

Lab: If Statement

Tutorial Lab 1: If Statements

The `if` statement allows for your program to make a decision about what it should do. It asks a simple question: “Is this condition true?” If yes, then the computer will execute certain commands.

```
int x = 5;

if (x < 10) {
    cout << "Less than 10" << endl;
}
```

Code Visualizer

An `if` statement is comprised of the keyword `if`, followed by a boolean expression surrounded by parentheses `()`. Any code that should run if the boolean expression is true is surrounded by curly braces `{}`. It is best practice to indent this code, but it does not affect how the code runs.

If the boolean expression is false, the code in curly braces is skipped, and the program continues as normal.

```
int x = 20;

if (x < 10) {
    cout << "Less than 10" << endl;
}

cout << "And the program continues...";
```

Code Visualizer

`if` statements can be used to test multiple conditions. These conditions exist as boolean expressions inside the parentheses `()`. In addition, an `if` statement can exist *inside* another `if` statement. Both blocks of code below accomplish the same exact task.

```
int age = 15;

if (age < 20) {
    if (age > 10) {
        cout << "Teenager";
    }
}
```

```
int age = 15;

if ((age < 20) && (age > 10)) {
    cout << "Teenager";
}
```

Code Visualizer

Lab: If Else Statement

Tutorial Lab 2: If Else Statements

The `if else` statement gives your program the ability to ask a question, perform certain actions if the answer is true, and then perform another set of actions if the answer is false.

```
int x = 10;

if (x > 50) {
    cout << to_string(x) + " is greater than 50" << endl;
}
else {
    cout << to_string(x) + " is less than 50" << endl;
}
```

Code Visualizer

The `if` part of the `if else` statement is written as before. The `else` keyword is **not** indented; it should be aligned with the `if` keyword. `else` is followed by an open curly brace `{`. You do not use a boolean expression with `else`. All code that should run when the boolean expression is false should go before the closing curly brace `}`.

Be careful when expressing your boolean expression in terms of “less than or greater than”. This does not take into account when the values being compared are equal. Consider the code from above, but with `x` having the value of 50.

```
int x = 50;

if (x > 50) {
    cout << to_string(x) + " is greater than 50" << endl;
}
else {
    cout << to_string(x) + " is less than 50" << endl;
}
```

Code Visualizer

The output of the program does not make sense. 50 is not less than 50. Sometimes using `<=` and `>=` need to be used. Another solution is to update the output to be more inclusive such as replacing `is less than 50` with `is less than or equal to 50`. In either case, be sure to think through all of the possible outcomes, and make sure your code can function properly in all of those scenarios.

Lab: Switch Statement

Tutorial Lab 3: Switch Statement

The `switch` case statement gives your program the ability to perform different actions based on the *value* of a given variable.

```
int size = 3;

switch (size) {
    case 1: cout << "Short"; break;
    case 2: cout << "Tall"; break;
    case 3: cout << "Grande"; break;
    case 4: cout << "Venti"; break;
    case 5: cout << "Trenta"; break;
    default: cout << "Grande";
}
```

Code Visualizer


The variable used to make the decision is in parentheses following the `switch` keyword. Inside curly braces, the cases listing the different values to check are followed by a `:` and then the code that should run if the variable is equal to that case's value comes next. The last case, `default`, runs if none of the other cases are true. Each code segment except the last one ends with `break;` to signal the program to jump to the closing curly brace.

Remember to include `break;` statements at the end of each case. Check out what happens when you remove them.

```
int size = 3;

switch (size) {
    case 1: cout << "Short";
    case 2: cout << "Tall";
    case 3: cout << "Grande";
    case 4: cout << "Venti";
    case 5: cout << "Trenta";
    default: cout << "Grande";
}
```

Code Visualizer

The output of the program does not make sense because the program continues through all of the cases after the initial case is matched to a value. In the example above, the program prints the command for case 3 as well as all of the commands that follow. 

Lab Challenge: Month of the Year

Conditionals Challenge

Write a program that determines the month of the year based on the value of a variable called `month`. The variable will be a number from 1 to 12 (1 is January, 2 is February, etc.). Use a `cout <<` statement to write the month to the screen.

Click to compile your code

Test your code with a few different values