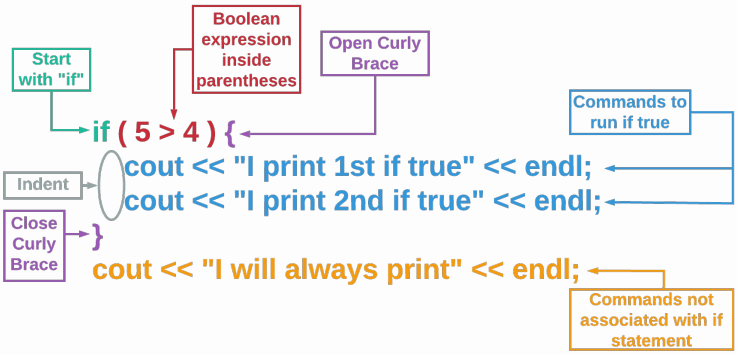
Learning Objectives: If Statement

- **Describe `if` statement syntax**
- **Use a conditional statement to make decisions**
- **Use two or more boolean expressions in an `if` statement**
- **Identify when to use compound conditionals and when not to use them**

If Statement Syntax

If Statement Syntax

Conditionals are pieces of code that make a decision about what the program is going to do next. The most common conditional is the `if` statement.



.guides/img/IfSyntax

If statements in C++ must contain the following items:
* The keyword `if`.
* A boolean expression in parentheses, `()`.
* Curly braces, `{}`, surrounding all lines of code that will run if the boolean expression is true.

It is best practice to also indent the lines of code inside the curly braces to visually differentiate them from the commands that will always run.

Copy the code below into the text editor on the left and then click the `TRY IT` button to see the output. You can also click on the ++Code Visualizer++ link below to see how the program runs behind-the-scenes. If the visualizer does not open properly after a few seconds, click the `Refresh code` code button at the top to restart it.

```
if (5 > 4) {
  cout << "I print 1st if true" << endl;
  cout << "I print 2nd if true" << endl;
}
cout << "I will always print" << endl;
```

Code Visualizer

If Statement

If Statement

`if` statements test to see if a certain condition is true. If yes, then specific commands are run. The simple `if` statement does not do anything if the boolean expression is false.

```
if (7 != 10) {
  cout << "The above statement is true" << endl;
  cout << "The above statement is still true" << endl;
}
cout << "This is not related to the if statement" << endl;
```

Code Visualizer

challenge

What happens if you:

- Change `!=` in the code above to `==`?
- Change `7 != 10` in the code above to `true`?
- Change `7 != 10` in the code above to `false`?
- Remove the curly braces `{}` with the condition set to `if (false)`?

Code Visualizer

Testing Multiple Cases

You will find yourself needing to test the same variable multiple times. Be sure that you set up your conditionals to test *all* possible values of the variable.

```cpp
int grade = 90;

if (grade > 70) {
  cout << "Congrats, you passed the class" << endl;
}

if (grade < 70) {
  cout << "Condolences, you did not pass the class" << endl;
}
```

Code Visualizer

challenge

What happens if you:

- Assign `int grade` to `60`?
- Assign `int grade` to `70`?
- Change `grade > 70` in the code above to `grade >= 70`?

Code Visualizer

Compound Conditional Statements

Compound Conditional Statements

Conditional statements (`if` statements) are used to match an action with a condition being true. For example, print `Even` if a number is even. If you want to test for a number being even *and* greater than 10, you will need *two* conditionals.

```
int num = 16;

if (num % 2 == 0 && num > 10) {
  cout << "Even and greater than 10" << endl;
}
```

challenge

What happens if you:

- Assign `num` to `8`?
- Change `&&` in the code above to `||`?
- Change `==` in the code above to `!=`?

Why Use Compound Conditionals?

Both code snippets below do the same thing: Ask if `my_var` is greater than 15 and if `my_var` is less than 20. If both of these are true, then C++ will print the value of `my_var`.

```
int my_var = 19;

if (my_var > 15) {
    if (my_var < 20) {
        cout << my_var << endl;
    }
}
```

```
int my_var = 19;

if (my_var > 15 && my_var < 20) {
    cout << my_var << endl;
}
```

.guides/img/CompoundConditional

The code on the left is a **nested** `if` statement - which means an `if` statement is *inside* another `if` statement.

The code with the **compound conditional** (on the right) has fewer lines of code, and is easier for a human to read. In fact, it almost reads like a sentence.