Understanding Conditional Statements in C: `if`, `if-else`, and `if-else if-else`

Abdo Wael

August 11, 2025

1 Introduction

Conditional statements in programming allow us to make decisions and execute different parts of code depending on certain conditions. In C, the three most common forms are:

- `if`
- `if-else`
- `if-else if-else`

These statements are fundamental for controlling the flow of a program and are used extensively in problem-solving.

2 Syntax and Basic Idea

2.1 `if` Statement

The `if` statement executes a block of code only if a specified condition evaluates to `true`.

```
// Syntax
if (condition) {
    // code to execute if condition is true
}
```

2.2 `if-else` Statement

The `if-else` statement executes one block of code if the condition is true, and another block if the condition is false.

```
// Syntax
if (condition) {
    // code if condition is true
} else {
    // code if condition is false
}
```

2.3 `if-else if-else` Statement

This allows multiple conditions to be checked sequentially. The first condition that evaluates to `true` triggers its block, and the rest are skipped.

```
// Syntax
if (condition1) {
    // code if condition1 is true
} else if (condition2) {
    // code if condition2 is true
} else {
    // code if none of the above are true
}
```

3 Example: Number to Word Conversion

Consider the following problem:

> Given a positive integer `n`, print the lowercase English word corresponding to numbers 1 through 9. For numbers greater than 9, print "Greater than 9".

This can be solved using `if-else if-else` chains as follows:

```
if (n == 1) {
    printf("one");
} else if (n == 2) {
    printf("two");
}
// ... other else if cases for 3 to 9
else {
    printf("Greater than 9");
}
```

This straightforward use of conditional statements allows the program to select the correct word or print the fallback message.

4 How This Idea Extends to More Complex Problems

Conditionals are the building blocks for decision-making in programs. As problems become more complex, conditionals are often combined with loops, functions, and data structures to:

- Handle multiple cases dynamically (e.g., switch statements, arrays for mapping).
- Validate input data and ensure correctness.
- Implement algorithms that require branching logic (sorting, searching, parsing).
- Control state changes in larger systems (e.g., state machines).

For instance, in a more advanced problem, instead of using multiple `if-else` statements, you might use an array of strings indexed by the number, or even hash maps, to simplify and optimize the lookup process.

5 Summary

- `if` executes code only when a condition is true.
- `if-else` chooses between two paths.
- `if-else if-else` chains allow multiple conditions to be checked sequentially.
- These constructs allow programs to make decisions and handle different inputs dynamically.
- Mastery of conditionals is essential for tackling both simple and complex programming problems.