Common Confusion 1
& vs. &&
Matthew Woodring

The distinction between the '&' and '&&' operator is typically one of the first confusions students run into. While the two operators are quite similar, there are certain instances where it is preferred to use one over the other. It should also be noted that all of this applies to the 'OR' operator as well except with the rules of an 'OR' statement instead of the rules of an 'AND' statement.

The '&' operator is an element-wise operator. When using this operator, a logical 'AND' is performed on the elements and an array of logical '1's and '0's is returned. This operator does not have short-circuiting behavior *unless* it used in the context of an 'if' statement or 'while' loop. This means that the operator will still keep comparing the elements even if a logical '0' is returned. For example, imagine we have two logical expressions 'E1' and 'E2'. If we compare these expressions like this:

'E1 & E2'

Regardless of whether 'E1' is true or false, 'E2' will still be evaluated since the '&' has no short-circuiting behavior *unless* it used in the context of an 'if' statement or 'while' loop.

As a sidenote, we will typically be using the '&' operator in the context of comparing logical arrays or logical statements such as:

'X & Y' or '(X > 2) & (X < 5)'

However, the '&' operator can also be used to compare two arrays containing a mixture of non-logical zero and non-zero elements. A logical '1' is produced when the two arrays both contain a non-zero element at the same index.

The operands, or items you are comparing, must be either scalars, vectors, matrices, or multidimensional arrays. The operands must also be the same size or sizes that are compatible.

The '&' operator is equivalent to the 'and' function in MATLAB.

Common Confusion 1
& vs. &&
Matthew Woodring

The '&&' operator is a logical short-circuiting operator. An important note about this operator is that is can *only* be used with scalar logical conditions. You cannot use it to compare arrays or matrices like you can with the '&' operator. Since this operator does have short-circuiting behavior, if two scalars are compared and the first one is a logical '0', the second one will not be evaluated. For example, imagine we have two logical expressions 'E1' and 'E2'. If we compare these expressions like this:

'E1 && E2'

'E2' will not be evaluated if 'E1' is false. Remember, both 'E1' and 'E2' *must* be scalar logical conditions. In the context of our course, this will typically look like the following:

'(X > 2) && (X < 5)'

The '&&' operator is useful for two main reasons. First, it can be more efficient than the '&' operator in certain instances such as when you are comparing a long list of conditions. However, in this class, the efficiency is negligible. The second, and much more important, reason is that the '&&' operator can guarantee that a run-time error does *not* occur. For example, imagine we ran the following line of code using the '&' element-wise operator:

'X = (B ~= 0) & (A / B > 5)'

This statement is saying "put the result of whether 'B' does not equal 0 AND 'A' divided by 'B' is greater than 5 into the 'X' variable". Now, imagine that 'B' is equal to '0'. In this case, we will get a run-time error since '(A / B > 5)' will run regardless of whether '(B ~= 0)' is true or not. If we had used the '&&' operator in the above example, then the run-time error would not occur because MATLAB would have stopped evaluating the line when '(B ~= 0)' evaluated to false.

So, should you use the '&' or '&&' operator in this class? In most cases, I recommend just using the '&&' operator all the time *unless* a problem requires you to use the '&' operator. Regardless of which one you use, you will probably be fine since you will likely be able to quickly identify and fix any errors that arise from using these operators.