© attator/iStockphoto.

Chapter Five: Functions

by Cay Horstmann

Chapter Goals

- To be able to implement functions
- To become familiar with the concept of parameter passing
- To appreciate the importance of function comments
- To develop strategies for decomposing complex tasks into simpler ones
- To be able to determine the scope of a variable
- To recognize when to use value and reference parameters

Topic 1

1. Functions as black boxes
2. Implementing functions
3. Parameter passing
4. Return values
5. Functions without return values
6. Reusable functions
7. Stepwise refinement
8. Variable scope and globals
9. Reference parameters
10. Recursive functions

What Is a Function? Why Functions?

A function is a sequence of instructions with a name.

A function packages a computation into a form that can be easily understood and reused.

Calling a Function

A programmer *calls* a function to have its instructions run.

```
int main()
{
    double z = pow(2, 3);
    . . .
}
```

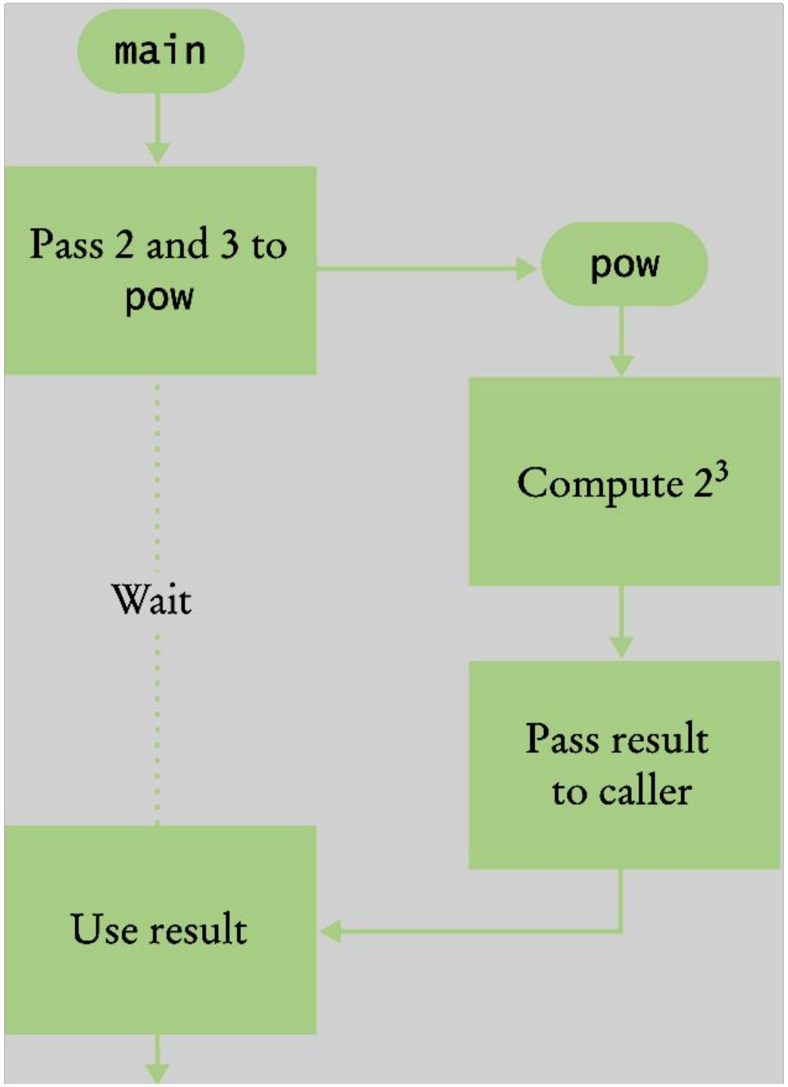
By using the expression: **`pow(2, 3)`**
`main` *calls* the **`pow`** function, asking it to compute **`2`**^{3}.

The **`main`** function is temporarily suspended.

The instructions of the **`pow`** function execute and compute the result.

The pow function *returns* its result back to **`main`**, and the **`main`** function resumes execution.

Flowchart: Calling a Function



Execution flow during a function call

Big C++ by Cay Horstmann

Parameters

```
int main()
{
    double z = pow(2, 3);
   ...
}
```

When another function calls the `pow` function, it provides "inputs", such as the values 2 and 3 in the call `pow(2, 3).`

In order to avoid confusion with inputs that are provided by a human user (`cin >>`), these values are called *parameter values*.

The "output" that the `pow` function computes is called the *return value* (not output using <<).

An Output Statement Does Not Return a Value

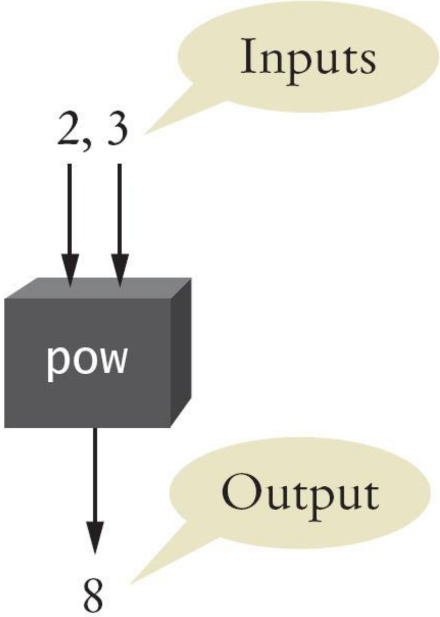
output ≠ return

- The **`return`** statement does not display output
  - Rather, it causes execution to resume in the calling program and ends the called function.
  - `return` may also pass a “value” back to the calling program

An output statement using << communicates *only* with the user running the program.

The Black Box Concept

- You can think of a function as a “black box” where you can’t see what’s inside but you know what it does.



- How did the **`pow`** function do its job?
- You don’t need to know.
- You only need to know its *specification*.

Big C++ by Cay Horstmann

Topic 2

1. Functions as black boxes
2. Implementing functions
3. Parameter passing
4. Return values
5. Functions without return values
6. Reusable functions
7. Stepwise refinement
8. Variable scope and globals
9. Reference parameters
10. Recursive functions

Implementing Functions

Example: Calculate the volume of a cube

When writing this function, you need to:

- Pick a good, descriptive name for the function

- Give a type and a name for each parameter. There will be one parameter for each piece of information the function needs to do its job.

- Specify the type of the return type:

```
double cube_volume(double side_length)
```

- Then write the body of the function, as statements enclosed in braces

```
{ }
```

`cube_volume` Function

The comments at the top are the standard Java format which you should follow for any function you write (even in C++). They can be processed by the Doxygen program to automatically generate documentation of your function libraries.

```
/**
   Computes the volume of a cube.
   @param side_length the side length of the cube
   @return the volume
*/
double cube_volume(double side_length)
{
  double volume = side_length * side_length * side_length;
  return volume;
}
```

Test your Functions

You should always test the function.

ch05/cube.cpp

You'll write a **main** function to do this.

```
#include <iostream>
using namespace std;

/**
   Computes the volume of a cube.
   @param side_length the side length of the cube
   @return the volume
*/
double cube_volume(double side_length)
{
   double volume = side_length * side_length *
  side_length;
   return volume;
}
```

A Testbench Program (main)

```
int main()
{
   double result1 = cube_volume(2);
   double result2 = cube_volume(10);
   cout << "A cube with side length 2 has volume "
      << result1 << endl;
   cout << "A cube with side length 10 has volume "
      << result2 << endl;

   return 0;
}
```

Topic 3

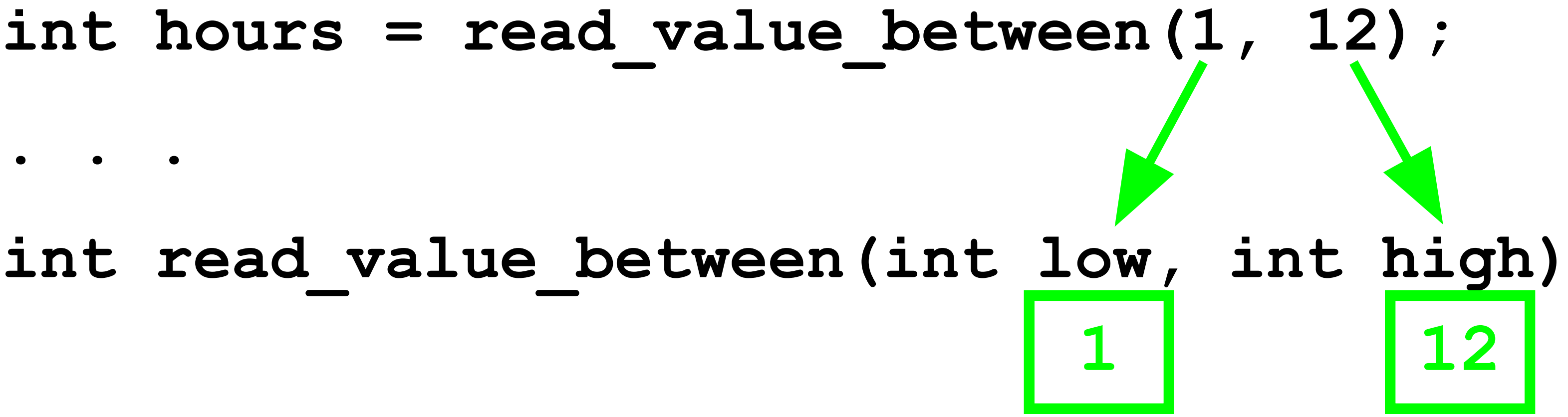
1. Functions as black boxes
2. Implementing functions
3. Parameter passing
4. Return values
5. Functions without return values
6. Reusable functions
7. Stepwise refinement
8. Variable scope and globals
9. Reference parameters
10. Recursive functions

Big C++ by Cay Horstmann

Parameter Passing

When a function is called, a *parameter variable* is created for each value passed in.

Each parameter variable is *initialized* with the corresponding parameter value from the call.



```
int hours = read_value_between(1, 12);
. . .
int read_value_between(int low, int high)
```

Parameter Passing, `cube_volume` example

Here is a call to the **`cube_volume`** function:

```
double result1 = cube_volume(2);
```

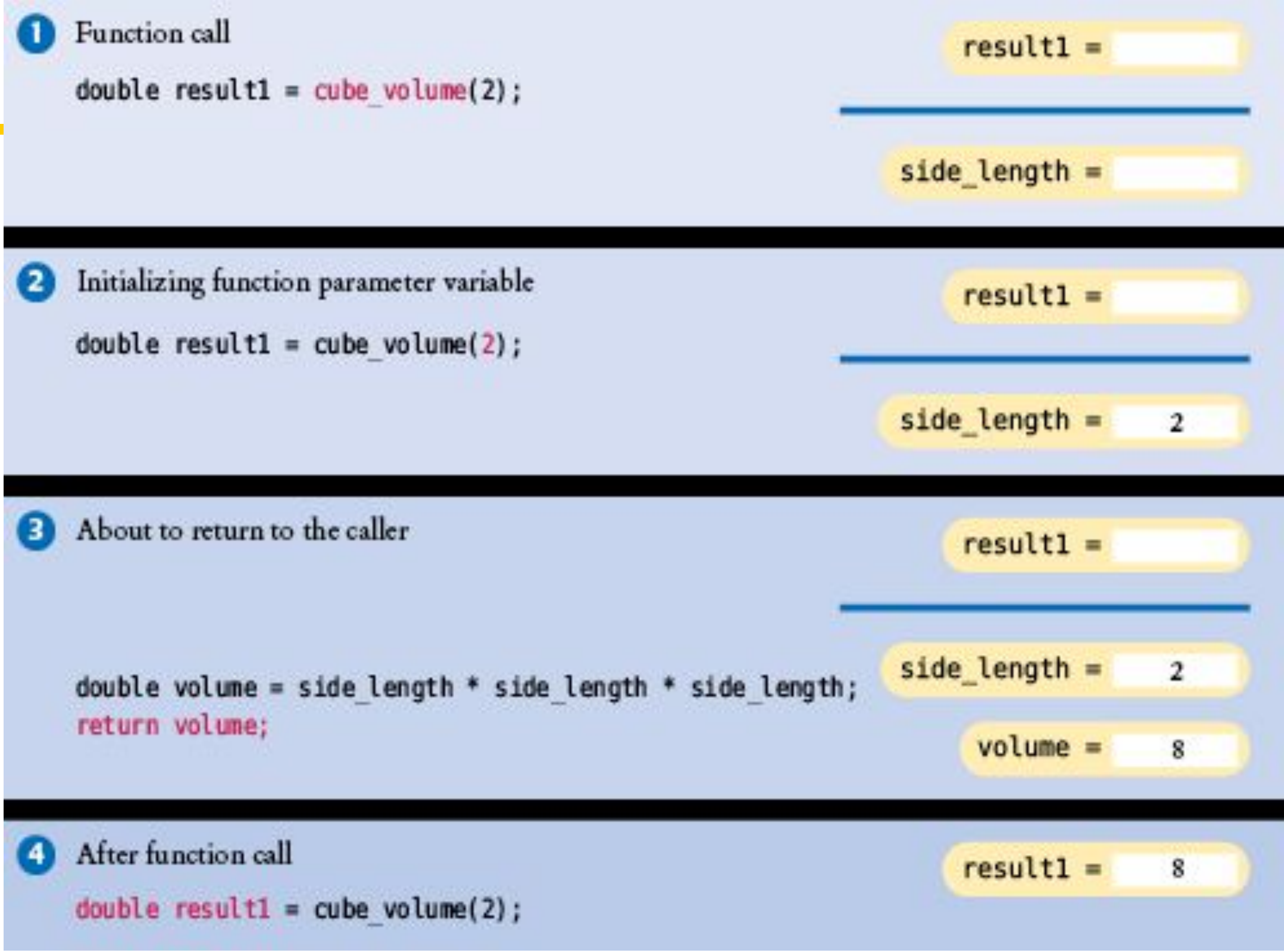
Here is the function definition:

```
double cube_volume(double side_length)
{
  double volume = side_length * side_length * side_length;
  return volume;
}
```

We'll keep up with their variables and parameters:

```
result1
side_length
volume
```

Parameter Passing



1 Function call

```
double result1 = cube_volume(2);
```

2 Initializing function parameter variable

```
double result1 = cube_volume(2);
```

3 About to return to the caller

```
double volume = side_length * side_length * side_length;
return volume;
```

4 After function call

```
double result1 = cube_volume(2);
```

In the calling function (`main`), the variable **`result1`** is declared. When the **`cube_volume`** function is called, the parameter variable **`side_length`** is created & initialized with the value that was passed in the call (2).

After the return statement, the local variables `side_length` and `volume` disappear from memory.

Big C++ by Cay Horstmann