Contents

[Introduction to Conditionals & Logic 1](#_Toc56960961)

[If Statement 1](#_Toc56960962)

[Relational Operators 2](#_Toc56960963)

[Else Clause 2](#_Toc56960964)

[Else If 3](#_Toc56960965)

[Switch Statement 4](#_Toc56960966)

[Here are some of the major concepts: 5](#_Toc56960967)

[Introduction to Logical Operators 5](#_Toc56960968)

[Logical Operator: && 6](#_Toc56960969)

[Logical Operator: II 6](#_Toc56960970)

[Logical Operator: ! 7](#_Toc56960971)

## Introduction to Conditionals & Logic

Every program we’ve seen so far has only had one possible path of execution — they all execute line by line, from top to bottom. And every time you run one of those programs, it gives you the same exact result. But it’s the twenty-first century, and we like options!

In this lesson, we will explore how programs make decisions by evaluating conditions and introduce logic into our code!

We’ll be covering the following concepts:

* if, else if, and else statements
* switch statements
* Relational operators
* Logical operators

So… *if* you’ve already learned these concepts in another language, go to the next lesson — *else*, prepare yourself and let’s get started!

## If Statement

An if statement is used to test an expression for truth and execute some code based on it. Here’s a simple form of the if statement:

if (condition) {

some code

}

If the *condition* is true, then the *statements* within are executed. Otherwise, the *statements* are skipped and the program continues on.

if (flip == 1) {  
   
  std::cout << "Heads\n";  
   
}

The if keyword is followed by a set of parentheses (). Inside the parentheses (), a condition is provided that evaluates to true or false:

* If the condition evaluates to true, the code inside the curly braces {} executes.
* If the condition evaluates to false, the code won’t execute.

So in the code above, if flip is equal to 1, the program outputs “Heads”; if it does not, then nothing happens and the program continues.

## Relational Operators

When writing conditional statements, sometimes we need to use different types of operators to compare values. These operators are called *relational operators*.

To have a condition, we need relational operators:

* == equal to
* != not equal to
* > greater than
* < less than
* >= greater than or equal to
* <= less than or equal to

Relational operators compare the value on the left with the value on the right.

## Else Clause

We can also add an else clause to an if statement to provide code that will only be executed if the condition is false. Here’s a form of an if statement that includes an else clause:

if (condition) {

do something

} else {

do something else

}

* If *condition* is true, *statement1* is executed. Then the program *skips* *statement2* and executes any code statements following the if/else clause.
* If *condition* is false, *statement1* is *skipped* and *statement2* is executed. After *statement2* completes, the program executes any code statements following the if/else clause.

if (coin == 1) {  
   
  std::cout << "Heads\n";  
   
}  
else {  
   
  std::cout << "Tails\n";  
   
}

So in the code above, if coin is equal to 1, the program outputs “Heads”; if it does not, then it outputs “Tails”.

**Note:** It’s either or — only one of them will execute!

## Else If

So, what happens if you want more than two possible outcomes?

This is where else if comes in!

if (condition) {

some code

} else if (condition) {

some code

} else {

some code

}

The else if statement always comes after the if statement and before the else statement. The else if statement also takes a condition.

And you can have more than one of them! Here’s an example with three of them:

if (grade == 9) {  
   
  std::cout << "Freshman\n";  
   
}   
else if (grade == 10) {  
   
  std::cout << "Sophomore\n";  
   
}  
else if (grade == 11) {  
   
  std::cout << "Junior\n";  
   
}   
else if (grade == 12) {  
   
  std::cout << "Senior\n";  
   
}  
else {  
   
  std::cout << "Super Senior\n";  
   
}

## Switch Statement

Now that we know how if, else if, else work, we can write programs that have multiple outcomes. Programs with multiple outcomes are so common that C++ provides a special statement for it… the switch statement!

A switch statement provides an alternative syntax that is easier to read and write. However, you are going to find these less frequently than if, else if, else in the wild.

A switch statement looks like this:

switch (grade) {  
   
  case 9:  
    std::cout << "Freshman\n";  
    break;  
  case 10:  
    std::cout << "Sophomore\n";  
    break;  
  case 11:  
    std::cout << "Junior\n";  
    break;  
  case 12:  
    std::cout << "Senior\n";  
    break;  
  default:  
    std::cout << "Invalid\n";  
    break;  
   
}

* The switch keyword initiates the statement and is followed by (), which contains the value that each case will compare. In the example, the value or expression of the switch statement is grade. One restriction on this expression is that it must evaluate to an integral type (int, char, short, long, long long, or enum).
* Inside the block, {}, there are multiple cases.
* The case keyword checks if the expression matches the specified value that comes after it. The value following the first case is 9. If the value of grade is equal to 9, then the code that follows the : would run.
* The break keyword tells the computer to exit the block and not execute any more code or check any other cases inside the code block.
* At the end of each switch statement, there is a default statement. If none of the cases are true, then the code in the default statement will run. It’s essentially the else part.

In the code above, suppose grade is equal to 10, then the output would be “Sophomore”.

**Note:** Without the break keyword at the end of each case, the program would execute the code for the first matching case and *all* subsequent cases, including the default code. This behavior is different from if / else conditional statements which execute only one block of code.

## Here are some of the major concepts:

* An if statement checks a condition and will execute a task if that condition evaluates to true.
* if / else statements make binary decisions and execute different code blocks based on a provided condition.
* We can add more conditions using else if statements.
* Relational operators, including <, >, <=, >=, ==, and != can compare two values.
* A switch statement can be used to simplify the process of writing multiple else if statements. The break keyword stops the remaining cases from being checked and executed in a switch statement.

## Introduction to Logical Operators

Often, when we are trying to create a control flow for our program, we’ll encounter situations where the logic cannot be satisfied with a single condition.

*Logical operators* are used to combine two or more conditions. They allow programs to make more flexible decisions. The result of the operation of a logical operator is a bool value of either true or false.

There are three logical operators that we will cover:

* &&: the and logical operator
* ||: the or logical operator
* !: the not logical operator

We will also discuss the order of operations.

Let’s get started!

## Logical Operator: &&

The and logical operator is denoted by &&.

It returns true if the condition on the left *and* the condition on the right are both true. Otherwise, it returns false.

Here’s the truth table:

| **a** | **b** | **a && b** |
| --- | --- | --- |
| false | false | false |
| false | true | false |
| true | false | false |
| true | true | true |

For instance:

* ( 1 < 2 && 2 < 3 ) returns true
* ( 1 < 2 && 2 > 3 ) returns false

**Note:** The keyword and can also be used in the place of &&.

## Logical Operator: II

The or logical operator is denoted by ||.

It returns true when the condition on the left is true *or* the condition on the right is true. Only one of them needs to be true.

Here’s the truth table:

| **a** | **b** | **a || b** |
| --- | --- | --- |
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |

For instance:

* ( 1 < 2 || 2 > 3 ) returns true
* ( 1 > 2 || 2 > 3 ) returns false

**Note:** The keyword or can be used in the place of ||.

## Logical Operator: !

The not logical operator is denoted by !.

It reverses the bool outcome of the expression that immediately follows.

Here’s the truth table:

| **a** | **!a** |
| --- | --- |
| false | true |
| true | false |

For instance:

* ( !true ) returns false
* ( !false ) returns true
* ( !(10 < 11) ) returns false

**Note:** The keyword not can be used in the place of !.

 In this mini-lesson, we’ve added more operators to our toolbox:

* &&: the and logical operator
* ||: the or logical operator
* !: the not logical operator