If statements in C

By Alex Allain

The ability to control the flow of your program, letting it make decisions on what code to execute, is valuable to the programmer. The if statement allows you to control if a program enters a section of code or not based on whether a given condition is true or false. One of the important functions of the if statement is that it allows the program to select an action based upon the user's input. For example, by using an if statement to check a user-entered password, your program can decide whether a user is allowed access to the program.

Without a conditional statement such as the if statement, programs would run almost the exact same way every time, always following the same sequence of function calls. If statements allow the flow of the program to be changed, which leads to more interesting code.

Before discussing the actual structure of the if statement, let us examine the meaning of TRUE and FALSE in computer terminology. A true statement is one that evaluates to a nonzero number. A false statement evaluates to zero. When you perform comparison with the relational operators, the operator will return 1 if the comparison is true, or 0 if the comparison is false. For example, the check 0 == 2 evaluates to 0. The check 2 == 2 evaluates to a 1. If this confuses you, try to use a printf statement to output the result of those various comparisons (for example printf ("%d", 2 == 1);)

When programming, the aim of the program will often require the checking of one value stored by a variable against another value to determine whether one is larger, smaller, or equal to the other.

There are a number of operators that allow these checks.

Here are the relational operators, as they are known, along with examples:

```
1  >     greater than              5 > 4 i
2  <     less than                 4 < 5 i
3  >=    greater than or equal     4 >= 4
4  <=    less than or equal        3 <= 4
5  ==    equal to                  5 == 5
6  !=    not equal to              5 != 4
```

It is highly probable that you have seen these before, probably with slightly different symbols. They should not present any hindrance to understanding. Now that you understand TRUE and FALSE well as the comparison operators, let us look at the actual structure of if statements.

Basic If Syntax

The structure of an if statement is as follows:

```
1  if ( statement is TRUE )
2      Execute this line of code
```

Here is a simple example that shows the syntax:

```
1  if ( 5 < 10 )
2      printf( "Five is now less than ten,
```

Here, we're just evaluating the statement, "is five less than ten", to see if it is true or not; with any luck, it is! If you want, you can write your own full program including stdio.h and put this in the main function and run it to test.

To have more than one statement execute after an if statement that evaluates to true, use braces, like we did with the body of the main function. Anything inside braces is called a compound statement, or a block. When using if statements, the code that depends on the if statement is called the "body" of the if statement.

For example:

```
if ( TRUE ) {
    /* between the braces is the body c
    Execute all statements inside the b
}
```

I recommend always putting braces following if statements. If you do this, you never have to remember to put them in when you want more than one statement to be executed, and you make the body of the if statement more visually clear.

Else

Sometimes when the condition in an if statement evaluates to false, it would be nice to execute some code instead of the code executed when the statement evaluates to true. The "else" statement effectively says that whatever code after it (whether a single line or code between brackets) is executed if the if statement is FALSE.

It can look like this:

```
if ( TRUE ) {
    /* Execute these statements if TRUE
}
else {
    /* Execute these statements if FALS
}
```

Else if

Another use of else is when there are multiple conditional statements that may all evaluate to true, yet you want only one if statement's body to execute. You can use an "else if" statement following an if statement and its body; that way, if the first statement is true, the "else if" will be ignored, but if the if statement is false, it will then check the condition for the else if statement. If the if statement was true the else statement will not be checked. It is possible to use numerous else if statements to ensure that only one block of code is executed.

Let's look at a simple program for you to try out on your own.

```
#include <stdio.h>

int main()
{
    int age;

    printf( "Please enter your age: "
    scanf( "%d", &age );
    if ( age < 100 ) {
        printf ("You are pretty young!
    }
    else if ( age == 100 ) {
        printf( "You are old\n" );
    }
    else {
        printf( "You are really old\n'
    }
  return 0;

}
```

More interesting conditions using boolean operators

Boolean operators allow you to create more complex conditional statements. For example, if you wish to check if a variable is both greater than five and less than ten, you could use the Boolean AND to ensure both var > 5 and var < 10 are true. In the following discussion of Boolean operators, I will capitalize the Boolean operators in order to distinguish them from normal English. The actual C operators of equivalent function will be described further along into the tutorial - the C symbols are not: OR, AND, NOT, although they are of equivalent function.

When using if statements, you will often wish to check multiple different conditions. You must understand the Boolean operators OR, NOT, and AND. The boolean operators function in a similar way to the comparison operators: each returns 0 if evaluates to FALSE or 1 if it evaluates to TRUE.

NOT: The NOT operator accepts one input. If that input is TRUE, it returns FALSE, and if that input is FALSE, it returns TRUE. For example, NOT (1) evaluates to 0, and NOT (0) evaluates to 1. NOT (any number but zero) evaluates to 0. In C NOT is written as !. NOT is evaluated prior to both AND and OR.

AND: This is another important command. AND returns TRUE if both inputs are TRUE (if 'this' AND 'that' are true). (1) AND (0) would evaluate to zero because one of the inputs is false (both must be TRUE for it to evaluate to TRUE). (1) AND (1) evaluates to 1. (any number but 0) AND (0) evaluates to 0. The AND operator is written && in C. Do not be confused by thinking it checks equality between numbers: it does not. Keep in mind that the AND operator is evaluated before the OR operator.

OR: Very useful is the OR statement! If either (or both) of the two values it checks are TRUE then it returns TRUE. For example, (1) OR (0) evaluates to 1. (0) OR (0) evaluates to 0. The OR is written as || in C. Those are the pipe characters. On your keyboard, they may look like a stretched colon. On my computer the pipe shares its key with \. Keep in mind that OR will be evaluated after AND.

It is possible to combine several Boolean operators in a single statement; often you will find doing so to be of great value when creating complex expressions for if statements. What is !(1 && 0)? Of course, it would be TRUE. It is true is because 1 && 0 evaluates to 0 and !0 evaluates to TRUE (i.e., 1).

Try some of these - they're not too hard. If you have questions about them, feel free to stop by our forums.

```
1  A. !( 1 || 0 )          ANSWER: 0
2  B. !( 1 || 1 && 0 )     ANSWER: 0 (AND i
3  C. !( ( 1 || 0 ) && 0 )  ANSWER: 1 (Par
```

If you find you enjoyed this section, then you might want to look more at Boolean Algebra.

Quiz yourself
Previous: The Basics of C
Next: Loops
Back to C Tutorial Index