Conditional Expressions

C's `if` statement allows a program to perform one of two actions depending on the value of a condition. C also provides an *operator* that allows an expression to produce one of two *values* depending on the value of a condition.

The ***conditional operator*** consists of two symbols (`?` and `:`), which must be used together in the following way:

conditional expression

expr1 `?` *expr2* `:` *expr3*

expr1, *expr2*, and *expr3* can be expressions of any type. The resulting expression is said to be a ***conditional expression***. The conditional operator is unique among C operators in that it requires *three* operands instead of one or two. For this reason, it is often referred to as a ***ternary*** operator.

The conditional expression *expr1* `?` *expr2* `:` *expr3* should be read "if *expr1* then *expr2* else *expr3*." The expression is evaluated in stages: *expr1* is evaluated first; if its value isn't zero, then *expr2* is evaluated, and its value is the value of the entire conditional expression. If the value of *expr1* is zero, then the value of *expr3* is the value of the conditional.

The following example illustrates the conditional operator:

```
int i, j, k;

i = 1;
j = 2;
k = i > j ? i : j;          /* k is now 2 */
k = (i >= 0 ? i : 0) + j;   /* k is now 3 */
```

The conditional expression `i > j ? i : j` in the first assignment to `k` returns the value of either `i` or `j`, depending on which one is larger. Since `i` has the value 1 and `j` has the value 2, the `i > j` comparison fails, and the value of the conditional is 2, which is assigned to `k`. In the second assignment to `k`, the `i >= 0` comparison succeeds; the conditional expression `(i >= 0 ? i : 0)` has the value 1, which is then added to `j` to produce 3. The parentheses are necessary, by the way; the precedence of the conditional operator is less than that of the other operators we've discussed so far, with the exception of the assignment operators.

Conditional expressions tend to make programs shorter but harder to understand, so it's probably best to avoid them. There are, however, a few places in which they're tempting; one is the `return` statement. Instead of writing

```
if (i > j)
  return i;
else
  return j;
```

many programmers would write

```
return i > j ? i : j;
```