Chapter 6

Repetition

Brooks/Cole
Thomson Learning™

OBJECTIVES

After studying this chapter you will be able to:

- **Understand the basic components of a loop: initialization, control expression, and update.**
- **Understand and use pretest, post-test, and count-controlled loops.**
- **Differentiate between event-controlled and counter-controlled loops.**
- **Write loops in C++ using *while*, *for*, and *do...while* loops.**
- **Understand the limitations and use of *break* and *continue* statements in loops.**
- **Design structure charts for programs using loops.**
- **Understand how recursion works in a C++ program.**
- **Analyze the efficiency of algorithms using Big-O theory.**

CONCEPT OF A LOOP

6.2

Pretest and post-test loops

Brooks/Cole
Thomson Learning™

Note:

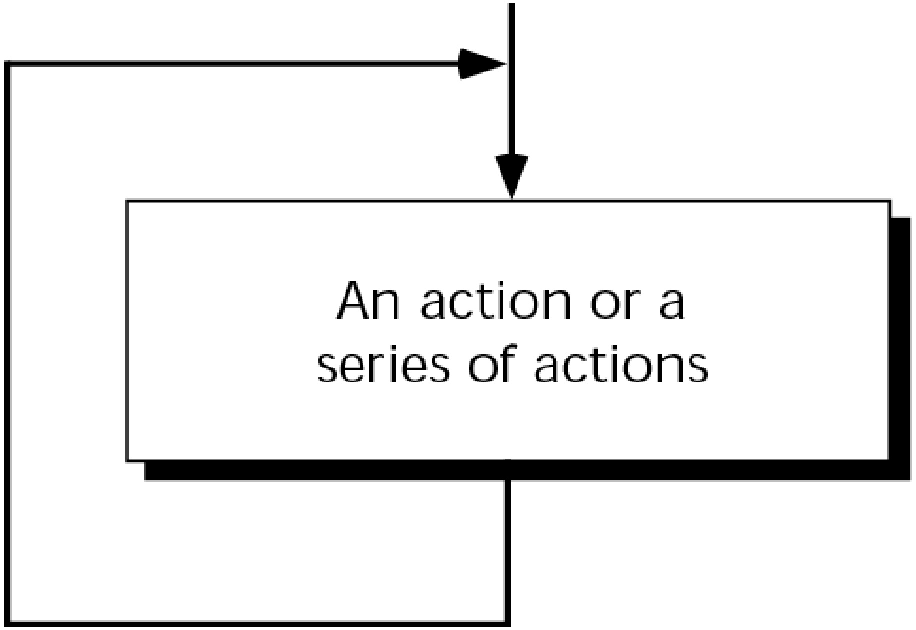
Pretest Loop

In each iteration, the loop control expression is tested first. If it is true, the loop action(s) is executed; if it is false, the loop is terminated.

Post-test Loop

In each iteration, the loop action(s) are executed. Next, the loop control expression is tested. If it is true, a new iteration is started; otherwise, the loop terminates.

Figure 6-1 The concept of a loop



An action or a
series of actions

Figure 6-2 Pretest and post-test loops

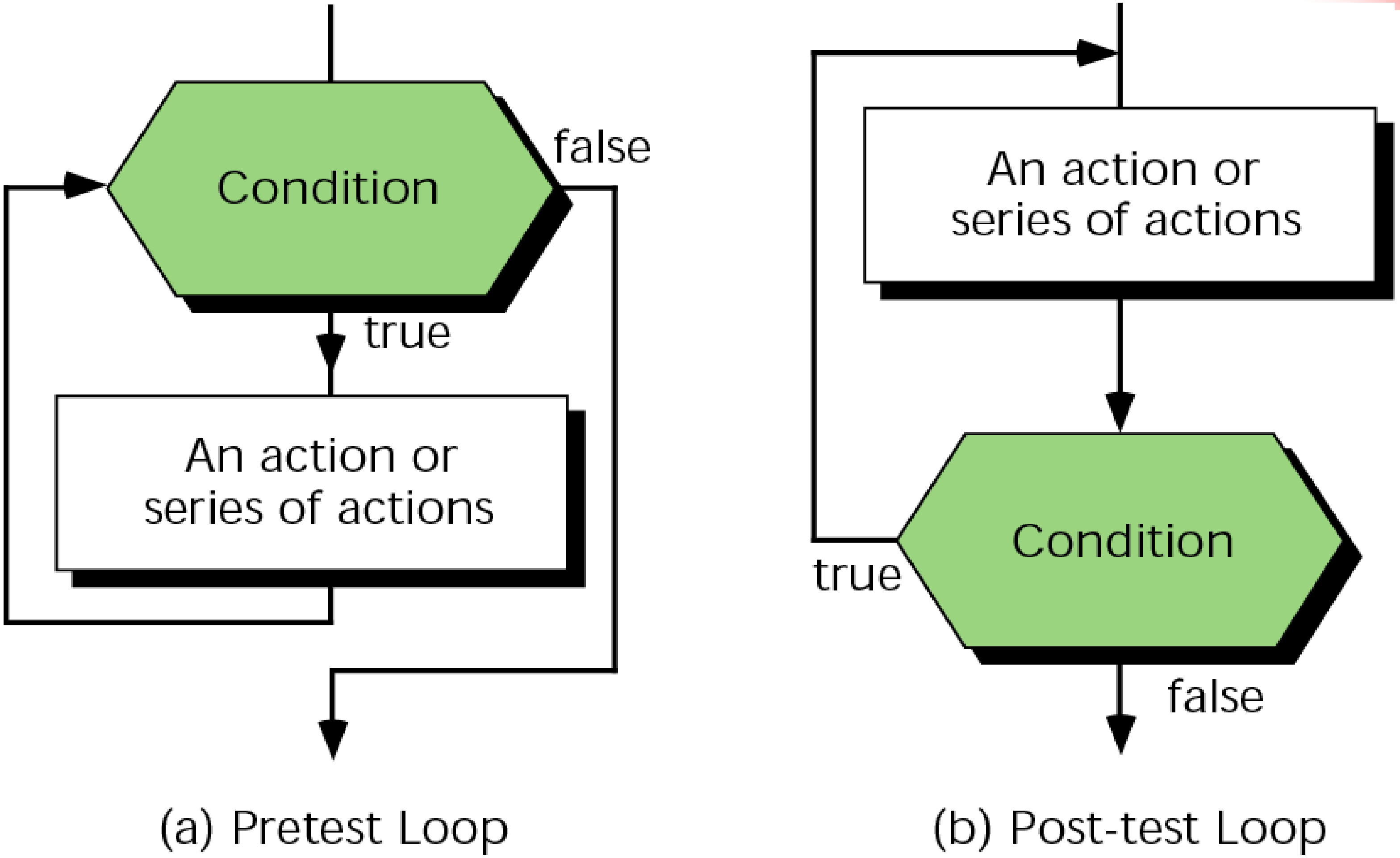
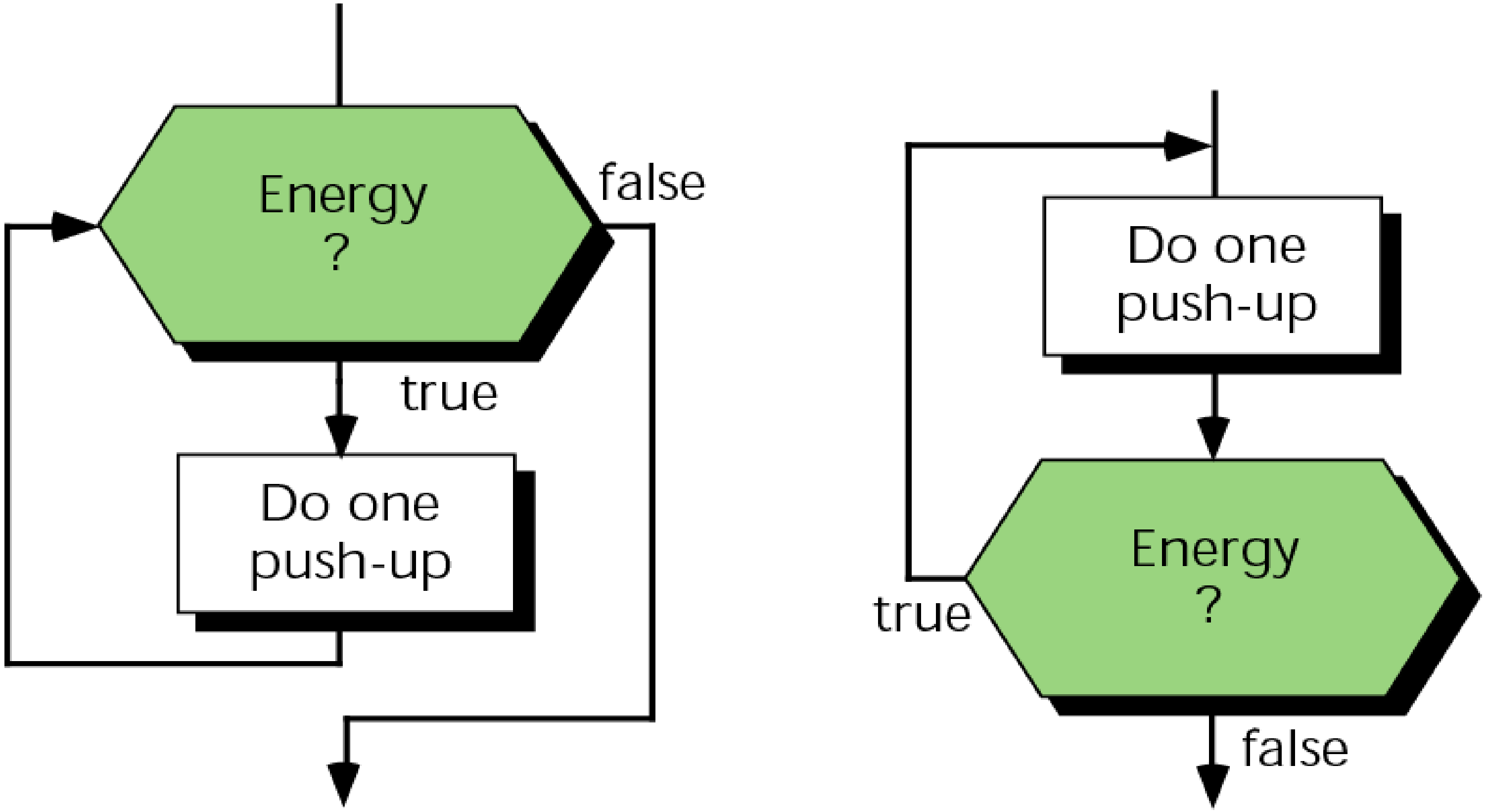


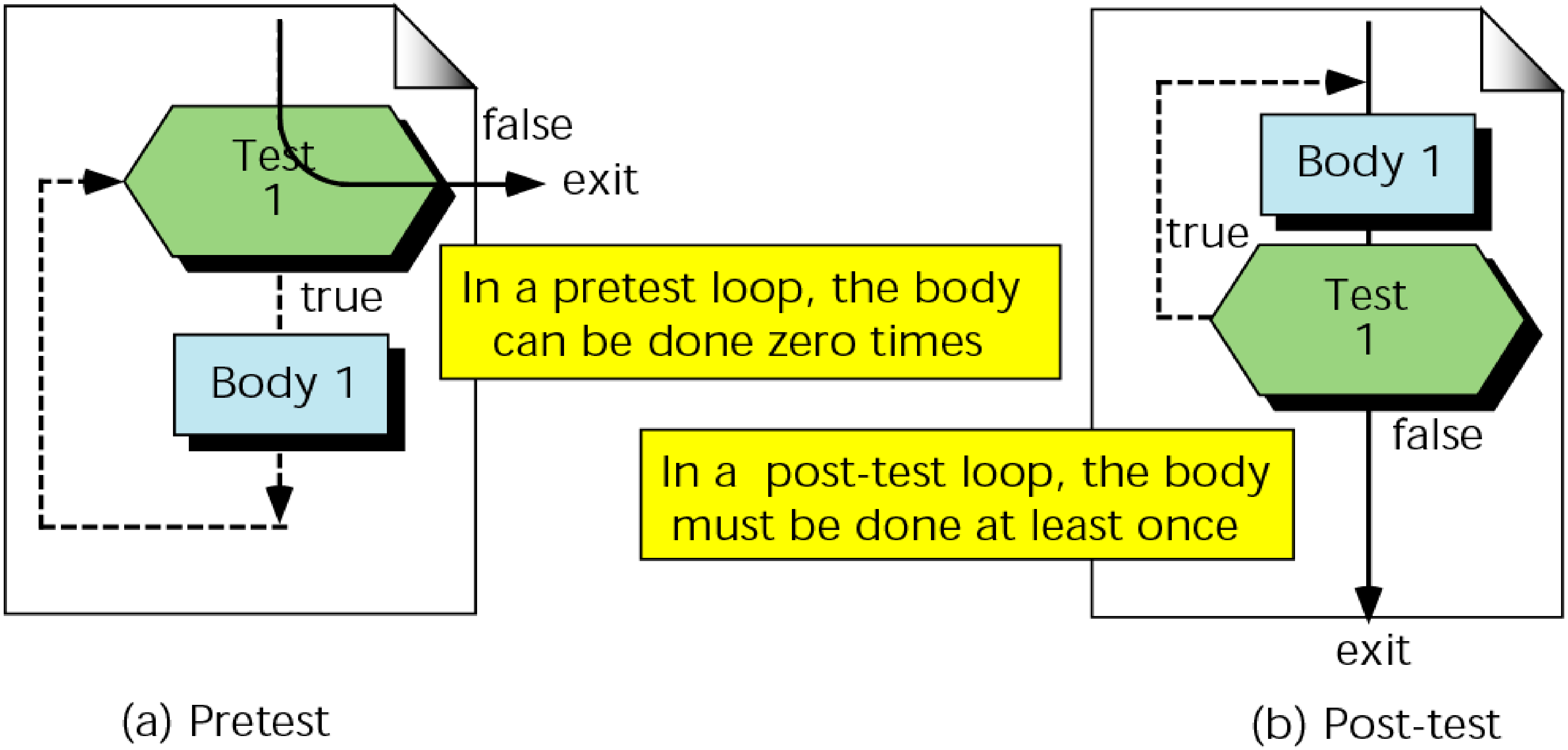
Figure 6-3 Two different strategies for starting exercise



(a) Pretest Loop

(b) Post-test Loop

Figure 6-4 Minimum number of iterations in two loops



In a pretest loop, the body can be done zero times

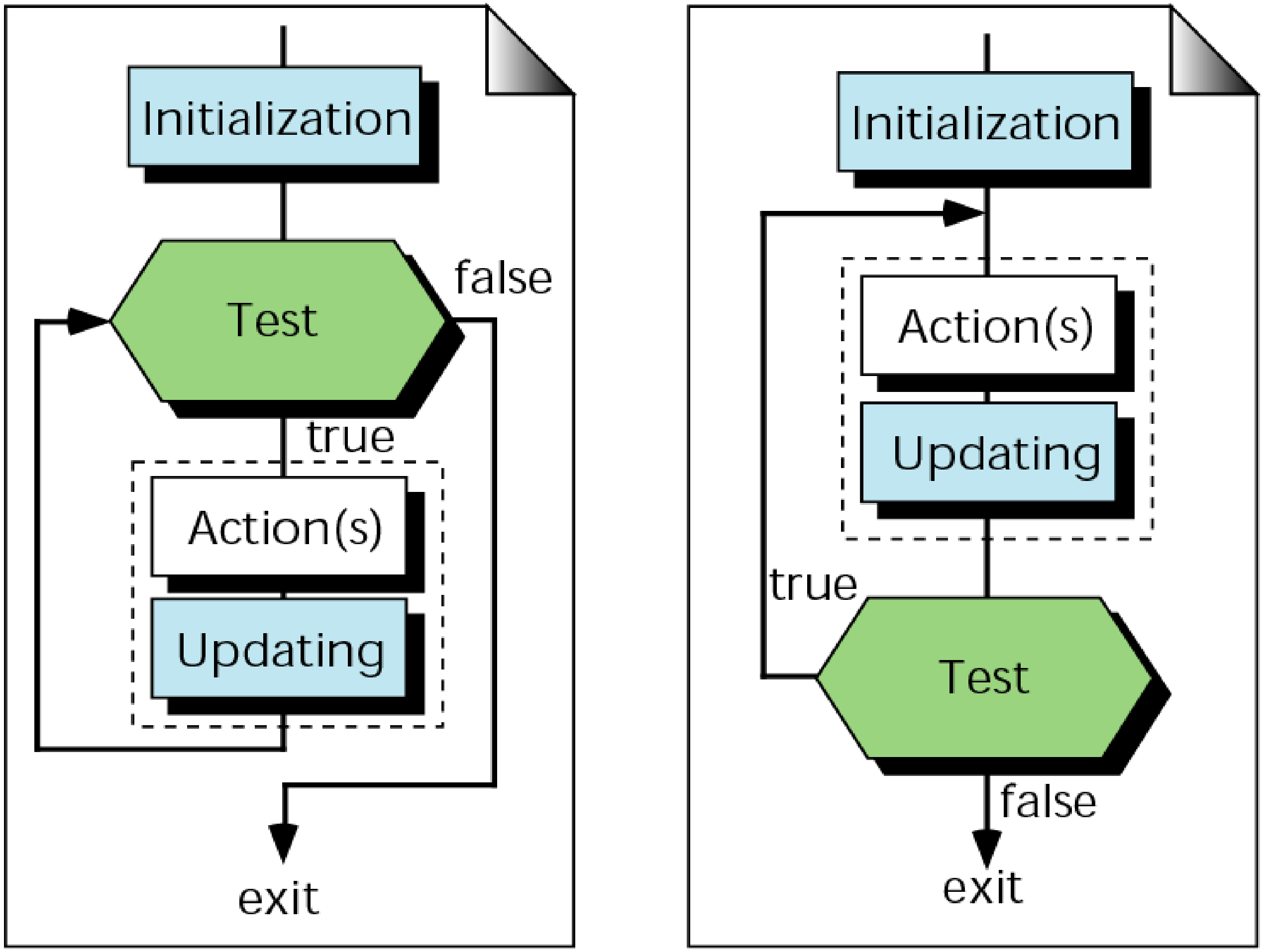
In a post-test loop, the body must be done at least once

6.3

INITIALIZATION AND UPDATING

Brooks/Cole
Thomson Learning™

Figure 6-5 Loop initialization and updating

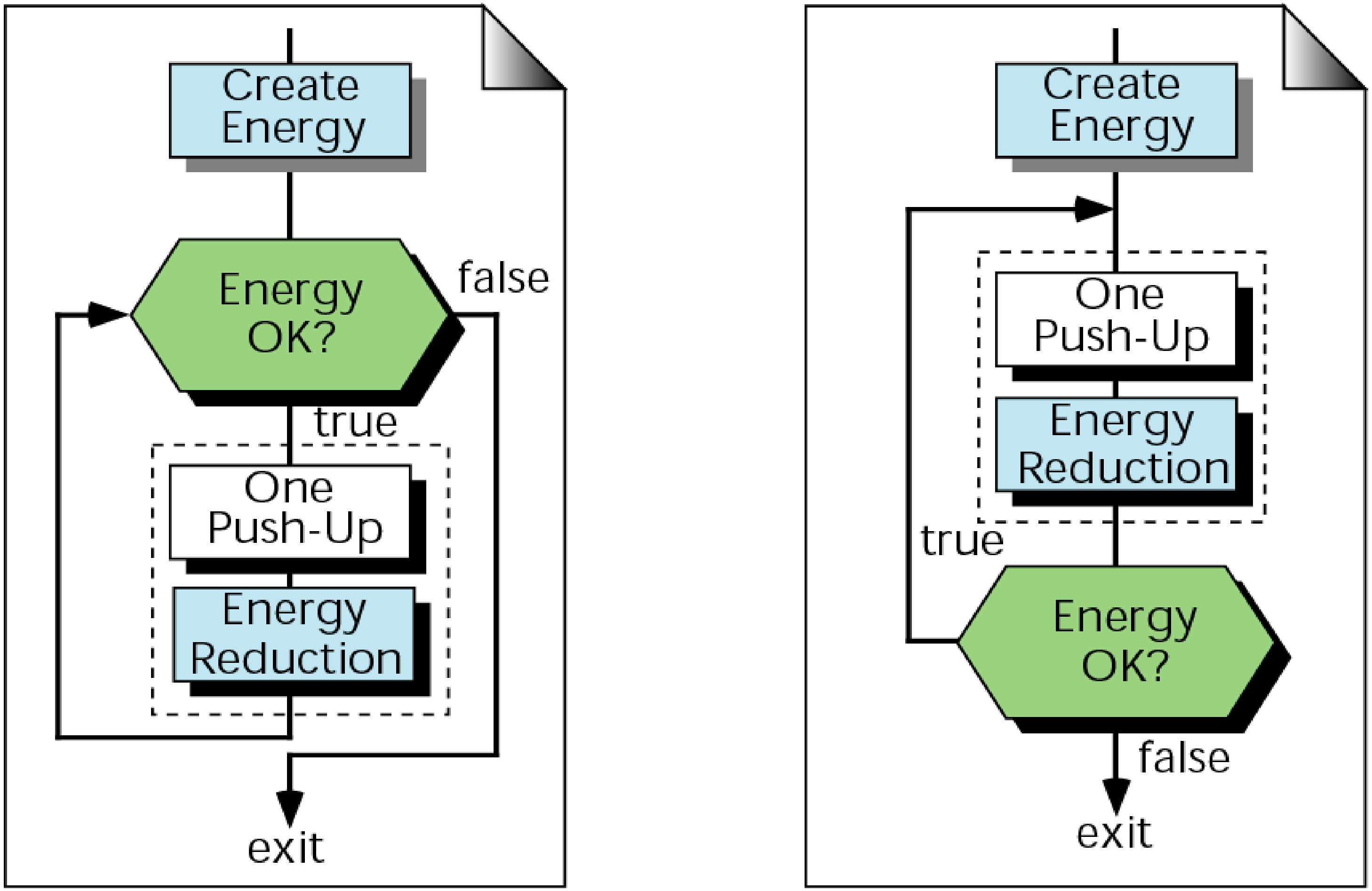


(a) Pretest Loop

(b) Post-test Loop

Brooks/Cole
Thomson Learning™

Figure 6-6 Initialization and updating for exercise



(a) Pretest Loop

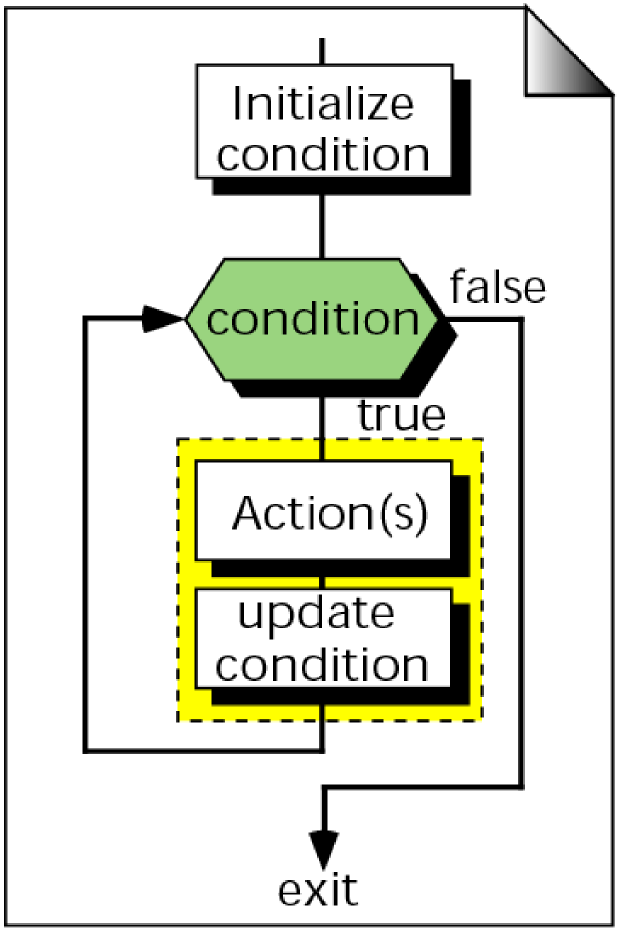
(b) Post-test Loop

Brooks/Cole
Thomson Learning

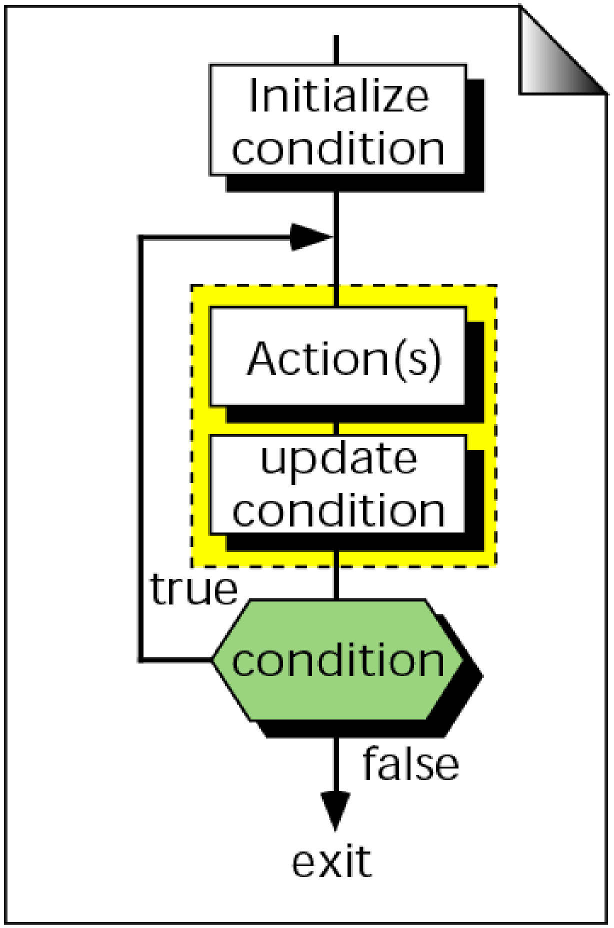
6.4

EVENT-CONTROLLED AND COUNTER-CONTROLLED LOOPS

Figure 6-7 Event-controlled loop concept



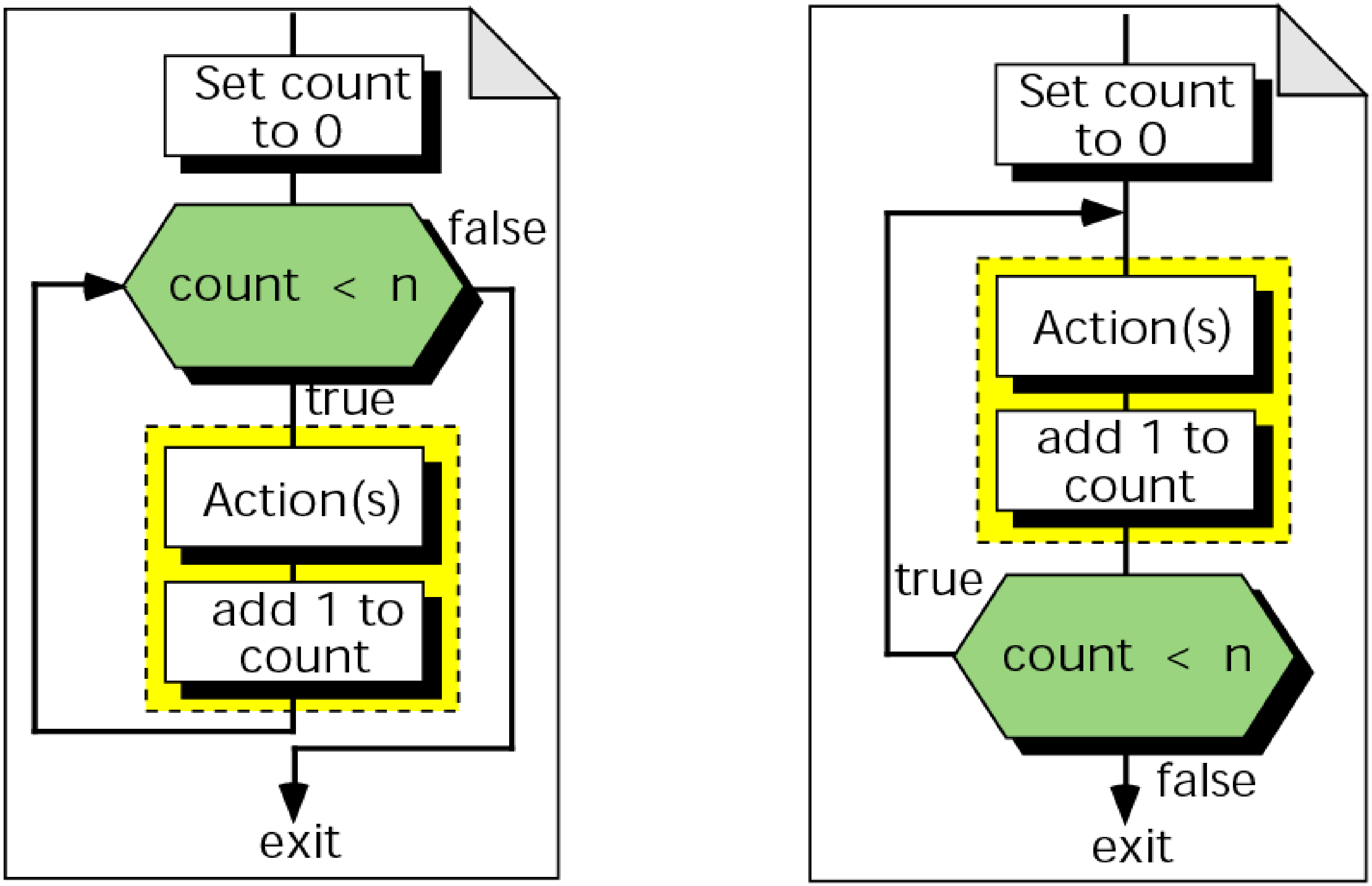
(a) Pretest Loop



(b) Post-test Loop

Brooks/Cole
Thomson Learning™

Figure 6-8 Counter-controlled loop concept



(a) Pretest Loop

(b) Post-test Loop

LOOPS IN C++

Figure 6-9 C++ loop constructs

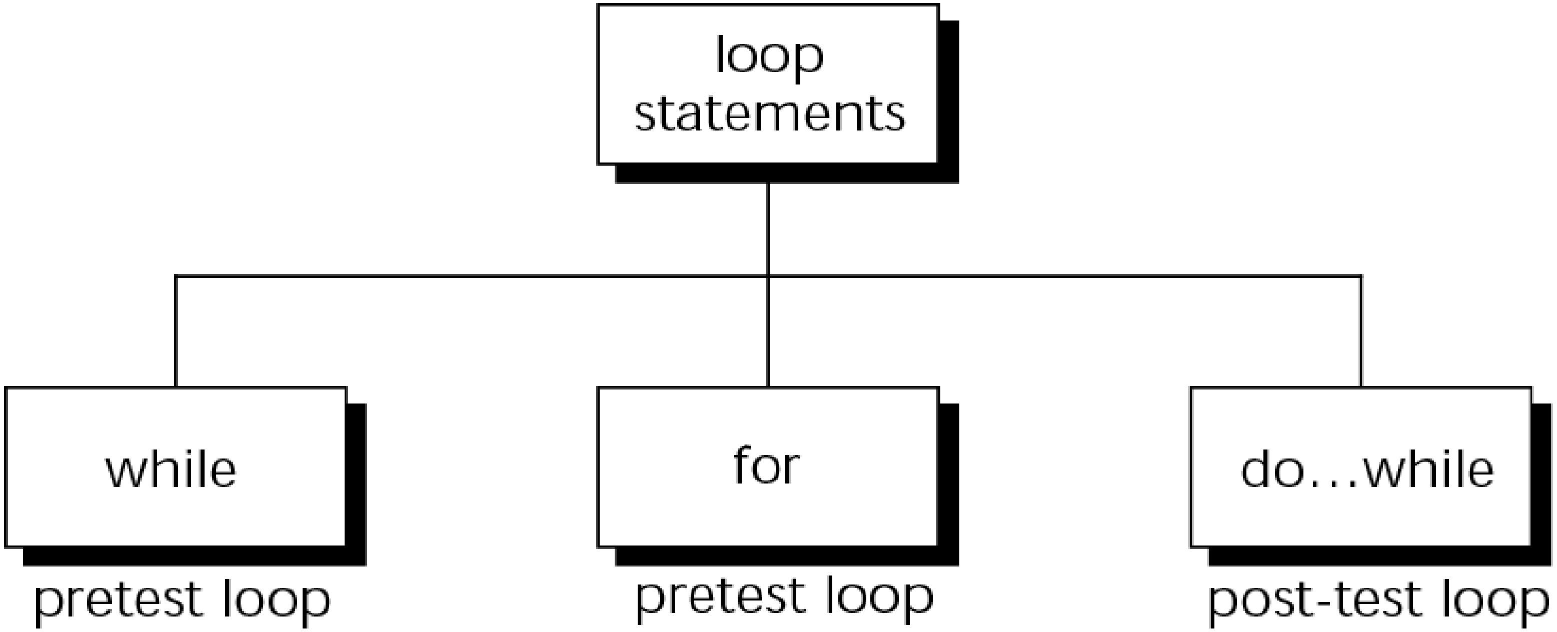
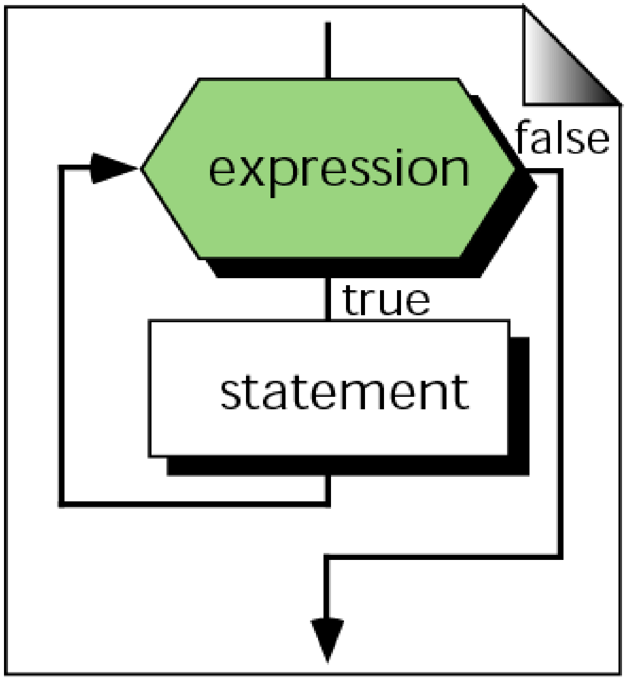


Figure 6-10 The while statement



(a) Flowchart

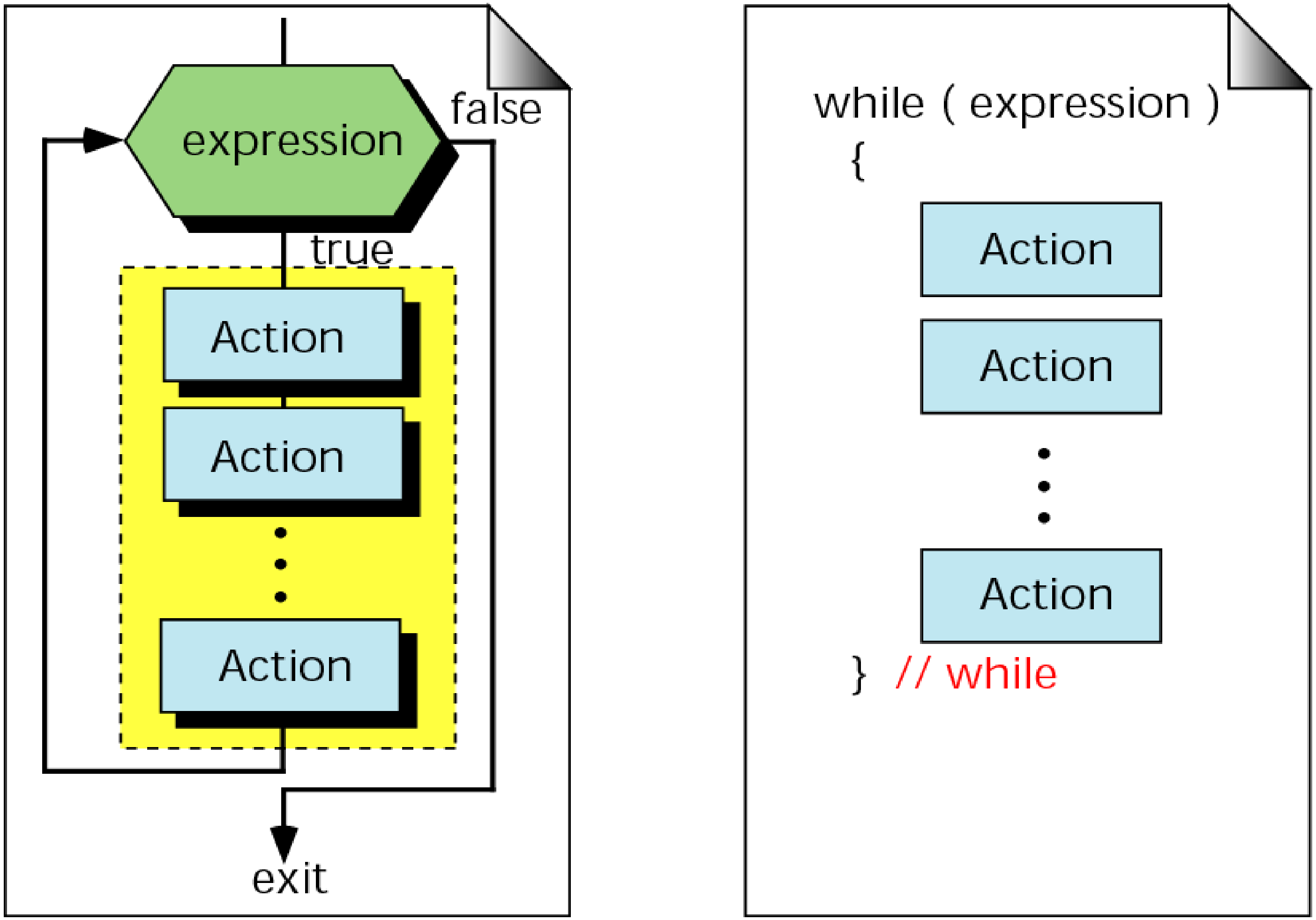
```
while ( expression)

    statement
```

(b) Sample Code

Brooks/Cole
Thomson Learning™

Figure 6-11 Compound *while* statement



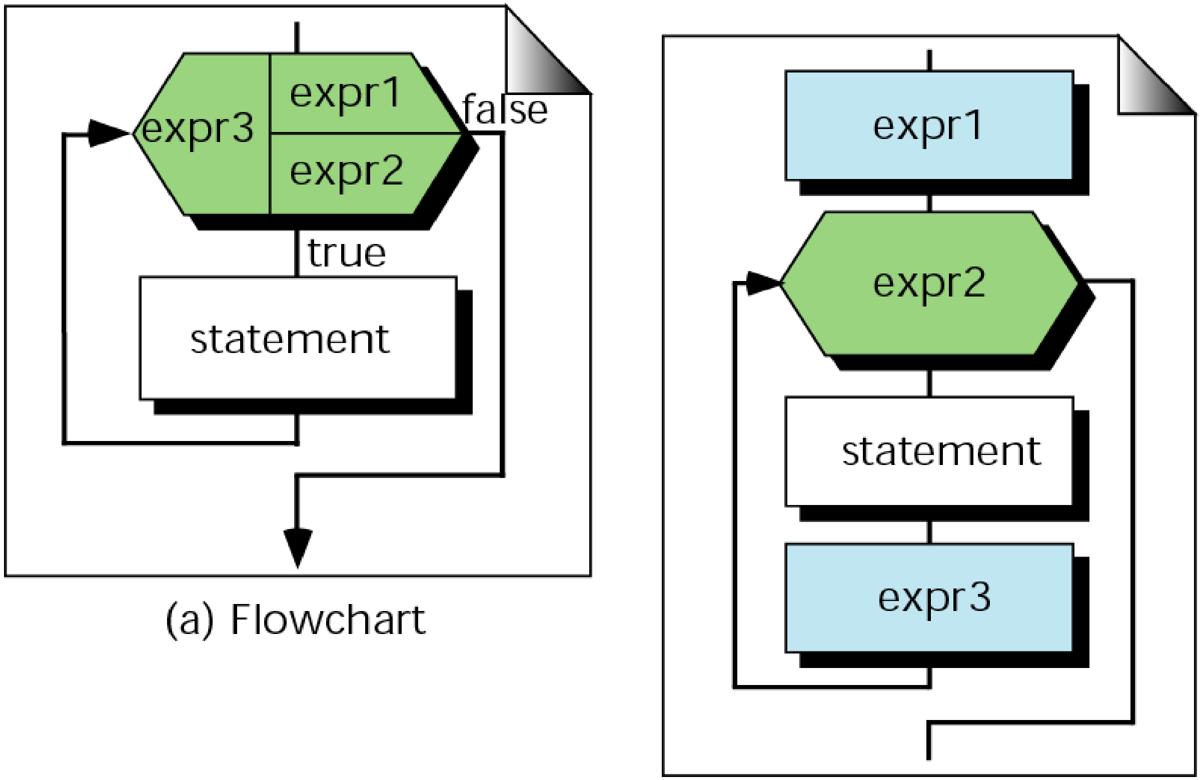
```
while ( expression )
  {
        Action
        Action
          •
          •
          •
        Action
  }  // while
```

(a) Flowchart

(b) C++ Language

Brooks/Cole
Thomson Learning™

Figure 6-12 *for* statement



(a) Flowchart

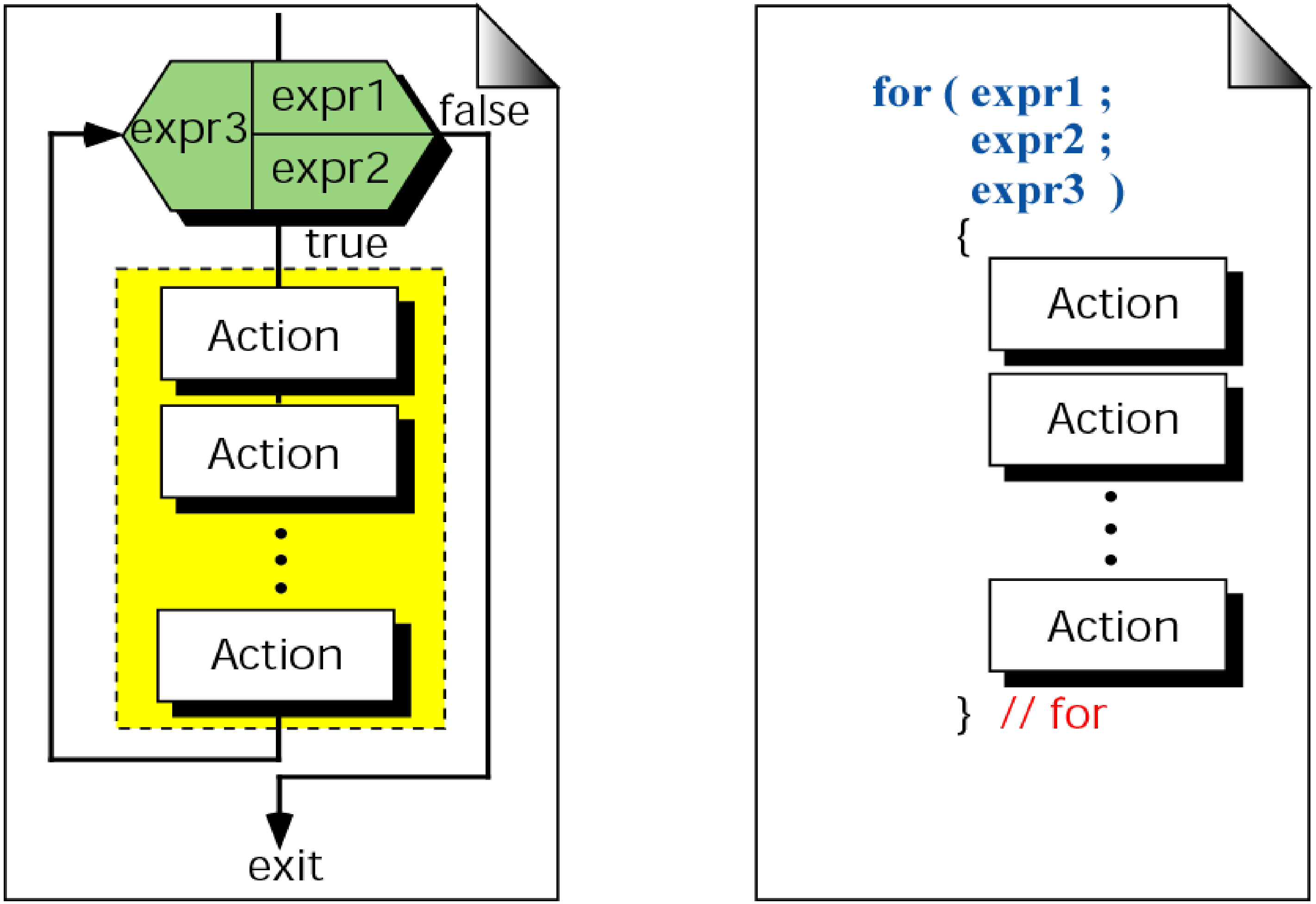
(b) Expanded Flowchart

```
for ( expr1 ; expr2 ; expr3 )
    statement
```

Note:

A for loop is used when your loop is to be executed a known number of times. You can do the same thing with a while loop, but the for loop is easier to read and more natural for counting loops.

Figure 6-13 Compound *for* statement



```
for ( expr1 ;
      expr2 ;
      expr3  )
     {
       Action
       Action
       ...
       Action
     }  // for
```

(a) Flowchart

(b) C++ Language

Figure 6-14 Comparing *for* and *while* loops

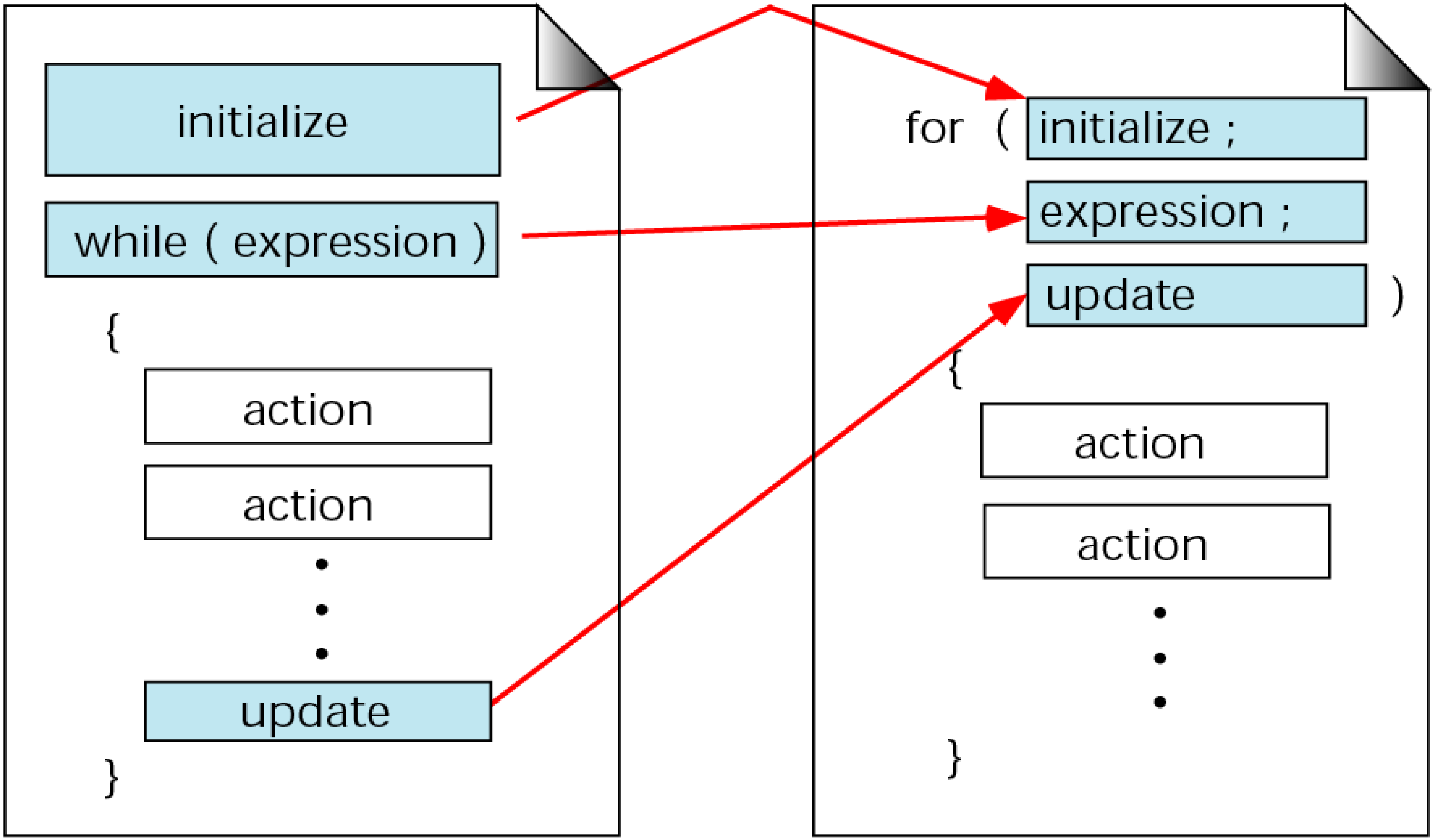
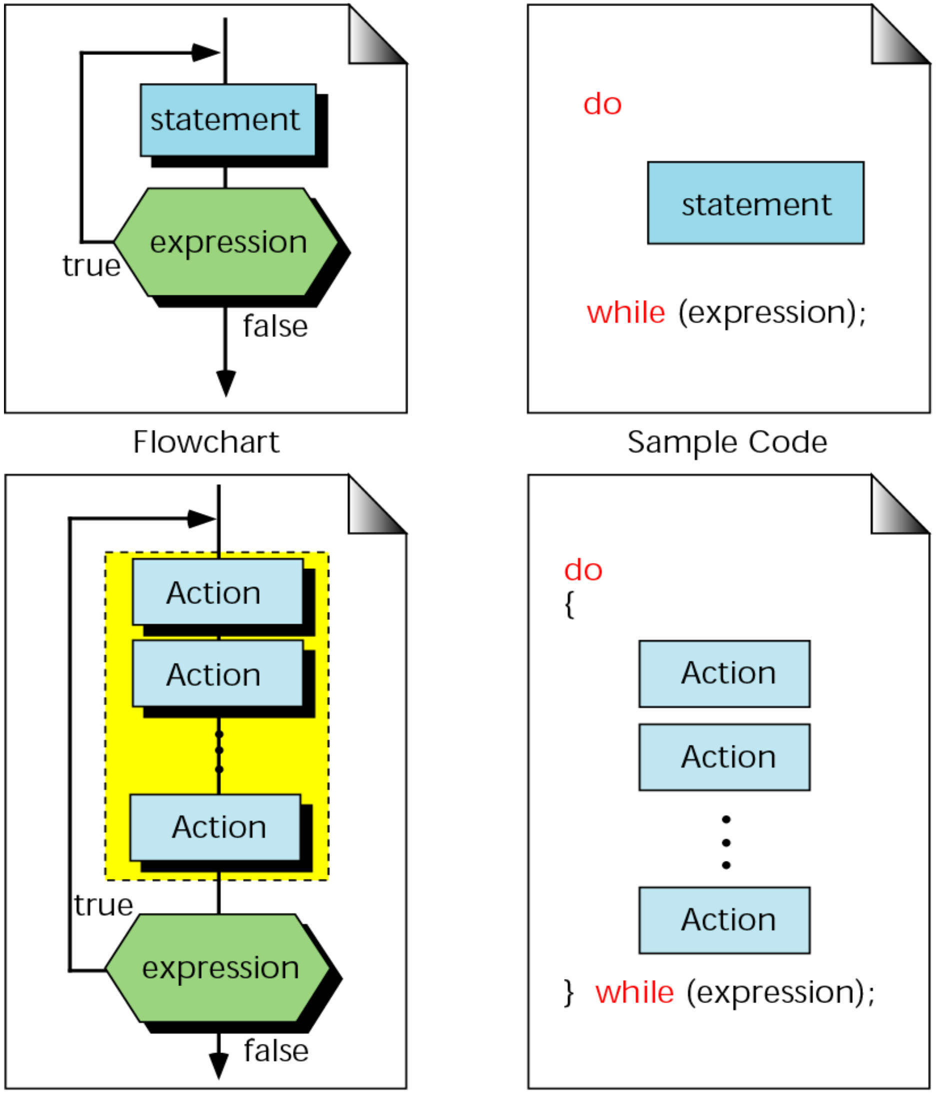


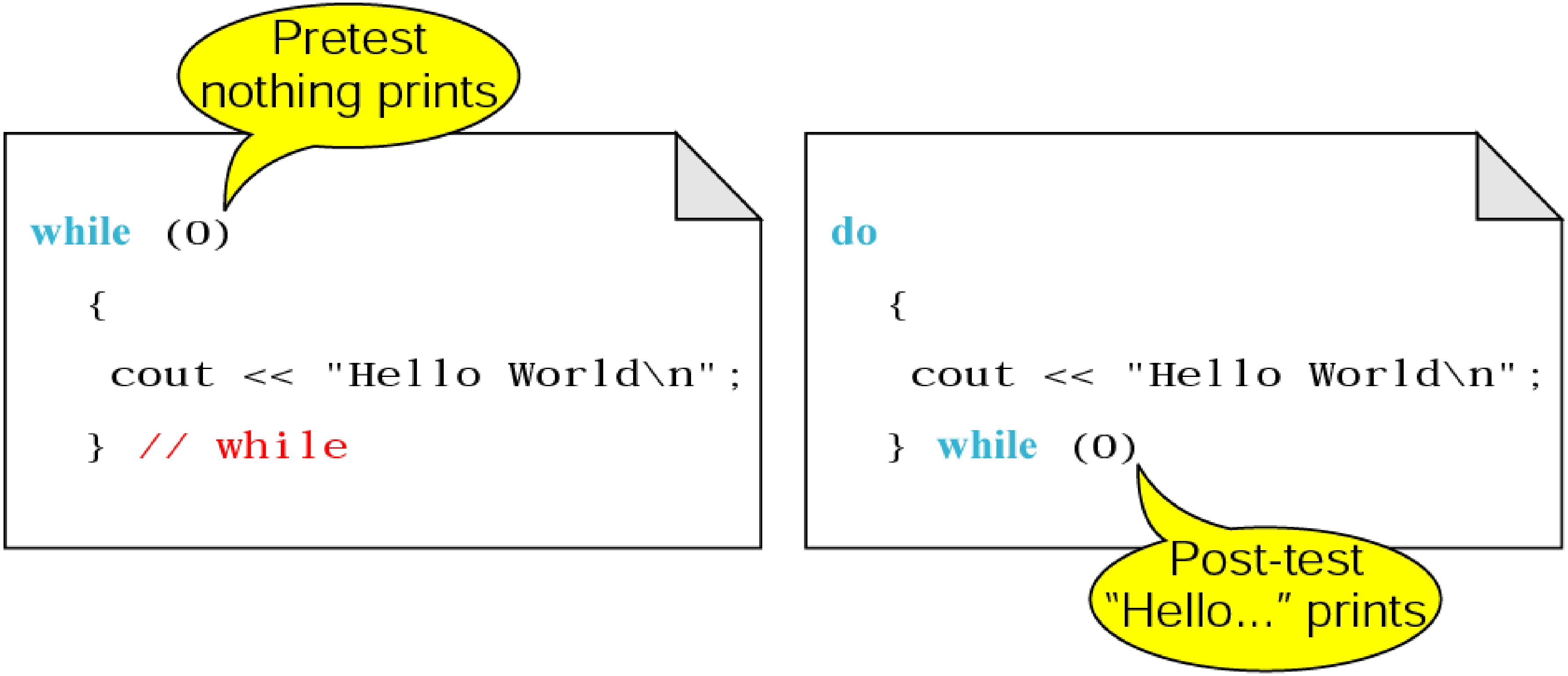
Figure 6-15 Format of the *do…while* statement



```
do
        statement
while (expression);
```

```
do
{
        Action
        Action
        ⋮
        Action
}  while (expression);
```

Figure 6-16   Pre- and post-test loops



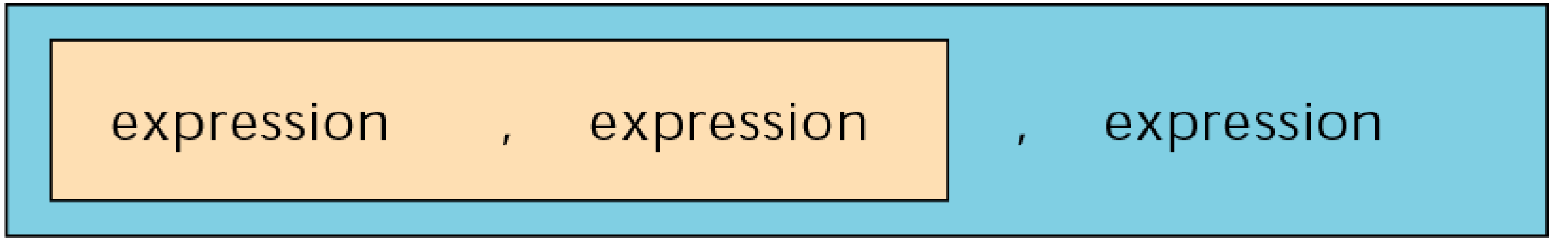
Pretest
nothing prints

```
while  (0)
   {
    cout << "Hello World\n";
   } // while
```

```
do
   {
    cout << "Hello World\n";
   } while  (0)
```

Post-test
"Hello…" prints

Figure 6-17 Nested comma expression

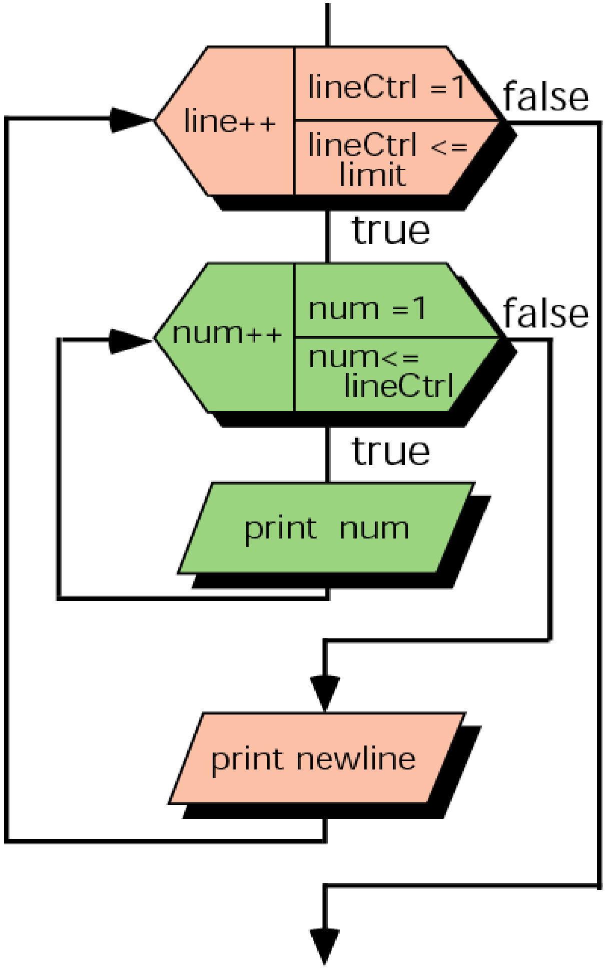


6.6

Loop Examples

Brooks/Cole
Thomson Learning™

Figure 6-18 Print triangle flowchart and pseudocode

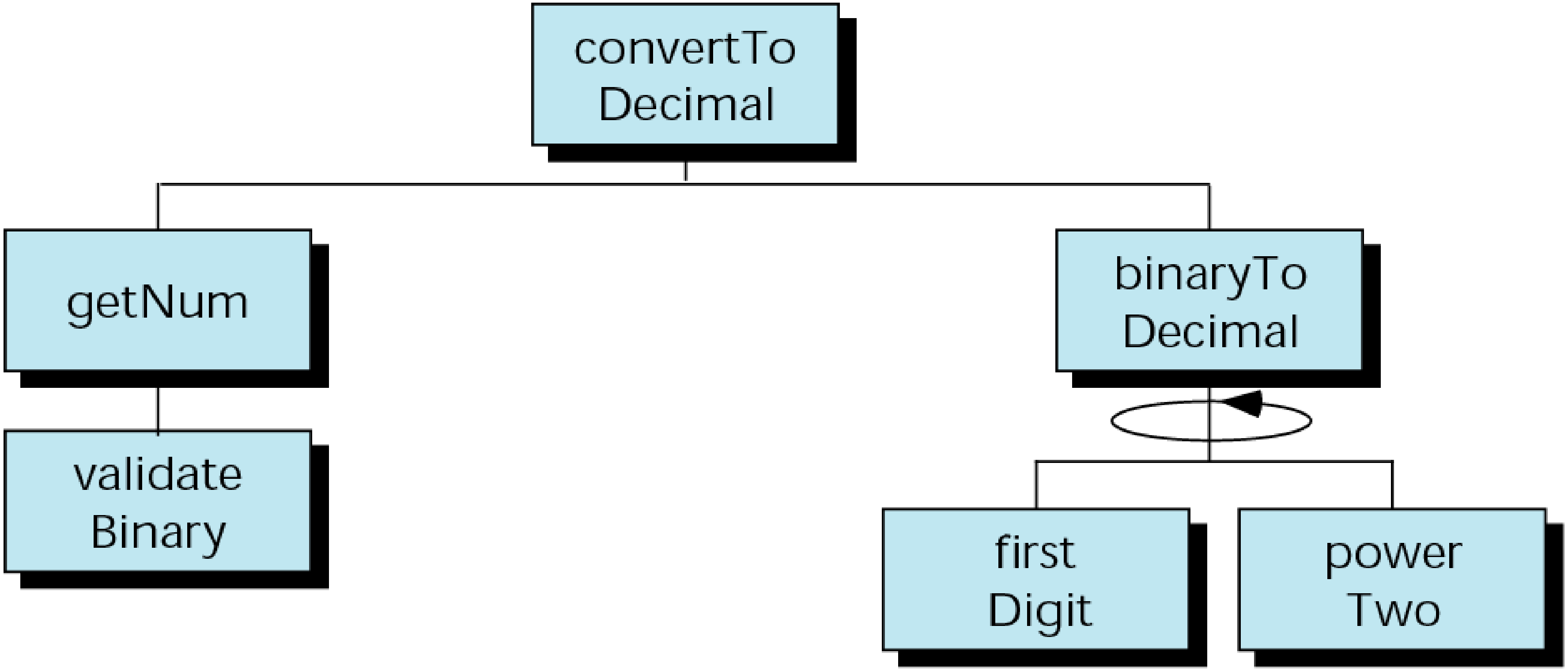


(a) Flowchart

```
for lineCtrl = 1 to limit
    for num = 1 to lineCtrl
        print num
    end for
    advance line
end for
```

(b) Pseudocode

Figure 6-19 Design for binary to decimal



6.7

Other Statements Related to Looping

Figure 6-20 Jump statements

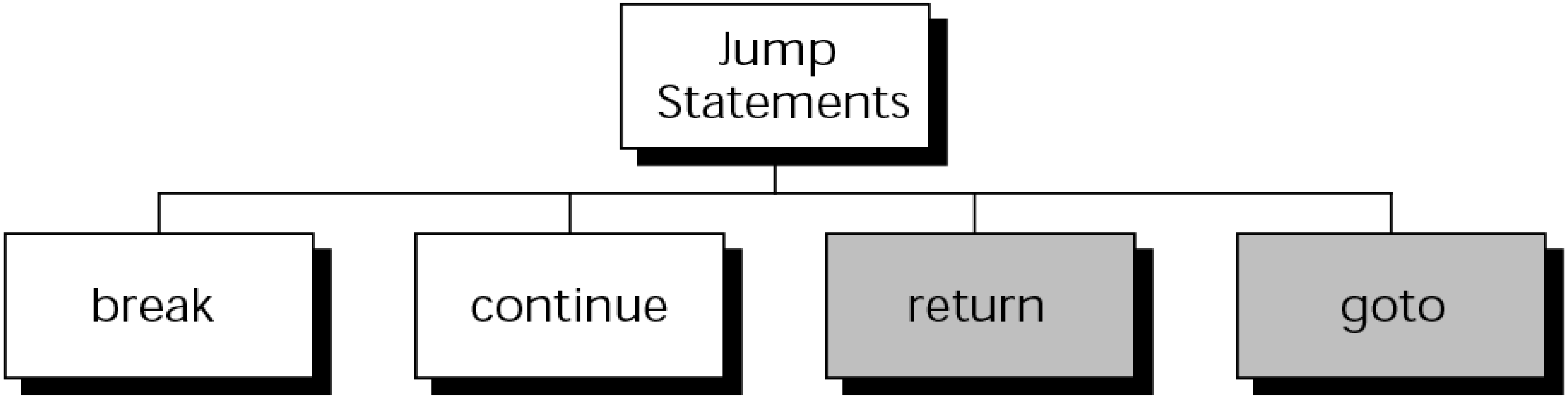
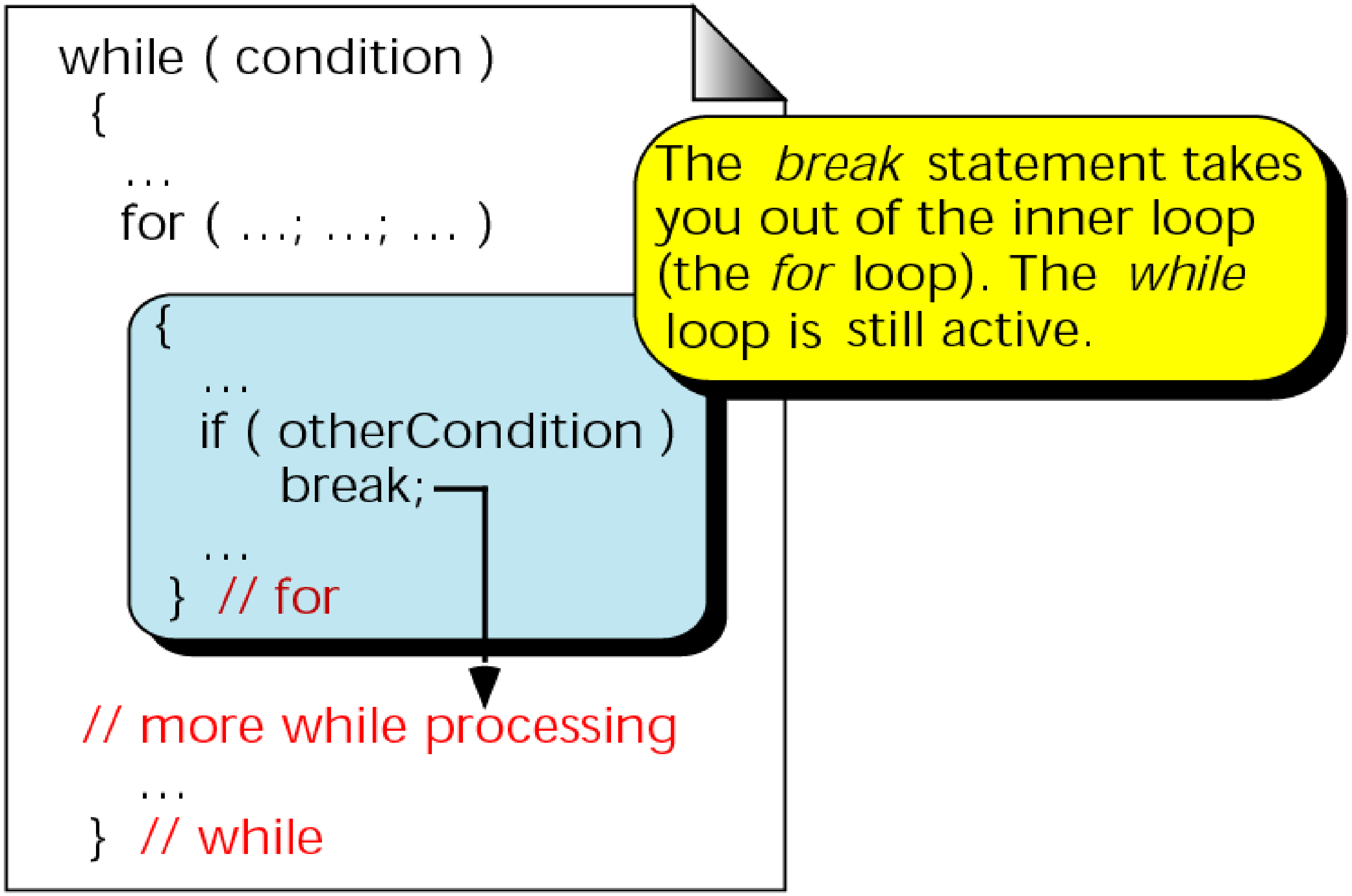


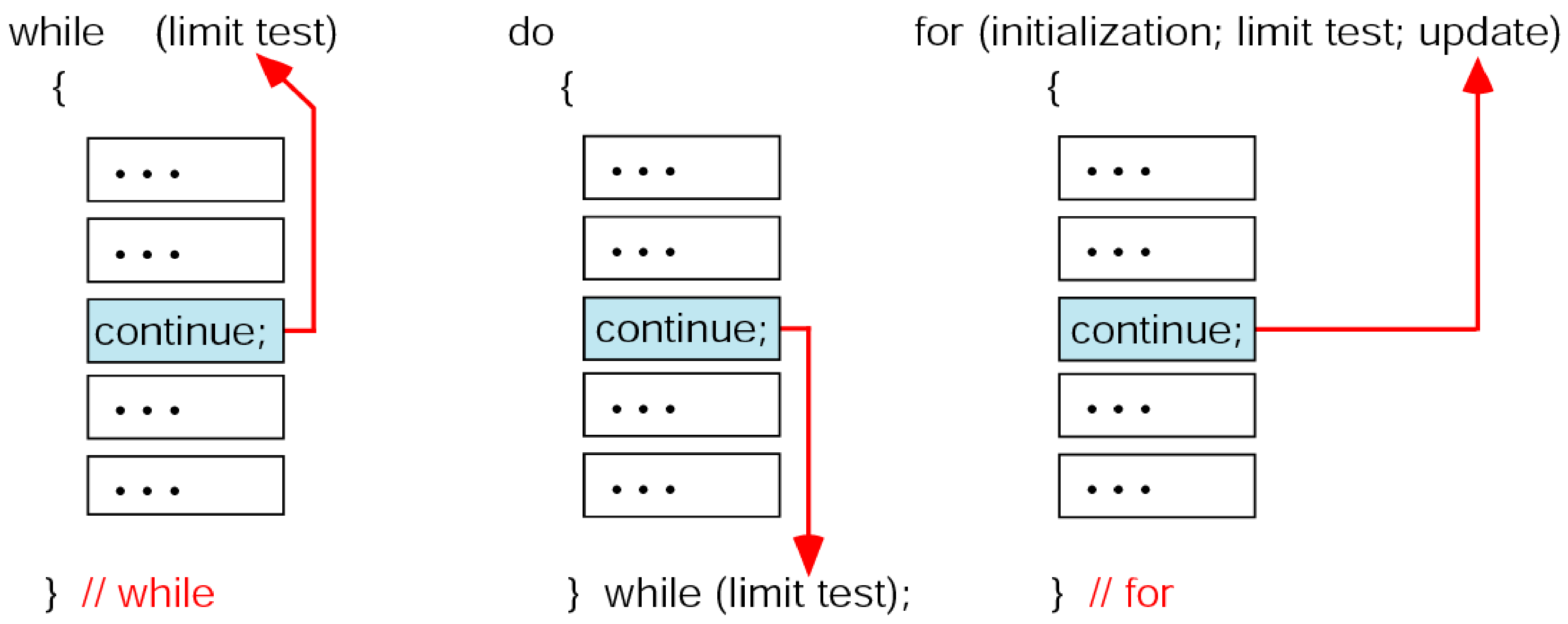
Figure 6-21 *break* and inner loops



```
while ( condition )
  {
    ...
    for ( ...; ...; ... )
      {
        ...
        if ( otherCondition )
            break;
        ...
      } // for
  // more while processing
    ...
  } // while
```

The *break* statement takes you out of the inner loop (the *for* loop). The *while* loop is still active.

Figure 6-22 The *continue* statement



Brooks/Cole
Thomson Learning™

6.8

Looping Applications

Figure 6-23 Summation and product loops

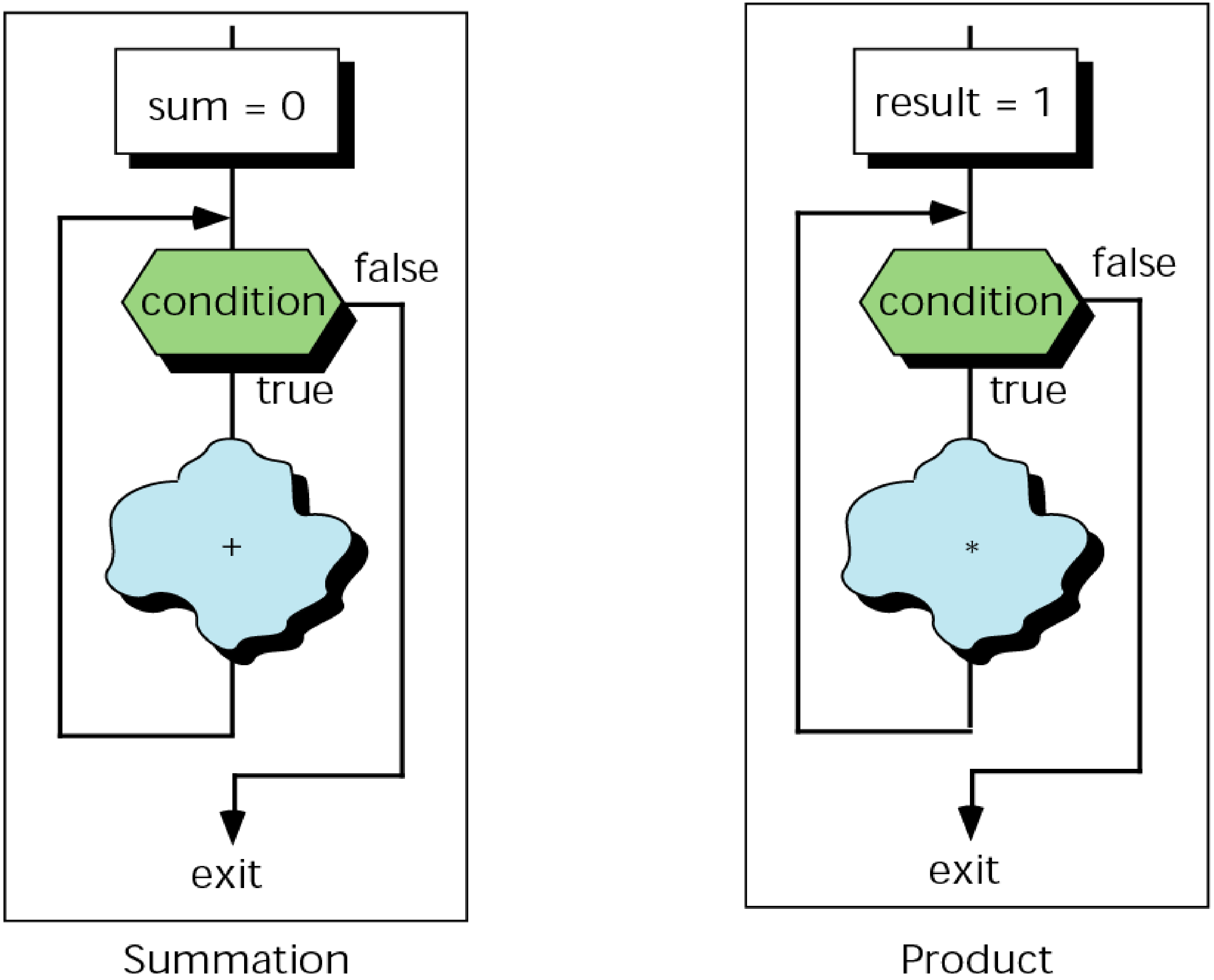
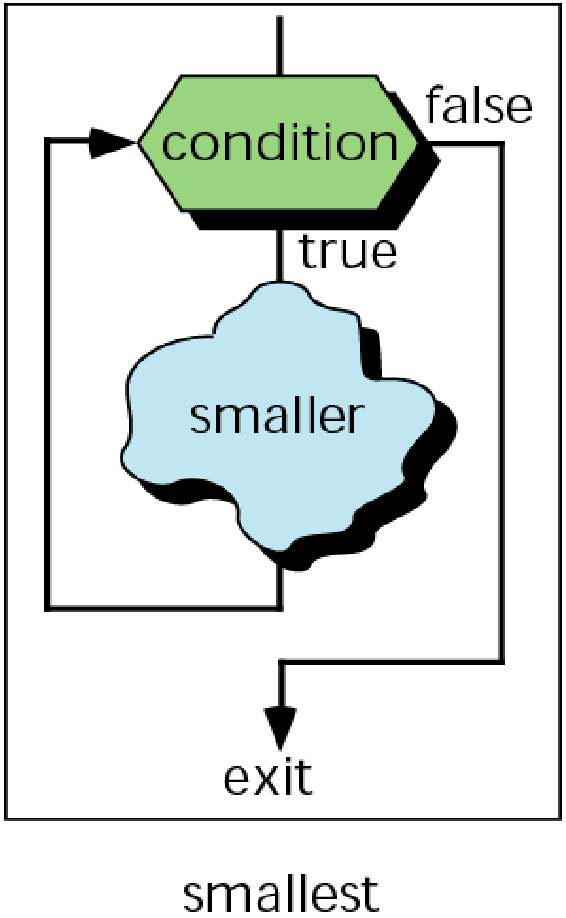


Figure 6-24 Smallest and largest loops



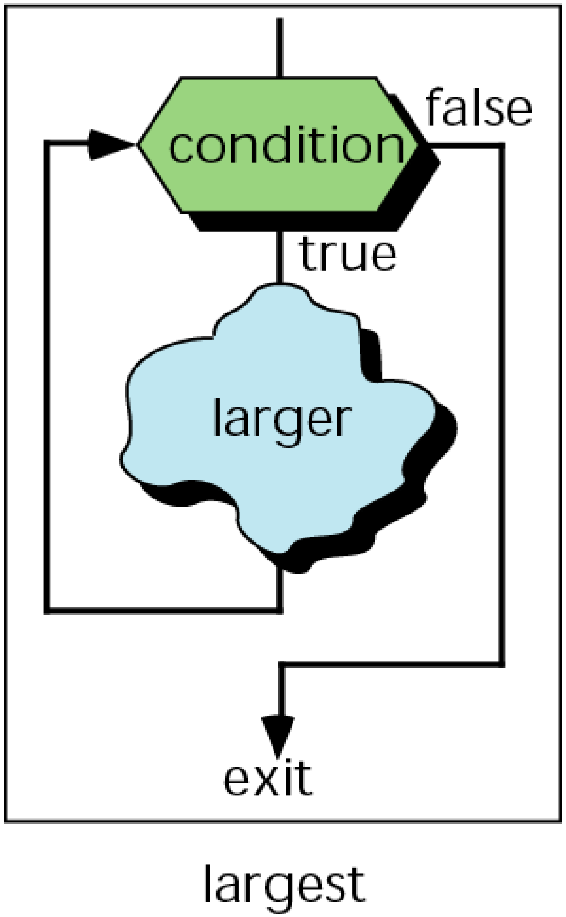
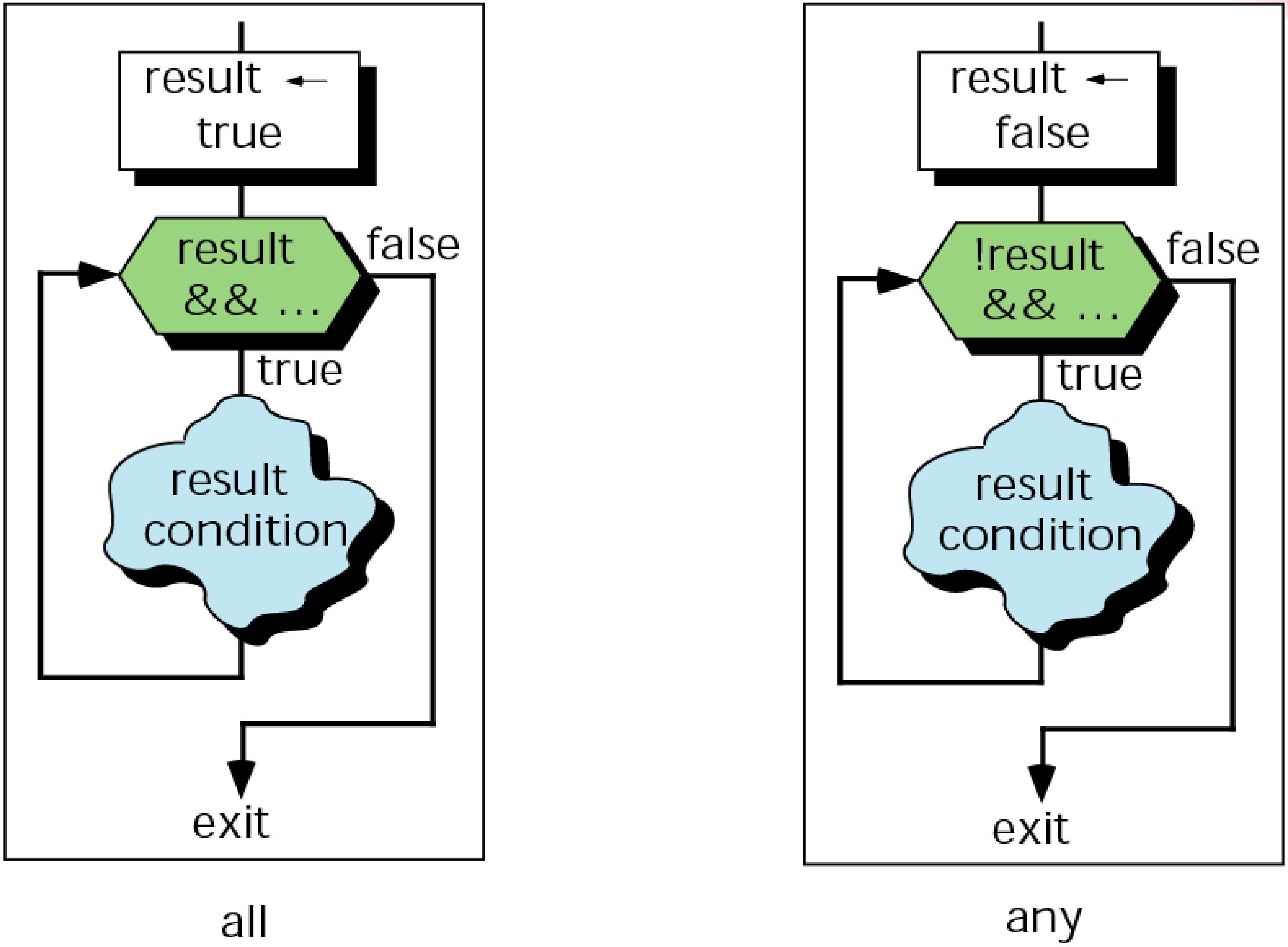


Figure 6-25 any and all inquiries

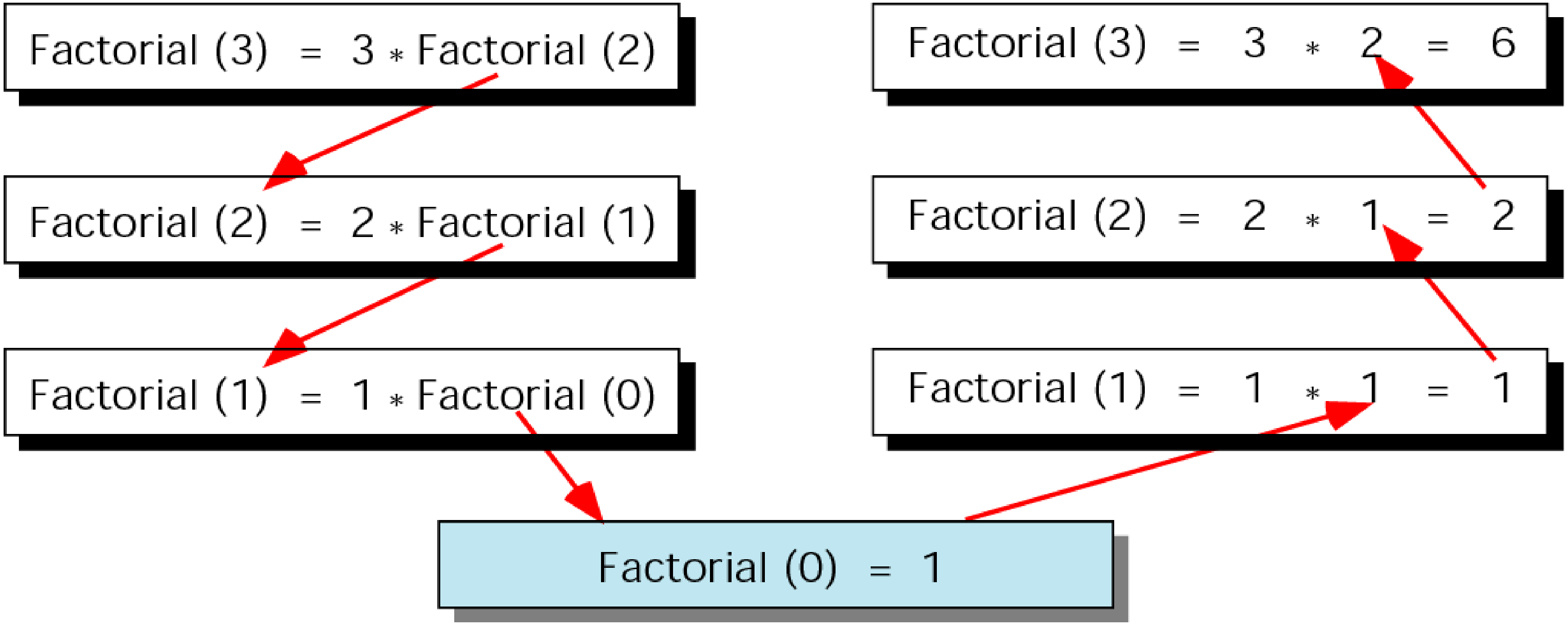


Brooks/Cole
Thomson Learning™

6.9

RECURSION

Figure 6-26 Factorial (3) recursively



Factorial (3) = 3 * Factorial (2)

Factorial (2) = 2 * Factorial (1)

Factorial (1) = 1 * Factorial (0)

Factorial (0) = 1

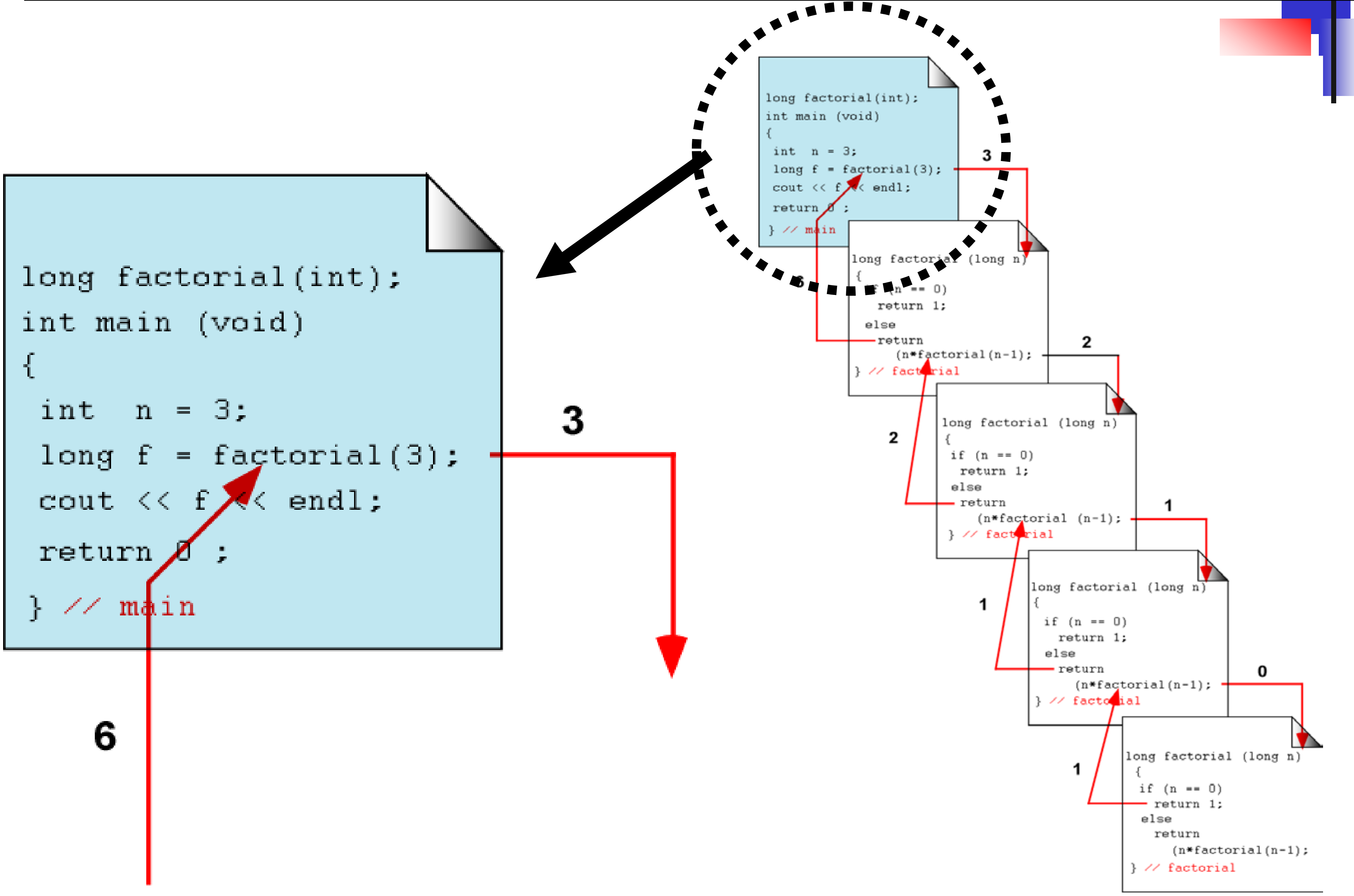
Factorial (1) = 1 * 1 = 1

Factorial (2) = 2 * 1 = 2

Factorial (3) = 3 * 2 = 6

Brooks/Cole
Thomson Learning™

Figure 6-27 Calling a recursive function



```
long factorial(int);
int main (void)
{
 int  n = 3;
 long f = factorial(3);
 cout << f << endl;
 return 0 ;
} // main
```

```
long factorial(int);
int main (void)
{
 int  n = 3;
 long f = factorial(3);
 cout << f << endl;
 return 0 ;
} // main
```

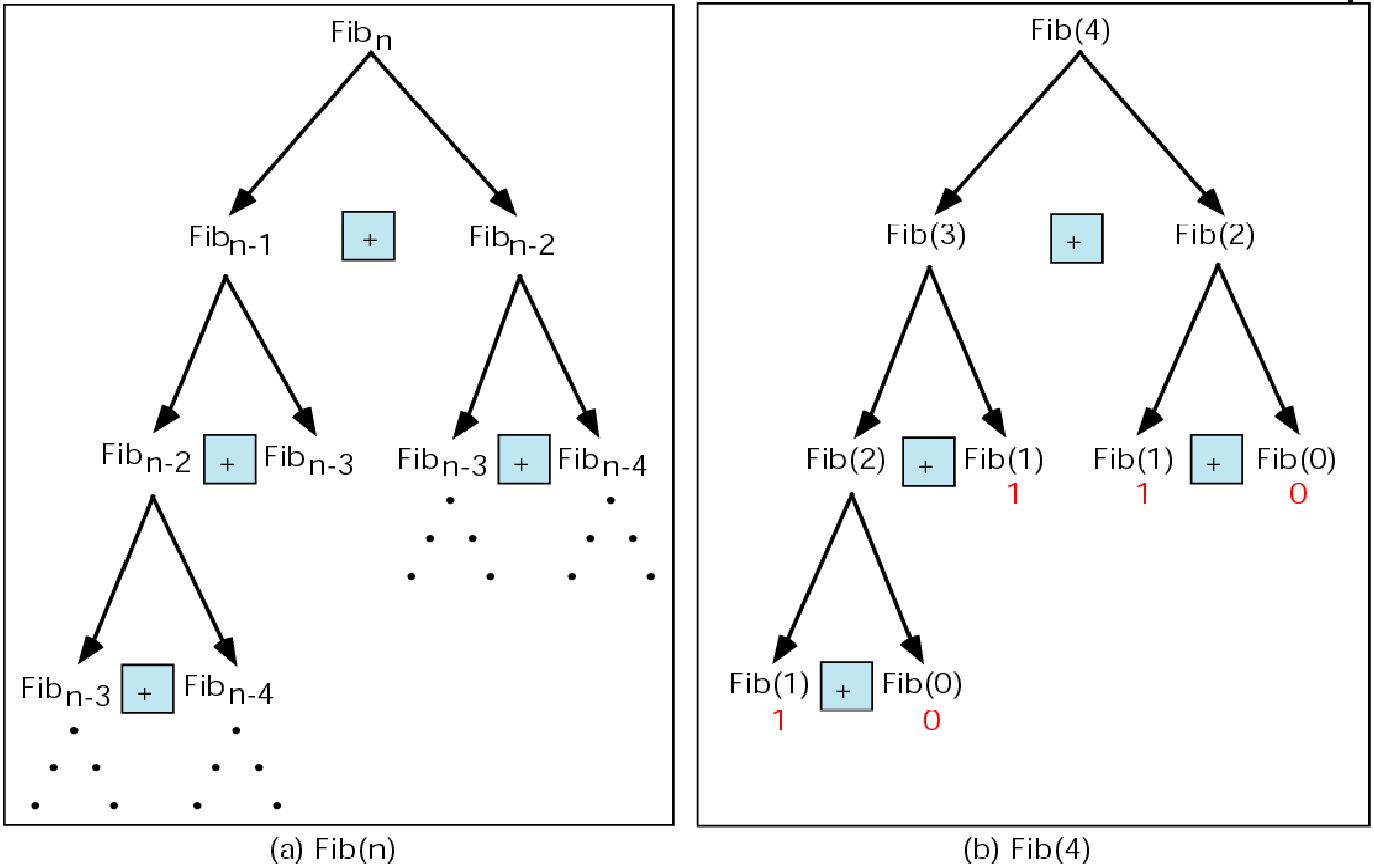
```
long factorial (long n)
{
 if (n == 0)
   return 1;
 else
   return
     (n*factorial(n-1);
} // factorial
```

```
long factorial (long n)
{
 if (n == 0)
   return 1;
 else
   return
     (n*factorial (n-1);
} // factorial
```

```
long factorial (long n)
{
 if (n == 0)
   return 1;
 else
   return
     (n*factorial(n-1);
} // factorial
```

```
long factorial (long n)
{
 if (n == 0)
   return 1;
 else
   return
     (n*factorial(n-1);
} // factorial
```

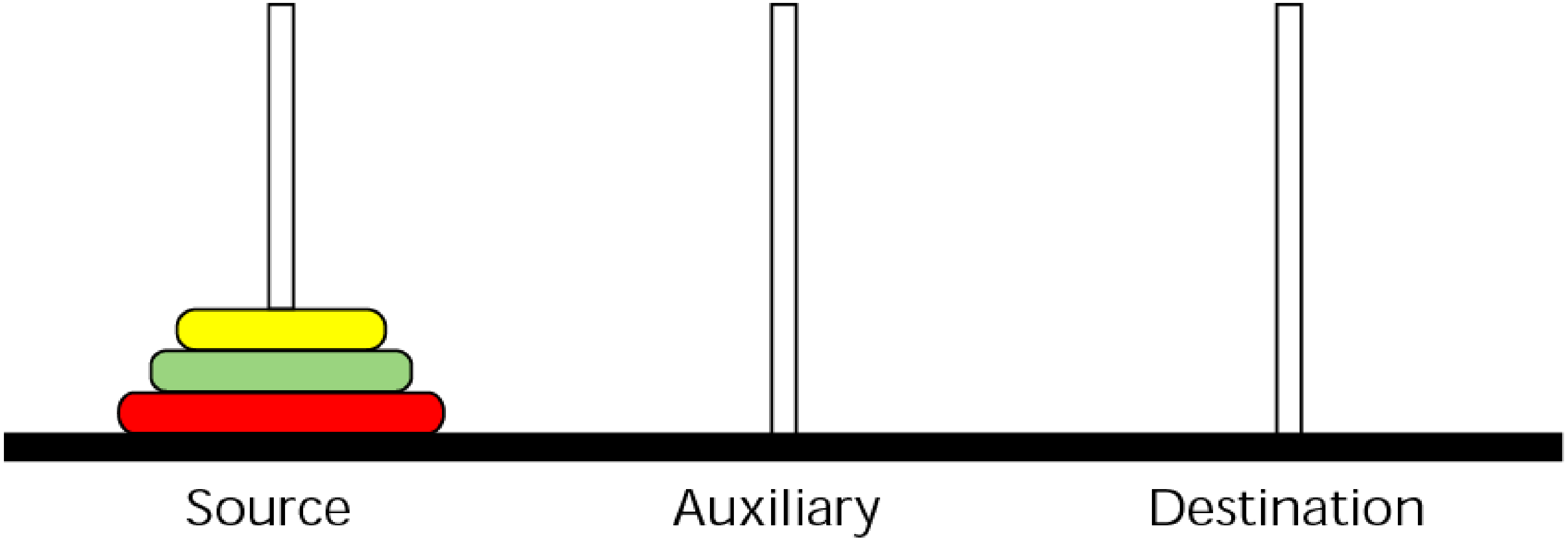
Figure 6-28 Fibonacci numbers



Note:

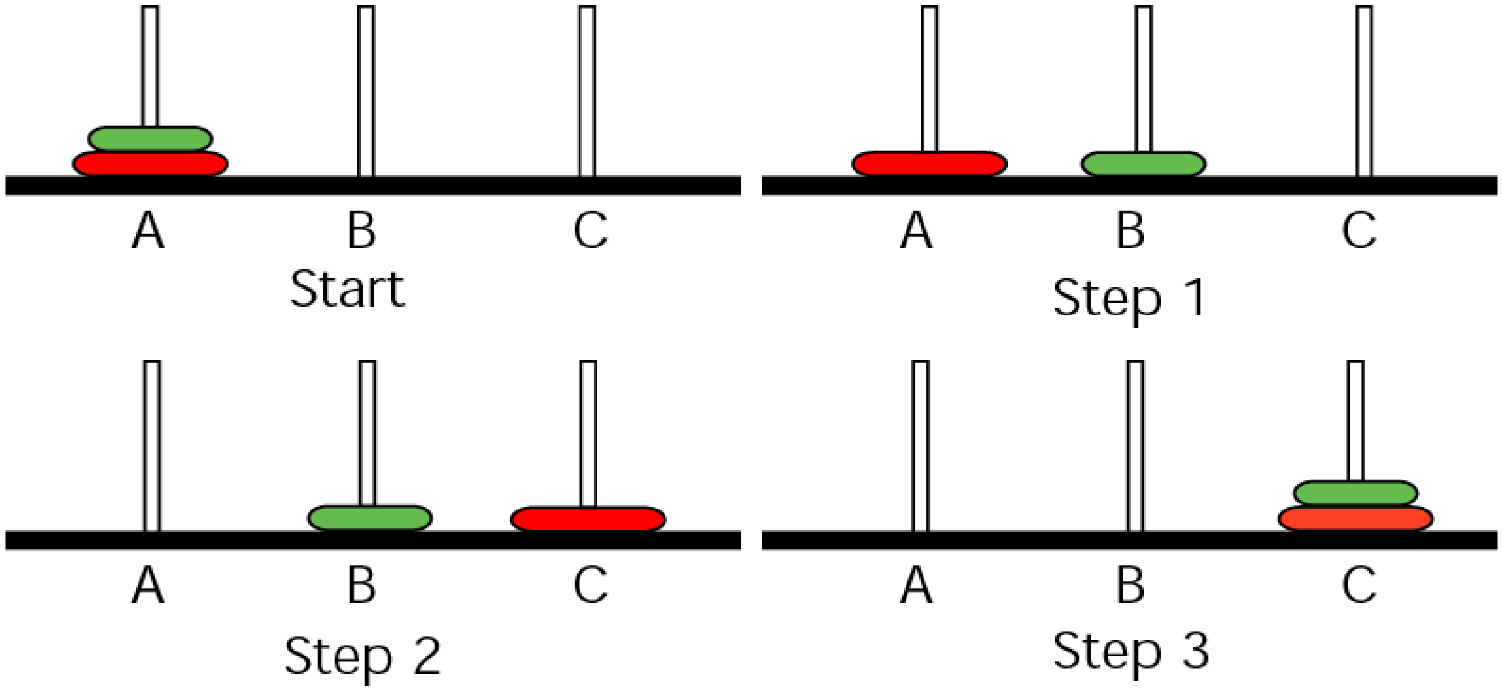
Every recursive call must either solve part of the problem or reduce the size of the problem.

Figure 6-29 Towers of Hanoi—start position



Brooks/Cole
Thomson Learning™

Figure 6-30 Towers solution for two disks



Brooks/Cole
Thomson Learning™

Figure 6-31 Towers solution for three disks

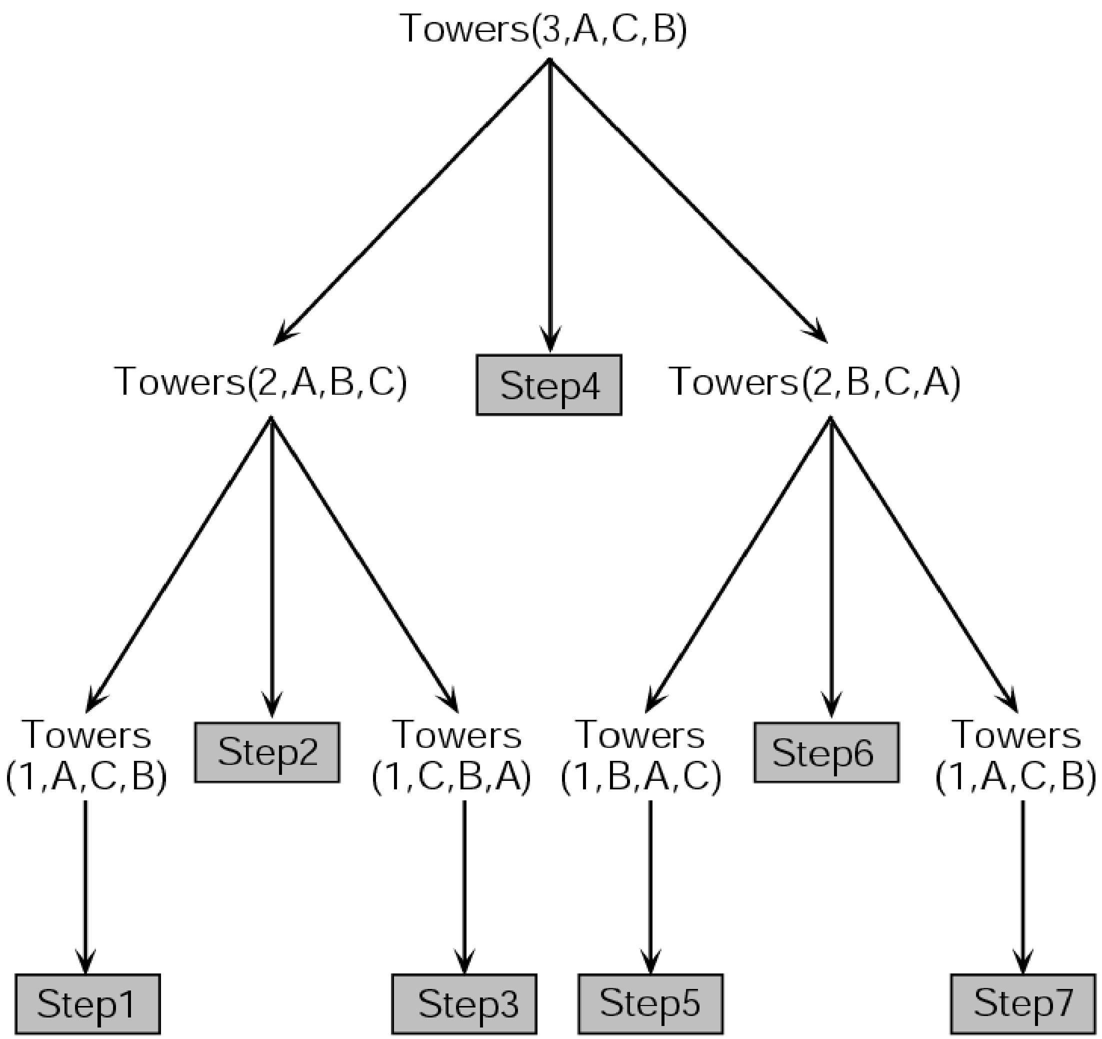
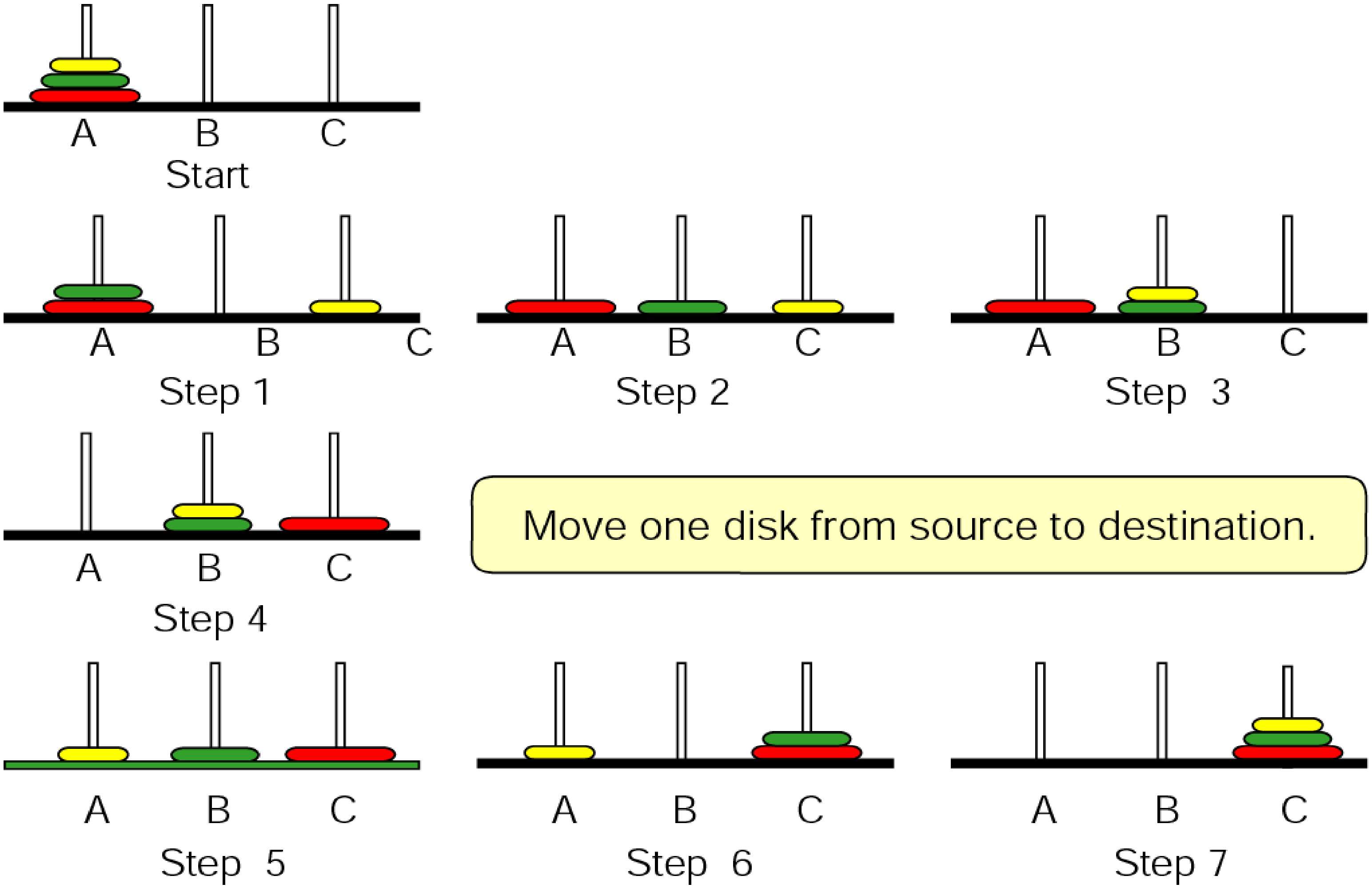


Figure 6-31 Towers solution for three disks (continued)



Move one disk from source to destination.

Brooks/Cole
Thomson Learning

6.10

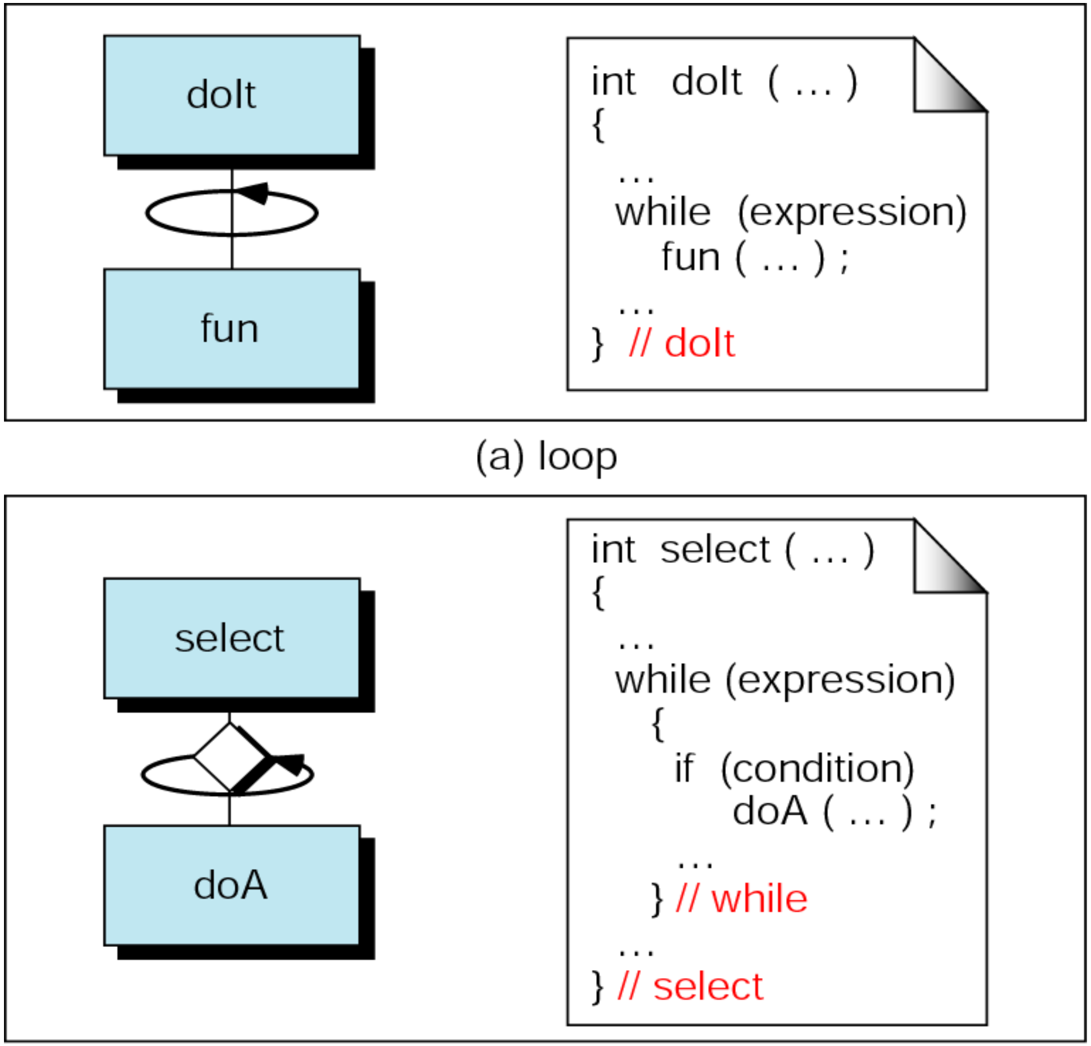
A Programming Example—The Calculator Program

6.11

Software Engineering and Programming Style

Brooks/Cole
Thomson Learning™

Figure 6-32 Structure chart symbols for loops



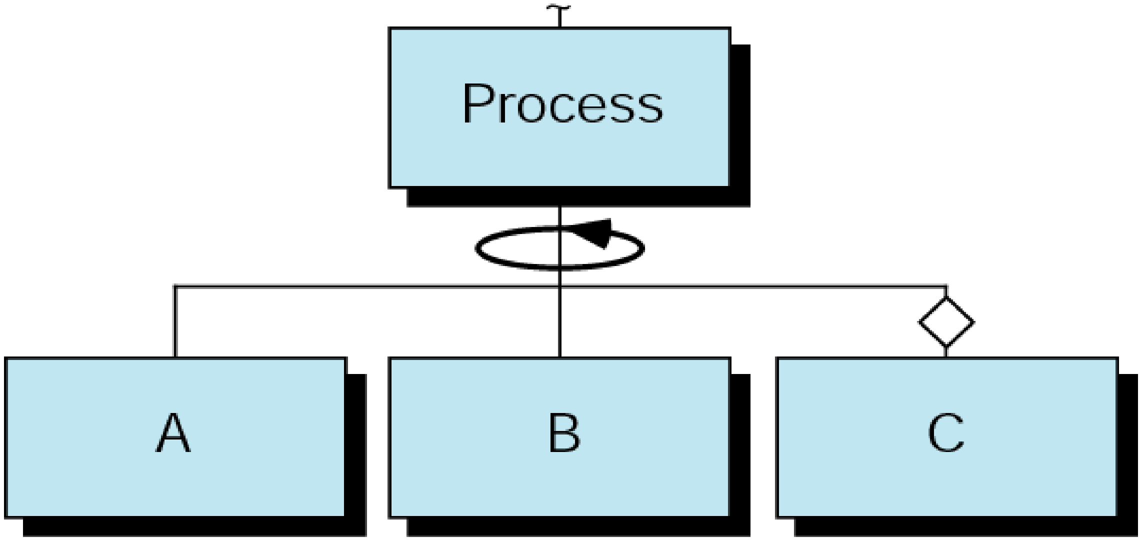
```
int   doIt  ( … )
{
  …
  while  (expression)
     fun ( … ) ;
  …
}  // doIt
```

(a) loop

```
int  select ( … )
{
  …
  while (expression)
     {
       if  (condition)
            doA ( … ) ;
       …
     } // while
  …
} // select
```

(b) conditional loop

Figure 6-33 Structure chart for process

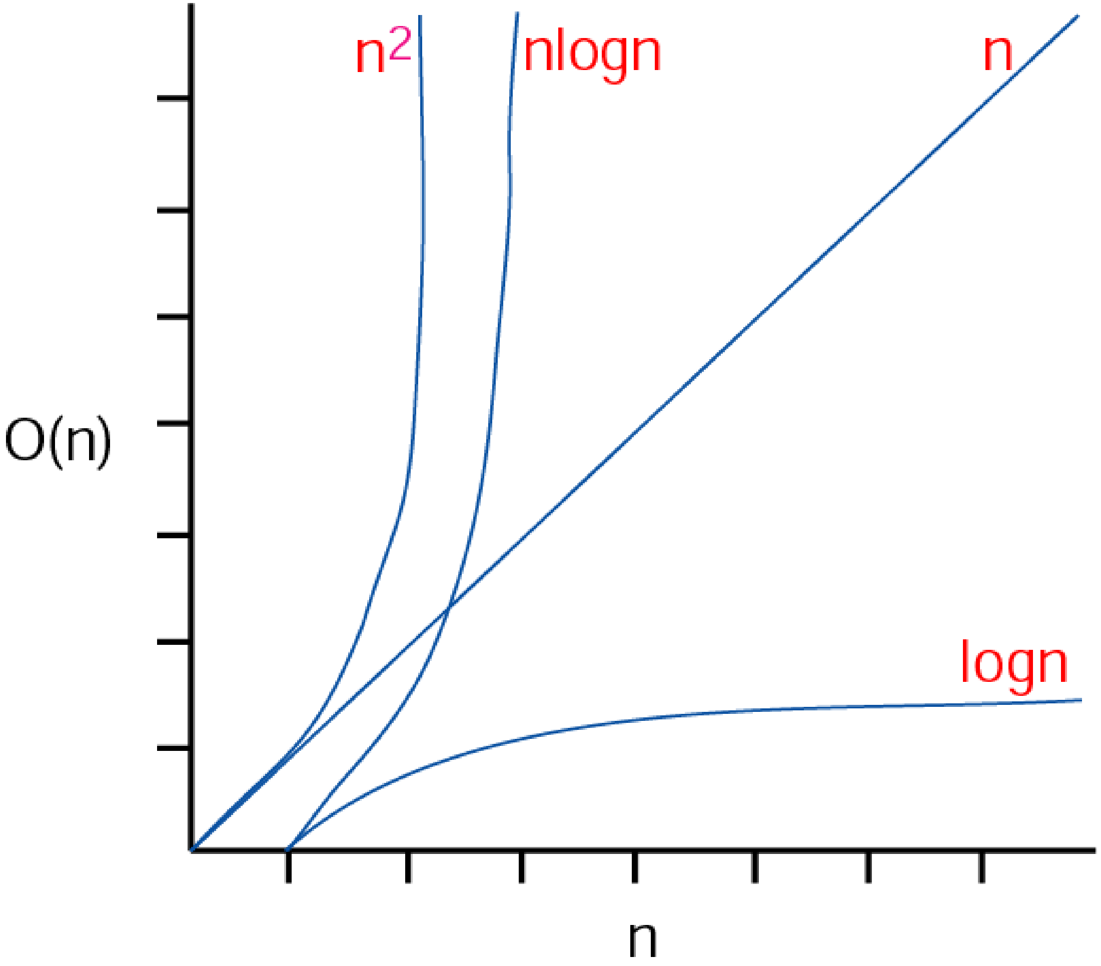


(a) Design

```
while ( … )
  {
    A ( … ) ;
    . . .
    B ( … ) ;
    . . .
    if  (  …  )
        C ( … ) ;
    } // while
```

(b) Code

Figure 6-34 Measures of efficiency


n^2
nlogn
n
logn
O(n)
n