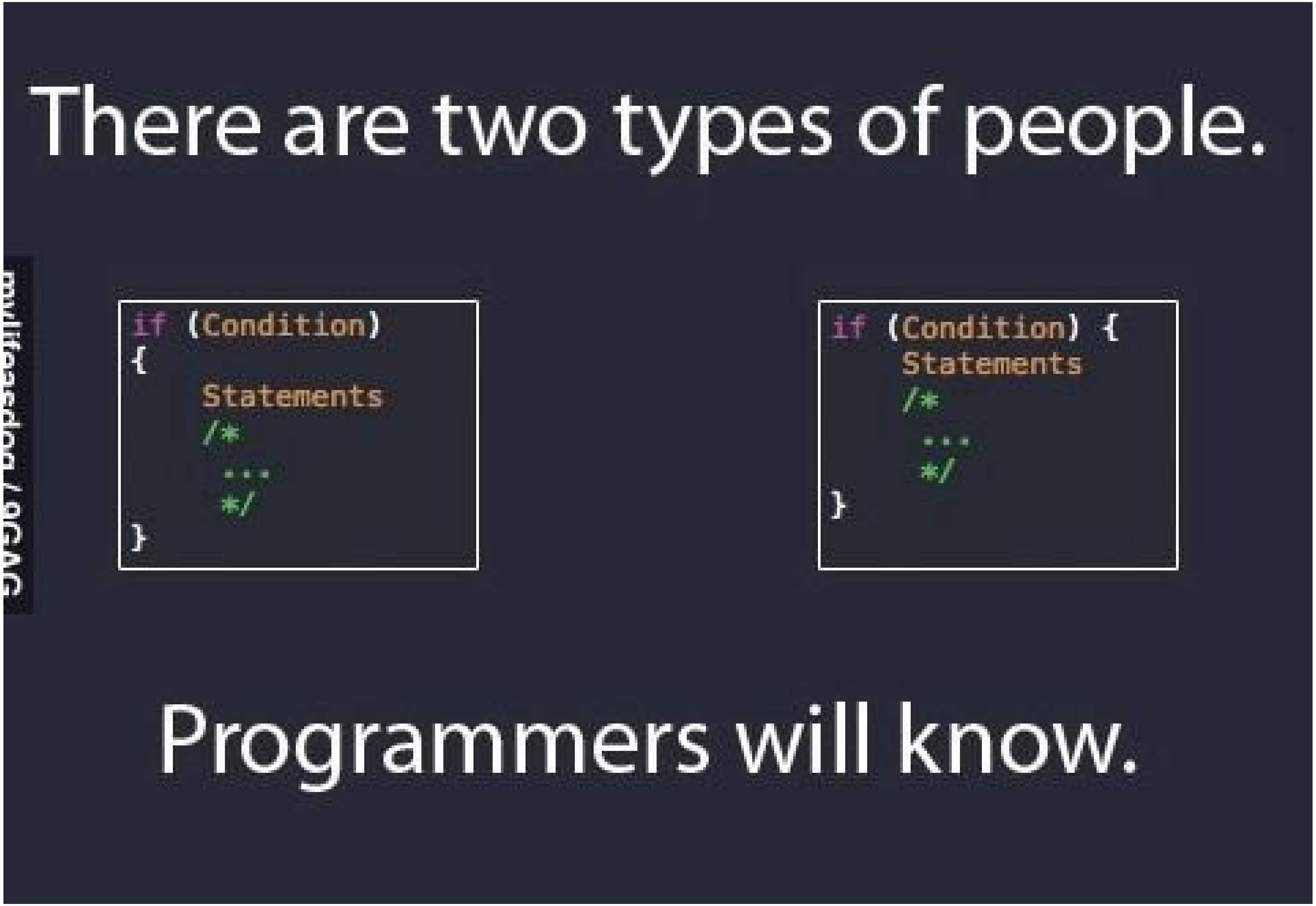
6. Everything in the universe is either a potato or not a potato.

Decisions



There are two types of people.

```
if (Condition)
{
    Statements
    /*
     ...
     */
}
```

```
if (Condition) {
    Statements
    /*
     ...
     */
}
```

Programmers will know.

mylifeasdog / 9GAG

Due this week

- **Homework 2**
  - Write solutions in VSCode and paste in Autograder, **Homework 2 CodeRunner**.
  - Zip your .cpp files and submit on canvas **Homework 2**.
- Extra-credit: start early bonus (2 points)
- Start going through the textbook readings and watch the videos
  - Take **Quiz 3**.
- 3-2-1 on Friday
- Check the due date! **No late submissions!!**

Today

- Boolean variables
- Relational operators
- Logical Operators
- The `if` statement

Boolean Variables & Operators

Boolean Variables and Operators

- Sometimes you need to evaluate a logical condition in one part of a program and use it elsewhere.
- To store a condition that can be `true` or `false`, you use a Boolean variable
- Variables of type `bool` can hold exactly two values, `false` or `true`.
  - **not** strings.
  - **not** integers; they are special values, just for Boolean variables.

- BUT actually zero is **false**, and any non-zero value is treated as **true**.

Relational Operators

C++	Math Notation	Description
>	>	Greater than
>=	≥	Greater than or equal
<	<	Less than
<=	≤	Less than or equal
==	=	Equal
!=	≠	Not equal

Boolean Variables

- Here is a declaration of a Boolean variable, initialized to false:

```
bool failed = false;
```

- Here’s another example:

```
// If the value of x is negative, set the boolean variable to True
bool isNegative = x < 0;
```

Boolean Variables - `cout`

- Boolean variables that hold the value True, print the value 1 when displayed to the console via cout
- Boolean variables that hold the value False, print the value 0 when displayed to the console via cout
- Here’s an example:

```
int x = -3;
bool isNegative = (x < 0);
bool isPositive = (x > 0);
cout << isNegative << “ “ << isPositive << endl;
```

Output: `1 0`

Expression	Value	Comment
3 <= 4	🟩	3 is less than 4; <= tests for “less than or equal”.
3 =< 4	🟩	The “less than or equal” operator is <=, not =<. The “less than” symbol comes first.
3 > 4	🟩	> is the opposite of <=.
4 < 4	🟩	The left-hand side of < must be strictly smaller than the right-hand side.
4 <= 4	🟩	Both sides are equal; <= tests for “less than or equal”.

Relational Operators – Some Notes

- The == operator is initially confusing to beginners.
- In C++, = already has a meaning, namely assignment
- The == operator denotes equality testing:

```
floor = 13; // Assign the value 13 to floor
floor == 13; // Check whether value of floor equals 13
```

- You can compare strings as well:

```
         if (input == "Quit") ...
```

Confusing = and ==

- In C++, assignments have values.
- The value of the assignment expression `floor = 13` is 13.
- These two features conspire to make a horrible pitfall:

```
if (floor = 13) …
```

- is <u>legal</u> C++.

- The code sets floor to 13, and since that value is not zero, the condition of the if statement is always true.

SO… Use only == inside tests/conditions.

Use = outside tests/conditions.

Expression	Value	Comment
3 == 5-2	🟩	== tests for equality.
3 != 5-1	🟩	!= tests for inequality. It is true that 3 is not 5 – 1.
3 = 6 / 2	🟩	Use == to test for equality.
1.0 / 3.0 == 0.333333333	🟩	Although the values are very close to one another, they are not exactly equal. See Common Error 3.3.
"10" > 5	🟩	You cannot compare a string to a number.

Logical Operators

- **Example:** you need to write a program to process temperature values, and tests whether a given temperature corresponds to liquid water or to solid ice.
- At sea level, water freezes at 0 degrees Celsius and boils at 100 degrees Celsius.
- Water is liquid IF the temperature is greater than 0 AND less than 100

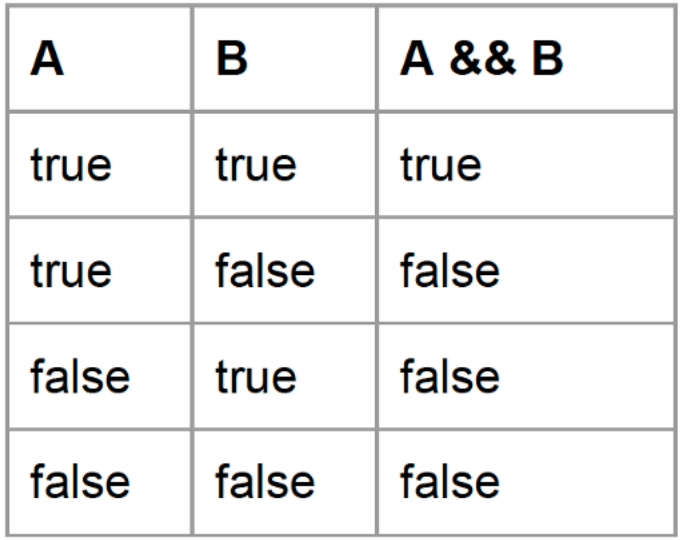
Logical Operators: And &&

- **Example:** you need to write a program to process temperature values, and tests whether a given temperature corresponds to liquid water or to solid ice.
- At sea level, water freezes at 0 degrees Celsius and boils at 100 degrees Celsius.
- Water is liquid IF the temperature is greater than 0 AND less than 100
- In C++, the && operator (called “and”) yields true only when both conditions that it joins are true:

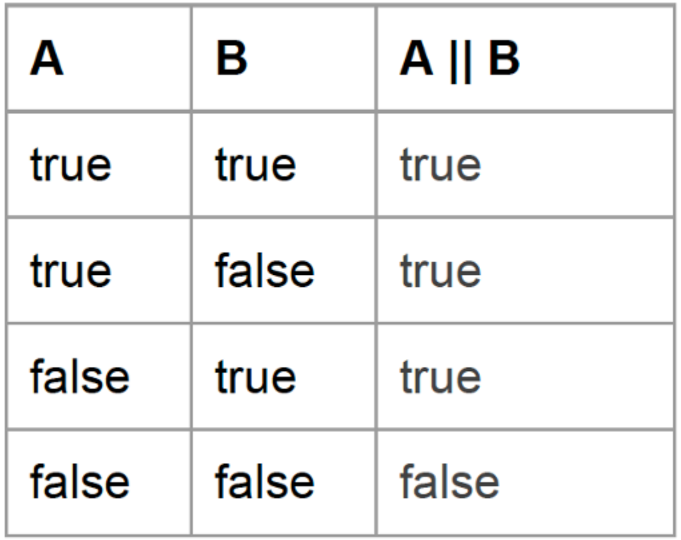
```
if (temp > 0 && temp < 100)
{
      cout << “Liquid” << endl;
}
```

Truth Tables

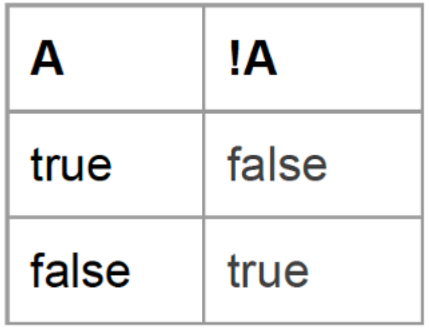
- **Definition:** A truth table displays the value of a Boolean operator expression for all possible combinations of its constituent expressions.
- (You’ll look at truth tables a lot more in CSCI 2824 (Discrete))
- So if A and B denote bool variables or Boolean expressions, we have:



A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false



A	B	A || B
true	true	true
true	false	true
false	true	true
false	false	false



A	!A
true	false
false	true

Logical Operators: And &&

```
if (temp > 0 && temp < 100)
{
      cout << “Liquid” << endl;
}
else
{
      cout < “Not liquid” << endl;
}
```

- If temp is within the 0 to 100 range, then both the left-hand side and right-hand side are true, so the whole expression in parens ( ) has value = true
- In all other cases, the whole expression’s value is false

Logical Operators: Or ||

- The || operator (called or) yields the result true if at least one of the conditions connected by it is true
- Written as two adjacent vertical bar symbols (above the Enter key)

```
if (temp <= 0 || temp >= 100)
{
     cout < “Not liquid” << endl;
}
```

- If either of the left-hand or right-hand side expressions is true, then the whole expression has value true
- **Question:** What is the only case in which “Not liquid” would appear?

Logical Operators: Not !

- Sometimes, you need to invert a condition with the logical not operator: !
- The ! operator takes a single condition and evaluates to true if the condition is false, and to false if the condition is true

```
if (!frozen)
{
      cout < “Not frozen” << endl;
}
```

- “Not frozen” will be written only when frozen contains the value false
- **Question:** What is the value of !false ?

Examples

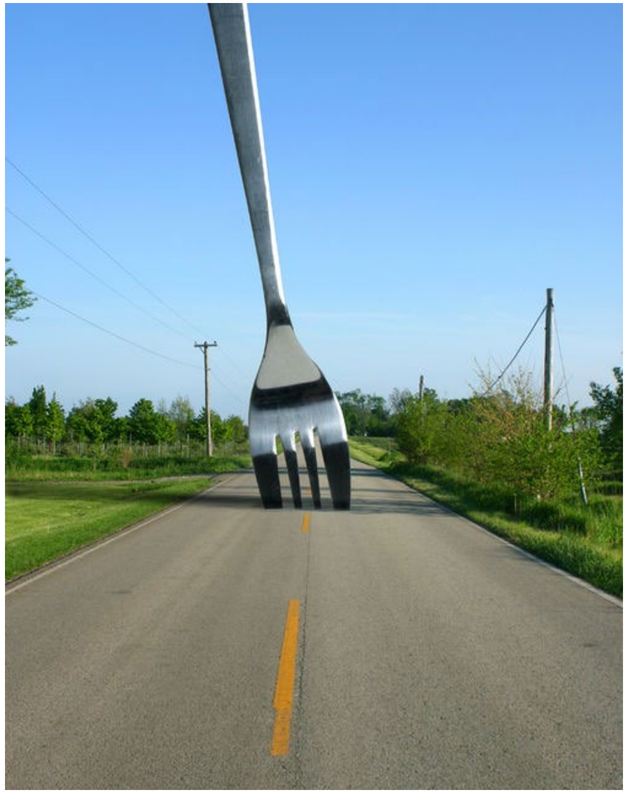
- 0 < 200 && 200 < 100
- 0 < 200 || 200 < 100
- 0 < 200 || 100 < 200
- 0 < 200 < 100
- !(0 < 200)
- -10 && 10 > 0
- 0 < x && x < 100 || x == -1
- (!0 < x && x < 100) || x == -1

The `if` statement

How do you know that class has started?

The `if` Statement

- The **`if`** statement is used to implement a decision
  - When a condition is fulfilled,
    one set of statements is executed
  - Otherwise,
    another set of statements is executed
- Like a fork in the road



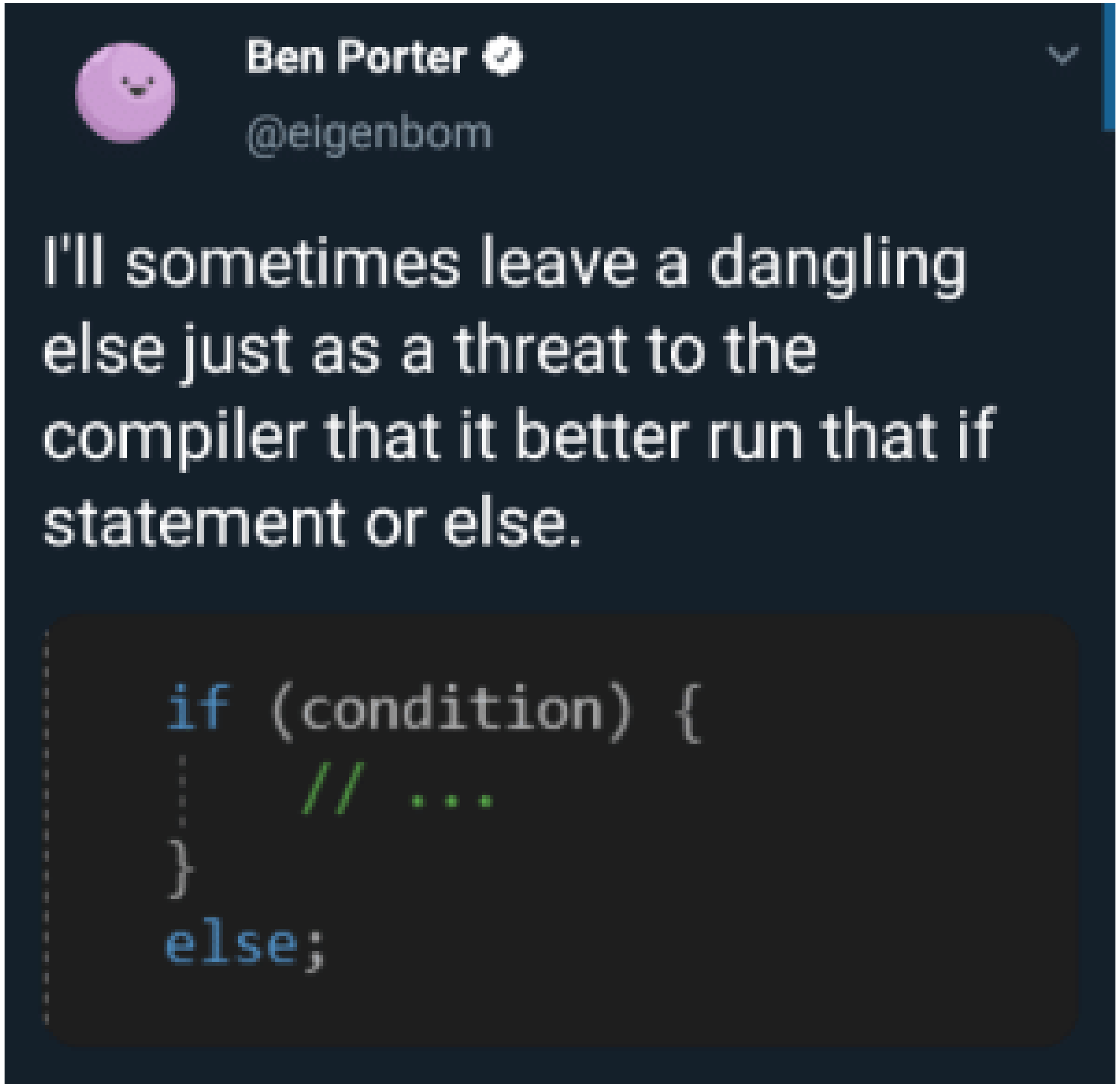
Syntax of the `if()` Statement

```
if (condition)//never put a semicolon after the parentheses!!
{
   statement1; //executed if condition is true
}
else   //the else part is optional
{
   statement2; //executed if condition false
} //braces are optional but recommended
```

Common Error – The Do-nothing Statement

- This is *not* a compiler error.
- The compiler does not complain.
- It interprets this **`if`** statement as follows:
    - If floor is greater than 13, execute the do-nothing statement (semicolon by itself is the do-nothing statement)
    - Then execute the code enclosed in the braces.
- Any statements enclosed in the braces are no longer a part of the if statement.

```
if (floor > 13); // ERROR?
{
   floor--;
}
```



Ben Porter

@eigenbom

I'll sometimes leave a dangling else just as a threat to the compiler that it better run that if statement or else.

```
if (condition) {
    // ...
}
else;
```

```
if (bool){
    System.out.println("TRUE!");
}
```

```
if (bool == false){

} else {
    System.out.println("TRUE!");
}
```

```
if (bool == true && bool != false){
    System.out.println("TRUE!");
}
```

```
while (bool == true && bool != false){
    System.out.println("TRUE!");
    break;
}
```

```
    ifStatement(bool, new Runnable() {
        @Override
        public void run() {
            System.out.println("TRUE!");
        }
    });

static void ifStatement(boolean b, Runnable r) {
    while (b == true && b != false) {
        r.run();
        break;
    }
}
```

The `if` Statement: Elevator Example

We must write the code to control the elevator.

How can we skip the 13th floor?

`if()` Elevator Example Code

- If the user inputs 20, the program must set the actual floor to 19.
- Otherwise, we simply use the supplied floor number.

We need to decrement the input only under a certain condition:

if() Elevator Example Code

```
int floor;
cout << “Enter the desired floor: “;
cin >> floor;
int actual_floor;
if (floor > 13)   //never put a semicolon after the parentheses!!
{
   actual_floor = floor - 1; //
}
else
{
   actual_floor = floor;
}
```

Is the **else** part necessary?

`if()` Elevator Example without `else`

Here is another way to write this code:
We only need to decrement when the floor is greater than 13.

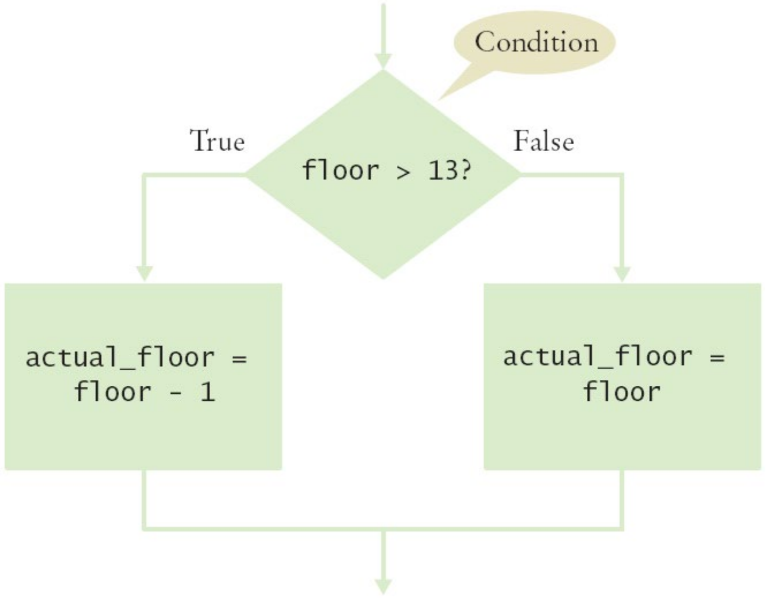
We can set **actual_floor** before testing:

```
int actual_floor = floor;
if (floor > 13)
{
    actual_floor--;
} // No else needed
```

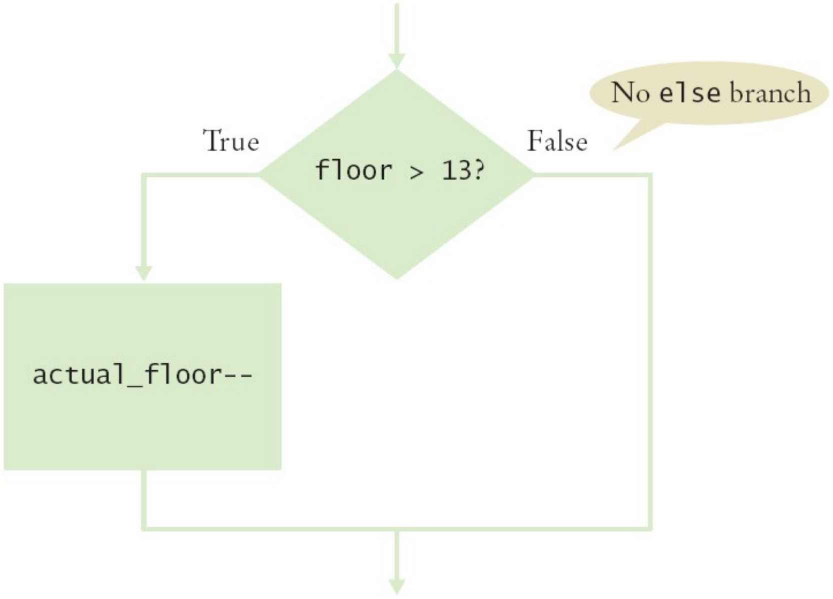
(And you'll notice we used the decrement operator this time.)

Brief C++ by Cay Horstmann

The `if` Statement Flowcharts



With else



Without else

The `if` Statement – A Complete Elevator Program

ch03/elevator1.cpp

```
#include <iostream>
using namespace std;

int main()
{
   int floor;
   cout << "Floor: ";
   cin >> floor;
   int actual_floor;
   if (floor > 13)
   {
      actual_floor = floor - 1;
   }
```

```
else
   {
      actual_floor = floor;
   }

   cout << "The elevator will
 travel to the actual floor "
      << actual_floor << endl;

   return 0;
}
```

The `if` Statement – Always use Braces

- When the body of an **`if`** statement consists of a single statement, you need not use braces:

```
if (floor > 13)
   floor--;
```

- However, it is a good idea to always include the braces:
  - the braces makes your code easier to read, and
  - you are less likely to make errors

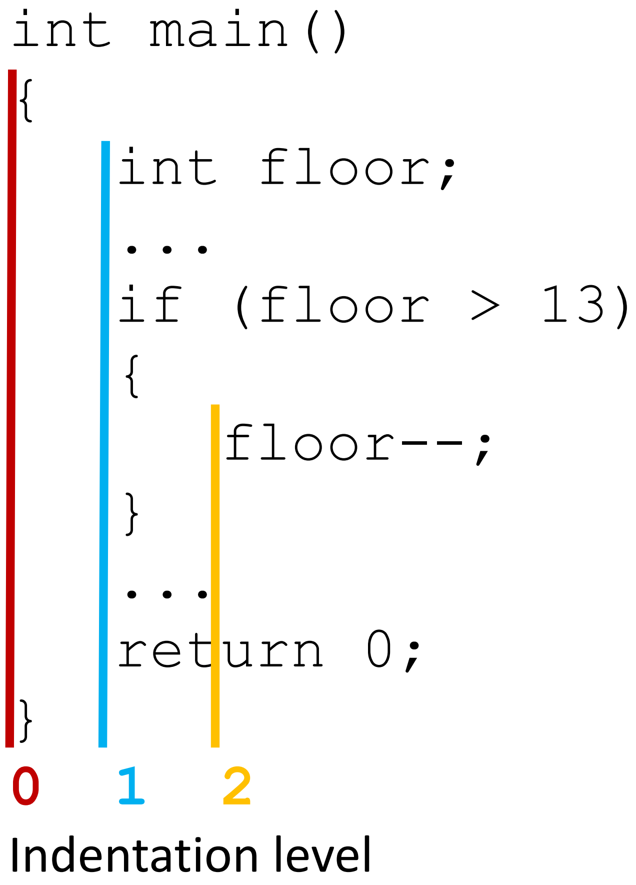
The `if` Statement – Brace Layout

- Making your code easy to read is good practice.
- Lining up braces vertically helps.

```
if (floor > 13)
{
   floor--;
}
```

The `if` Statement – Indent when Nesting

Block-structured code has the property that *nested* statements are indented by one or more levels.



```
int main()
{
    int floor;
    ...
    if (floor > 13)
    {
        floor--;
    }
    ...
    return 0;
}
```

The `if` Statement – Removing Duplication

```
if (floor > 13)
{
     actual_floor = floor - 1;
     cout << "Actual floor: " << actual_floor << endl;
}
else
{
     actual_floor = floor;
     cout << "Actual floor: " << actual_floor << endl;
}
```

- Do you find anything redundant in this code?

The `if` Statement – Removing Duplication

```
if (floor > 13)
{
      actual_floor = floor - 1;
}
else
{
      actual_floor = floor;
}
cout << "Actual floor: " << actual_floor << endl;
```

You can remove the duplication by moving the two identical cout statements outside of and after the braces, and of course deleting one of the two.