PROGRAM Computing Averages

Suppose we often need to compute the average of two `double` values. The C library doesn't have an "average" function, but we can easily define our own. Here's what it would look like:

```
double average(double a, double b)
{
  return (a + b) / 2;
}
```

The word `double` at the beginning is `average`'s ***return type:*** the type of data that the function returns each time it's called. The identifiers `a` and `b` (the function's ***parameters***) represent the two numbers that will be supplied when `average` is called. Each parameter must have a type (just like every variable has a type); in this example, both `a` and `b` have type `double`. (It may look odd, but the word `double` must appear twice, once for `a` and once for `b`.) A function parameter is essentially a variable whose initial value will be supplied later, when the function is called.

Q&A

Every function has an executable part, called the ***body***, which is enclosed in braces. The body of `average` consists of a single `return` statement. Executing this statement causes the function to "return" to the place from which it was called; the value of `(a + b) / 2` will be the value returned by the function.

To call a function, we write the function name, followed by a list of ***arguments***. For example, `average(x, y)` is a call of the `average` function. Arguments are used to supply information to a function; in this case, `average` needs to know which two numbers to average. The effect of the call `average(x, y)` is to copy the values of `x` and `y` into the parameters `a` and `b`, and then execute the body of `average`. An argument doesn't have to be a variable; any expression of a compatible type will do, allowing us to write `average(5.1, 8.9)` or `average(x/2, y/3)`.

We'll put the call of `average` in the place where we need to use the return value. For example, we could write

```
printf("Average: %g\n", average(x, y));
```

to compute the average of `x` and `y` and then print it. This statement has the following effect:

1. The `average` function is called with `x` and `y` as arguments.
2. `x` and `y` are copied into `a` and `b`.
3. `average` executes its `return` statement, returning the average of `a` and `b`.
4. `printf` prints the value that `average` returns. (The return value of `average` becomes one of `printf`'s arguments.)

Note that the return value of `average` isn't saved anywhere; the program prints it and then discards it. If we had needed the return value later in the program, we could have captured it in a variable: