Calls of `printf` can sometimes benefit from condition expressions. Instead of

```
if (i > j)
  printf("%d\n", i);
else
  printf("%d\n", j);
```

we could simply write

```
printf("%d\n", i > j ? i : j);
```

macro definitions ➤ *14.3*

Conditional expressions are also common in certain kinds of macro definitions.

Boolean Values in C89

For many years, the C language lacked a proper Boolean type, and there is none defined in the C89 standard. This omission is a minor annoyance, since many programs need variables that can store either *false* or *true*. One way to work around this limitation of C89 is to declare an `int` variable and then assign it either 0 or 1:

```
int flag;

flag = 0;
...
flag = 1;
```

Although this scheme works, it doesn't contribute much to program readability. It's not obvious that `flag` is to be assigned only Boolean values and that 0 and 1 represent false and true.

To make programs more understandable, C89 programmers often define macros with names such as `TRUE` and `FALSE`:

```
#define TRUE 1
#define FALSE 0
```

Assignments to `flag` now have a more natural appearance:

```
flag = FALSE;
...
flag = TRUE;
```

To test whether `flag` is true, we can write

```
if (flag == TRUE) ...
```

or just

```
if (flag) ...
```

The latter form is better, not only because it's more concise, but also because it will still work correctly if `flag` has a value other than 0 or 1.

To test whether `flag` is false, we can write

```
if (flag == FALSE) ...
```