Function Calls

A function call consists of a function name followed by a list of arguments, enclosed in parentheses:

```
average(x, y)
print_count(i)
print_pun()
```

If the parentheses are missing, the function won't get called:

```
print_pun;    /*** WRONG ***/
```

Q&A The result is a legal (albeit meaningless) expression statement that looks correct, but has no effect. Some compilers issue a warning such as *"statement with no effect."*

A call of a `void` function is always followed by a semicolon to turn it into a statement:

```
print_count(i);
print_pun();
```

A call of a non-`void` function, on the other hand, produces a value that can be stored in a variable, tested, printed, or used in some other way:

```
avg = average(x, y);
if (average(x, y) > 0)
  printf("Average is positive\n");
printf("The average is %g\n", average(x, y));
```

The value returned by a non-`void` function can always be discarded if it's not needed:

```
average(x, y);    /* discards return value */
```

expression statements ➤4.5

This call of `average` is an example of an expression statement: a statement that evaluates an expression but then discards the result.

Ignoring the return value of `average` is an odd thing to do, but for some functions it makes sense. The `printf` function, for example, returns the number of characters that it prints. After the following call, `num_chars` will have the value 9:

```
num_chars = printf("Hi, Mom!\n");
```

Since we're probably not interested in the number of characters printed, we'll normally discard `printf`'s return value:

```
printf("Hi, Mom!\n");    /* discards return value */
```

To make it clear that we're deliberately discarding the return value of a function, C allows us to put `(void)` before the call: