



Decisions

Due this week

- **Homework 1**

- Write solutions in VS Code
 - Paste in Autograder, **Homework 1 CodeRunner**.
 - Complete the quiz
- Check the due date! **No late submissions!!**

Homework 1 - CodeRunner

▼ Week 2: Decisions	
Content	
📄	Week 2 Overview
Assessments	
📄	Homework 1
📄	Homework 1 - Coderunner Sep 8 38 pts
🚀	Homework 1 Quiz Sep 8 12 pts
🚀	Recitation 1 Quiz (optional)



Question 1

Not complete

Points out of 2.00

🚩 Flag question

⚙️ Edit question

Write a C++ program to print:

Hello, World!

Answer: (penalty regime: 0 %)

1

Today

- Boolean variables
- Relational 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
>=	\geq	Greater than or equal
<	<	Less than
<=	\leq	Less than or equal
==	=	Equal
!=	\neq	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 is_negative = 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 is_negative = (x < 0);  
bool is_positive = (x > 0);  
cout << is_negative << " " << is_positive << endl;
```

Output: 1 0

Expression	Value	Comment
<code>3 <= 4</code>	true	3 is less than 4; <= tests for “less than or equal”.
<code>3 =< 4</code>	Error	The “less than or equal” operator is <=, not =<. The “less than” symbol comes first.
<code>3 > 4</code>	false	> is the opposite of <=.
<code>4 < 4</code>	false	The left-hand side of < must be strictly smaller than the right-hand side.
<code>4 <= 4</code>	true	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:

```
input == "Quit"
```

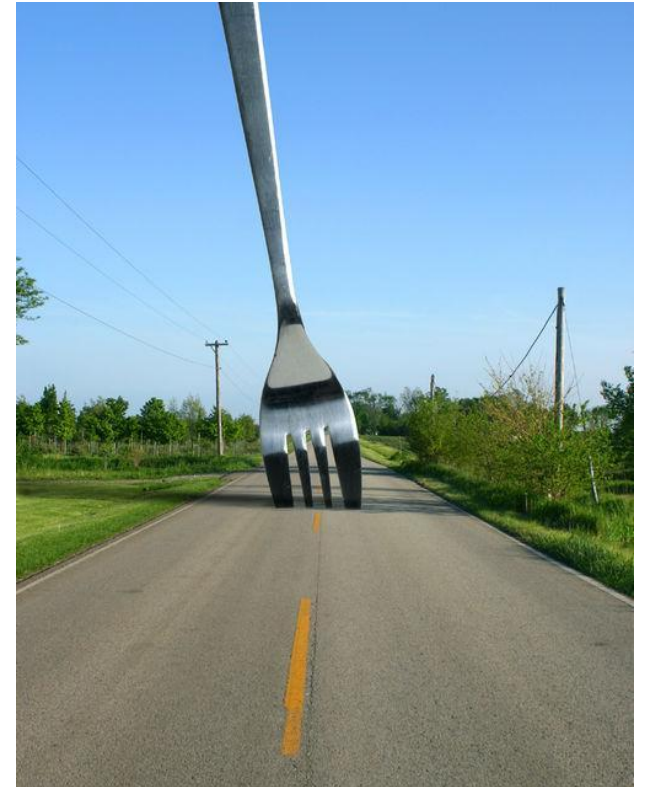
Expression	Value	Comment
<code>3 == 5-2</code>	true	<code>==</code> tests for equality.
<code>3 != 5-1</code>	true	<code>!=</code> tests for inequality. It is true that 3 is not 5 – 1.
<code>3 = 6 / 2</code>	Error	Use <code>==</code> to test for equality.
<code>1.0 / 3.0 == 0.333333333</code>	false	Although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

The `if` statement

How do you know that class has ended?

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

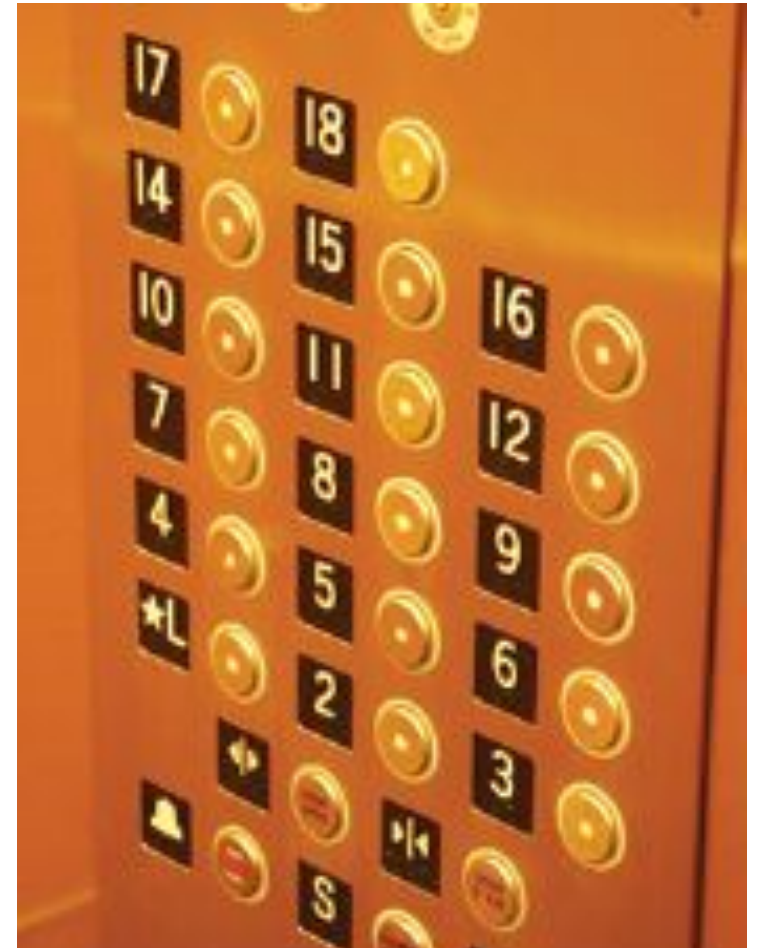
- This is *not* a compiler error.
- The compiler does not complain.
- It interprets this **if** statement as follows:
 - If floor is greater than 13, execute the do-nothing statement (semicolon by itself is the do-nothing statement)
 - Then execute the code enclosed in the braces.
- Any statements enclosed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

```
if (floor > 13); // ERROR?  
{  
    floor = floor - 1;  
}
```

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`

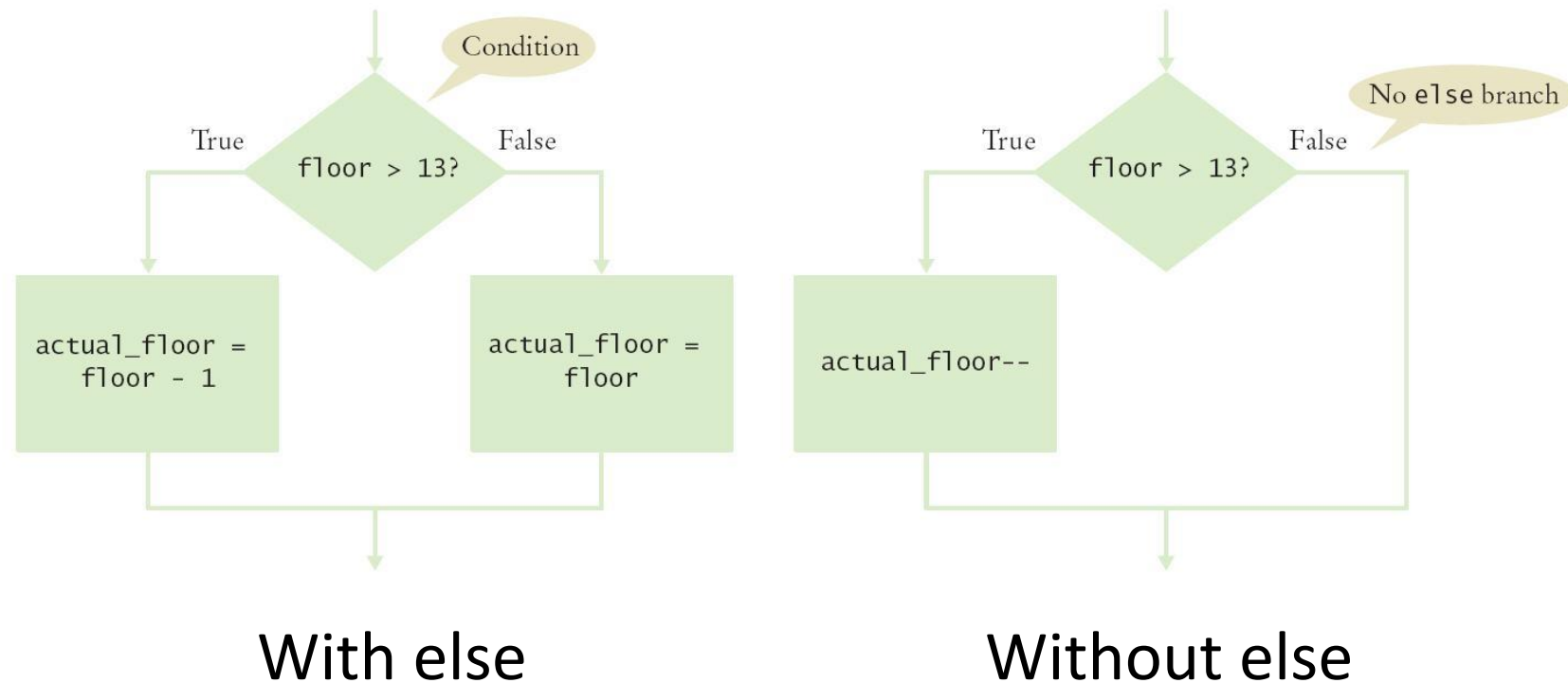
Here is another way to write this code:

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 = floor - 1;
} // No else needed
```

The `if` Statement Flowcharts



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 = floor - 1;
```

- 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--;
}
```