Chapter 4

Functions

OBJECTIVES

After studying this chapter you will be able to:

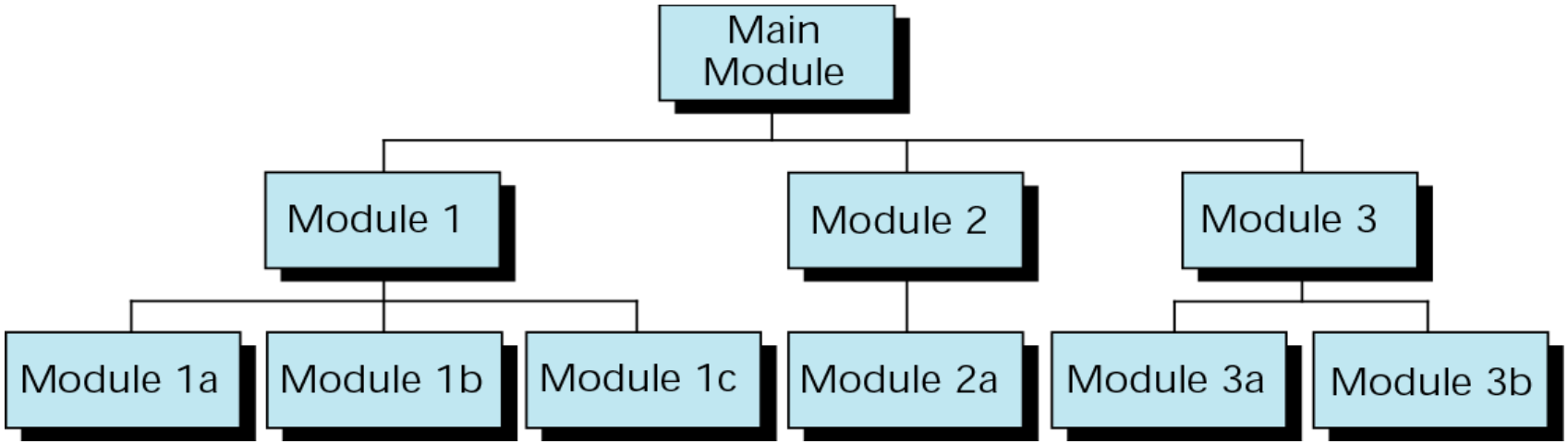
- Structure a program into functions designed to do a specific task.
- Understand the relationship between parameters and arguments
- Understand the concept of local variables.
- Pass data to a function using pass-by-value or pass-by-reference.
- Use a structure chart to design a program.
- Write function prototype statements.
- Write programs with multiple functions.
- Use mathematical and standard library functions in programs.
- Understand the concept of scope in a C++ program.
- Understand the basic concept of functional cohesion.
- Implement a program using top-down development and stubs.

DESIGNING STRUCTURED PROGRAMS

Note:

In top-down design, a program is divided into a main module and its related modules. Each module is in turn divided into submodules until the resulting modules are intrinsic; that is, until they are implicitly understood without further division.

Figure 4-1 Structure chart



Note:

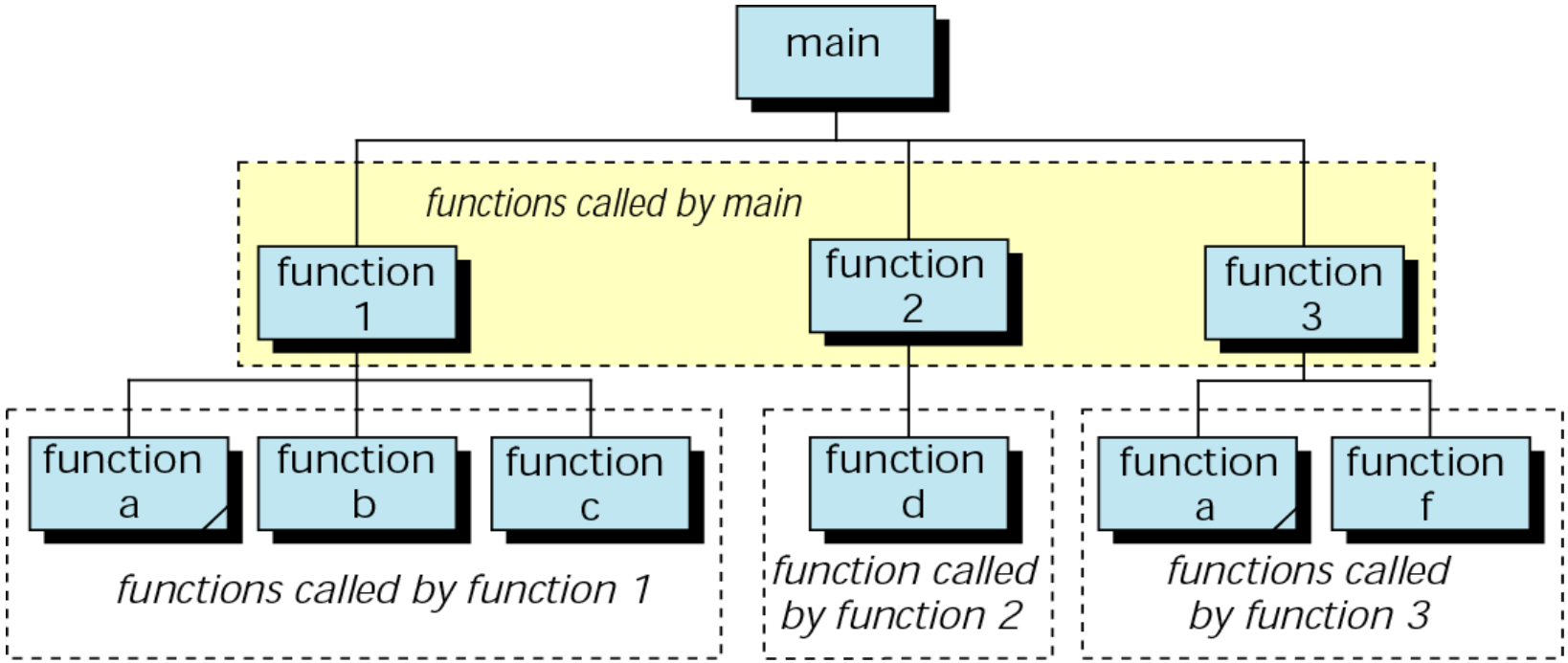
In a structure chart, a module can be called by one and only one higher module.

FUNCTIONS IN C++

Note:

In C++, a program is made of one or more functions, one and only one of which must be named main. The execution of the program always starts with main, but it can call other functions to do some part of the job.

Figure 4-2 Structure chart for a C++ program

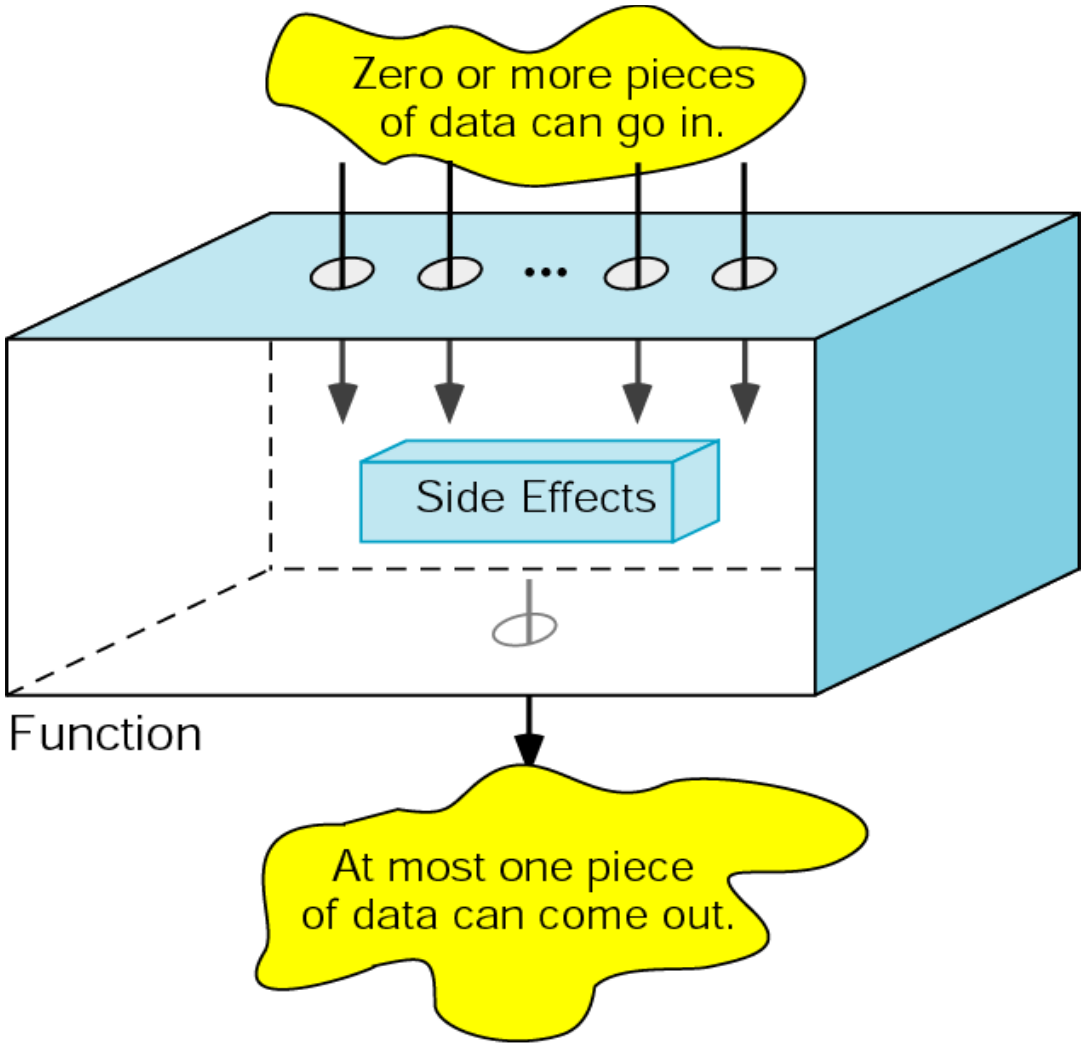


Note:

A function in C++ can have a value, a side effect, or both.

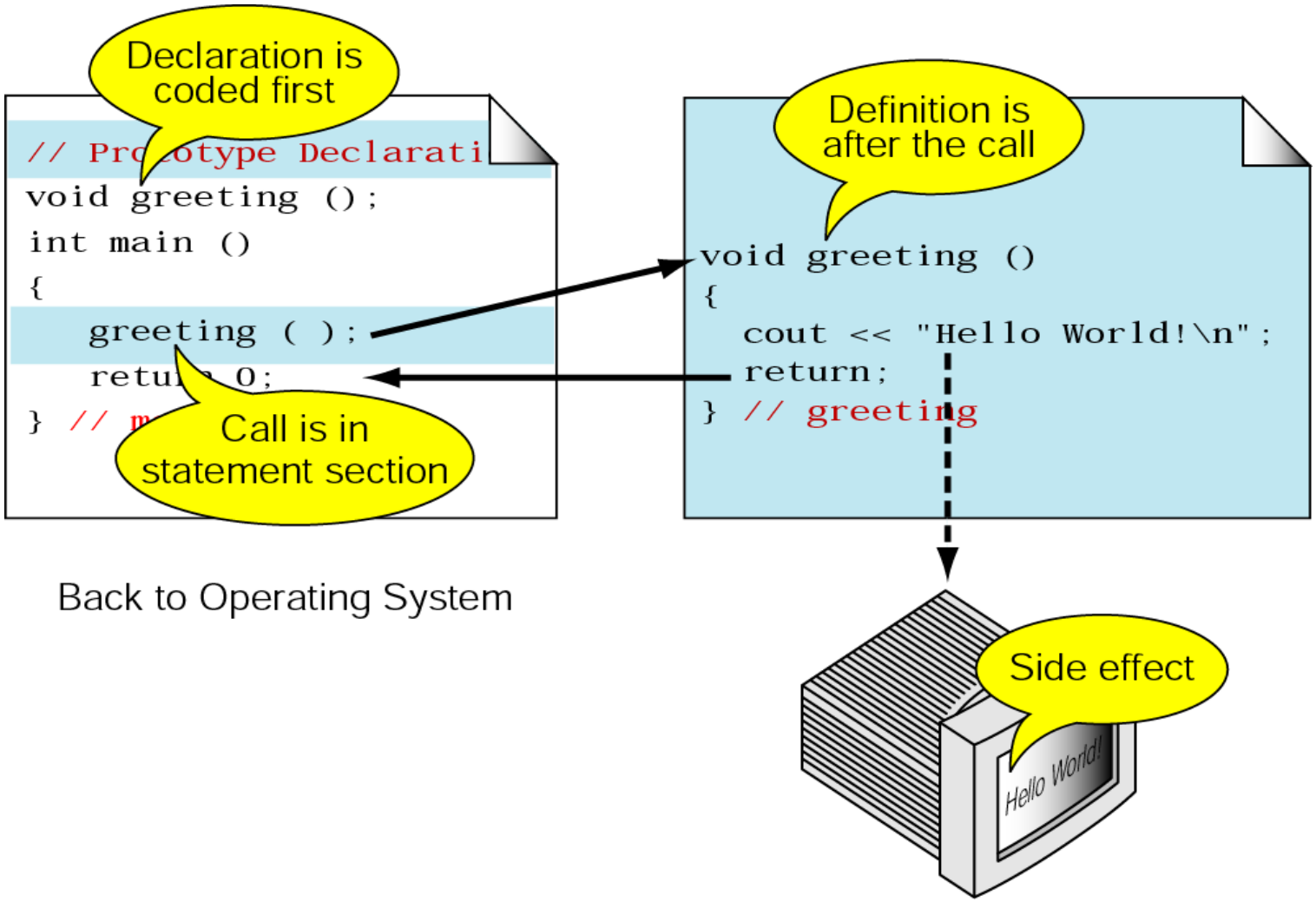
- ***The side effect occurs before the value is returned.***
- ***The function's value is the value of the expression in the return statement.***
- ***A function can be called for its value, its side effect, or both.***

Figure 4-3 Function concept



USER-DEFINED FUNCTIONS

Figure 4-4 Declaring, calling, and defining functions



Note:

The name of a function is used in three ways: for declaration, in a call, and for definition.

Figure 4-5 *void* function with parameters

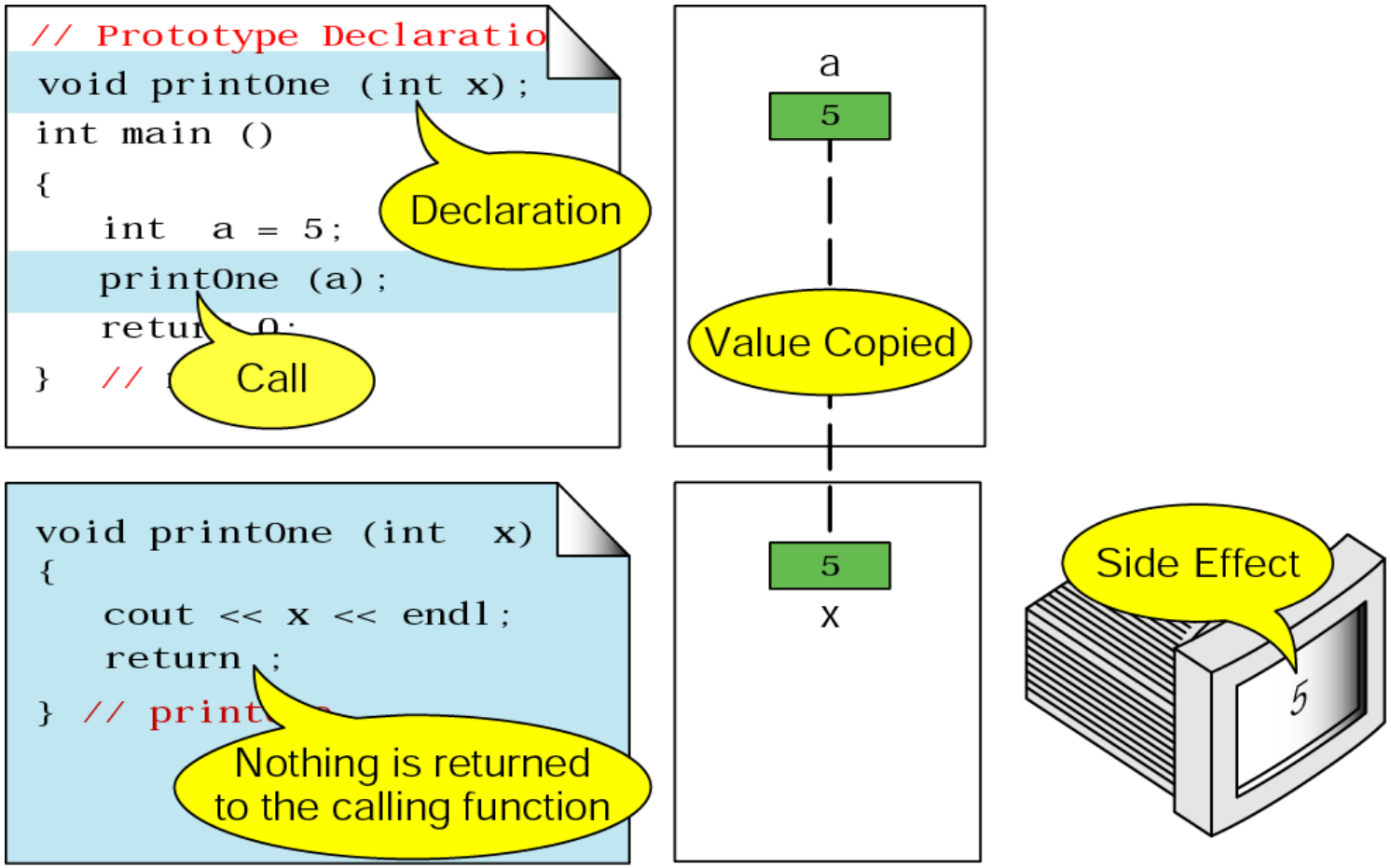
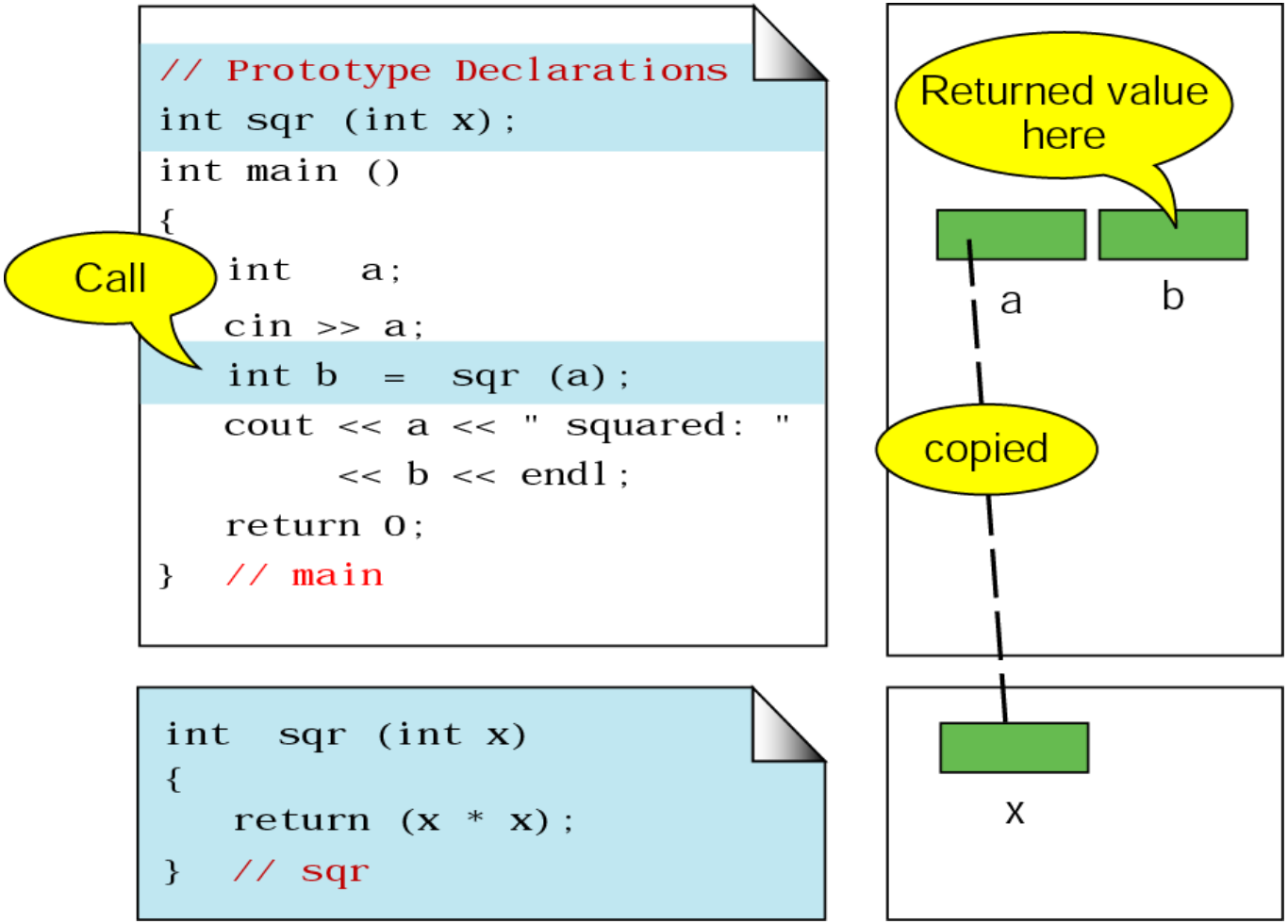


Figure 4-6 Calling a function that returns a value



Note:

void functions cannot be used in an expression; they must be a separate statement.

Functions that return a value may be used in an expression or as a separate statement.

Figure 4-7 Function definition

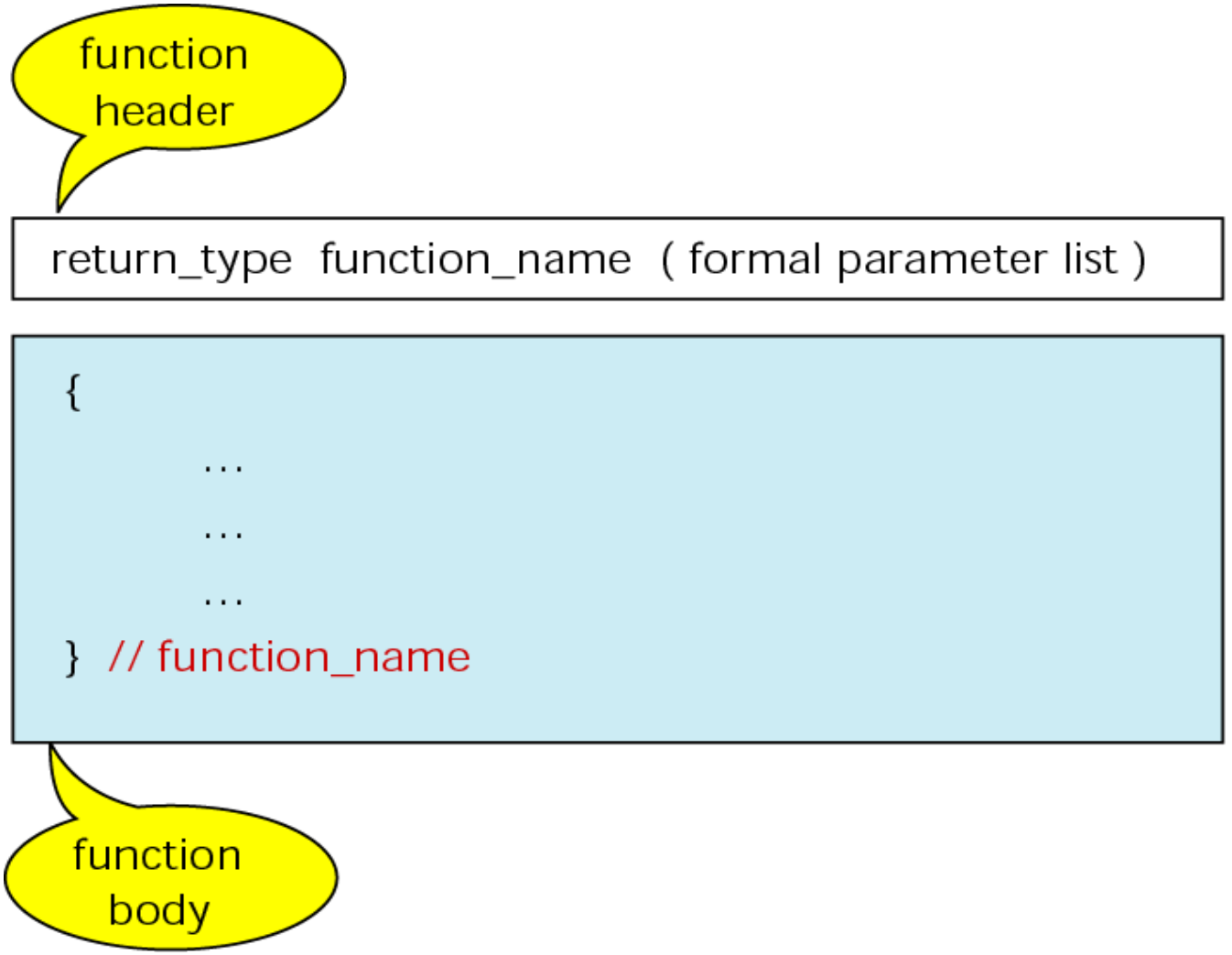
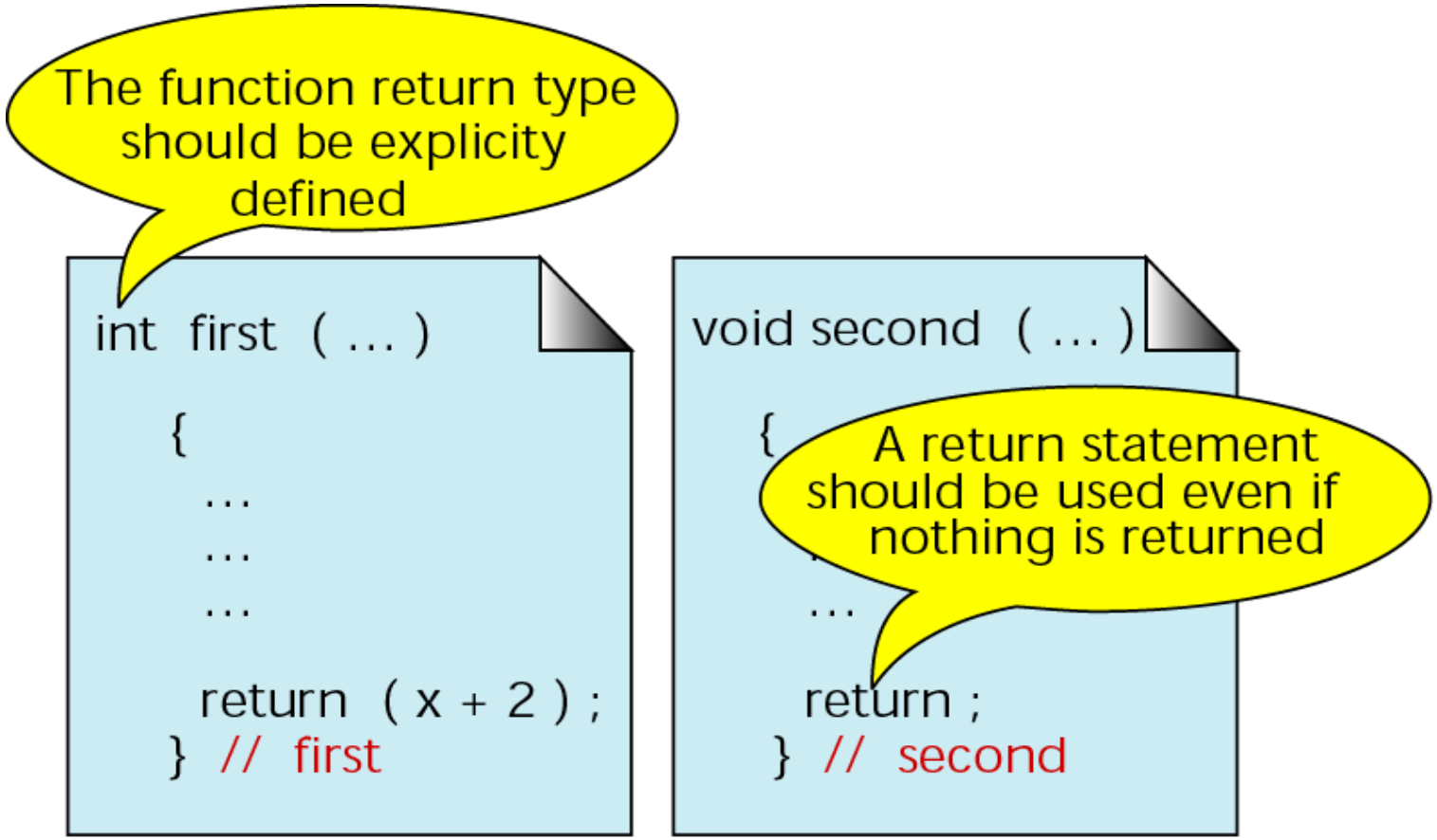


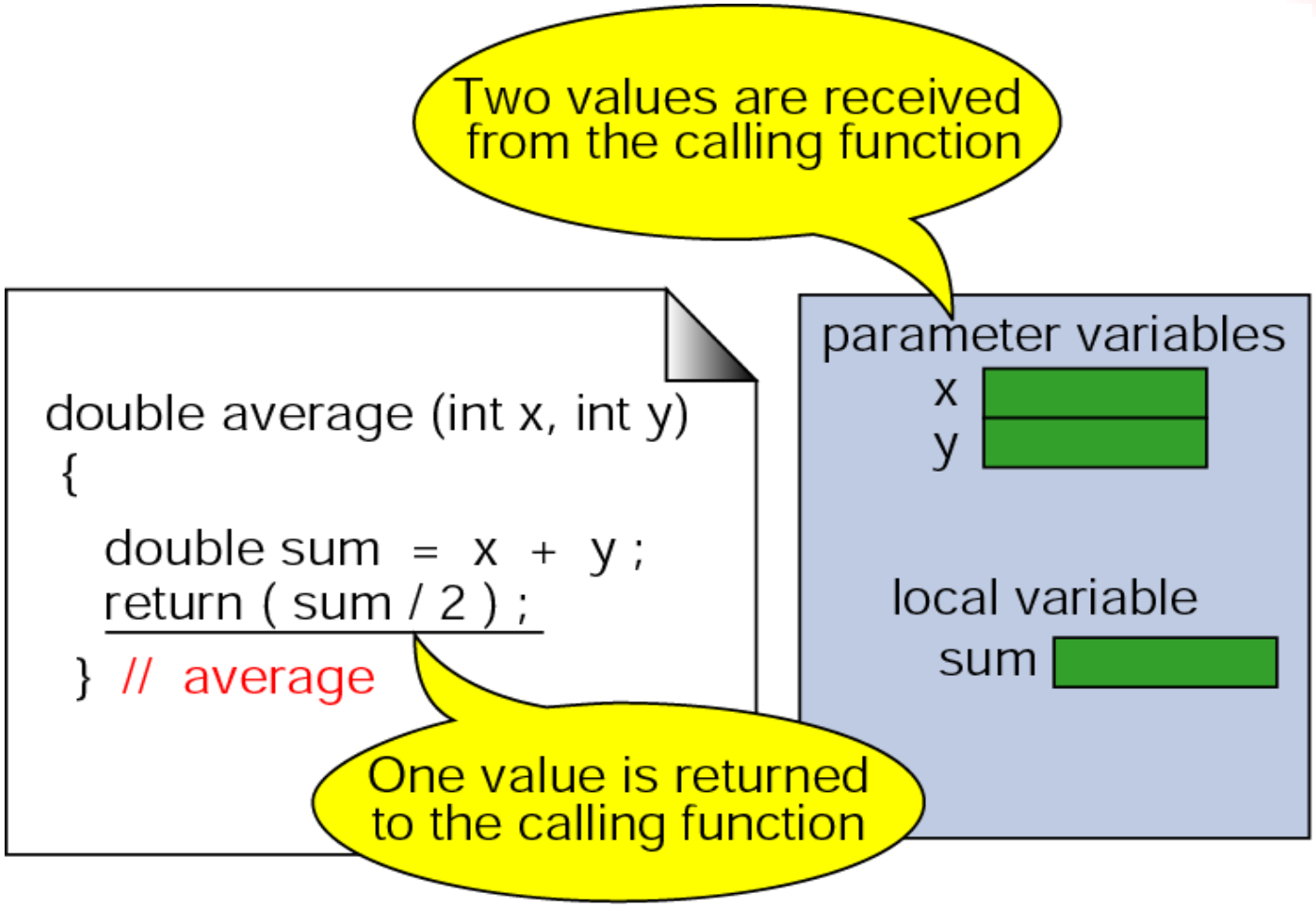
Figure 4-8 Function return statements



Note:

The type of the expression in the return statement must match the return type in the function header.

Figure 4-9 Function local variables



Note:

- ***Formal parameters are variables that are declared in the header of the function definition.***
- ***Actual parameters are the expressions in the calling statement.***
- ***The formal and actual parameters must match exactly in type, order, and number. Their names, however, do not need to be the same.***

Figure 4-10 Parts of a function call

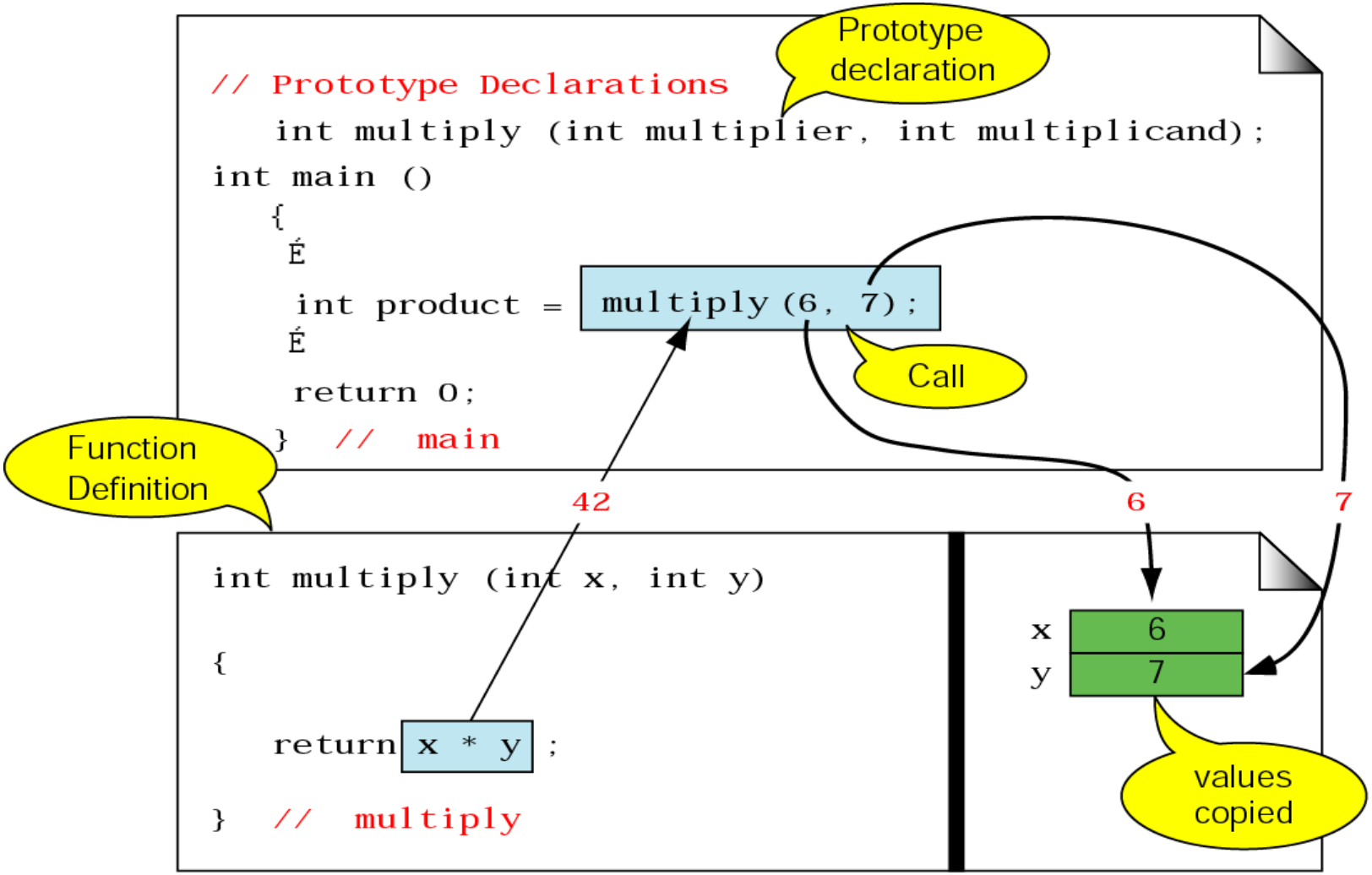


Figure 4-11 Examples of function calls

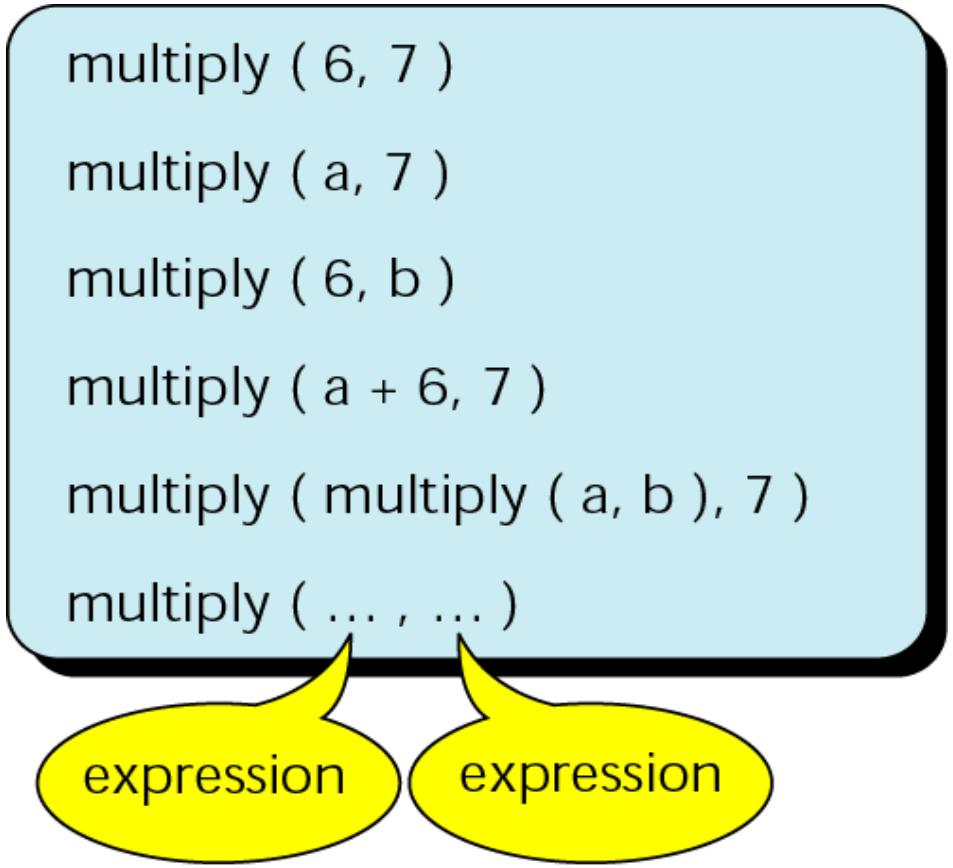
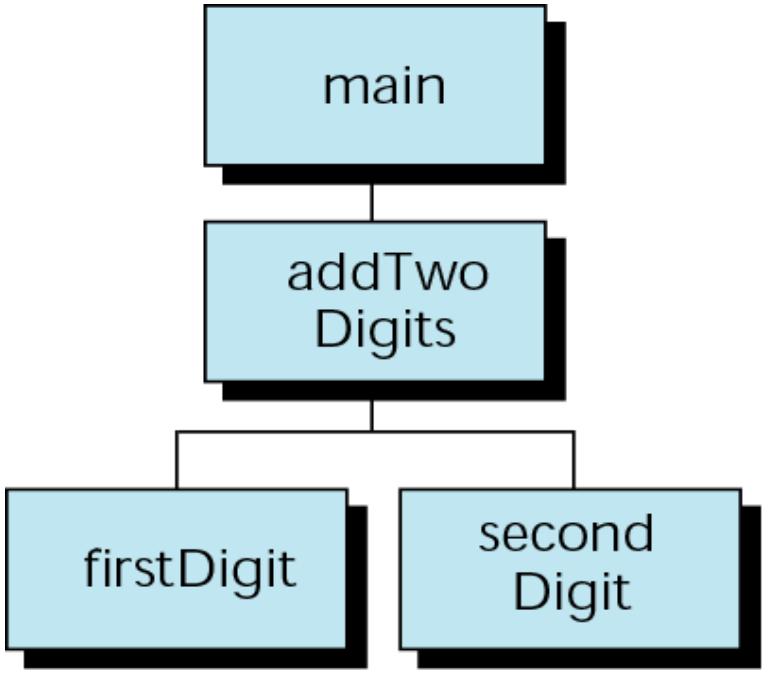


Figure 4-12 Design for addTwoDigits



Note:

It is the nature of the task to be performed, not the amount of code, that determines if a function should be used.

Figure 4-13 Design for Strange College fees

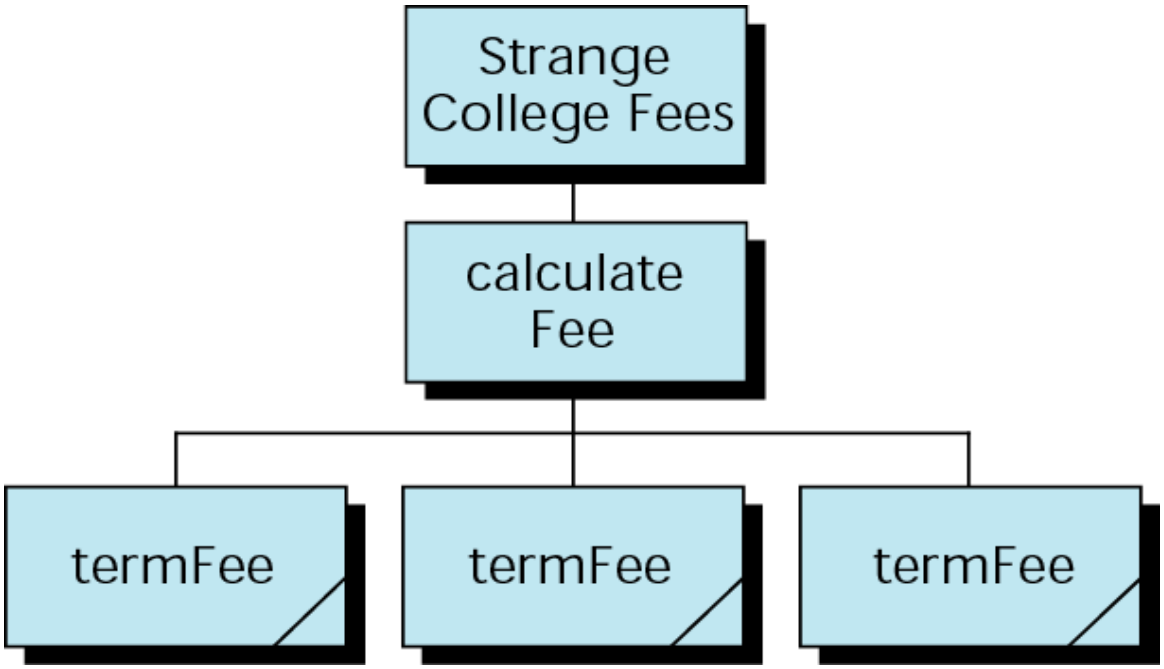


Figure 4-14 Pass by value

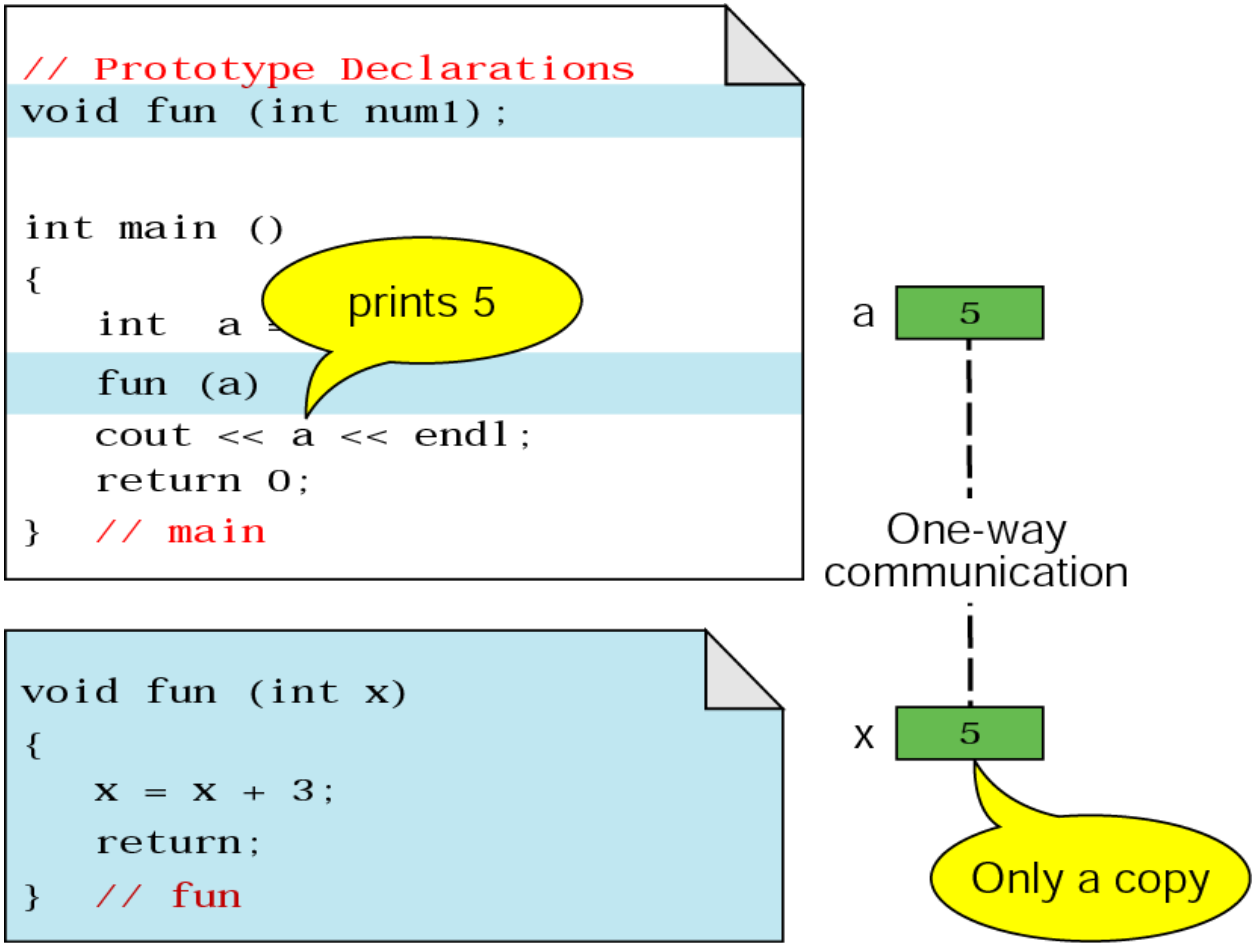


Figure 4-15 Pass by reference

```
// Prototype Statements
   void  exchange   ( int&  num1,
                      int&  num2 ) ;
int  main  ( )
{
   int   a ;
   int   b ;
   ...
   exchange ( a,  b ) ;
   cout << a << " " << b << endl;
   ...
   return  0 ;
}  // main
```

```
void  exchange   ( int&  num1,
                   int&  num2 )
{
      int   hold    =  num1 ;
            num1    =  num2 ;
            num2    =  hold ;
      return  ;
}  // exchange
```

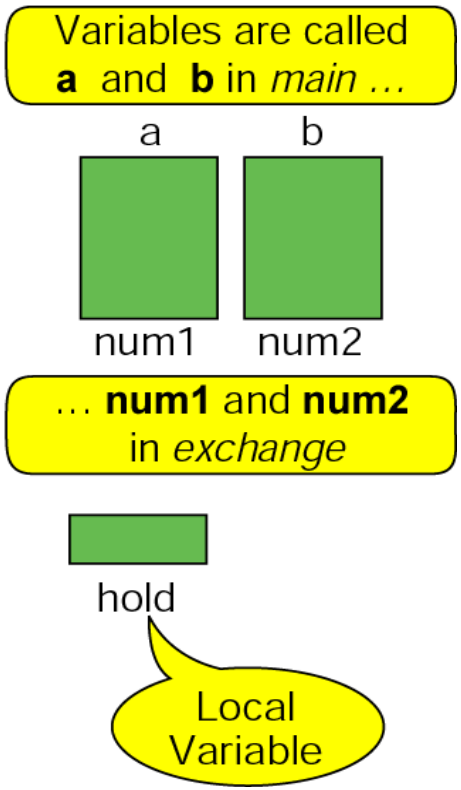


Figure 4-16 A bad exchange

```
// Prototype Statements
void exchange (int num1, int num2);

int  main  ( )
{
    int   a = 7;
    int   b = 5;
    ...
    exchange ( a,  b );
    cout << a << " " << b << endl;
    ...
    return  0;
 }  // main
```

```
void exchange (int num1, int num2)
{
   int  hold    =  num1;
        num1    =  num2;
        num2    =  hold;
   return;
 }  // exchange
```

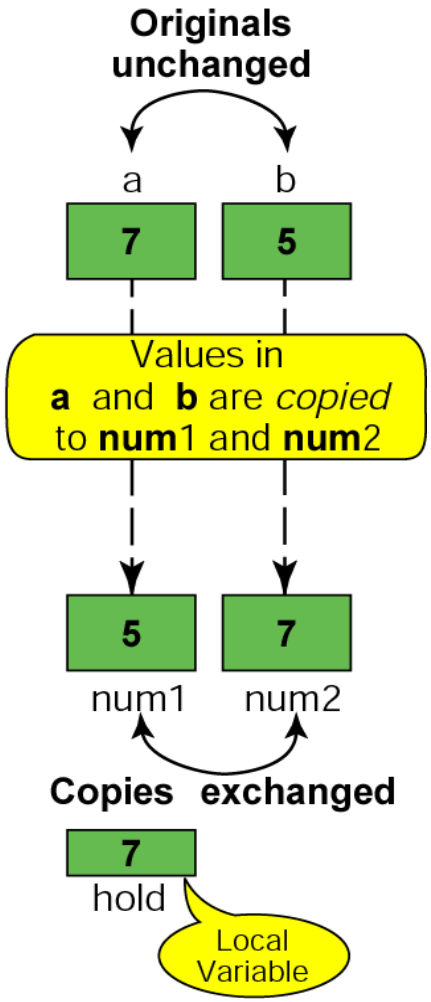


Figure 4-17 Calculate quotient and remainder

```
// Prototype Statements
void  divide ( int    divnd,  int    divsr,
               int&   quot,   int&   rem);

int  main  ( )
{
   int   a;
   int   b;
   int   c;
   int   d;
   …
   divide (a, b, c, d);
   …
   return 0;
}  // main
```

```
void  divide   ( int     divnd,
                 int     divsr,
                 int&    quot,
                 int&    rem )
{
   quot    =  divnd / divsr;
   rem     =  divnd % divsr;
   return  ;
}  // divide
```

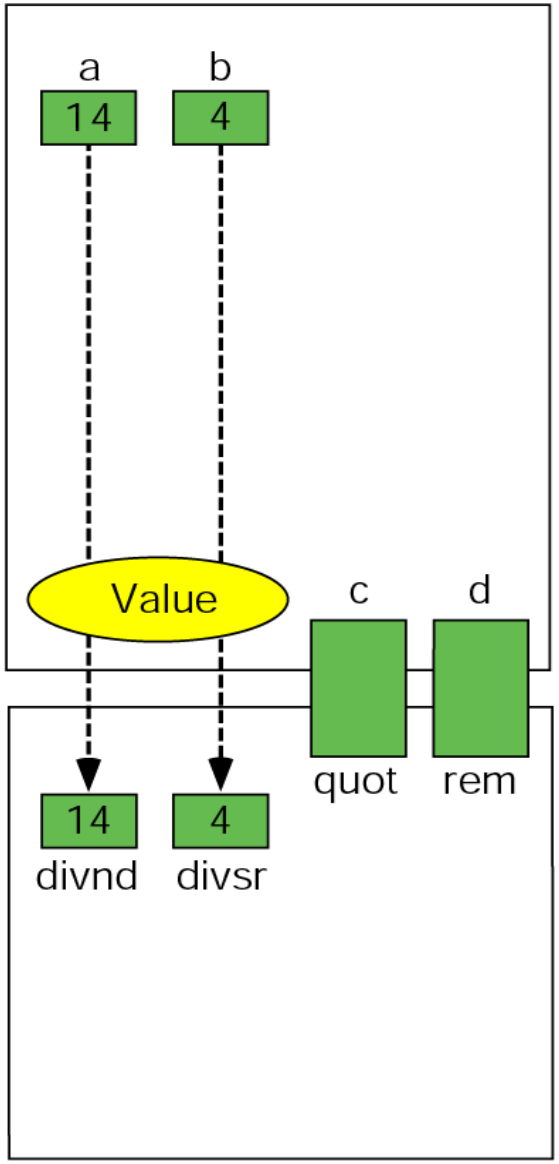
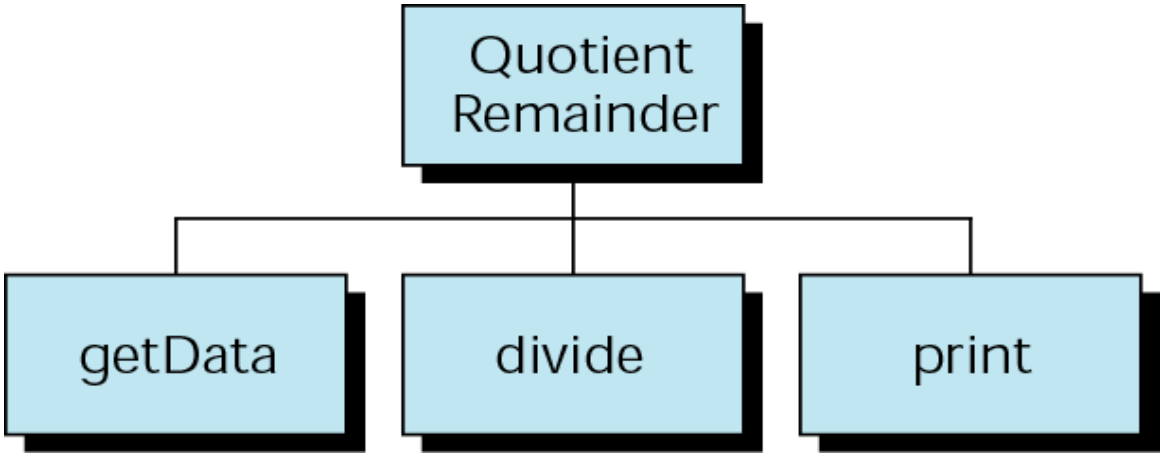


Figure 4-18 Quotient and remainder design



DEFAULT PARAMETER ARGUMENTS

STANDARD LIBRARY FUNCTIONS

Figure 4-19 Library functions and the linker

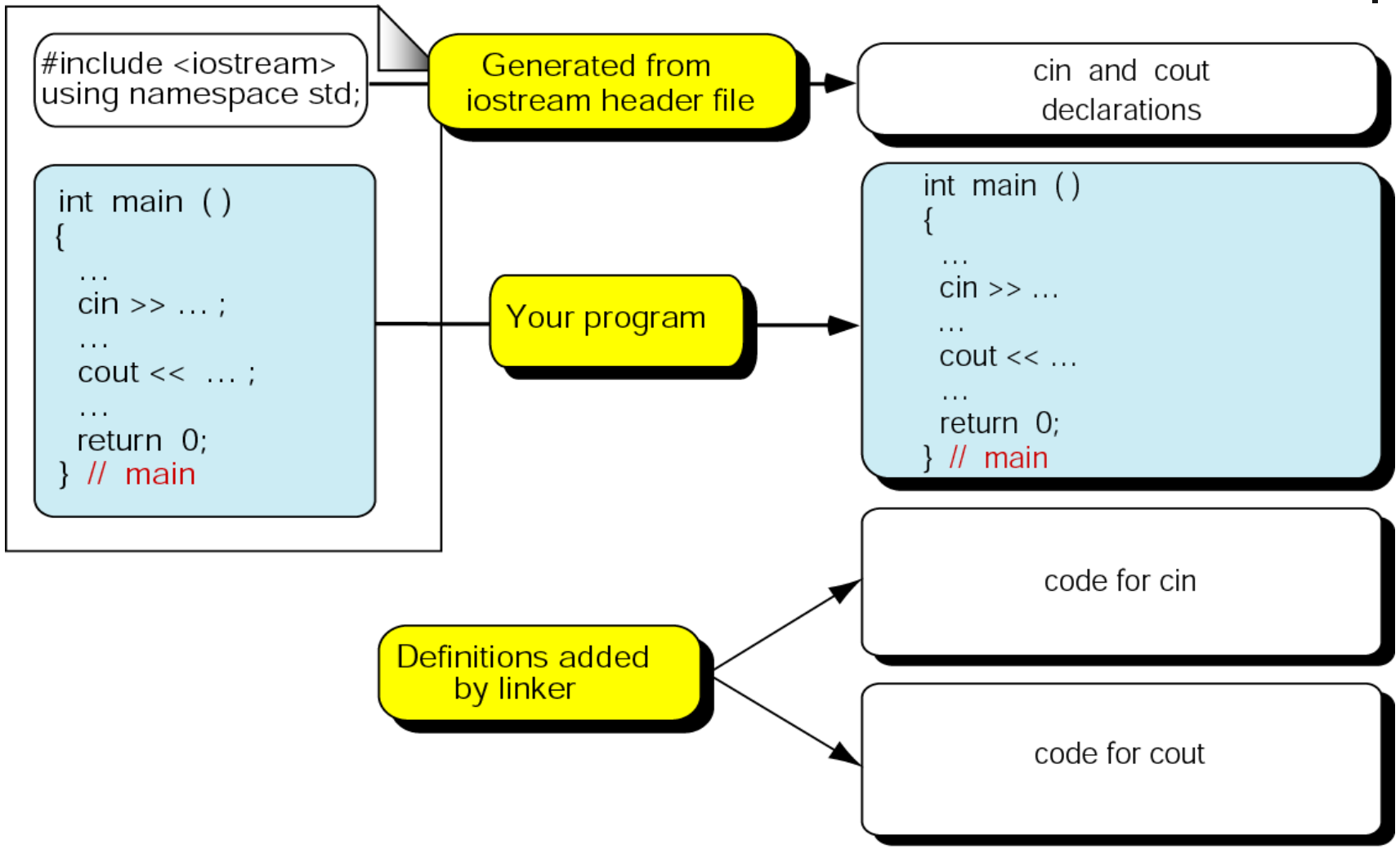


Figure 4-20 Floor and ceiling functions

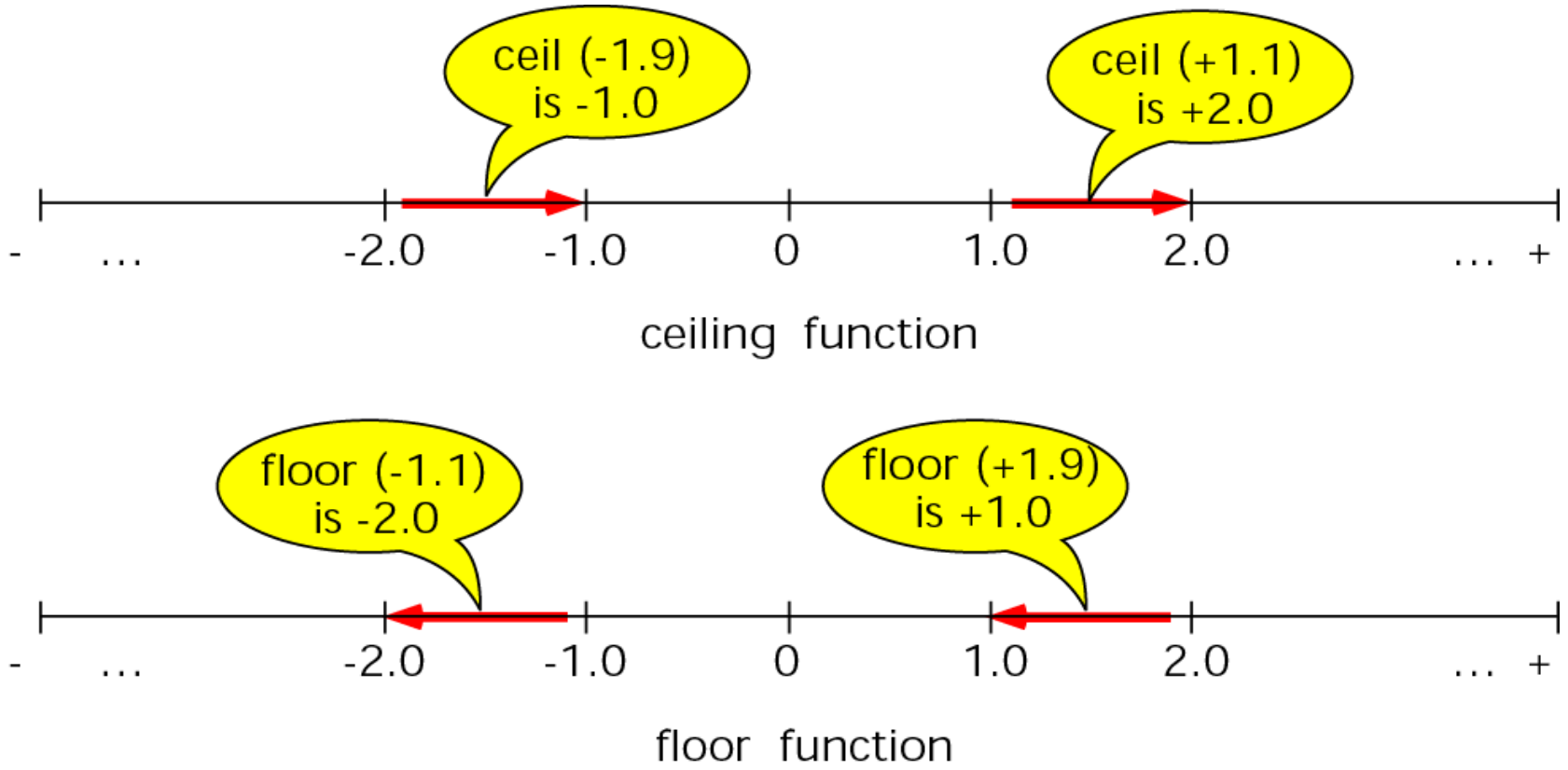


Figure 4-21 The random number seed

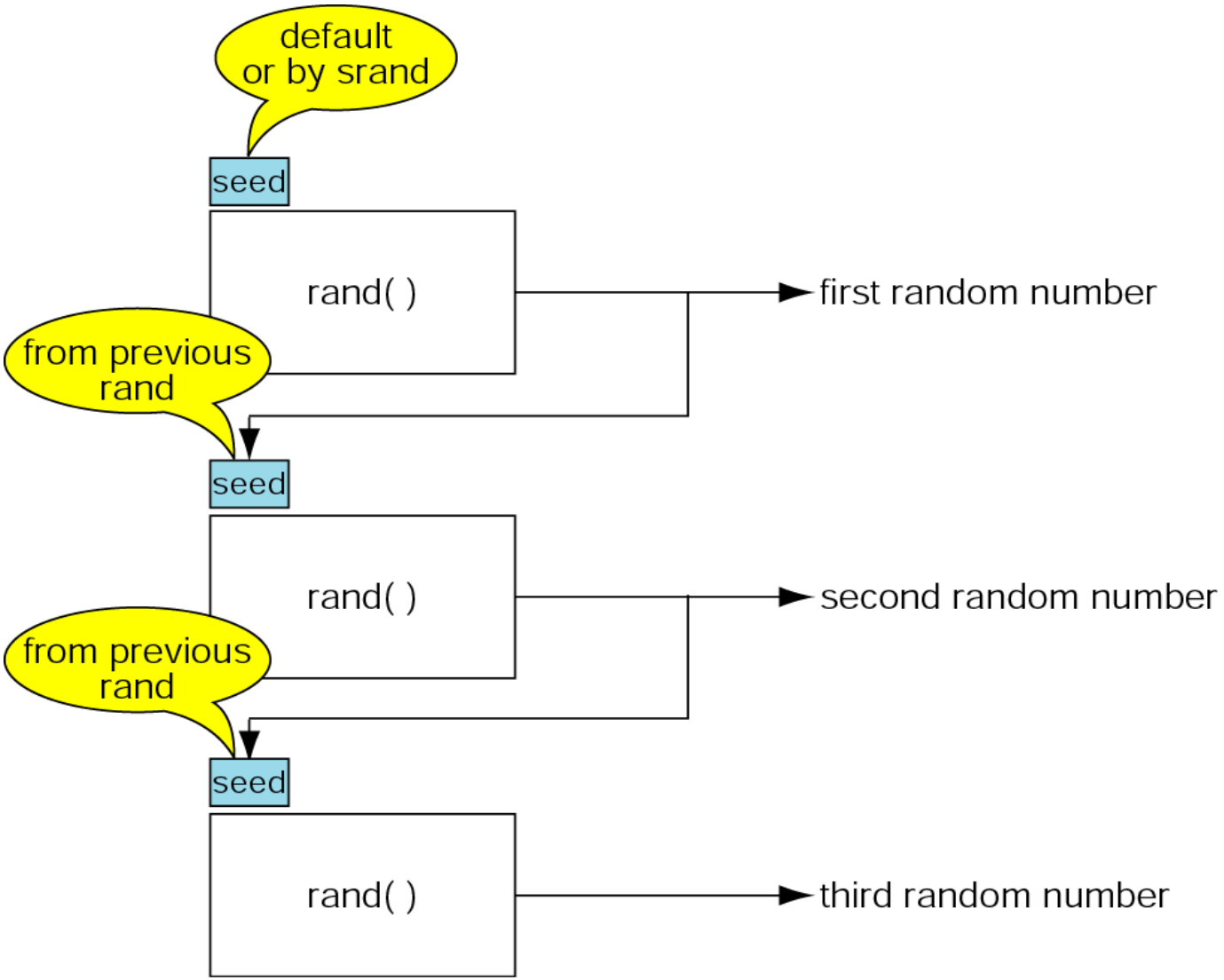
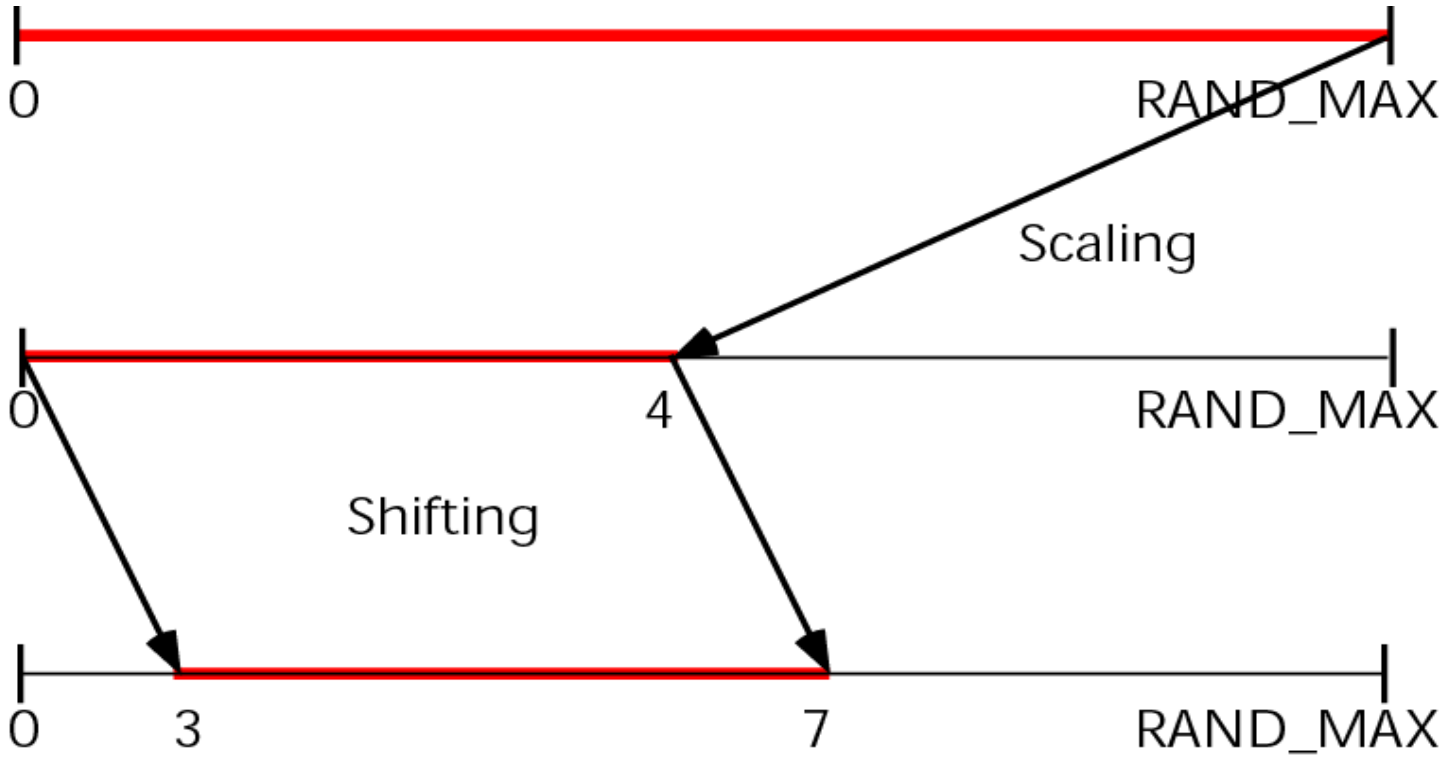
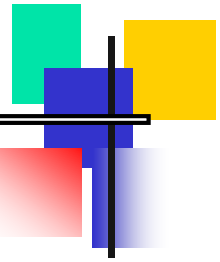


Figure 4-22 Random number scaling for 3-7



SCOPE

Figure 4-23 Scope for global and block areas



```
/*   This is a sample to demonstrate scope. The techniques used in
     this sample should never be used in practice.
*/
#include  <iostream>
using namespace std;
int   fun  (int a, int b);
```

Global Area

```
 int main ( )
 {
  int     a;
  int     b;
  float   y;
  ...
        { // Beginning of nested block
          float  a  =  y  / 2;
          float  y;
          float  z;
          ...
           z =  a * b / y;
          ...
        } // End of nested block
 ...

 } // End of Main
```

Main's Area

Nested Block Area

```
int   fun   (int   i, int   j)
{
  int  a;
  int  y;
  ...
} // fun
```

fun's Area

Note:

Variables are in scope from their point of definition until the end of their function or block.

Note:

It is poor programming style to reuse identifiers within the same scope.

A Programming Example—Calculator Program

Software Engineering and Programming Style

Figure 4-24 Structure chart symbols

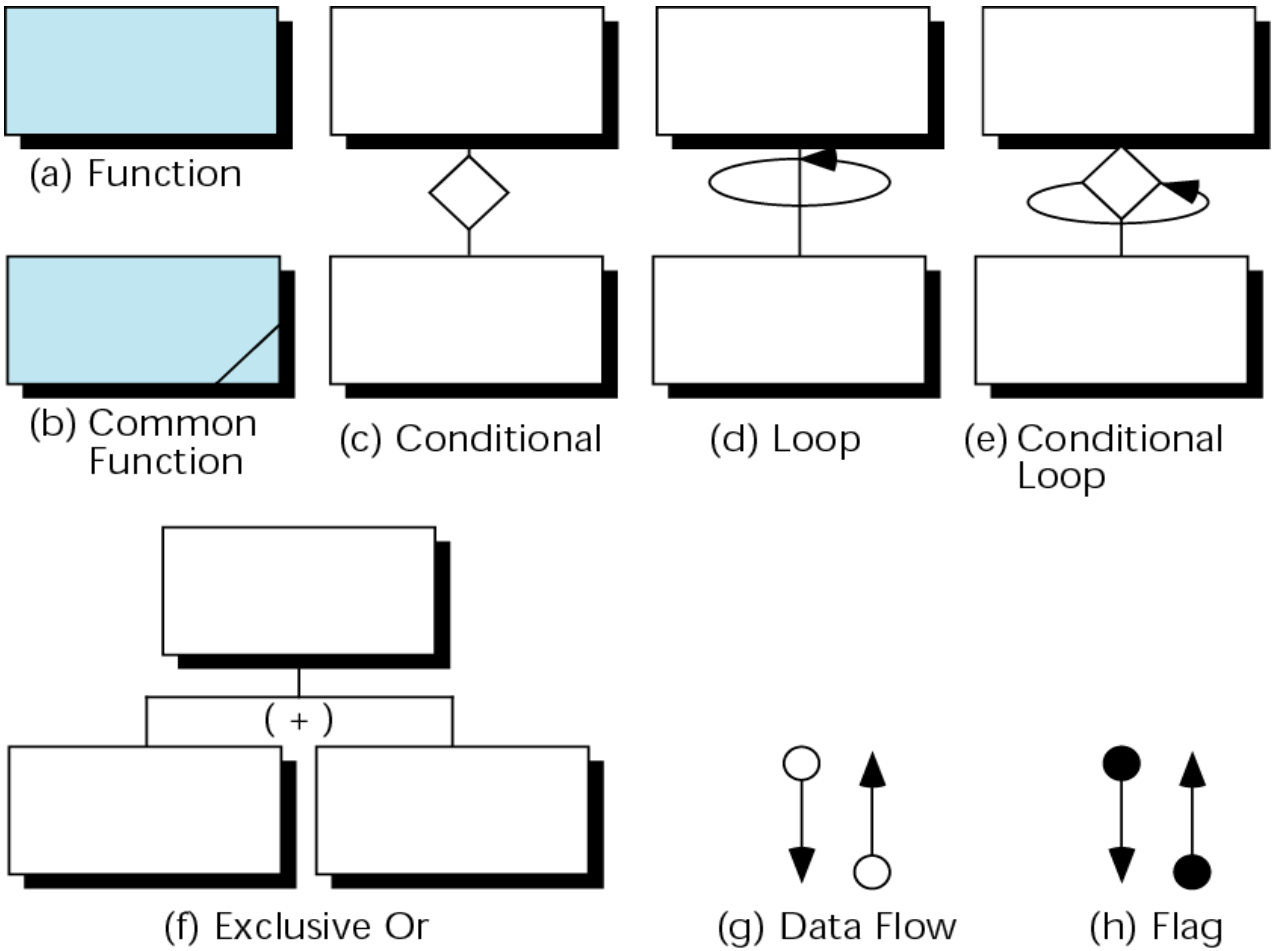
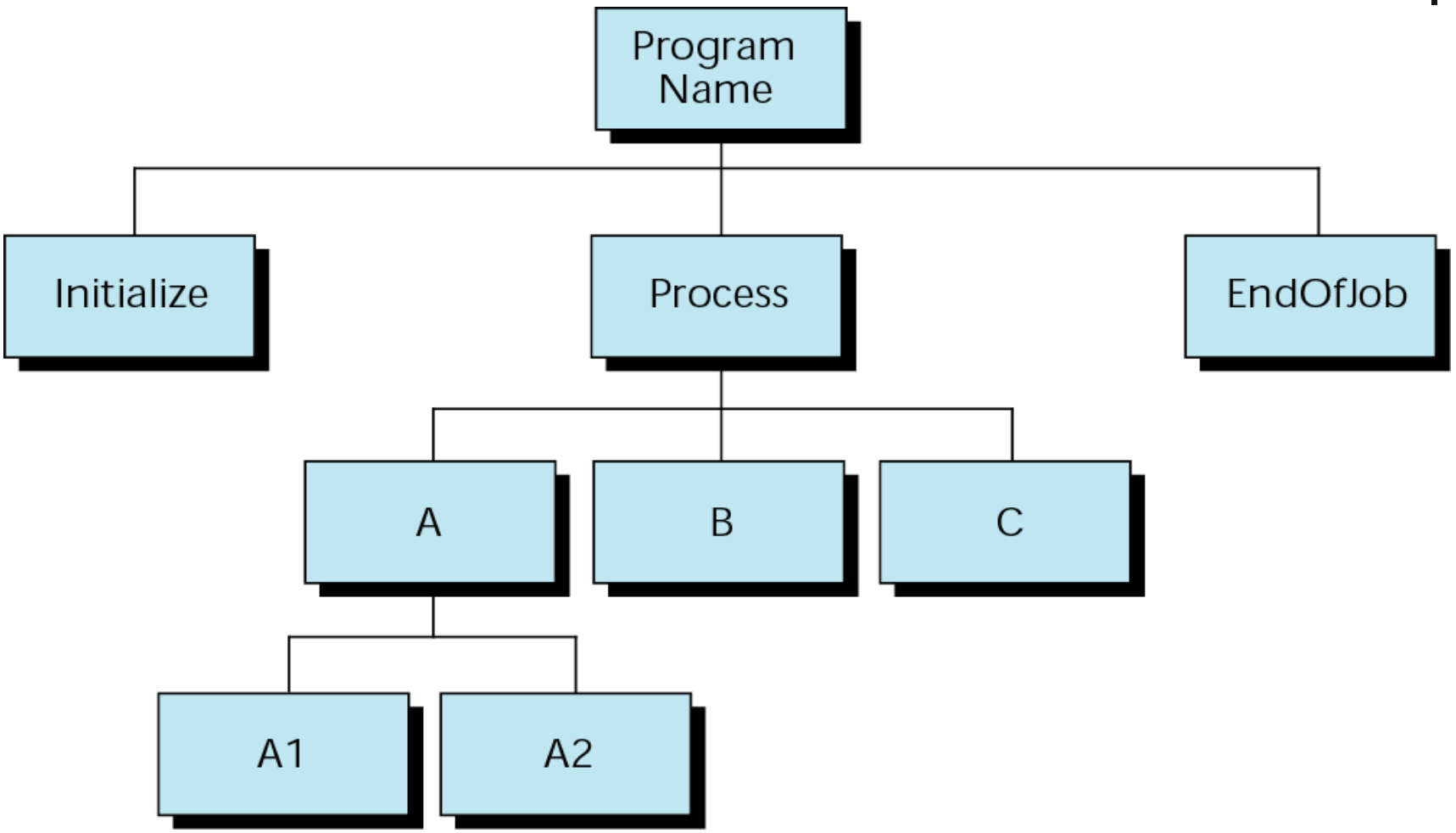


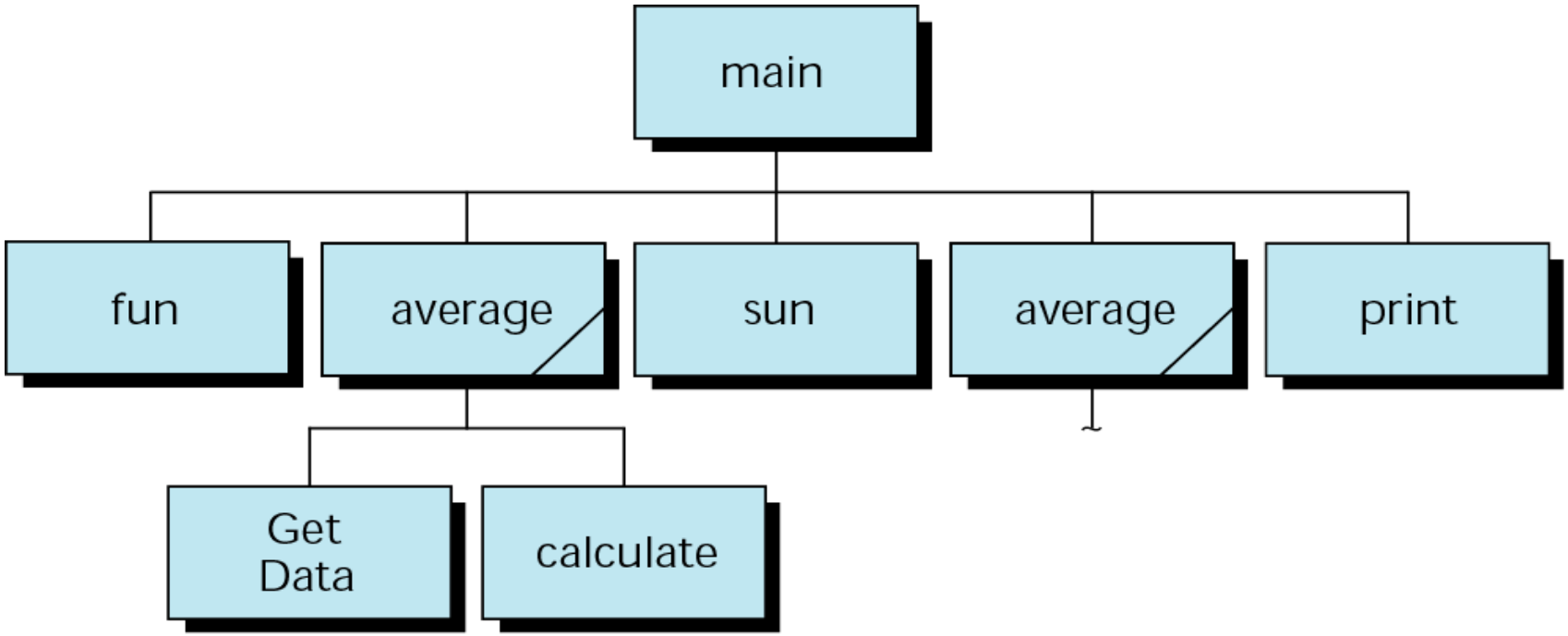
Figure 4-25 Structure chart design



Note:

Structure charts show only function flow; they contain no code.

Figure 4-26 Common functions in a structure chart



Note:

Rule of Thumb

When a function has more than four parameters, check its cohesion.

Figure 4-27 Calculate taxes

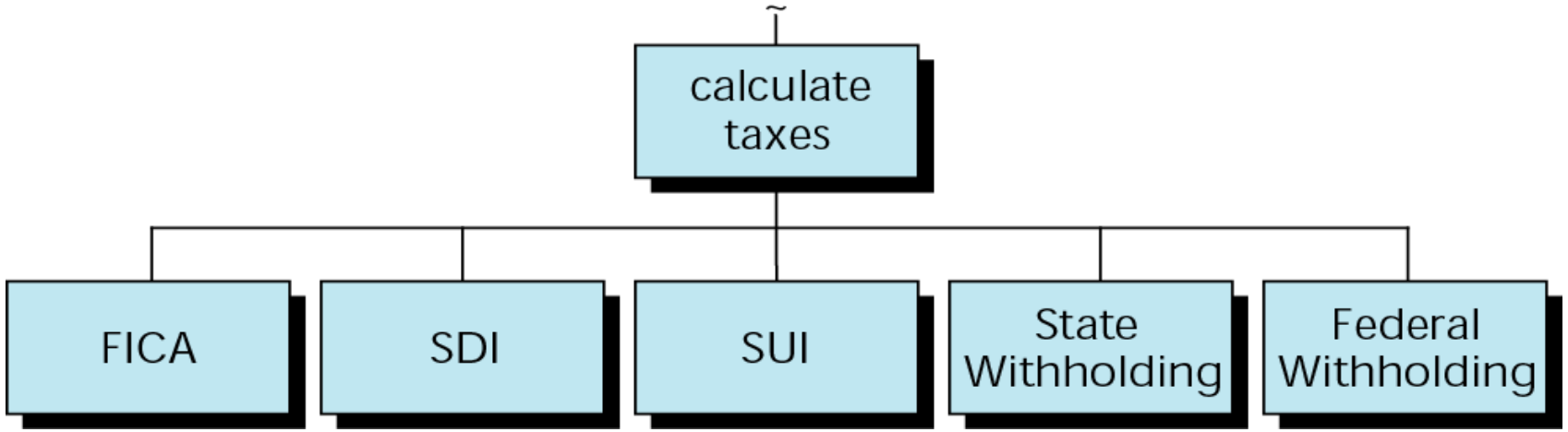


Figure 4-28 Design for print report