



Chapter Five: Functions

Slides by Evan Gallagher

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

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

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Call a Function to Get Something Done



If it's chilly in here... do something about it!

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

What Is a Function? Why Functions?

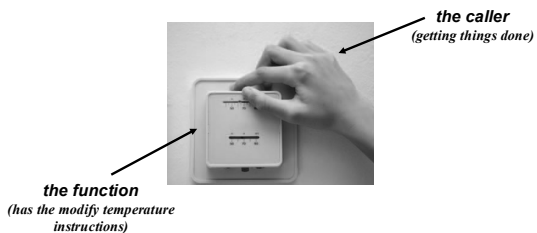
A function is a sequence of instructions with a name.

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

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Calling a Function

A programmer calls a function to have its instructions executed.



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Calling a Function

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

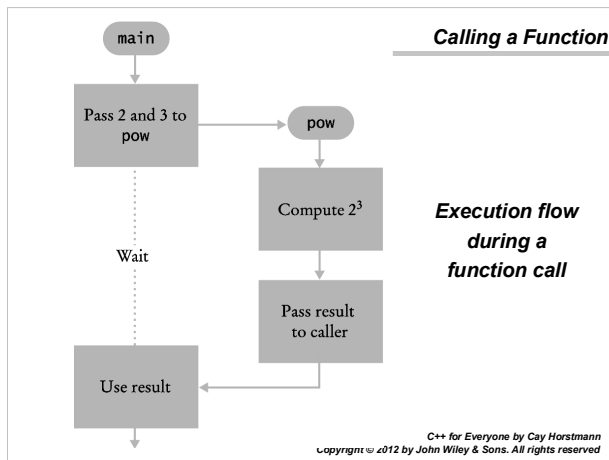
*By using the expression: `pow(2, 3)`
`main` calls the `pow` function, asking it to compute 2^3 .*

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.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved



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 `<<`).

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

An Output Statement Does Not Return a Value

output ≠ return

If a function needs to display something for a user to see, it cannot use a `return` statement.

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

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

The Return Statement Does Not Display (Good!)

output ≠ return

If a programmer needs the result of a calculation done by a function, the function must have a `return` statement.

An output statement using `<<` does not communicate with the calling programmer

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

The Return Statement Does Not Display (Good!)

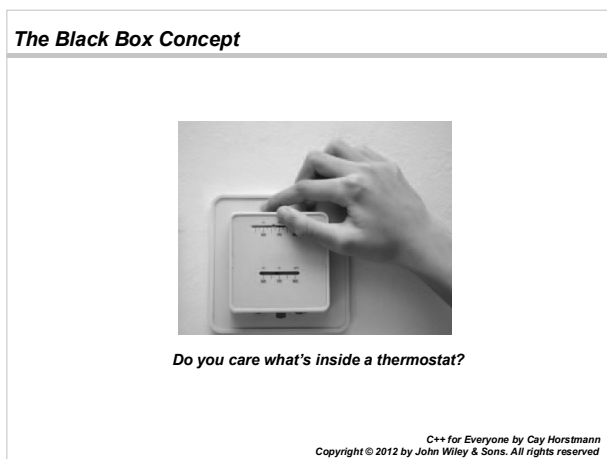
```

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

    // display result of calculation
    // stored in variable z
    cout << z << endl;

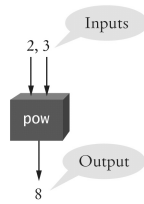
    // return from main - no output here!!!
    return 0;
}
  
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved



The Black Box Concept

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

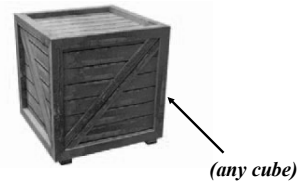


- How did the `pow` function do its job?
- You don’t need to know.
- You only need to know its specification.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Implementing Functions

Write the function that will do this:



Compute the volume of a cube with a given side length

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Implementing Functions

When writing this function, you need to:

- Pick a good, descriptive name for the function

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Implementing Functions

When writing this function, you need to:

- Pick a good, descriptive name for the function

(What else would a function named `cube_volume` do?)

`cube_volume`

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Implementing Functions

When writing this function, you need to:

- Pick a good, descriptive name for the function
- Give a type and a name for each parameter.

`cube_volume`

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Implementing Functions

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.

(And don’t forget the parentheses)

`cube_volume(double side_length)`

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Implementing Functions

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

```
cube_volume(double side_length)
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Implementing Functions

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)
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Implementing Functions

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

Now write the body of the function:

the code to do the cubing

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Implementing Functions

The code the function names must be in a block:

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

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Implementing Functions

The parameter allows the caller to give the function information it needs to do it's calculating.

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

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Implementing Functions

The return statement gives the function's result to the caller.

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

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Test Your Function

You should always test the function.

You'll write a main function to do this.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

A Complete Testing Program

ch05/cube.cpp

```
#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 Complete Testing Program

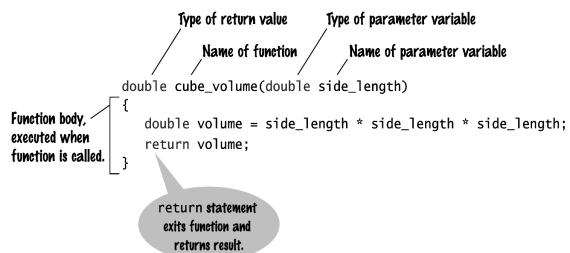
ch05/cube.cpp

```
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;
}
```

Implementing Functions

SYNTAX 5.1 Function Definition



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Commenting Functions

- **Whenever you write a function, you should comment its behavior.**
- **Comments are for human readers, not compilers**
- **There is no universal standard for the layout of a function comment.**
 - The layout used in the previous program is borrowed from the Java programming language and is used in some C++ tools to produce documentation from comments.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Commenting Functions

Function comments do the following:

- explain the purpose of the function
- explain the meaning of the parameters
- state what value is returned
- state any special requirements

Comments state the things a programmer who wants to use your function needs to know.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Calling Functions


Consider the order of activities when a function is called.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Parameter Passing

In the function call,
a value is supplied for each parameter,
called the parameter value.
(Other commonly used terms for this value
are: actual parameter and argument.)

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

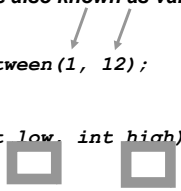


C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Parameter Passing

When a function is called,
a parameter variable is created for each value passed
in.
(Another commonly used term is formal parameter.)
(Parameters that take values are also known as value
parameters.)

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

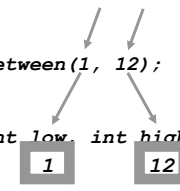


C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Parameter Passing

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

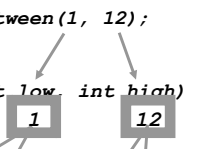
```
int hours = read_value_between(1, 12);  
...  
int read_value_between(int low, int high)
```



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Parameter Passing

```
int hours = read_value_between(1, 12);  
int read_value_between(int low, int high)  
{  
    int input;  
    do  
    {  
        cout << "Enter a value between "  
              << low << " and " << high << ": ";  
        cin >> input;  
    } while (input < low || input > high);  
    return input;  
}
```



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Parameter Passing

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
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Parameter Passing

1. In the calling function, the local variable `result1` already exists. When the `cube_volume` function is called, the parameter variable `side_length` is created.

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

1 Function call

result1 =

side_length =

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Parameter Passing

2. The parameter variable is initialized with the value that was passed in the call. In our case, `side_length` is set to 2.

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

2 Initializing function parameter

result1 =

side_length =

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Parameter Passing

1. The function computes the expression `side_length * side_length * side_length`, which has the value 8. That value is stored in the local variable `volume`.

```
[inside the function]
double volume = side_length * side_length * side_length;
```

3 About to return to the caller

result1 =

side_length =

volume =

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Parameter Passing

4. The function returns. All of its variables are removed. The return value is transferred to the caller, that is, the function calling the `cube_volume` function.

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

4 After function call

result1 =

The function executed: `return volume;`
which gives the caller the value 8

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Parameter Passing

4. The function returns. All of its variables are removed. The return value is transferred to the caller, that is, the function calling the `cube_volume` function.

```
double result1 = cube_volume(2);
    the returned 8 is about to be stored
```

4 After function call

result1 =

The function is over.
`side_length` and `volume` are gone.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Parameter Passing

The caller stores this value in their local variable `result1`.

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

4 After function call

result1 =

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Return Values

The `return` statement yields the function result.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Return Values

Also,

The `return` statement
– terminates a function call
– immediately

// you are here



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

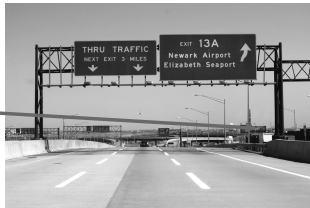
Return Values

Also,

The `return` statement
– terminates a function call
– immediately

// you are here

return
// now you are here



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Return Values

This behavior can be used to handle unusual cases.

*What should we do if the side length is negative?
We choose to return a zero and not do any calculation:*

```
double cube_volume(double side_length)
{
    if (side_length < 0) return 0;
    double volume = side_length * side_length * side_length;
    return volume;
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Return Values

The `return` statement can return the value of any expression.

Instead of saving the return value in a variable and returning the variable, it is often possible to eliminate the variable and return a more complex expression:

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

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Error – Missing Return Value

Your function always needs to return something.

*Consider putting in a guard against negatives
and also trying to eliminate the local variable:*

```
double cube_volume(double side_length)
{
    if (side_length >= 0)
    {
        return side_length * side_length *
            side_length; }
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Error – Missing Return Value

Consider what is returned if the caller does pass in a negative value!

```
double cube_volume(double side_length)
{
    if (side_length >= 0)
    {
        return side_length * side_length *
            side_length; }
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Error – Missing Return Value

Every possible execution path should return a meaningful value:

```
double cube_volume(double side_length)
{
    if (side_length >= 0)
    {
        return side_length * side_length *
            side_length;
    }
    ← ?
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Error – Missing Return Value

Depending on circumstances, the compiler might flag this as an error, or the function might return a random value.

This is always bad news, and you must protect against this problem by returning some safe value.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Functions Without Return Values

Consider the task of writing a string with the following format around it.

Any string could be used.

For example, the string "Hello" would produce:

```
-----
!Hello!
-----
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Functions Without Return Values – The void Type

A function for this task can be defined as follows:

```
void box_string(string str)
```

Notice the return type of this function: void

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Functions Without Return Values – The void Type

This kind of function is called a void function.

```
void box_string(string str)
```

Use a return type of void to indicate that a function does not return a value.

void functions are used to simply do a sequence of instructions
– They do not return a value to the caller.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

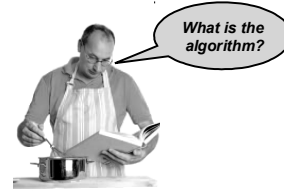
Functions Without Return Values – The void Type

void functions are used only to do a sequence of instructions.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Functions Without Return Values – The void Type

!Hello!



- Print a line that contains the '-' character $n + 2$ times, where n is the length of the string.
- Print a line containing the string, surrounded with a ! to the left and right.
- Print another line containing the - character $n + 2$ times.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Functions Without Return Values – The void Type

```
void box_string(string str)
{
    int n = str.length();
    for (int i = 0; i < n + 2; i++){ cout << "-"; }
    cout << endl;
    cout << "!" << str << "!" << endl;
    for (int i = 0; i < n + 2; i++) { cout << "-"; }
    cout << endl;
}
```

Note that this function doesn't compute any value.

It performs some actions and then returns to the caller
– without returning a value.
(The return occurs at the end of the block.)

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Functions Without Return Values – The void Type

Because there is no return value, you cannot use box_string in an expression.

You can make this call kind of call:

```
box_string("Hello");
```

but not this kind:

```
result = box_string("Hello");
// Error: box_string doesn't
//       return a result.
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Functions Without Return Values – The void Type

If you want to return from a void function before reaching the end, you use a return statement without a value. For example:

```
void box_string(string str)
{
    int n = str.length();
    int n = str.length();
    if (n == 0)
    {
        return; // Return immediately
    }
    . . . // None of these statements
          // will be executed
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

When you write nearly identical code multiple times, you should probably introduce a function.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

Consider how similar the following statements are:

```
int hours;
do
{
    cout << "Enter a value between 0 and 23:";
    cin >> hours;
} while (hours < 0 || hours > 23);

int minutes;
do
{
    cout << "Enter a value between 0 and 59: ";
    cin >> minutes;
} while (minutes < 0 || minutes > 59);
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

The values for the high end of the range are different.

```
int hours;
do
{
    cout << "Enter a value between 0 and 23:";
    cin >> hours;
} while (hours < 0 || hours > 23);

int minutes;
do
{
    cout << "Enter a value between 0 and 59: ";
    cin >> minutes;
} while (minutes < 0 || minutes > 59);
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

The names of the variables are different.

```
int hours;
do
{
    cout << "Enter a value between 0 and 23:";
    cin >> hours;
} while (hours < 0 || hours > 23);

int minutes;
do
{
    cout << "Enter a value between 0 and 59: ";
    cin >> minutes;
} while (minutes < 0 || minutes > 59);
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

But there is common behavior.

```
int hours;
do
{
    cout << "Enter a value between _ and _:";
    cin >> hours;
} while (hours < 0 || hours > );

int minutes;
do
{
    cout << "Enter a value between _ and _: ";
    cin >> minutes;
} while (minutes < 0 || minutes > );
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

Move the common behavior into one function.

```
int read_int_up_to(int high)
{
    int input;
    do
    {
        cout << "Enter a value between "
        << "0 and " << high << ": ";
        cin >> input;
    } while (input < 0 || input > high);
    return input;
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

Here we read one value, making sure it's within the range.

```
int read_int_up_to(int high)
{
    int input;
    do
    {
        cout << "Enter a value between "
        << "0 and " << high << ": ";
        cin >> input;
    } while (input < 0 || input > high);
    return input;
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

Then we can use this function as many times as we need:

```
int hours = read_int_up_to(23);
int minutes = read_int_up_to(59);
```

Note how the code has become much easier to understand.

And we are not rewriting code

– code reuse!

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

Perhaps we can make this function even better:

```
int months = read_int_up_to(12);
```

Can we use this function to get a valid month?
Months are numbered starting at 1, not 0.

We can modify the code to take two parameters:
the end points of the valid range.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

Again, consider how similar the following statements are:

```
int month;
do
{
    cout << "Enter a value between 1 and 12:";
    cin >> month;
} while (month < 1 || month > 12);

int minutes;
do
{
    cout << "Enter a value between 0 and 59: ";
    cin >> minutes;
} while (minutes < 0 || minutes > 59);
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

As before, the values for the range are different.

```
int month;
do
{
    cout << "Enter a value between 1 and 12:";
    cin >> month;
} while (month < 1 || month > 12);

int minutes;
do
{
    cout << "Enter a value between 0 and 59: ";
    cin >> minutes;
} while (minutes < 0 || minutes > 59);
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

But the names of the variables are different.

```
int month;
do
{
    cout << "Enter a value between 1 and 12:";
    cin >> month;
} while (month < 1 || month > 12);

int minutes;
do
{
    cout << "Enter a value between 0 and 59: ";
    cin >> minutes;
} while (minutes < 0 || minutes > 59);
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

Notice the common behavior?

```
int month;
do
{
    cout << "Enter a value between _ and _:";
    cin >> month;
} while (month < _ || month > _);

int minutes;
do
{
    cout << "Enter a value between _ and _:";
    cin >> minutes;
} while (minutes < _ || minutes > _);
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

Again, move the common behavior into one function.

```
int read_value_between(int low, int high)
{
    int input;
    do
    {
        cout << "Enter a value between "
              << low << " and " << high << ": ";
        cin >> input;
    } while (input < low || input > high);
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

A different name would need to be used, of course because it does a different activity.

```
int read_value_between(int low, int high)
{
    int input;
    do
    {
        cout << "Enter a value between "
              << low << " and " << high << ": ";
        cin >> input;
    } while (input < low || input > high);
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Designing Functions – Turn Repeated Code into Functions

We can use this function as many times as we need, passing in the end points of the valid range:

```
int hours = read_value_between(1, 12);
int minutes = read_value_between(0, 59);
```

Note how the code has become even better.

And we are still not rewriting code

– code reuse!

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

- One of the most powerful strategies for problem solving is the process of **stepwise refinement**.
- To solve a difficult task, break it down into simpler tasks.
- Then keep breaking down the simpler tasks into even simpler ones, until you are left with tasks that you know how to solve.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

Use the process of **stepwise refinement** to decompose complex tasks into simpler ones.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

Get coffee



This is the whole problem: this is like main.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

Get coffee

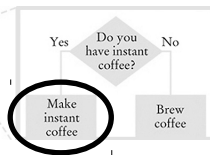


*The whole problem can be broken into:
if we can ask someone to give us coffee, we are done
but if not, we can make coffee (which we will
have to break into its parts)*

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

Make coffee



*The make coffee sub-problem can be broken into:
if we have instant coffee, we can make that
but if not, we can brew coffee
(maybe these will have parts)*

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

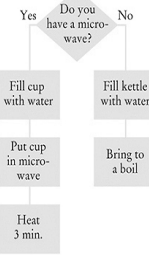
Make instant coffee



*Making instant coffee breaks into:
1. Boil Water
2. Mix (stir if you wish)
(Do these have sub-problems?)*

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

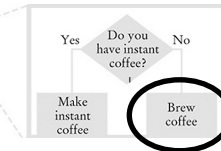


*Boiling water appears
not to be so easy.
Many steps,
but none have sub-steps.*

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

Make coffee



*Going back to the branch between
instant or brew, we need to think about brewing.
Can we break that into parts?*

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

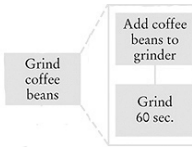


Or I can
brew

**Brewing coffee has several steps.
Do any need more breakdown
(grind coffee beans)?**

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement



Grinding...
Is that it?

**Grinding is a two step process
with no sub-sub-steps.**

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

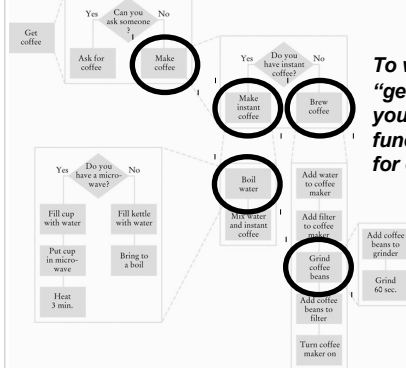
Stepwise Refinement



Are we
done yet?

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement – The Complete Process Shown



**To write the
“get coffee” program,
you would write
functions
for each sub-problem.**

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

**When writing a check by hand the recipient might be
tempted to add a few digits in front of the amount.**


C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

**To discourage this, when printing a check,
it is customary to write the check amount both
as a number (“\$274.15”) and as a text string
(“two hundred seventy four dollars and 15 cents”)**

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

	WILEY	John Wiley & Sons, Inc. 111 River Street Hoboken, NJ 07030-5774	Publishers' Bank Minnesota 2000 Pierce Blvd Jonesville, MN 55400	CHECK NUMBER	063331	74-99 311	567390
				Date	Amount		
PAY	4659484			04/29/09	*****274.15		
TWO HUNDRED SEVENTY FOUR AND 15 / 100 *****							
TO THE ORDER OF:							
JOHN DOE							
1109 Franklin Blvd							
Sunnyvale, CA 95014							

⑆ 4 7 8 0 6 2 4 0 ⑆ 2 0 0 6 2 0 3 5 ⑆ 1 3 0 ⑆

We will a program to take an amount and produce the text.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

***Sometimes we reduce the problem a bit when we start:
we will only deal with amounts less than \$1,000.***

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

Of course we will write a function to solve this sub-problem.

```
/**
 * Turns a number into its English name.
 * @param number a positive integer < 1,000
 * @return the name of number (e.g., "two hundred seventy four")
 */
string int_name(int number)
```

Notice that we started by writing only the comment and the first line of the function.

Also notice that the constraint of < \$1,000 is announced in the comment.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

Before starting to write this function, we need to have a plan.

Are there special considerations?

Are there subparts?

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

*If the number is between 1 and 9,
we need to compute "one" ... "nine".*

In fact, we need the same computation again for the hundreds (“two” hundred).

***Any time you need to do something more than once,
it is a good idea to turn that into a function:***

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

```
/**
 * Turns a digit into its English name.
 * @param digit an integer between 1 and 9
 * @return the name of digit ("one" ... "nine")
 */
string digitName(int digit)
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

Numbers between 10 and 19 are special cases.

Let's have a separate function `teen_name` that converts them into strings "eleven", "twelve", "thirteen", and so on:

```
/**
Turns a number between 10 and 19 into its English
name.
@param number an integer between 10 and 19
@return the name of the number ("ten" ...
        "nineteen")
*/
string teen_name(int number)
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

Next, suppose that the number is between 20 and 99. Then we show the tens as "twenty", "thirty", ..., "ninety". For simplicity and consistency, put that computation into a separate function:

```
/**
Gives the name of the tens part of a number between 20 and 99.
@param number an integer between 20 and 99
@return the name of the tens part of the number ("twenty"... "ninety")
*/
string tens_name(int number)
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement

- Now suppose the number is at least 20 and at most 99.
 - If the number is evenly divisible by 10, we use `tens_name`, and we are done.
 - Otherwise, we print the tens with `tens_name` and the ones with `digit_name`.
- If the number is between 100 and 999,
 - then we show a digit, the word "hundred", and the remainder as described previously.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement – The Pseudocode

```
part = number (The part that still needs to be
converted)
name = "" (The name of the number starts as
the empty string)

If part >= 100
    name = name of hundreds in part + "
    hundred"
    Remove hundreds from part

If part >= 20
    Append tens_name(part) to name
    Remove tens from part.
Else if part >= 10
    Append teen_name(part) to name
    part = 0

If (part > 0)
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement – The Pseudocode

- This pseudocode has a number of important improvements over the descriptions and comments.
 - It shows how to arrange the order of the tests, starting with the comparisons against the larger numbers
 - It shows how the smaller number is subsequently processed in further `if` statements.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement – The Pseudocode

- On the other hand, this pseudocode is vague about:
 - The actual conversion of the pieces, just referring to "name of hundreds" and the like.
 - Spaces—it would produce strings with no spaces:
"twohundredseventyfour"

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement – The Pseudocode

Compared to the complexity of the main problem, one would hope that spaces are a minor issue.

It is best not to muddy the pseudocode with minor details.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement – Pseudocode to C++

Now for the real code.

The last three cases are easy so let's start with them:

```
if (part >= 20)
{
    name = name + " " + tens_name(part);
    part = part % 10;
}
else if (part >= 10)
{
    name = name + " " + teen_name(part);
    part = 0;
}
if (part > 0)
{
    name = name + " " + digit_name(part);
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement – Pseudocode to C++

Finally, the case of numbers between 100 and 999. Because $part < 1000$, $part / 100$ is a single digit, and we obtain its name by calling `digit_name`. Then we add the “hundred” suffix:

```
if (part >= 100)
{
    name = digit_name(part / 100) + " hundred";
    part = part % 100;
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stepwise Refinement – Pseudocode to C++

Now for the complete program.

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

ch05/intname.cpp

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

The Complete Program

ch05/intname.cpp

```
/**
 * Turns a digit into its English name.
 * @param digit an integer between 1 and 9
 * @return the name of digit ("one" ... "nine")
 */
string digit_name(int digit)
{
    if (digit == 1) return "one";
    if (digit == 2) return "two";
    if (digit == 3) return "three";
    if (digit == 4) return "four";
    if (digit == 5) return "five";
    if (digit == 6) return "six";
    if (digit == 7) return "seven";
    if (digit == 8) return "eight";
    if (digit == 9) return "nine";
    return "";
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

The Complete Program

ch05/intname.cpp

```
/**
 * Turns a number between 10 and 19 into its English name.
 * @param number an integer between 10 and 19
 * @return the name of the given number ("ten" ... "nineteen")
 */
string teens_name(int number)
{
    if (number == 10) return "ten";
    if (number == 11) return "eleven";
    if (number == 12) return "twelve";
    if (number == 13) return "thirteen";
    if (number == 14) return "fourteen";
    if (number == 15) return "fifteen";
    if (number == 16) return "sixteen";
    if (number == 17) return "seventeen";
    if (number == 18) return "eighteen";
    if (number == 19) return "nineteen";
    return "";
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

The Complete Program

ch05/intname.cpp

```
/**
 * Gives the name of the tens part of a number between 20 and 99.
 * @param number an integer between 20 and 99
 * @return the name of the tens part of the number ("twenty" ...
 * "ninety")
 */
string tens_name(int number)
{
    if (number >= 90) return "ninety";
    if (number >= 80) return "eighty";
    if (number >= 70) return "seventy";
    if (number >= 60) return "sixty";
    if (number >= 50) return "fifty";
    if (number >= 40) return "forty";
    if (number >= 30) return "thirty";
    if (number >= 20) return "twenty";
    return "";
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

The Complete Program

ch05/intname.cpp

```
/**
 * Turns a number into its English name.
 * @param number a positive integer < 1,000
 * @return the name of the number (e.g. "two hundred seventy four")
 */
string int_name(int number)
{
    int part = number; // The part that still needs to be converted
    string name; // The return value

    if (part >= 100)
    {
        name = digit_name(part / 100) + " hundred";
        part = part % 100;
    }
    if (part >= 20)
    {
        name = name + " " + tens_name(part);
        part = part % 10;
    }
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

The Complete Program

ch05/intname.cpp

```
else if (part >= 10)
{
    name = name + " " + teens_name(part);
    part = 0;
}

if (part > 0)
{
    name = name + " " + digit_name(part);
}

return name;
}

int main()
{
    cout << "Please enter a positive integer: ";
    int input;
    cin >> input;
    cout << int_name(input) << endl;
    return 0;
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Good Design – Keep Functions Short

- **There is a certain cost for writing a function:**
 - You need to design, code, and test the function.
 - The function needs to be documented.
 - You need to spend some effort to make the function reusable rather than tied to a specific context.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Good Design – Keep Functions Short

- And you should keep your functions short.
- As a rule of thumb, a function that is so long that it will not fit on a single screen in your development environment should probably be broken up.
- Break the code into other functions



Whew!
That 'breaking'
word always
scares me

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Tracing Functions

When you design a complex set of functions, it is a good idea to carry out a manual walkthrough before entrusting your program to the computer.

This process is called tracing your code.

You should trace each of your functions separately.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Tracing Functions

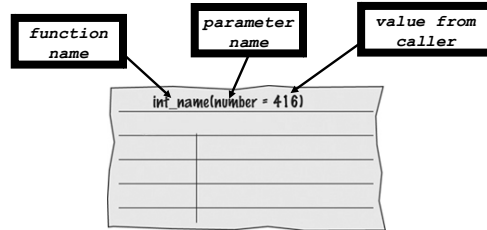
To demonstrate, we will trace the `int_name` function when 416 is passed in.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Tracing Functions

Here is the call: ... `int_name(416)` ...

Take an index card (or use the back of an envelope) and write the name of the function and the names and values of the parameter variables, like this:



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Tracing Functions

Then write the names and values of the function variables.

```
string int_name(int number)
{
    int part = number; // The part that still needs
                        // to be converted
    string name; // The return value, initially ""
```

Write them in a table since you will update them as you walk through the code.

part	name
416	""

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Tracing Functions

The test (`part >= 100`) is true so the code is executed.

```
if (part >= 100)
{
    name = digit_name(part / 100) + " hundred";
    part = part % 100;
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Tracing Functions

`part / 100` is 4

```
if (part >= 100)
{
    name = digit_name(part / 100) + " hundred";
    part = part % 100;
}
```

so `digit_name(4)` is easily seen to be "four".

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Tracing Functions

```
if (part >= 100)
{
    name = digit_name(part / 100) + " hundred";
    part = part % 100;
}
```

`part % 100` is 16.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Tracing Functions

name has changed to
`name + " " + digit_name(part / 100) + "hundred"`
which is the string "four hundred",

part has changed to `part % 100`, or 16.

int_name(number = 416)	
part	name
416	...

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Tracing Functions

name has changed to
`name + " " + digit_name(part / 100) + "hundred"`
which is the string "four hundred",

part has changed to `part % 100`, or 16.

Cross out the old values and write the new ones.

int_name(number = 416)	
part	name
416	...
16	"four hundred"

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Tracing Functions

If `digit_name`'s parameter had been complicated, you would have started another sheet of paper to trace that function call.

Your work table will probably be covered with sheets of paper (or envelopes) by the time you are done tracing!

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Tracing Functions

Let's continue...

Here is the status of the parameters and variables now:

int_name(number = 416)	
part	name
416	...
16	"four hundred"

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Tracing Functions

The test `(part >= 20)` is false but the test `(part >= 10)` is true so that code is executed.

```
if (part >= 20)...  
else if (part >= 10) {  
    name = name + " " + teens_name(part);  
    part = 0;  
}
```

teens_name(16) is "sixteen" part is set to 0, so do this:

int_name(number = 416)	
part	name
416	...
16	"four hundred"
0	"four hundred sixteen"

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Tracing Functions

Why is part set to 0?

```
if (part >= 20)...  
else if (part >= 10) {  
    name = name + " " + teens_name(part);  
    part = 0;  
}  
if (part > 0)  
{  
    name = name + " " + digit_name(part);  
}
```

After the `if-else` statement ends, name is complete.

The test in the following `if` statement needs to be "fixed" so that part of the code will not be executed
- nothing should be added to name.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stubs

- When writing a larger program, it is not always feasible to implement and test all functions at once.
- You often need to test a function that calls another, but the other function hasn't yet been implemented.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stubs

- You can temporarily replace the body of function yet to be implemented with a stub.
- A stub is a function that returns a simple value that is sufficient for testing another function.
- It might also have something written on the screen to help you see the order of execution.
- Or, do both of these things.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stubs

Here are examples of stub functions.

```
/**
 * Turns a digit into its English name.
 * @param digit an integer between 1 and 9
 * @return the name of digit ("one" ... "nine")
 */
string digit_name(int digit)
{
    return "mumble";
}

/**
 * Gives the name of the tens part of a number between 20 and 99.
 * @param number an integer between 20 and 99
 * @return the tens name of the number ("twenty" ... "ninety")
 */
string tens_name(int number)
{
    return "mumblety";
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Stubs

If you combine these stubs with the completely written `int_name` function and run the program testing with the value 274, this will be the result:

```
Please enter a positive integer: 274
mumble hundred mumblety mumble
```

which everyone knows indicates that the basic logic of the `int_name` function is working correctly.

(OK, only you know, but that is the important thing with stubs)

Now that you have tested `int_name`, you would "unstubby" another stub function, then another...

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Variable Scope



?



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Variable Scope



Which main?



Variable Scope

**You can only have one `main` function
but you can have as many variables and parameters
spread amongst as many functions as you need.**

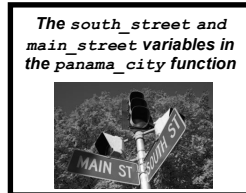
Can you have the same name in different functions?

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

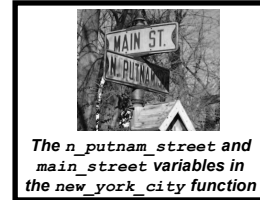
Variable Scope



**The `railway_avenue` and
`main_street` variables in
the `oklahoma_city` function**



**The `south_street` and
`main_street` variables in
the `panama_city` function**



**The `n_putnam_street` and
`main_street` variables in
the `new_york_city` function**

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Variable Scope

**A variable or parameter that is defined within a
function
is visible from the point at which it is defined until
the end of the block named by the function.**

This area is called the scope of the variable.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Variable Scope

**The scope of a variable is the part of
the program in which it is visible.**

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Variable Scope

**The scope of a variable is the part of
the program in which it is visible.**

**Because scopes do not overlap,
a name in one scope cannot
conflict with any name in another
scope.**

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Variable Scope

**The scope of a variable is the part of
the program in which it is visible.**

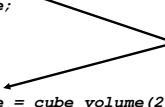
**Because scopes do not overlap,
a name in one scope cannot
conflict with any name in another
scope.**

**A name in one scope is “invisible”
in another scope**

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Variable Scope

```
double cube_volume(double side_len)
{
    double volume = side_len * side_len * side_len;
    return volume;
}
int main()
{
    double volume = cube_volume(2);
    cout << volume << endl;
    return 0;
}
```



Each volume variable is defined in a separate function, so there is not a problem with this code.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Variable Scope

Because of scope, when you are writing a function you can focus on choosing variable and parameter names that make sense for your function.

You do not have to worry that your names will be used elsewhere.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Variable Scope

Names inside a block are called local to that block.

A function names a block.

Recall that variables and parameters do not exist after the function is over—because they are local to that block.

But there are other blocks.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

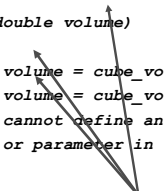
Variable Scope

It is not legal to define two variables or parameters with the same name in the same scope.

For example, the following is not legal:

ERRORS!!!

```
int test(double volume)
{
    double volume = cube_volume(2);
    double volume = cube_volume(10);
    // ERROR: cannot define another volume variable
    // ERROR: or parameter in the same scope
    ...
}
```

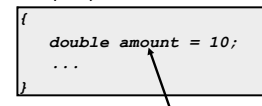


C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Variable Scope – Nested Blocks

However, you can define another variable with the same name in a nested block.

```
double withdraw(double balance, double amount)
{
    if (...)
    {
        double amount = 10;
        ...
    }
    ...
}
```



**a variable named amount local to the if's block
– and a parameter variable named amount.**

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Variable Scope – Nested Blocks

The scope of the parameter variable amount is the entire function, except the nested block.

Inside the nested block, amount refers to the local variable that was defined in that block.

You should avoid this potentially confusing situation in the functions that you write, simply by renaming one of the variables.

Why should there be a variable with the same name in the same function?

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Global Variables

- Generally, global variables are not a good idea.

But ...

here's what they are and how to use them

(if you must).

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Global Variables

Global variables are defined outside any block.

They are visible to every function defined after them.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Global Variables

In some cases, this is a good thing:

The `<iostream>` header defines these global variables:

```
cin
cout
```

This is good because there should only be one of each of these and every function who needs them should have direct access to them.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Global Variables

But in a banking program, how many functions should have direct access to a balance variable?

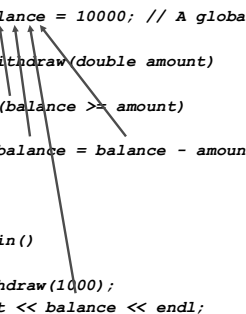
C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Global Variables

```
int balance = 10000; // A global variable

void withdraw(double amount)
{
    if (balance >= amount)
    {
        balance = balance - amount;
    }
}

int main()
{
    withdraw(1000);
    cout << balance << endl;
    return 0;
}
```



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Global Variables

In the previous program there is only one function that updates the balance variable.

But there could be many, many, many functions that might need to update balance each written by any one of a huge number of programmers in a large company.

Then we would have a problem.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Global Variables

When multiple functions update global variables, the result can be difficult to predict.

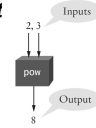
Particularly in larger programs that are developed by multiple programmers, it is very important that the effect of each function be clear and easy to understand.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Global Variables – Breaking Open the Black Box

When functions modify global variables, it becomes more difficult to understand the effect of function calls.

Programs with global variables are difficult to maintain and extend because you can no longer view each function as a “black box” that simply receives parameter values and returns a result something.

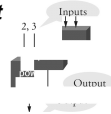


C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Global Variables – Breaking Open the Black Box

When functions modify global variables, it becomes more difficult to understand the effect of function calls.

Programs with global variables are difficult to maintain and extend because you can no longer view each function as a “black box” that simply receives parameter values and returns a result something.



And what good is a broken black box?

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Global Variables – Just Say “No”

*You should **avoid** global variables in your programs!*

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

- Suppose you would like a function to get the user's last name and ID number.
- The variables for this data are in your scope.
- But you want the function to change them for you.
- If you want to write a function that changes the value of a parameter, you must use a reference parameter.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

To understand the need for a different kind of parameter, you must first understand why the parameters you now know do not work.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

Consider a function that simulates withdrawing a given amount of money from a bank account, provided that sufficient funds are available.

If the amount of money is insufficient, a \$10 penalty is deducted instead.

The function would be used as follows:

```
double harrys_account = 1000;
withdraw(harrys_account, 100);
// Now harrys_account is 900
withdraw(harrys_account, 1000);
// Insufficient funds.
// Now harrys_account is 890
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

Here is a first attempt:

```
void withdraw(double balance, double amount)
{
    const double PENALTY = 10;
    if (balance >= amount)
    {
        balance = balance - amount;
    }
    else
    {
        balance = balance - PENALTY;
    }
}
```

But this doesn't work.

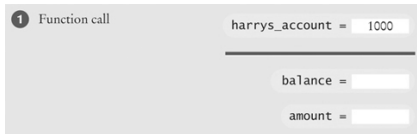
C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

What is actually happening?

Let's call the function passing in 100 to be taken from harrys_account.

```
double harrys_account = 1000;
withdraw(harrys_account, 100);
```

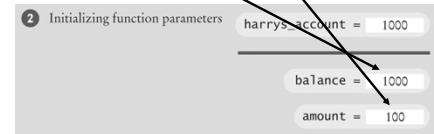


C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

The local variables, consts, and value parameters are initialized.

```
double harrys_account = 1000;
...
withdraw(harrys_account, 100);
...
void withdraw(double balance, double amount)
{
    const int PENALTY = 10;
```



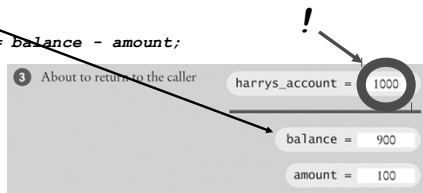
C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

The test is false, the LOCAL variable balance is updated

```
double harrys_account = 1000;
...
else
{
    balance = balance - amount;
}
```

NOTHING happens to harrys_balance because it is a separate variable (in a different scope)



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

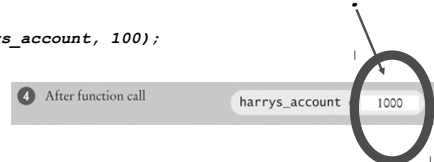
Reference Parameters

The function call has ended.

Local names in the function are gone and...

NOTHING happened to harrys_balance.

```
withdraw(harrys_account, 100);
```



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

A reference parameter refers to a variable that is supplied in a function call.

“refers” means that during the execution of the function, the reference parameter name is another name for the caller’s variable.

This “referring” is how a function can change non-local variables:

changes to its local parameters’ names cause changes to the callers variables they “refer” to.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

To indicate a reference parameter, you place an & after the type name.

```
void withdraw(double& balance, double amount)
```

To indicate a value parameter, you do not place an & after the type name.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

Here is correct code, using reference parameters:

```
void withdraw(double& balance, double amount)
{
    const int PENALTY = 10;
    if (balance >= amount)
    {
        balance = balance - amount;
    }
    else
    {
        balance = balance - PENALTY;
    }
}
```

Let’s see this in action.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

Now, using the function with reference parameters let’s again call the function passing in 100 to be taken from harrys_account.

```
double harrys_account = 1000;
withdraw(harrys_account, 100);
```

1 Function call

harrys_account =	1000
withdraw(harrys_account, 100);	
balance =	
amount =	

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

Notice that balance now refers to harrys_account.

```
double harrys_account = 1000;
...
withdraw(harrys_account, 100);
...
void withdraw(double balance, double amount)
{
    const int PENALTY = 10;
```

2 Initializing function parameters

harrys_account =	1000
withdraw(harrys_account, 100);	
balance =	
amount =	100

harrys_account and balance are both names for this memory

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

The test is false, the variable harrys_account is updated when the reference parameter balance is assigned to.

```
double harrys_account = 1000;
...
else
{
    balance = balance - amount;
}
```

recall that harrys_account and balance are both names for this memory

3 About to return to the caller

harrys_account =	900
balance = balance - amount;	
balance =	
amount =	100

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

**The function call has ended.
Local names in the function are gone and...**

harrys_balance was correctly changed!

```
withdraw(harrys_account, 100);
```

After function call

harrys_account = 900

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

ch05/account.cpp

```
/**  
 * Withdraws the amount from the given balance, or withdraws  
 * a penalty if the balance is insufficient.  
 * @param balance the balance from which to make the  
 * withdrawal  
 * @param amount the amount to withdraw  
 */  
void withdraw(double& balance, double amount)  
{  
    const double PENALTY = 10;  
    if (balance >= amount)  
    {  
        balance = balance - amount;  
    }  
    else  
    {  
        balance = balance - PENALTY;  
    }  
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

ch05/account.cpp

```
int main()  
{  
    double harrys_account = 1000;  
    double sallys_account = 500;  
    withdraw(harrys_account, 100);  
    // Now harrys_account is 900  
    withdraw(harrys_account, 1000); // Insufficient funds  
    // Now harrys_account is 890  
    withdraw(sallys_account, 150);  
    cout << "Harry's account: " << harrys_account << endl;  
    cout << "Sally's account: " << sallys_account << endl;  
  
    return 0;  
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

The type `double&` is pronounced:

**reference to double
or
double ref**

**(The type `double` is, of course, pronounced:
double)**

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

A reference parameter must always be called with a variable.

It would be an error to supply a number:

```
withdraw(1000, 500);  
// Error: reference parameter must be a variable
```

The reason is clear—the function modifies the reference parameter, but it is impossible to change the value of a number.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Reference Parameters

For the same reason, you cannot supply an expression:

```
withdraw(harrys_account + 150, 500);  
// Error: reference parameter must be a variable
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Prefer Return Values to Reference Parameters

Some programmers use reference parameters as a mechanism for setting the result of a function.

For example:

```
void cube_volume(double side_length, double& volume)
{
    volume = side_length * side_length * side_length;
}
```

However, this function is less convenient than our previous `cube_volume` function.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Prefer Return Values to Reference Parameters

```
void cube_volume(double side_length, double& volume)
{
    volume = side_length * side_length * side_length;
}
```

This function cannot be used in expressions such as:

```
cout << cube_volume(2)
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Prefer Return Values to Reference Parameters

Another consideration is that the `return` statement can return only one value.

If caller wants more than two values, then the only way to do this is with reference parameters (one for each wanted value).

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Constant References

It is not very efficient to have a value parameter that is a large object (such as a string value).

Copying the object into a parameter variable is less efficient than using a reference parameter.

With a reference parameter, only the location of the variable, not its value, needs to be transmitted to the function.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Constant References

You can instruct the compiler to give you the efficiency of a reference parameter and the meaning of a value parameter, by using a constant reference:

```
void shout(const string& str)
{
    cout << str << "!!!" << endl;
}
```

This is a bit more efficient than having `str` be a value parameter.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

CHAPTER SUMMARY

Understand the concepts of functions, arguments, and return values.



- A function is a named sequence of instructions.
- Arguments are supplied when a function is called. The return value is the result that the function computes.

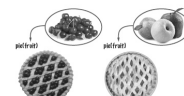
Be able to implement functions.



- When defining a function, you provide a name for the function, a variable for each argument, and a type for the result.
- Function comments explain the purpose of the function, the meaning of the parameter variables and return value, as well as any special requirements.

Describe the process of parameter passing.

- Parameter variables hold the argument values supplied in the function call.



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

CHAPTER SUMMARY

Describe the process of returning a value from a function.

- The return statement terminates a function call and yields the function result.



Design and implement functions without return values.



- Use a return type of void to indicate that a function does not return a value.

Develop functions that can be reused for multiple problems.

- Eliminate replicated code or pseudocode by defining a function.
- Design your functions to be reusable. Supply parameter variables for the values that can vary when the function is reused.

Apply the design principle of stepwise refinement.



- Use the process of stepwise refinement to decompose complex tasks into simpler ones.
- When you discover that you need a function, write a description of the parameter variables and return values.
- A function may require simpler functions to carry out its work.

CHAPTER SUMMARY

Determine the scope of variables in a program.

- The scope of a variable is the part of the program in which it is visible.
- A variable in a nested block shadows a variable with the same name in an outer block.
- A local variable is defined inside a function. A global variable is defined outside a function.
- Avoid global variables in your programs.



Describe how reference parameters work.

- Modifying a value parameter has no effect on the caller.
- A reference parameter refers to a variable that is supplied in a function call.
- Modifying a reference parameter updates the variable that was supplied in the call.



End Chapter Five