To see how these macros work, we'll use them to write a function named `max_int` that finds the maximum of *any* number of integer arguments. Here's how we might call the function:

```
max_int(3, 10, 30, 20)
```

The first argument specifies how many additional arguments will follow. This call of `max_int` will return 30 (the largest of the numbers 10, 30, and 20).

Here's the definition of the `max_int` function:

```
int max_int(int n, ...)   /* n must be at least 1 */
{
  va_list ap;
  int i, current, largest;

  va_start(ap, n);
  largest = va_arg(ap, int);

  for (i = 1; i < n; i++) {
    current = va_arg(ap, int);
    if (current > largest)
      largest = current;
  }

  va_end(ap);
  return largest;
}
```

The `...` symbol in the parameter list (known as an *ellipsis*) indicates that the parameter `n` is followed by a variable number of additional parameters.

The body of `max_int` begins with the declaration of a variable of type `va_list`:

```
va_list ap;
```

Declaring such a variable is mandatory for `max_int` to be able to access the arguments that follow `n`.

va_start

The statement

```
va_start(ap, n);
```

indicates where the variable-length part of the argument list begins (in this case, after `n`). A function with a variable number of arguments must have at least one "normal" parameter; the ellipsis always goes at the end of the parameter list, after the last normal parameter.

va_arg

The statement

```
largest = va_arg(ap, int);
```

fetches `max_int`'s second argument (the one after `n`), assigns it to `largest`, and automatically advances to the next argument. The word `int` indicates that we expect `max_int`'s second argument to have `int` type. The statement