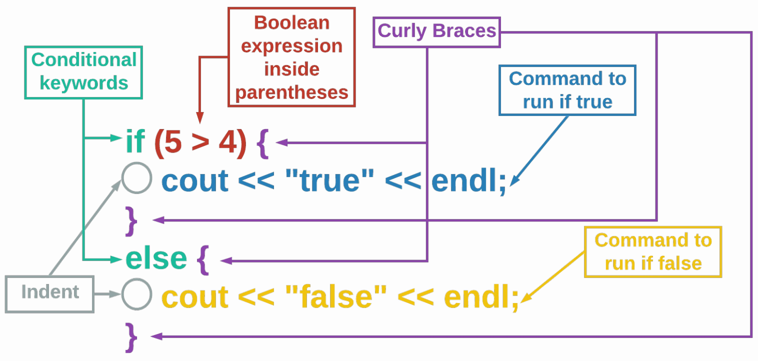
Learning Objectives: If Else Statement

- **Describe `if-else` statement syntax**
- **Explain the difference between an `if` statement and an `if-else` statement**

If Else Statement Syntax

If Else Syntax

The `if-else` statement checks to see if a condition is true, and then has specific actions that take place. However, it also provides a specific set of actions if the boolean expression is false. Use the `else` keyword to introduce the code to run when false is evaluated. Note that `else` is aligned with the `if` keyword (no indentation) and has its own set of curly braces `{}`. You do *not* write another boolean expression with `else`.



.guides/img/IfElseSyntax

It is best practice to indent the lines of code within the curly braces to differentiate them but the indention does not affect how the program runs.

```
if (5 > 4) {
  cout << "Print me if true" << endl;
}
else {
  cout << "Print me if false" << endl;
}
```

Code Visualizer

challenge

What happens if you:

- Change `4` in the code above to `6`? → Print me if false
- Remove all the curly braces `{}`?
- Add `cout << "False" << endl;` under `cout << "Print me if false" << endl;` *without* any curly braces `{}` in the code?
- Add `cout << "True" << endl;` under `cout << "Print me if true" << endl;` *without* any curly braces `{}` in the code?

Code Visualizer

important

IMPORTANT

You may have noticed that when there is only **one** command associated with an `if` or `else` statement the *curly braces* `{}` become **optional**.

When Are Curly Braces Mandatory?

Curly braces `{}` are mandatory whenever you have **more than one** command that is associated with an `if` or `else` statement. Here is a code snippet that will work *without* curly braces:

```
if (10 % 2 == 0)
  cout << "10 is even" << endl;
else
  cout << "10 is odd" << endl;
```

However, if you add more commands to the `if` or `else` statement, the program will not run properly. The examples below will not print as intended or will produce an error message.

```
if (10 % 2 == 0)
  cout << "10 is even" << endl;
else
  cout << "10 is odd" << endl;
```

```
if (10 % 2 == 0)
  cout << "10 is even"
  cout << "True" << end
else
```

Like indentations, it is best practice to always include curly braces even if they are optional in certain situations.

```
if (10 % 2 == 0) { // mandatory curly braces
  cout << "10 is even" << endl;
  cout << "True" << endl;
}
else { // optional curly braces
  cout << "10 is odd" << endl;
}
```

If Else Statement

If Else Statement

The `if-else` statement is used when you want something *specific* to happen if the boolean expression is true and something *else* to happen if it is false.

```
bool my_bool = true;

if (my_bool) {
  cout << "The value of my_bool is true" << endl; }
else {
  cout << "The value of my_bool is false" << endl; }
```

challenge

What happens if you:

- Assign `my_bool` to `false`?
- Assign `my_bool` to `! true && ! false`?

important

IMPORTANT

Did you notice that the code above has the closing curly brace `}` after the the semi-colon `;` instead of on the next line? Remember that curly braces `{}` are *optional* if the `if-else` statement only includes **one** command within the `if` and `else` bodies. However, they are *mandatory* when there is **more than one** command. When using curly braces, the decision of where to place them is entirely up to you. All of the commands below work exactly the same way:

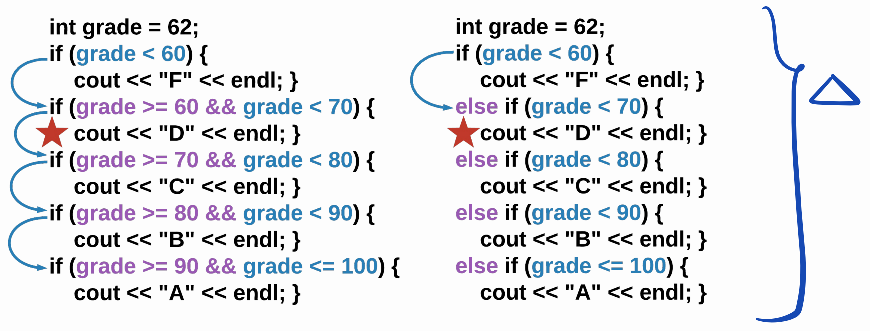
```
if (is_true) {
  cout << "1" << endl;
  cout << "2" << endl;
}
```

```
if (is_true) {
  cout << "1" << endl; cout << "2" << endl; }
```

```
if (is_true) { cout << "1" << endl; cout << "2" << endl; }
```

Testing Multiple Cases

You will find yourself needing to test the same variable multiple times. To simplify this, you can **nest** `if-else` statements – which means you can put an `if-else` structure inside of another `if-else` structure (as shown on the right below).



```
int grade = 62;
if (grade < 60) {
   cout << "F" << endl; }
if (grade >= 60 && grade < 70) {
   cout << "D" << endl; }
if (grade >= 70 && grade < 80) {
   cout << "C" << endl; }
if (grade >= 80 && grade < 90) {
   cout << "B" << endl; }
if (grade >= 90 && grade <= 100) {
   cout << "A" << endl; }
```

```
int grade = 62;
if (grade < 60) {
   cout << "F" << endl; }
else if (grade < 70) {
   cout << "D" << endl; }
else if (grade < 80) {
   cout << "C" << endl; }
else if (grade < 90) {
   cout << "B" << endl; }
else if (grade <= 100) {
   cout << "A" << endl; }
```

.guides/img/NestedElseIf

When nesting `if` and `else` **together**, use the keywords `else` and `if` side-by-side (`else if`). This nesting drastically changes the way the program flows once the correct case is found. On the left, the program checks *every* case no matter the value of the variable. On the right, the **nested** structure causes the program to jump out of the structure once the correct case is found. This is able to occur because the other `if` cases are inside the `else` statement, which will only run when the previous boolean expression is false.

```
int grade = 62;
if (grade < 60) {
  cout << "F" << endl; }
else if (grade < 70) {
  cout << "D" << endl; }
else if (grade < 80) {
  cout << "C" << endl; }
else if (grade < 90) {
  cout << "B" << endl; }
else if (grade <= 100) {
  cout << "A" << endl; }
```

Code Visualizer