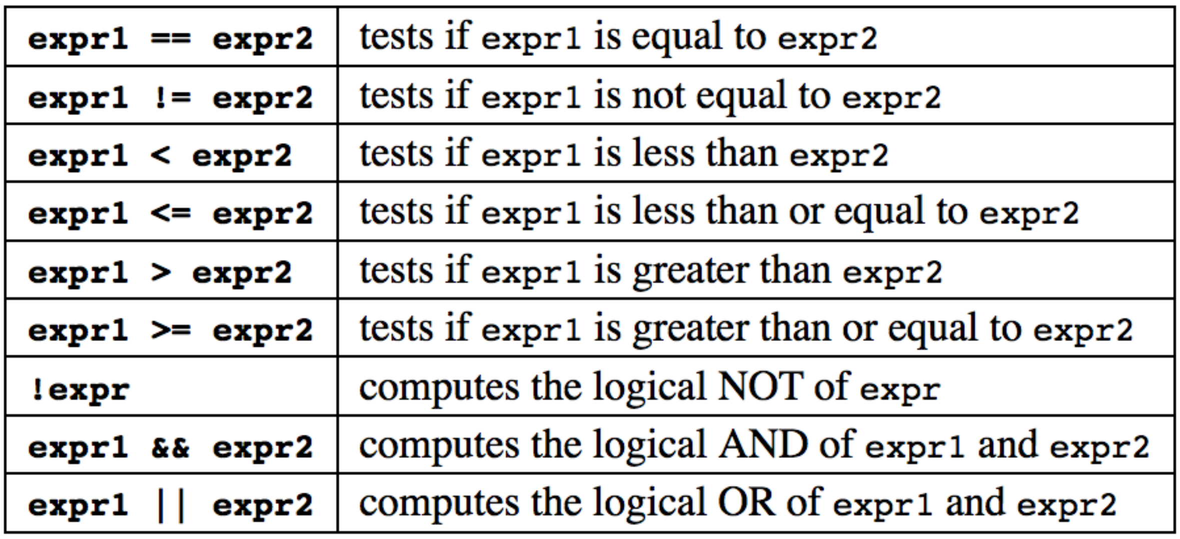
Conditional Statements

In addition to computing arithmetic combinations of their variables, programs often make decisions based on the values of their variables—executing different statements based on the value of expressions. In C, an if/else statement specifies that one block of code should be executed if a condition is true, and another block should be executed if that condition is false.

To write meaningful if/else statements, we need to introduce operators which allow us to compare two expressions and produce a Boolean outcome. In C, however, there are no distinct values for true or false, instead, false is 0, and anything which is non-zero is true. We will refer to true and false because they make more sense conceptually; the distinction should not make a practical difference in most cases.



`expr1 == expr2`	tests if `expr1` is equal to `expr2`
`expr1 != expr2`	tests if `expr1` is not equal to `expr2`
`expr1 < expr2`	tests if `expr1` is less than `expr2`
`expr1 <= expr2`	tests if `expr1` is less than or equal to `expr2`
`expr1 > expr2`	tests if `expr1` is greater than `expr2`
`expr1 >= expr2`	tests if `expr1` is greater than or equal to `expr2`
`!expr`	computes the logical NOT of `expr`
`expr1 && expr2`	computes the logical AND of `expr1` and `expr2`
`expr1 || expr2`	computes the logical OR of `expr1` and `expr2`

The table above shows the C operators for conditional expressions. The first six (==, !=, <, <=, >, and >=) are relational operators—they compare two expressions for equality or inequality. For any of these operators, both operands (the expressions on the left and right) are evaluated to a value, then compared appropriately. The operator then produces a true or false value.

The last three operators in the table (!, &&, and ||) are boolean operators—they operate on true/false values. The first of these, ! performs the boolean NOT operation. It is a unary operator—meaning that is has one operand—which evaluates to true if its operand is false, and evaluates to false if its operand is true.

The && and || operators perform the logical AND and logical OR operations respectively. The logical AND of two values is true if and only if both values are true, otherwise it is false. The logical OR of two values is true if and only if either of the values are true, otherwise it is false.

Unlike previous operators that we have seen, && and || may know their answer from only one argument. In the case of &&, if either operand is false, then the result is false, regardless of the other value. Similarly for ||, if either operand is true, then the result is true regardless of the other value. C exploits this fact in the way that it evaluates && and || by making them short circuit—they may only evaluate one operand. Specifically, the first operand is always evaluated to a value; however, if the value of that operand determines the result of the entire && or ||—false for && or true for ||—then the second operand is not evaluated at all.