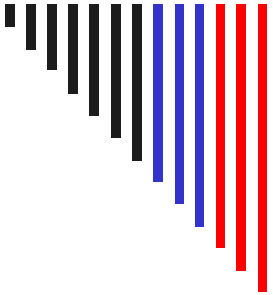


Chapter 6 Repetition

Objectives

- To understand basic loop concepts:
  - pretest loops and post-test loops
  - loop initialization and updating
  - event and counter controlled loops
- To understand and be able to select the best loop construct for a given problem.
- To write programs that use the *while*, *for*, or *do ... while* statements.
- To understand the basic concepts and usage of recursion algorithms.
- To understand and be able to determine the efficiency of an algorithm through an analysis of its looping constructs.

6-1 Concept of a loop

The real power of computers is in their ability to repeat an operation or a series of operations many times. This repetition, called looping, is one of the basic structured programming concepts.

Each loop must have an expression that determines if the loop is done. If it is not done, the loop repeats one more time; if it is done, the loop terminates.

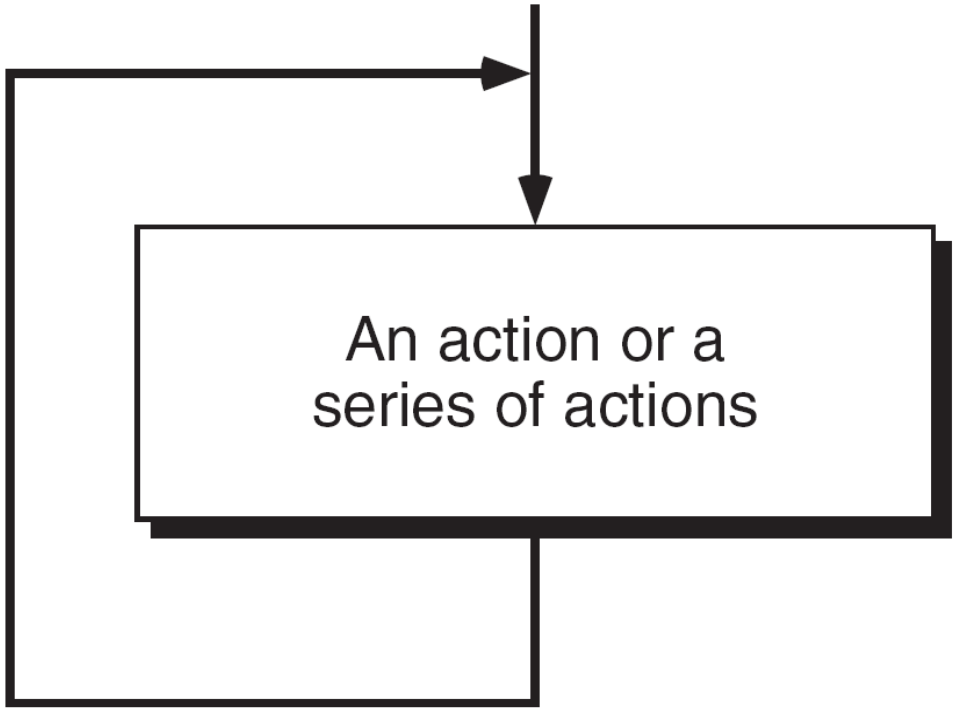


FIGURE 6-1 **Concept of a Loop**

6-2 Pretest and Post-test Loops

We need to test for the end of a loop, but where should we check it—before or after each iteration? We can have either a pre- or a post-test terminating condition.

In a pretest loop , the condition is checked at the beginning of each iteration.

In a post-test loop, the condition is checked at the end of each iteration.

Note

Pretest Loop

In each iteration, the control expression is tested first. If it is true, the loop continues; otherwise, the loop is terminated.

Post-test Loop

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

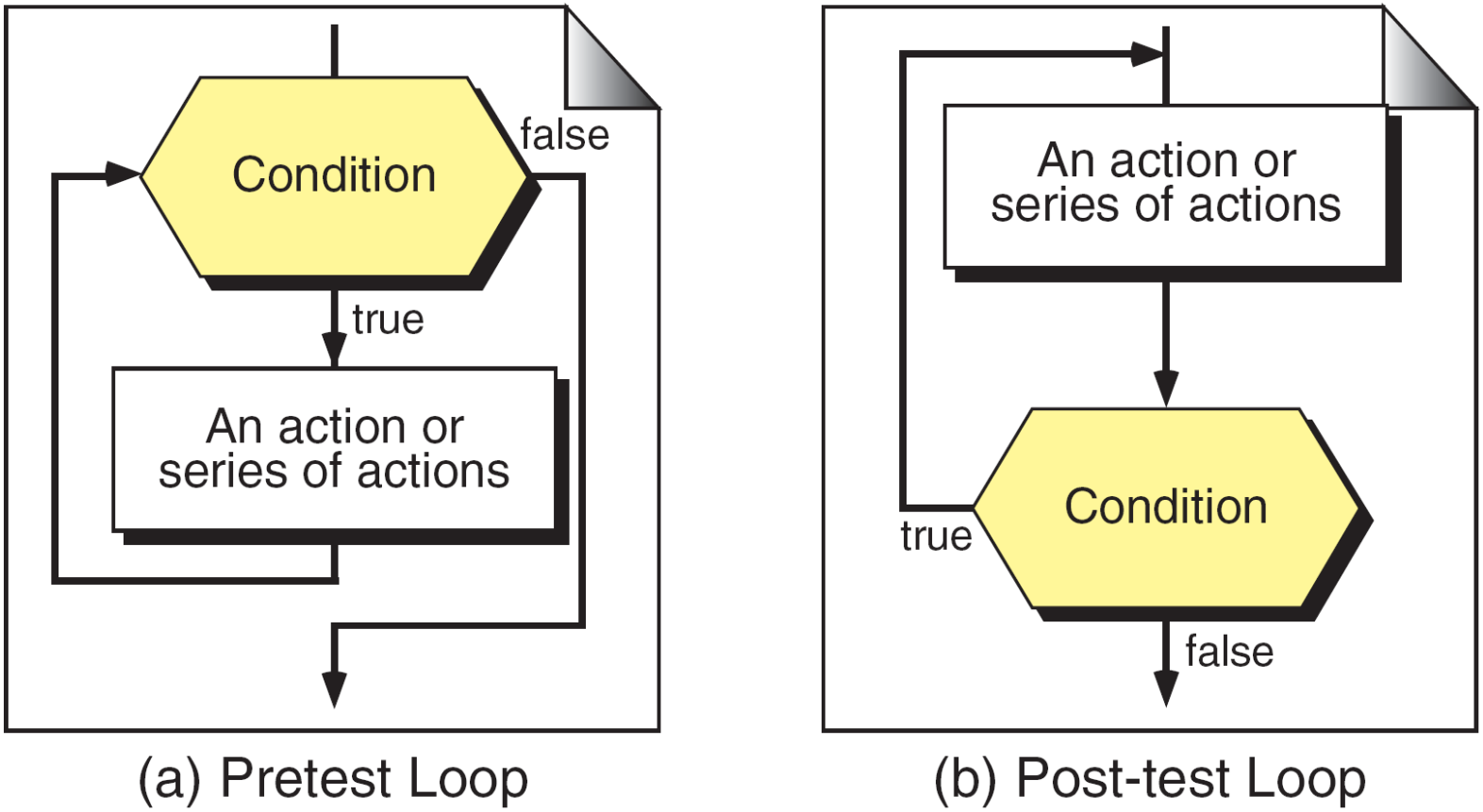


FIGURE 6-2 Pretest and Post-test Loops

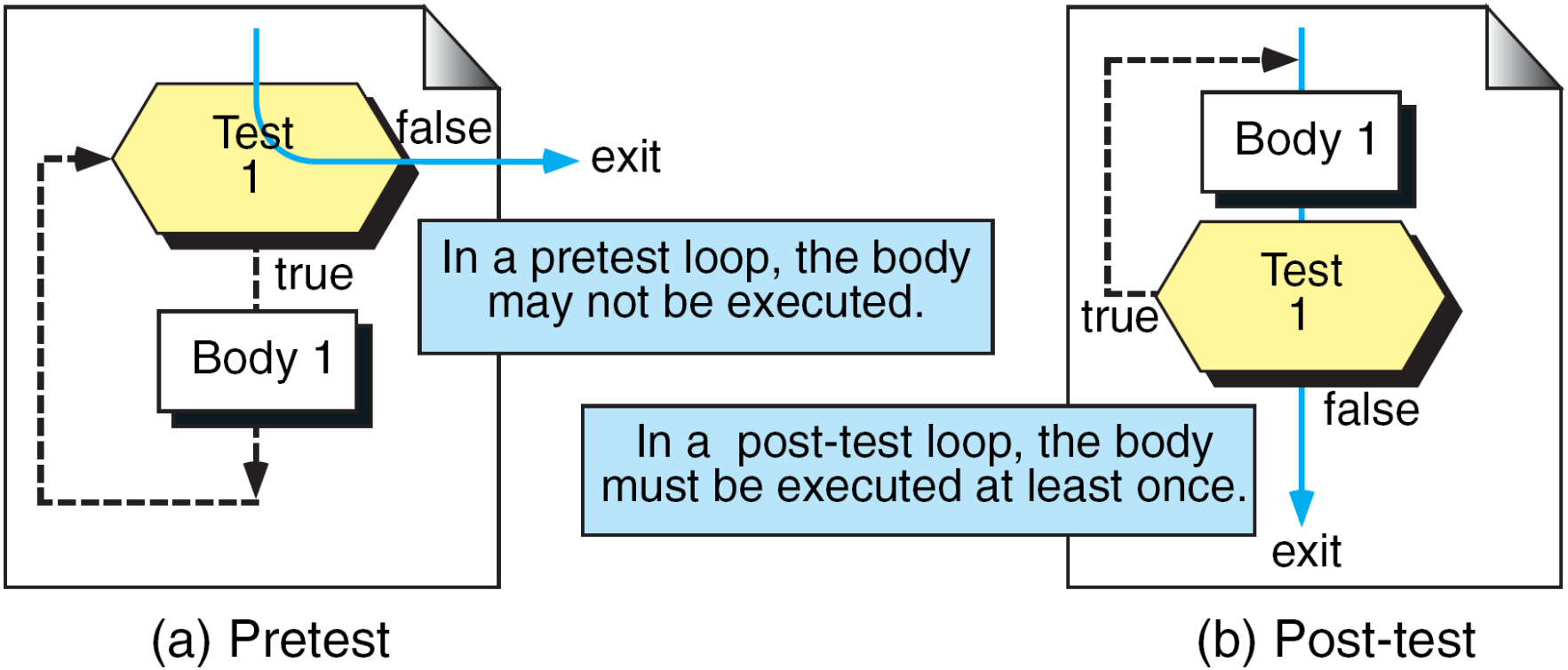


FIGURE 6-4 Minimum Number of Iterations in Two Loops

6-3 Initialization and Updating

In addition to the loop control expression, two other processes, initialization and updating, are associated with almost all loops.

Topics discussed in this section:

Loop Initialization

Loop Update

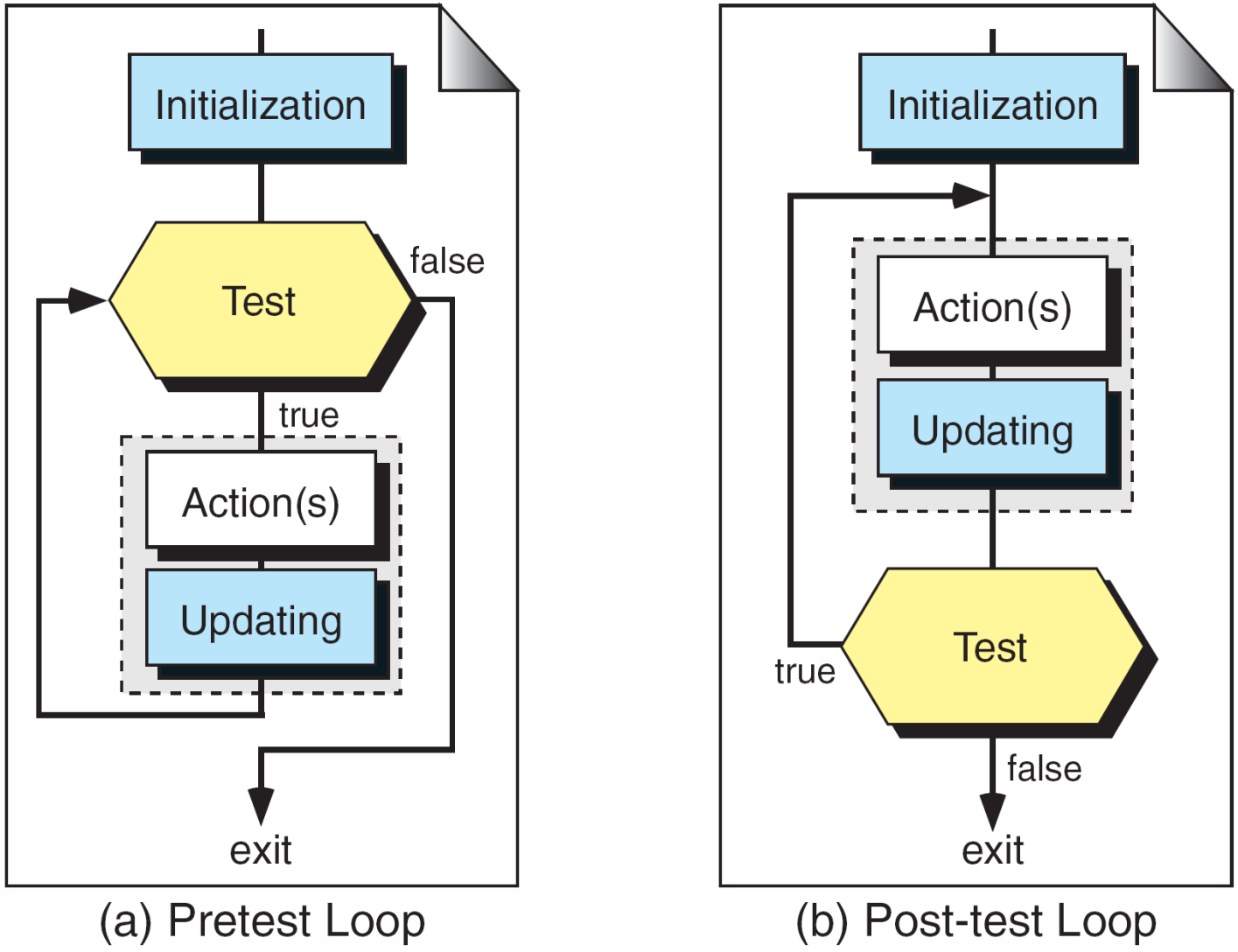


FIGURE 6-5 Loop Initialization and Updating

6-4 Event- and Counter-Controlled Loops

All the possible expressions that can be used in a loop limit test can be summarized into two general categories: event-controlled loops and counter-controlled loops.

Topics discussed in this section:

Event-Controlled Loops
Counter-Controlled Loops
Loop Comparison

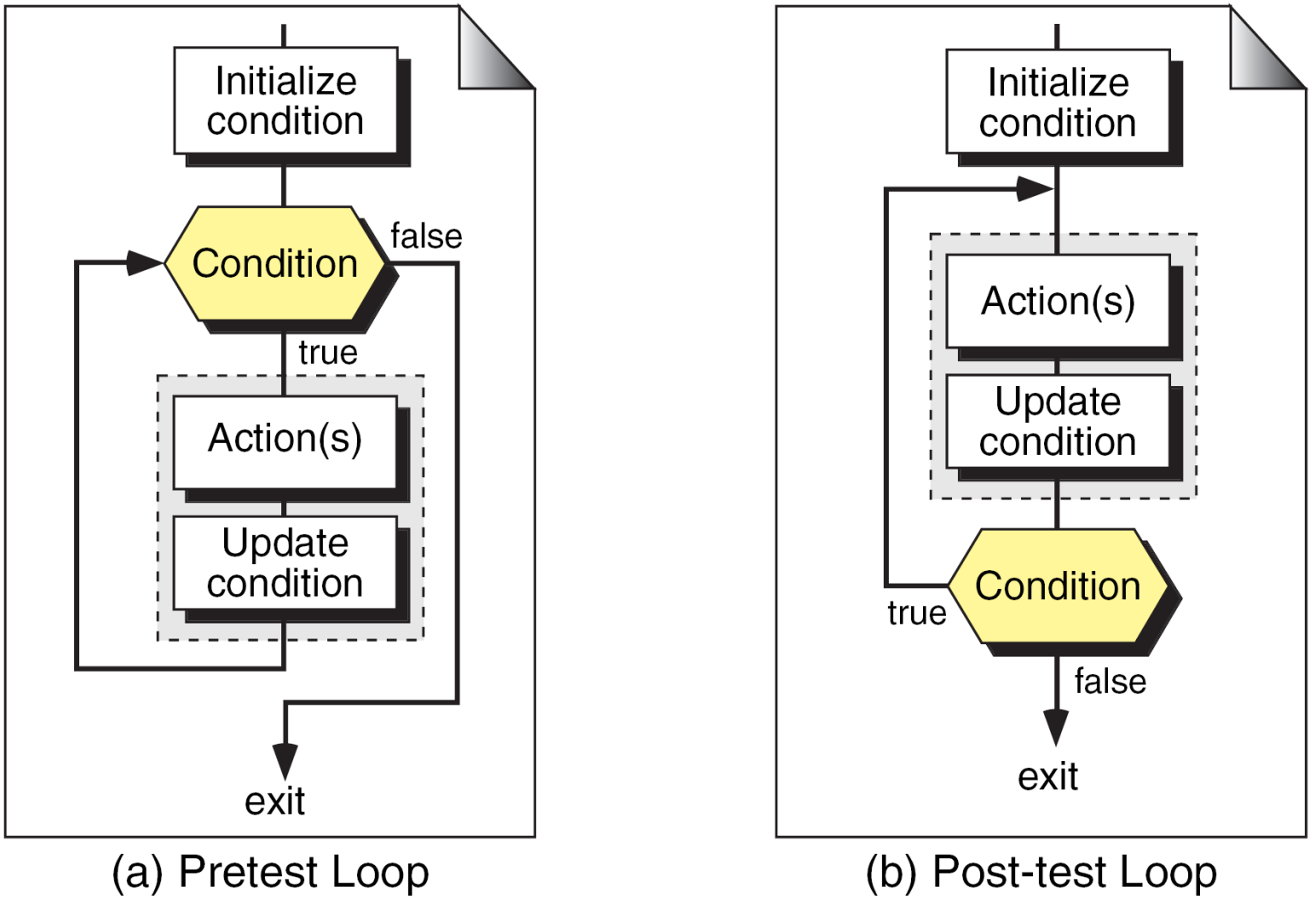


FIGURE 6-7 Event-controlled Loop Concept

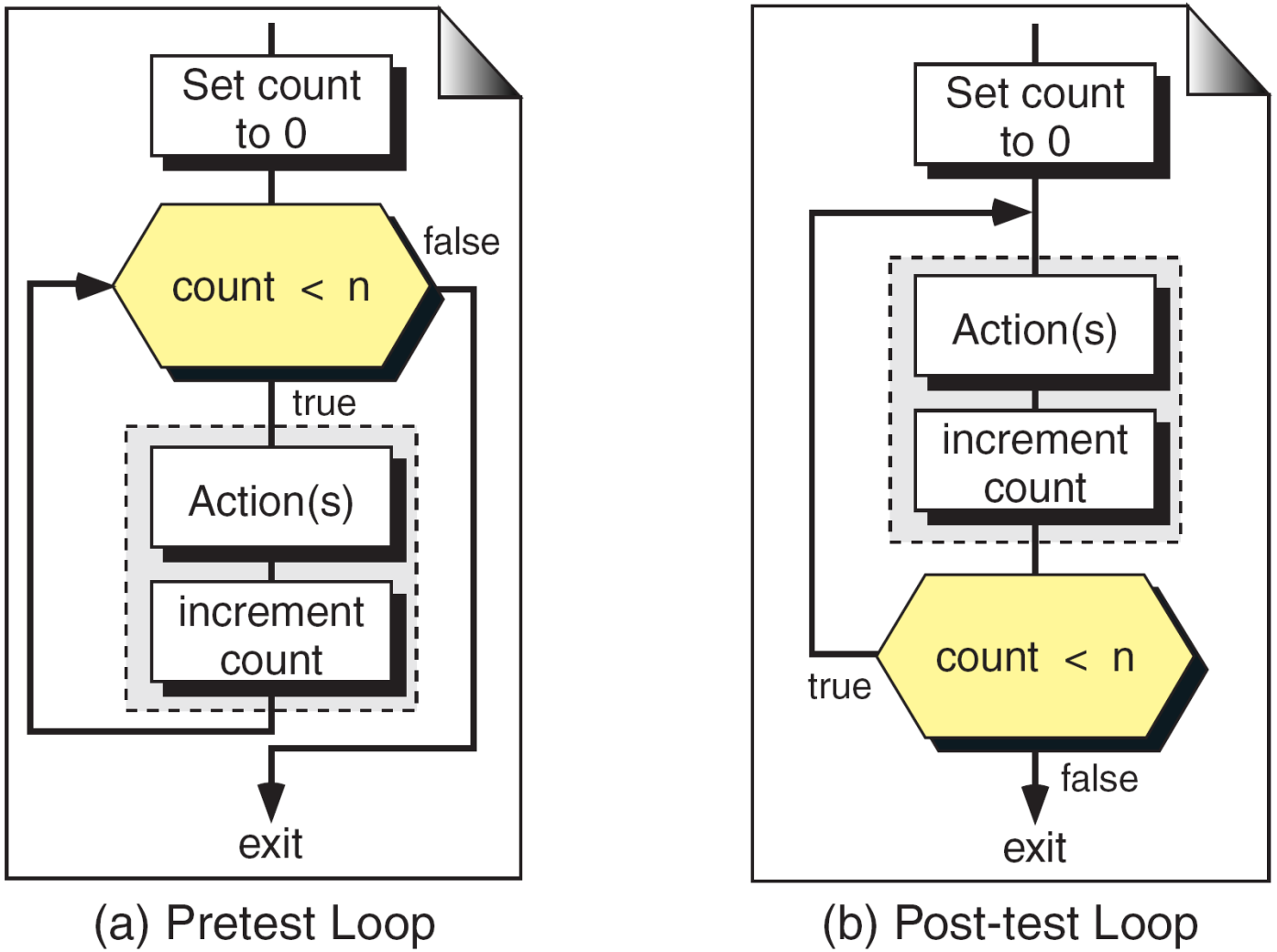


FIGURE 6-8 Counter-controlled Loop Concept

Pretest Loop		Post-test Loop	
Initialization:	1	Initialization:	1
Number of tests:	*n* + 1	Number of tests:	n
Action executed:	n	Action executed:	n
Updating executed:	n	Updating executed:	n
Minimum iterations:	0	Minimum iterations:	1

Table 6-1 Loop Comparisons

6-5 Loops in C

C has three loop statements: the* while*, the* for*, and the* do…while*. The first two are pretest loops, and the the third is a post-test loop. We can use all of them for event-controlled and counter-controlled loops.

Topics discussed in this section:

The *while* Loop
The *for* Loop
The *do…while* Loop
The Comma Expression

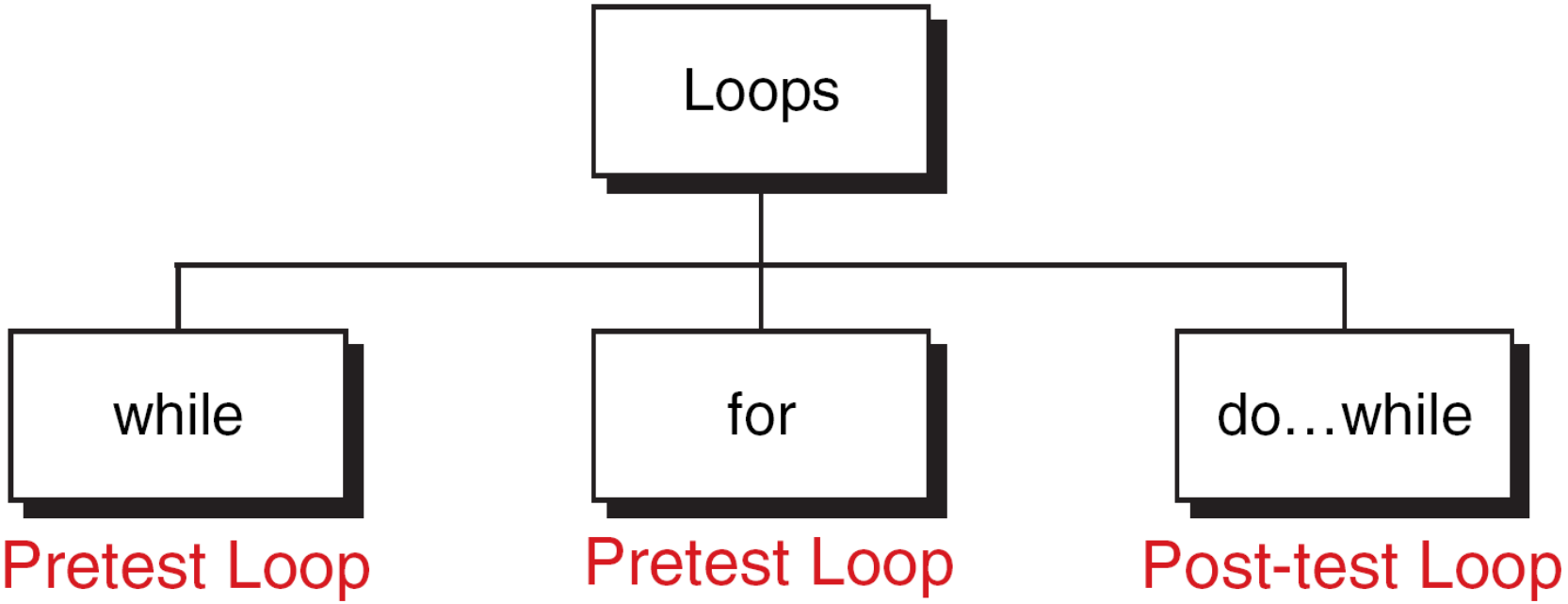
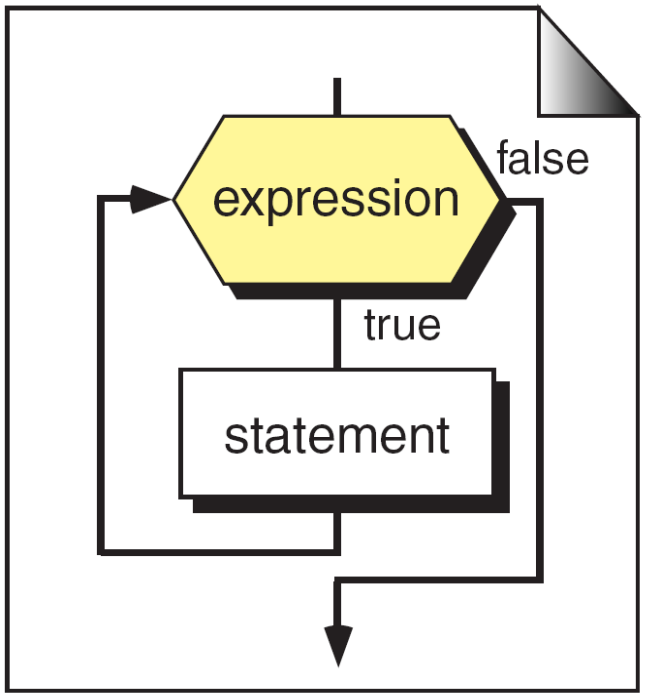


FIGURE 6-9 **C Loop Constructs**



(a) Flowchart

```
while (expression)

    statement
```

(b) Sample Code

FIGURE 6-10 The while Statement

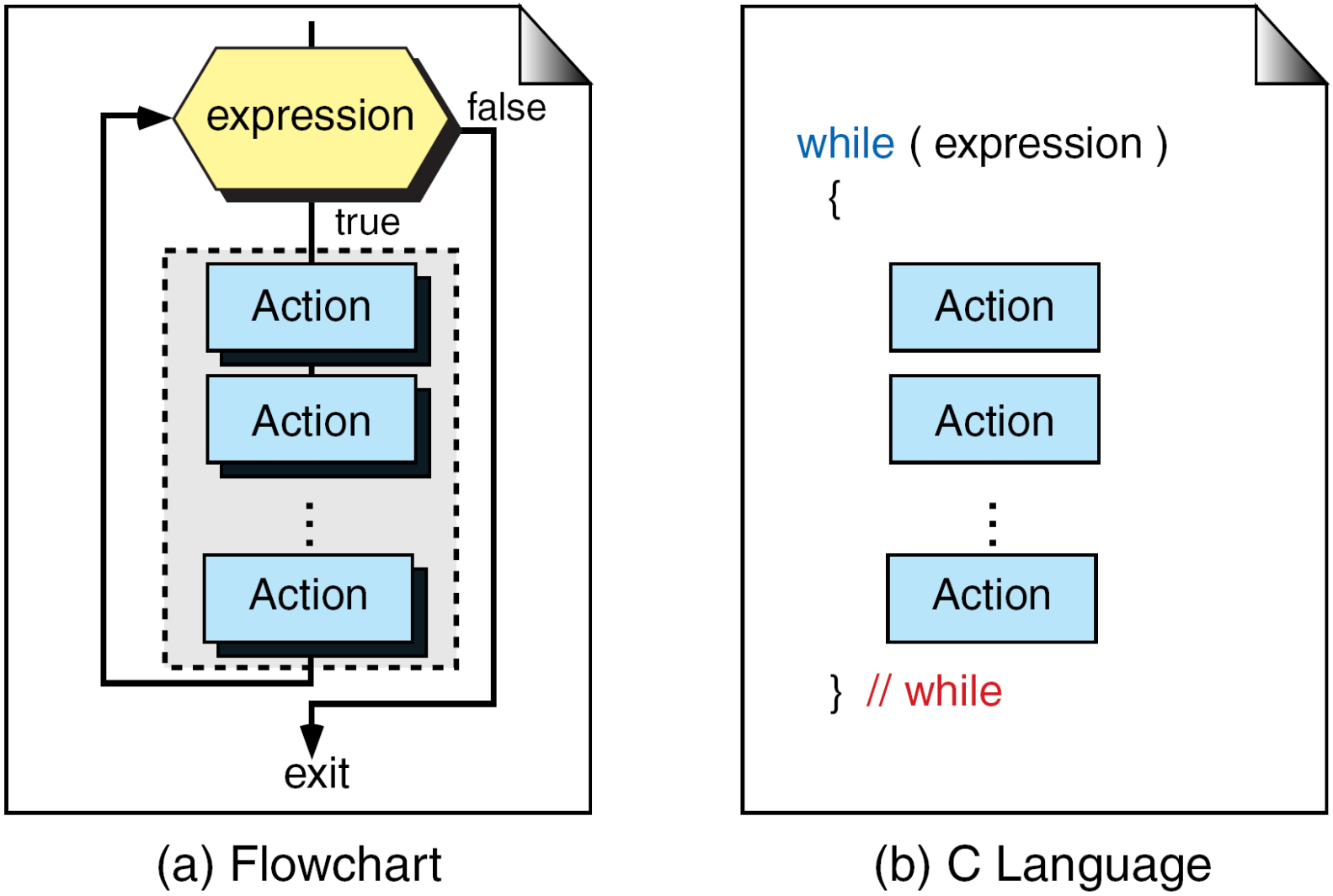


FIGURE 6-11 Compound while Statement

PROGRAM 6-1 Process-control System Example

```
 1  while (true)
 2     {
 3      temp = getTemperature();
 4      if (temp < 68)
 5          turnOnHeater();
 6      else if (temp > 78)
 7          turnOnAirCond();
 8      else
 9         {
10          turnOffHeater();
11          turnOffAirCond();
12         } // else
13     } // while true
```

PROGRAM 6-2 A *while* Loop to Print Numbers

```
 1  /* Simple while loop that prints numbers 10 per line.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Local Declarations
10     int num;
11     int lineCount;
12
13  // Statements
14     printf ("Enter an integer between 1 and 100: ");
15     scanf  ("%d", &num);          // Initialization
16
17     // Test number
18     if (num > 100)
19         num = 100;
20
```

PROGRAM 6-2 A *while* Loop to Print Numbers

```
21      lineCount = 0;
22      while (num > 0)
23          {
24           if (lineCount < 10)
25               lineCount++;
26           else
27              {
28               printf("\n");
29               lineCount = 1;
30              } // else
31           printf("%4d", num--);        // num-- updates loop
32          } // while
33      return 0;
34  }   // main
```

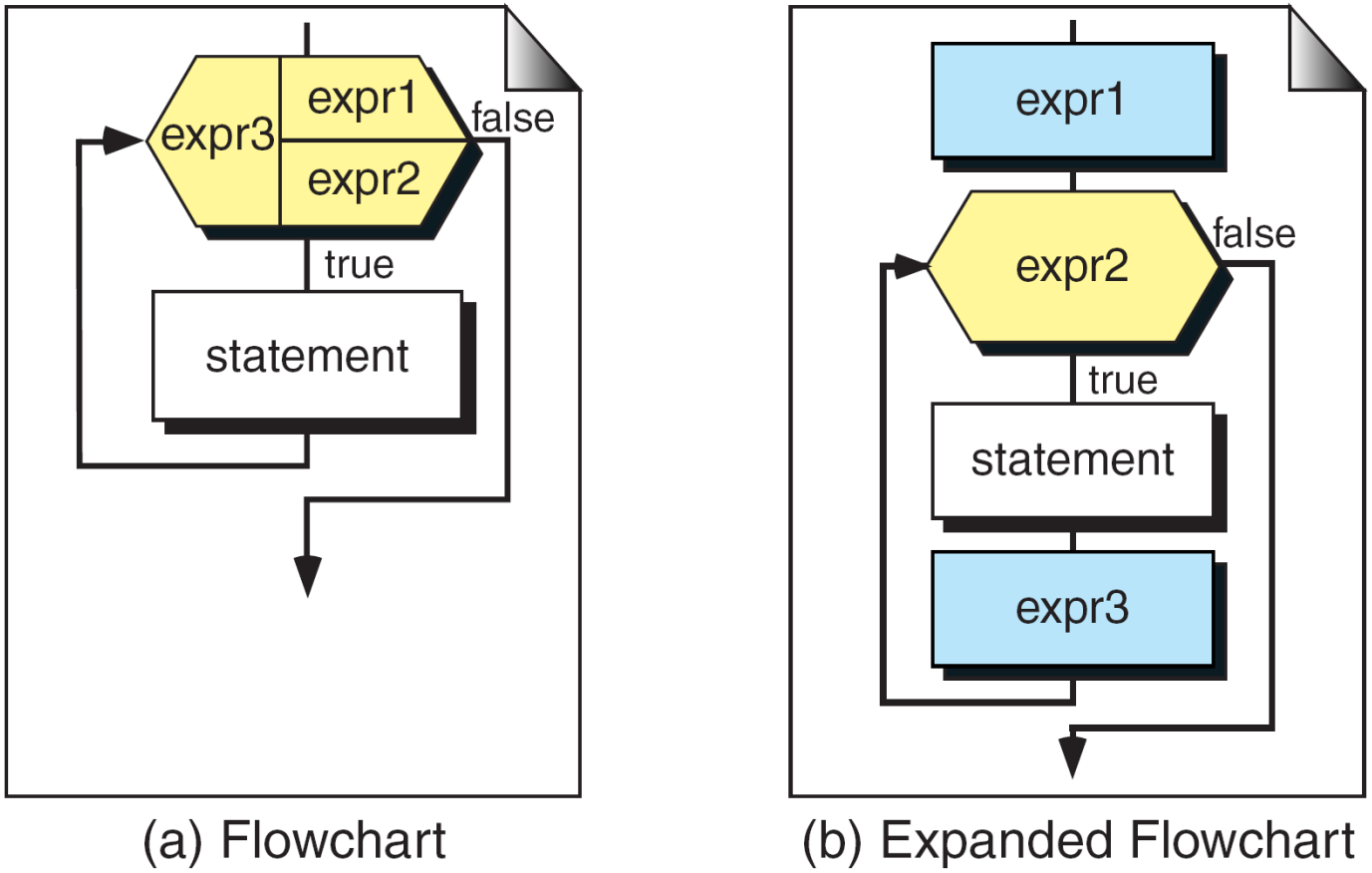
```
Results:
Enter an integer between 1 and 100: 15
  15   14   13   12   11   10    9    8    7    6
   5    4    3    2    1
```

PROGRAM 6-3 Adding a List of Numbers

```
 1  /* Add a list of integers from the standard input unit
 2     Written by:
 3     Date:
 4  */
 5  #include <stdio.h>
 6  int main (void)
 7  {
 8  // Local Declarations
 9     int x;
10     int sum = 0;
11
12  // Statements
13     printf("Enter your numbers: <EOF> to stop.\n");
14     while (scanf("%d", &x) != EOF)
15        sum += x;
16     printf ("\nThe total is: %d\n", sum);
17     return  0;
18  }  // main
```

PROGRAM 6-3 Adding a List of Numbers

```
Results:
Enter your numbers: <EOF> to stop
15
22
3^d
The total is: 40
```




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

FIGURE 6-12 ***for*** **Statement**

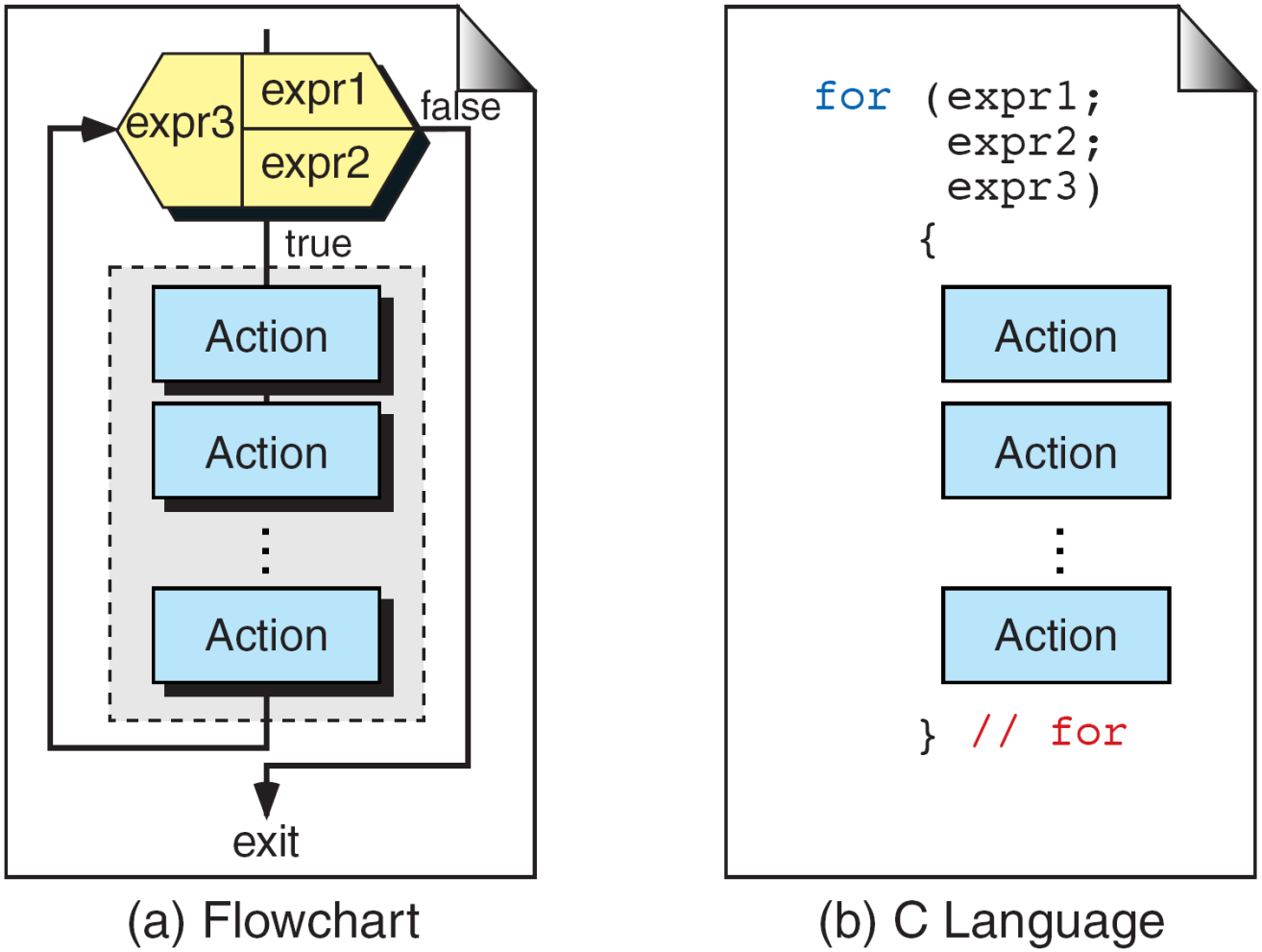


FIGURE 6-13 Compound *for* Statement

Note

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

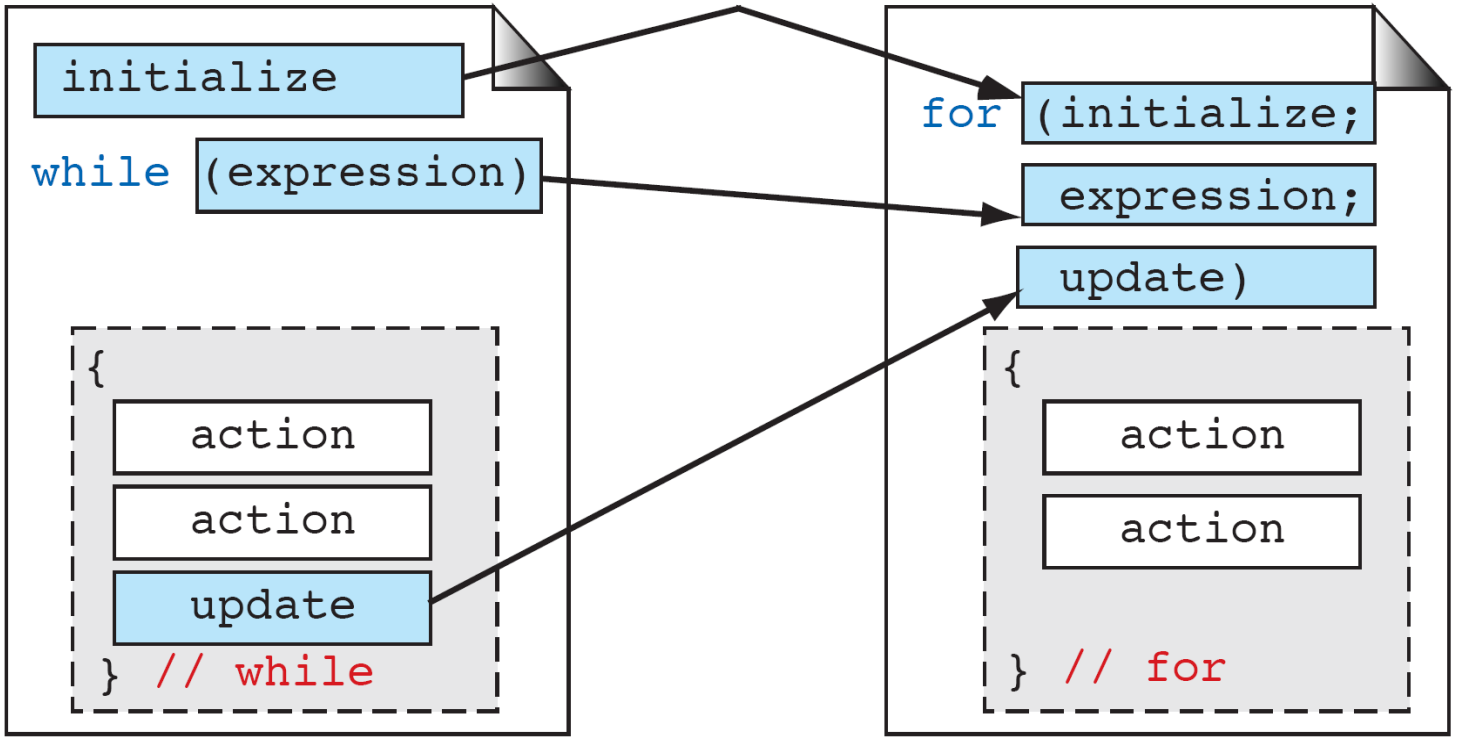


FIGURE 6-14 **Comparing *for* and *while* Loops**

PROGRAM 6-4 Example of a *for* Loop

```
1  /* Print number series from 1 to user-specified limit.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  int main (void)
7  {
8  // Local Declarations
9     int limit;
10
11 // Statements
12    printf ("\nPlease enter the limit: ");
13    scanf ("%d", &limit);
14    for (int i = 1; i <= limit; i++)
15       printf("\t%d\n", i);
16    return 0;
17 }  // main
```

PROGRAM 6-4 Example of a *for* Loop

```
Results:
Please enter the limit: 3
 1
 2
 3
```

PROGRAM 6-5 A Simple Nested *for* Loop

```
 1  /* Print numbers on a line.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Statements
10     for (int i = 1; i <= 3; i++)
11        {
12         printf("Row %d: ", i);
13         for (int j = 1; j<= 5; j++)
14              printf("%3d", j);
15         printf("\n");
16        } // for i
17     return 0;
18  }  // main
```

PROGRAM 6-5 A Simple Nested *for* Loop

```
Results:
Row 1:    1   2   3   4   5
Row 2:    1   2   3   4   5
Row 3:    1   2   3   4   5
```

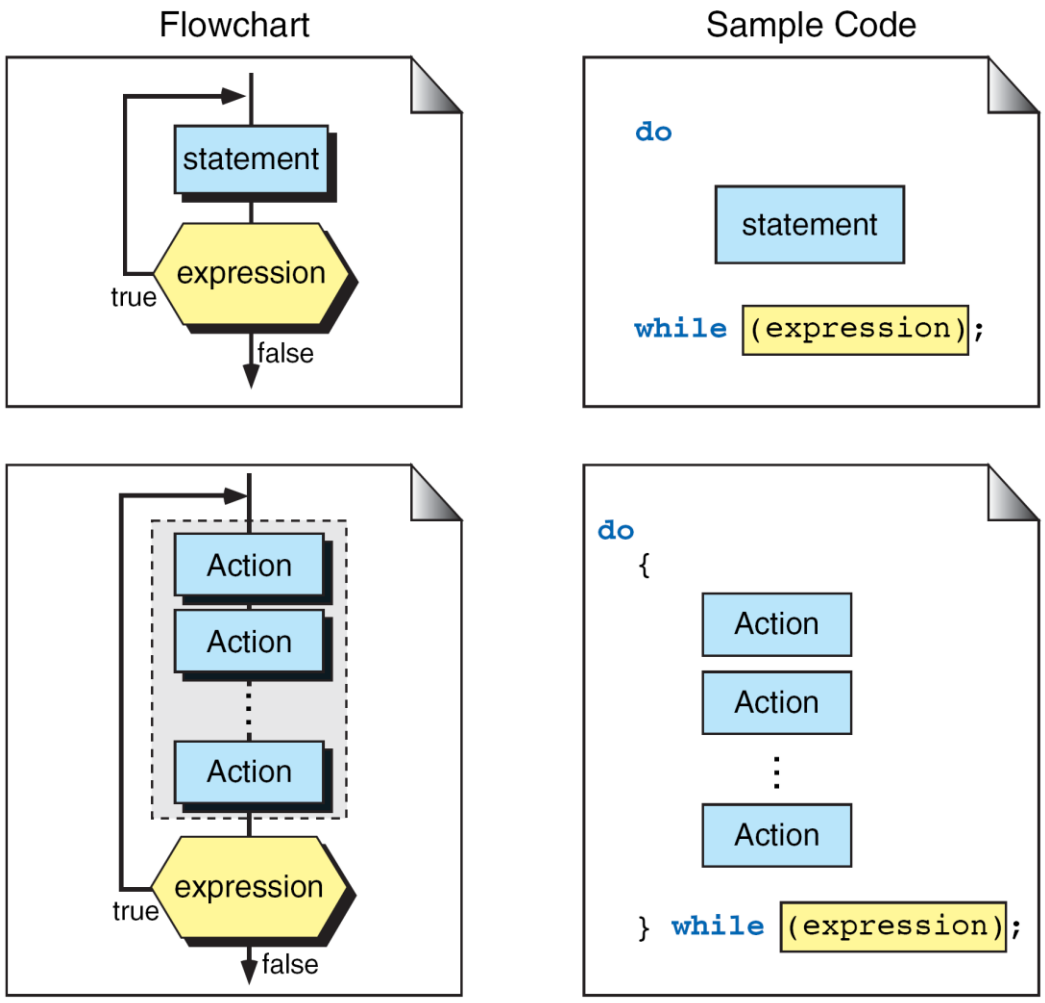


FIGURE 6-15 ***do…while* Statement**

PROGRAM 6-6 Two Simple Loops

```
 1  /* Demonstrate while and do...while loops.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Local Declarations
10     int loopCount;
11
12  // Statements
13     loopCount = 5;
14     printf("while loop      : ");
15     while (loopCount > 0)
16        printf ("%3d", loopCount--);
17     printf("\n\n");
18
```

PROGRAM 6-6 Two Simple Loops

```
19      loopCount = 5;
20      printf("do...while loop: ");
21      do
22         printf ("%3d", loopCount--);
23      while (loopCount > 0);
24      printf("\n");
25      return 0;
26  }   // main
```

```
Results
while loop      :   5   4   3   2   1

do...while loop:   5   4   3   2   1
```

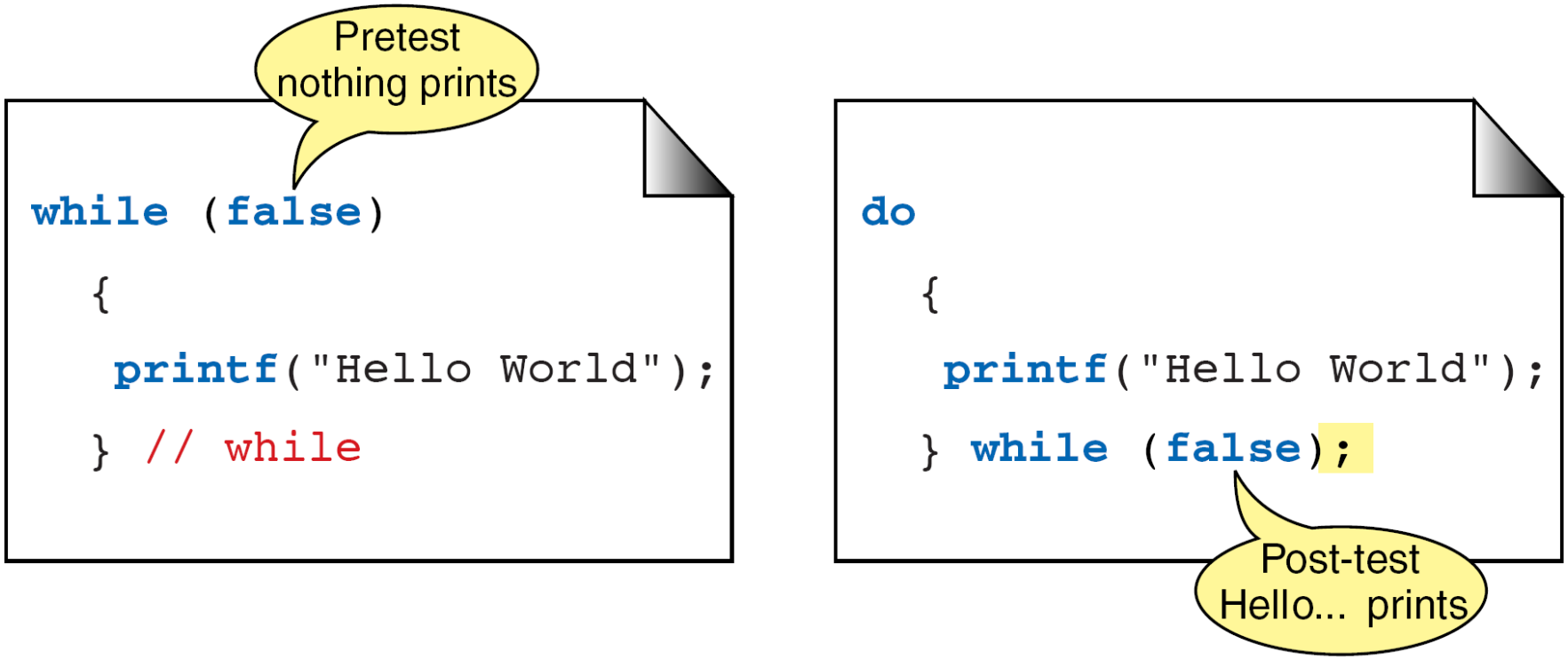


FIGURE 6-16 Pre- and Post-test Loops

PROGRAM 6-7 Adding a List with the *do…while*

```
 1  /* Add a list of integers from the standard input unit
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Local Declarations
10     int x;
11     int sum = 0;
12     int testEOF;
13
14     // Statements
15     printf("Enter your numbers: <EOF> to stop.\n");
16     do
17        {
18         testEOF = scanf("%d", &x);
```

PROGRAM 6-7 Adding a List with the *do…while*

```
19          if (testEOF != EOF)
20              sum += x;
21        } while (testEOF != EOF);
22     printf ("\nTotal: %d\n", sum);
23     return  0;
24  }  // main
```

```
Results:
Run 1:
   Enter your numbers: <EOF> to stop.
   10 15 20 25 ^d
   Total: 70

Run 2:
   Enter your numbers: <EOF> to stop.
   ^d
   Total: 0
```

The Comma Expression

- Complex expression made up of two expressions separated by a comma
- Most often used in *for* statements

```
for (sum = 0, i = 1; i <= 20; i++) {
        scanf("%d", &a);
        sum += a;
} // for
```

FIGURE 6-17 Nested Comma Expression

PROGRAM 6-8 Comparison of *while* and *do…while*

```
 1  /* Demonstrate while and do…while loops.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Local Declarations
10     int loopCount;
11     int testCount;
12
13  // Statements
14     loopCount = 1;
15     testCount = 0;
16     printf("while loop:       ");
17     while (testCount++, loopCount <= 10)
18        printf("%3d", loopCount++);
19     printf("Loop Count:       %3d\n", loopCount);
```

PROGRAM 6-8 Comparison of *while* and *do…while*

```
20    printf("Number of tests:  %3d\n", testCount);
21
22    loopCount = 1;
23    testCount = 0;
24    printf("\ndo...while loop:  ");
25    do
26       printf("%3d",    loopCount++);
27    while (testCount++,  loopCount <= 10);
28
29    printf("\nLoop Count:       %3d\n", loopCount);
30    printf("Number of tests:  %3d\n", testCount);
31    return 0;
32 }  // main
```

```
Results:
while loop:          1   2   3   4   5   6   7   8   9 10
Loop Count:         11
Number of tests:    11

do...while loop:     1   2   3   4   5   6   7   8   9 10
Loop Count:         11
Number of tests:    10
```

6-6 Loop Examples

This section contains several short examples of loop applications. Each program demonstrates one or more programming concepts that you will find helpful in solving other problems.

Topics discussed in this section:

***for* Loops**
***while* LOOPS**
***do…while* LOOPS**

PROGRAM 6-9 Compound Interest

```
 1  /* Print report showing value of investment.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Local Declarations
10     double presVal;
11     double futureVal;
12     double rate;
13     int    years;
14     int    looper;
15
16  // Statements
17     printf("Enter value of investment:   ");
```

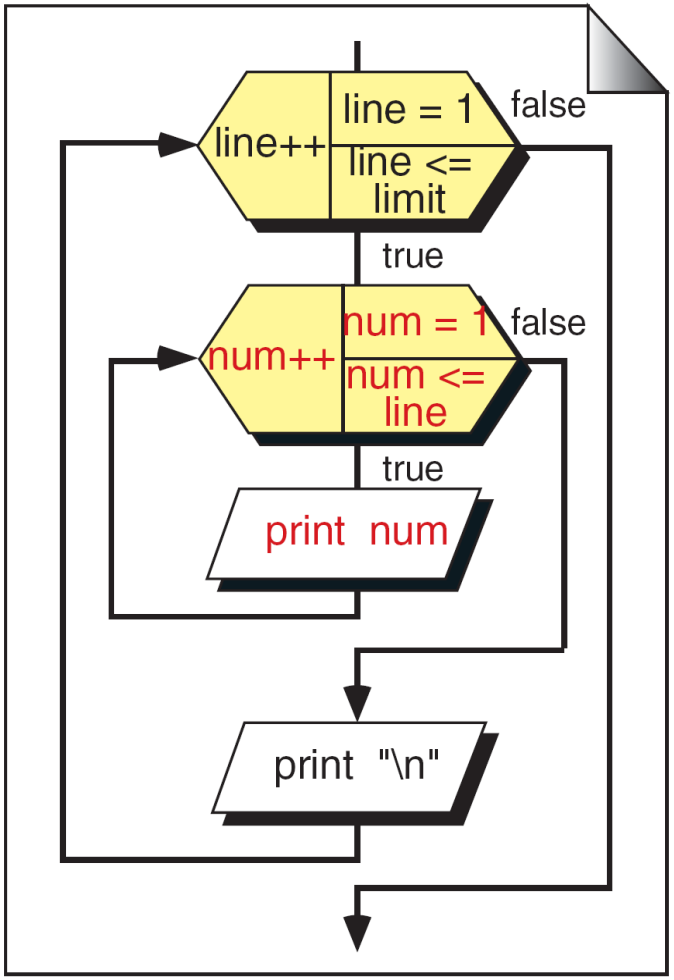
PROGRAM 6-9 Compound Interest

```
18      scanf ("%lf", &presVal);
19      printf("Enter rate of return (nn.n): ");
20      scanf ("%lf", &rate);
21      printf("Enter number of years:       ");
22      scanf ("%d", &years);
23
24      printf("\nYear    Value\n");
25      printf("====  ========\n");
26      for (futureVal = presVal, looper = 1;
27           looper <= years;
28           looper++)
29          {
30           futureVal = futureVal * (1 + rate/100.0);
31           printf("%3d%11.2lf\n", looper, futureVal);
32          } // for
33      return 0;
34  }  // main
```

PROGRAM 6-9 Compound Interest

```
Results:
Enter value of investment:   10000
Enter rate of return (nn.n): 7.2
Enter number of years:       5

Year       Value
====     ========
  1      10720.00
  2      11491.84
  3      12319.25
  4      13206.24
  5      14157.09
```



(a) Flowchart

```
1  set line to 1
2  loop (line not greater than limit)
     1  set num to 1
     2  loop (num not greater than line)
          1  print num
          2  increment num
     3  end loop
     4  advance line
     5  increment line
3  end loop
```

(b) Pseudocode

FIGURE 6-18 Print Right Triangle Flowchart and Pseudocode

PROGRAM 6-10 Print Right Triangle Using Nested *for*

```
1  /* Print a number series from 1 to a user-specified limit
2     in the form of a right triangle.
3        Written by:
4        Date:
5  */
6  #include <stdio.h>
7
8  int main (void)
9  {
10 // Local Declarations
11    int limit;
12
13 // Statements
14    // Read limit
15    printf("\nPlease enter a number between 1 and 9: ");
16    scanf("%d", &limit);
17
18    for (int lineCtrl = 1; lineCtrl <= limit; lineCtrl++)
19       {
```

PROGRAM 6-10 Print Right Triangle Using Nested *for*

```
20          for (int numCtrl = 1;
21                   numCtrl <= lineCtrl;
22                   numCtrl++)
23              printf("%1d", numCtrl);
24
25          printf("\n");
26        } // for lineCtrl
27     return 0;
28  }  // main
```

```
Results:
Please enter a number between 1 and 9: 6
1
12
123
1234
12345
123456
```

PROGRAM 6-11 Print Number Series Using User-specified Limit

```
1  /* Print number series from 1 to a user-specified limit
2     in the form of a rectangle.
3        Written by:
4        Date:
5  */
6  #include <stdio.h>
7
8  int main (void)
9  {
10 // Local Declarations
11    int limit;
12
13 //Statements
14    // Read limit
15    printf("Please enter a number between 1 and 9: ");
16    scanf("%d", &limit);
17
```

PROGRAM 6-11 Print Number Series Using User-specified Limit

```
19          {
20           for (int col = 1; col <= limit; col++)
21                if (row >= col)
22                   printf("%d", col);
23                else
24                   printf("*");
25           printf("\n");
26          } // for row ...
27       return 0;
28   }  // main
```

```
Results:
Please enter a number between 1 and 9: 6
1*****
12****
123***
1234**
12345*
123456
```

PROGRAM 6-12 Print Calendar Month

```
1  /* Test driver for function to print a calendar month.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6
7  // Prototype Declarations
8  void printMonth (int startDay, int days);
9
10 int main (void)
11 {
12 // Statements
13    printMonth (2, 29);              // Day 2 is Tuesday
14    return 0;
15 }  // main
16
17 /* ================= printMonth ================
18    Print one calendar month.
19       Pre   startDay is day of week relative
20                to Sunday (0)
```

PROGRAM 6-12 Print Calendar Month

```
21              days is number of days in month
22        Post  Calendar printed
23  */
24  void printMonth (int startDay, int days)
25  {
26  // Local Declarations
27     int weekDay;
28
29  // Statements
30     // print day header
31     printf("Sun Mon Tue Wed Thu Fri Sat\n");
32     printf("--- --- --- --- --- --- ---\n");
33
34     // position first day
35     for (weekDay = 0; weekDay < startDay; weekDay++)
36         printf("    ");
37
38     for (int dayCount = 1; dayCount <= days; dayCount++)
39         {
```

PROGRAM 6-12 Print Calendar Month

```
40            if (weekDay > 6)
41               {
42                printf("\n");
43                weekDay = 1;
44               } // if
45            else
46                weekDay++;
47            printf("%3d ", dayCount);
48           } // for
49       printf("\n--- --- --- --- --- --- ---\n");
50       return;
51   }  // printMonth
```

Results:

```
Sun Mon Tue Wed Thu Fri Sat
--- --- --- --- --- --- ---
              1   2   3   4   5
  6   7   8   9  10  11  12
 13  14  15  16  17  18  19
 20  21  22  23  24  25  26
 27  28  29
--- --- --- --- --- --- ---
```

Note

Never use one variable to control two processes.

PROGRAM 6-13 Print Sum of Digits

```
 1  /* Print the number and sum of digits in an integer.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Local Declarations
10     int number;
11     int count = 0;
12     int sum   = 0;
13
14  // Statements
15     printf("Enter an integer: ");
16     scanf ("%d", &number);
17     printf("Your number is:   %d\n\n", number);
18
```

PROGRAM 6-13 Print Sum of Digits

```
19      while (number != 0)
20          {
21           count++;
22           sum +=  number % 10;
23           number /= 10;
24          } // while
25      printf("The number of digits is:  %3d\n", count);
26      printf("The sum of the digits is: %3d\n", sum);
27      return 0;
28  }   // main
```

```
Results:
Enter an integer: 12345
Your number is:   12345

The number of digits is:    5
The sum of the digits is:  15
```

PROGRAM 6-14 Print Number Backward

```
 1  /* Use a loop to print a number backward.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Local Declarations
10     long num;
11     int  digit;
12
13  // Statements
14     printf("Enter a number and I'll print it backward: ");
15     scanf ("%d", &num);
16
17     while (num > 0)
18        {
```

PROGRAM 6-14 Print Number Backward

```
19          digit = num % 10;
20          printf("%d", digit);
21          num   = num / 10;
22        } // while
23     printf("\nHave a good day.\n");
24     return 0;
25  }  // main
```

```
Results:
Enter a number and I'll print it backward: 12345678
87654321
Have a good day.
```

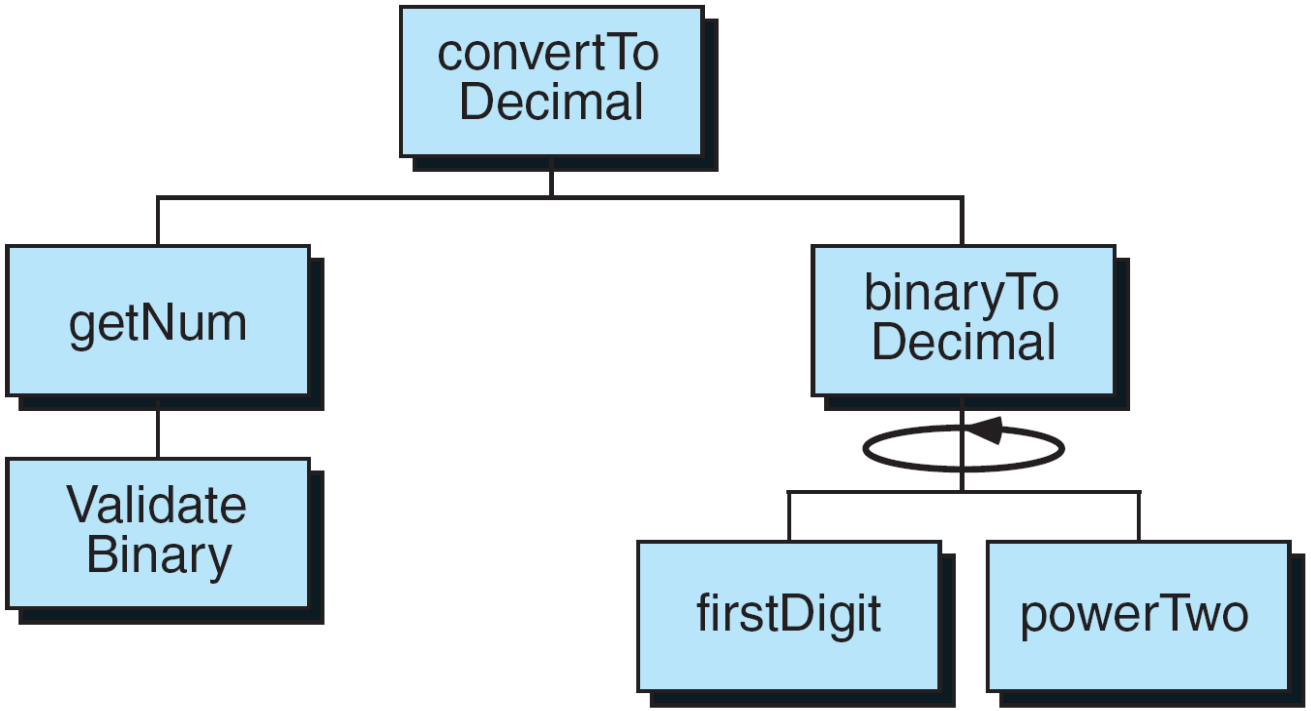


FIGURE 6-19 Design for Binary to Decimal

PROGRAM 6-15 Convert Binary to Decimal

```
 1  /* Convert a binary number to a decimal number.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  #include <stdbool.h>
 7
 8  // Function Declarations
 9  long long getNum           (void);
10  long long binaryToDecimal (long long binary);
11  long long powerTwo         (long long num);
12  long long firstDigit       (long long num);
13  bool      validateBinary   (long long binary);
14
15  int main (void)
16  {
```

PROGRAM 6-15 Convert Binary to Decimal

```
17  // Local Declarations
18     long long binary;
19     long long decimal;
20
21  // Statements
22     binary   = getNum ();
23     decimal  = binaryToDecimal (binary);
24     printf("The binary number was: %lld", binary);
25     printf("\nThe decimal number is: %lld", decimal);
26     return 0;
27  }  // main
28
29  /* ====================== getNum ====================
30     This function reads and validates a binary number
31     from the keyboard.
32        Pre   nothing
33        Post  a valid binary number is returned
34  */
```

PROGRAM 6-15 Convert Binary to Decimal

```
35  long long getNum (void)
36  {
37  // Local Declarations
38     bool      isValid;
39     long long binary;
40
41  // Statements
42     do
43      {
44       printf("Enter a binary number (zeros and ones): ");
45       scanf ("%lld", &binary);
46       isValid = validateBinary (binary);
47       if (!isValid)
48          printf("\a\aNot binary. Zeros/ones only.\n\n");
49      } while (!isValid);
50     return binary;
51  } // getNum
52
```

PROGRAM 6-15 Convert Binary to Decimal

```
53 /* ================ binaryToDecimal ==================
54    Change a binary number to a decimal number.
55      Pre  binary is a number containing only 0 or 1
56      Post returns decimal number
57 */
58 long long binaryToDecimal (long long binary)
59 {
60 // Local Declarations
61    long long  decimal;
62
63 // Statements
64    decimal = 0;
65    for (int i = 0; binary != 0; i++)
66        {
67         decimal += firstDigit (binary) * powerTwo (i);
68         binary /= 10;
69        } // for i
70    return decimal;
71 } // binaryToDecimal
```

PROGRAM 6-15 Convert Binary to Decimal

```
72
73  /* ================ validateBinary ==================
74     Check the digits in a binary number for only 0 and 1.
75        Pre   binary is a number to be validated
76        Post  returns true if valid; false if not
77  */
78  bool validateBinary (long long binary)
79  {
80  // Statements
81     while (binary != 0)
82        {
83         if (!(binary % 10 ==  0 || binary % 10 ==  1))
84             return false;
85         binary /= 10;
86        } // while
87     return true;
88  }  // validateBinary
89
```

PROGRAM 6-15 Convert Binary to Decimal

```
 90  /* ==================== powerTwo ====================
 91     This function raises 2 to the power num
 92        Pre   num is exponent
 93        Post  Returns 2 to the power of num
 94  */
 95  long long powerTwo (long long num)
 96  {
 97  // Local Declarations
 98     long long  power = 1;
 99
100  // Statements
101     for (int i = 1; i <= num; i++)
102              power *= 2;
103     return power;
104  }   // powerTwo
105
```

PROGRAM 6-15 Convert Binary to Decimal

```
106 /* ================== firstDigit ==================
107    This function returns the right most digit of num
108       Pre   the integer num
109       Post  the right digit of num returned
110 */
111 long long firstDigit (long long num)
112 {
113 // Statements
114    return (num % 10);
115 }  // firstDigit
```

```
Results:
Enter a binary number (zeros and ones): 10001
The binary number was: 10001
The decimal number is: 17
```

6-7 Other Statements Related to Looping

Three other C statements are related to loops: break, continue, and goto. The last statements, the goto, is not valid for structured programs and therefore is not discussed in this text.

Topics discussed in this section:

break

continue

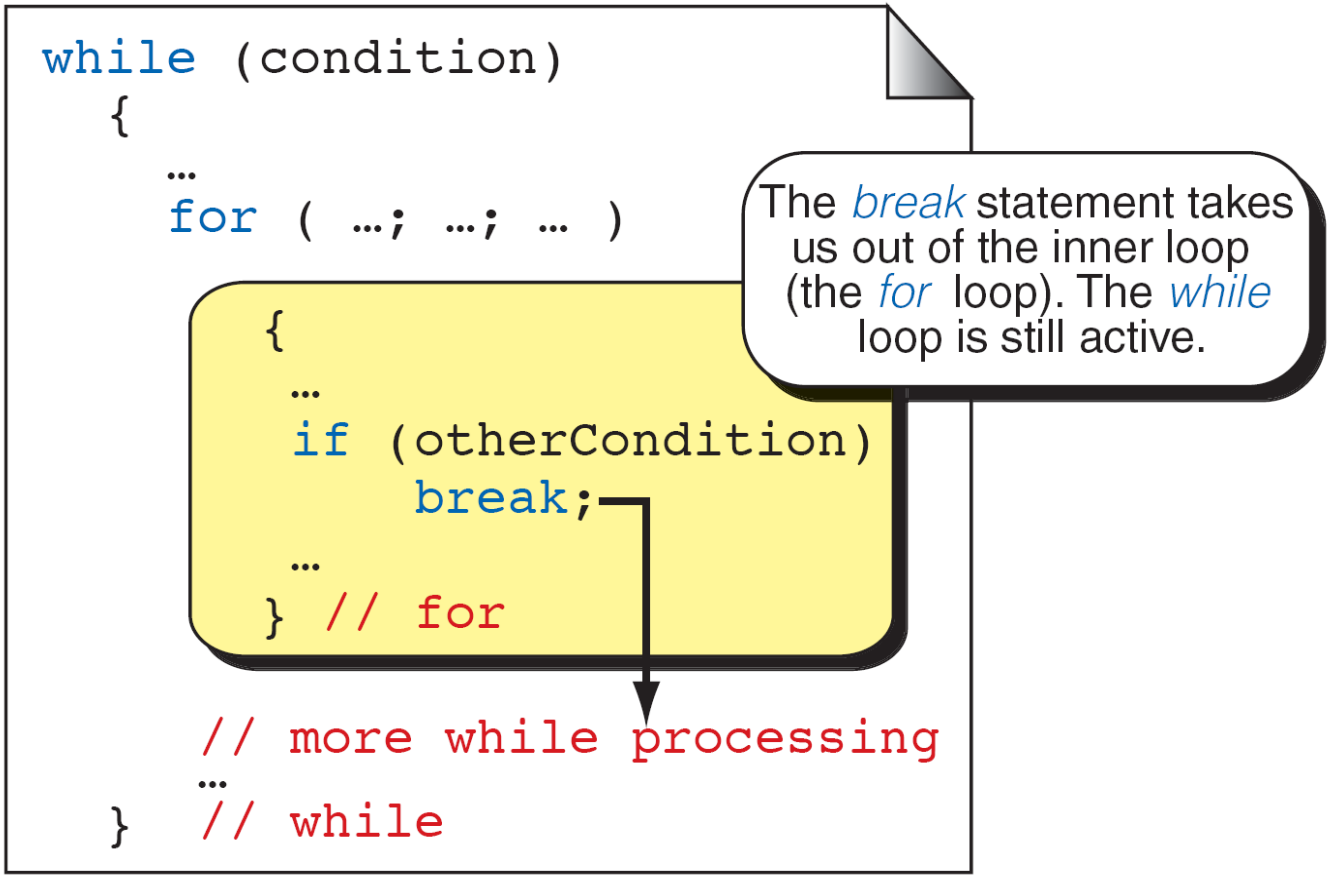


FIGURE 6-20 ***break*** **and Inner Loops**

PROGRAM 6-16 The *for* and *while* as Perpetual Loops

```
1  // A bad loop style
2  for ( ; ; )
3     {
4      …
5      if (condition)
6         break;
7     } // for
```

```
1  // A better loop style
2  for ( ; !condition ; )
3     {
4       …
5     } // for
```

```
1  while (x)
2     {
3      …
4      if (condition)
5         break;
6      else
7         …
8     } // while
```

```
1  while  (x && !condition)
2     {
3      …
4      if (!condition)
5           …;
6     } // while
```

PROGRAM 6-17 Using a *break* Flag

```
1  breakFlag = 0;
2  while (!breakFlag)
3      {
4       …
5       if (x && !y || z)              // Complex limit test
6          breakFlag = 1;
7       else
8          …;
9      } // while
```

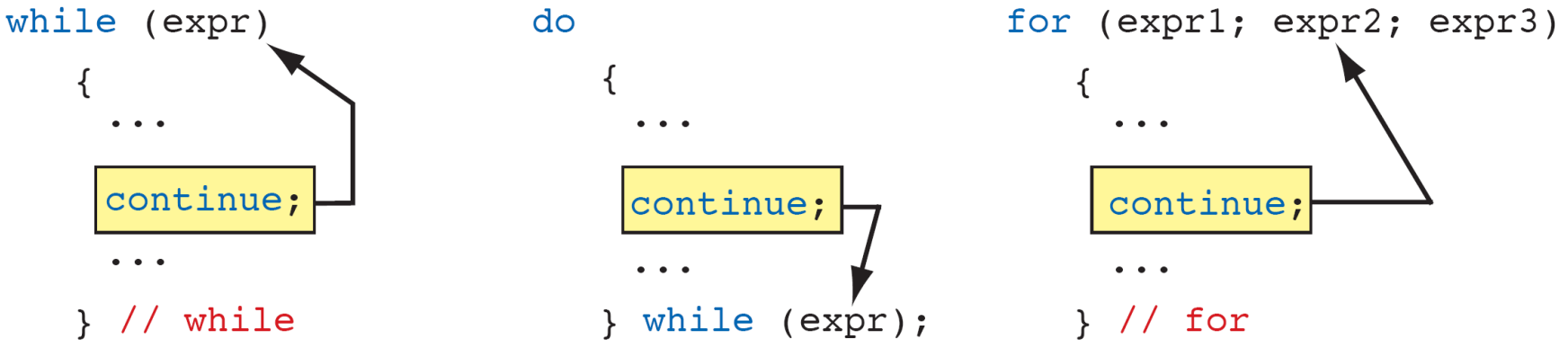


FIGURE 6-21 The *continue* Statement

PROGRAM 6-18 *continue* Example

```
 1  float readAverage (void)
 2  {
 3  // Local Declarations
 4  int   count = 0;
 5
 6  int   n;
 7  float sum = 0;
 8
 9  // Statements
10  while(scanf("%d",&n)
11        != EOF)
12     {
13      if (n == 0)
14         continue;
15      sum += n;
16      count++;
17     }  // while
18
19  return (sum / count);
20  } // readAverage
```

```
float readAverage (void)
{
// Local Declarations
int   count = 0;

int   n;
float sum = 0;

// Statements
while(scanf("%d",&n)
      != EOF)
   {
    if (n != 0)
       {
        sum += n;
        count++;
       } // if
   } // while
return (sum / count);
} // readAverage
```

6-8 Looping Applications

In this section, we examine four common applications for loops: summation, product, smallest and largest, and inquiries. Although the uses for loops are virtually endless, these problems illustrate many common applications.

Topics discussed in this section:

Summation
Powers
Smallest and Largest
Inquiries

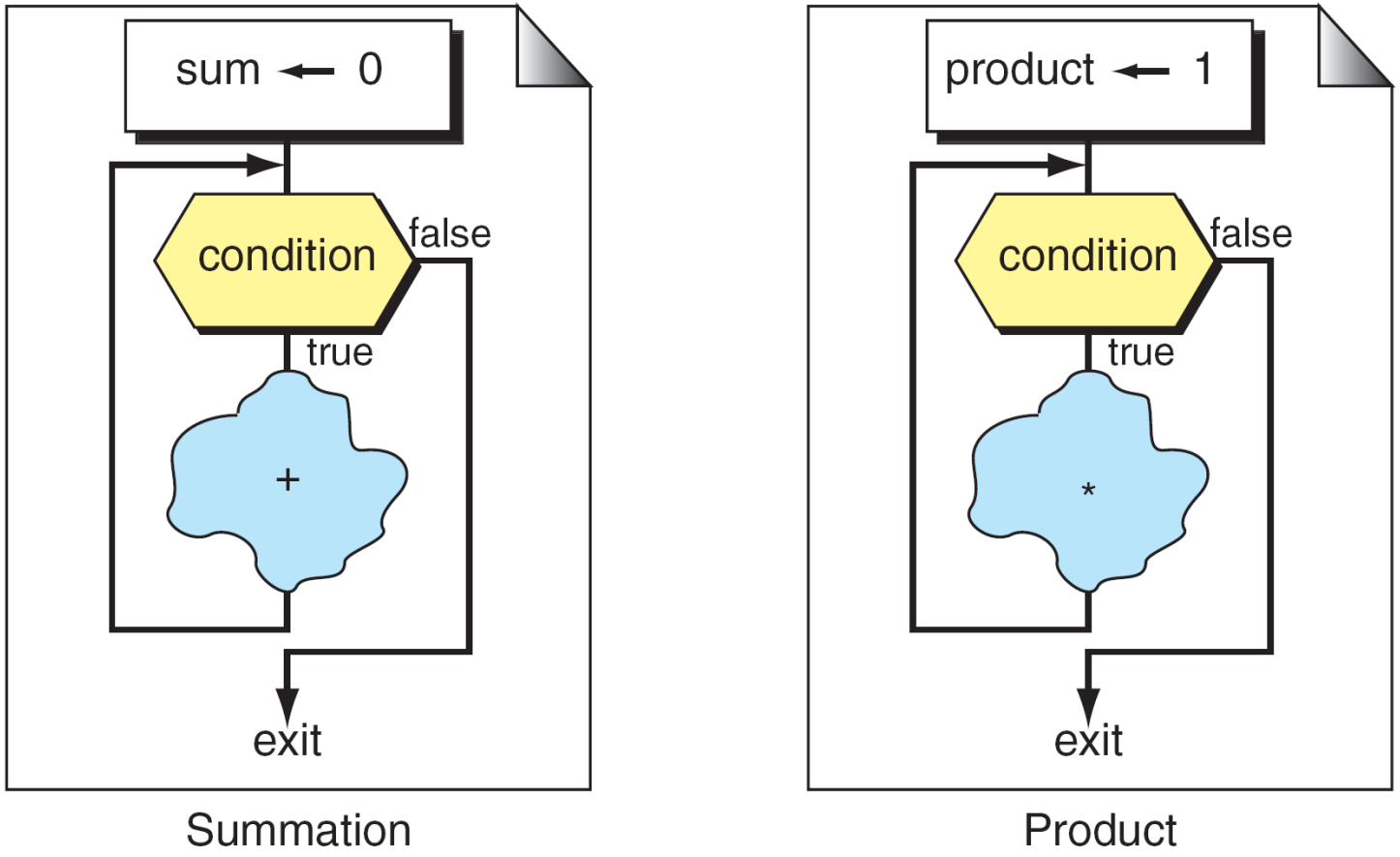


FIGURE 6-22 Summation and Product Loops

PROGRAM 6-19 Sum to EOF Function

```
 1  /* Read a series of numbers, terminated by EOF, and
 2     return their sum to the calling program.
 3        Pre   nothing
 4        Post  data read and sum returned
 5  */
 6  int sumEOF (void)
 7  {
 8  // Local Declarations
 9     int nmbr;
10     int sum;
11
12  // Statements
13     sum = 0;
14     printf ("Please enter an integer:    ");
15
16     while (scanf("%d", &nmbr) != EOF)
```

PROGRAM 6-19 Sum to EOF Function

```
17          {
18           sum += nmbr;
19           printf("Next integer <EOF> to stop: ");
20          } // while
21      return sum;
22  }   // sumEOF
```

PROGRAM 6-20 Powers Function

```
1  /* Raise base to an integral power, exp. If the
2     exponent is zero, return 1.
3        Pre   base & exp are both positive integer values
4        Post  return either (a) the result of raising the
5                 base to the exp power
6              or (b) zero if the parameters are invalid
7  */
8  int powers (int base, int exp)
9  {
10 // Local Declarations
11    int result = 1;
12
13 // Statements
14    if (base < 1 || exp < 0)
15       // Error Condition
16       result = 0;
```

PROGRAM 6-20 Powers Function

```
17      else
18          for  (int i = 1; i <= exp; i++)
19                result *= base;
20      return result;
21  }   // powers
```

Note

To find the sum of a series, the result is initialized to 0; to find the product of a series, the result is initialized to 1.

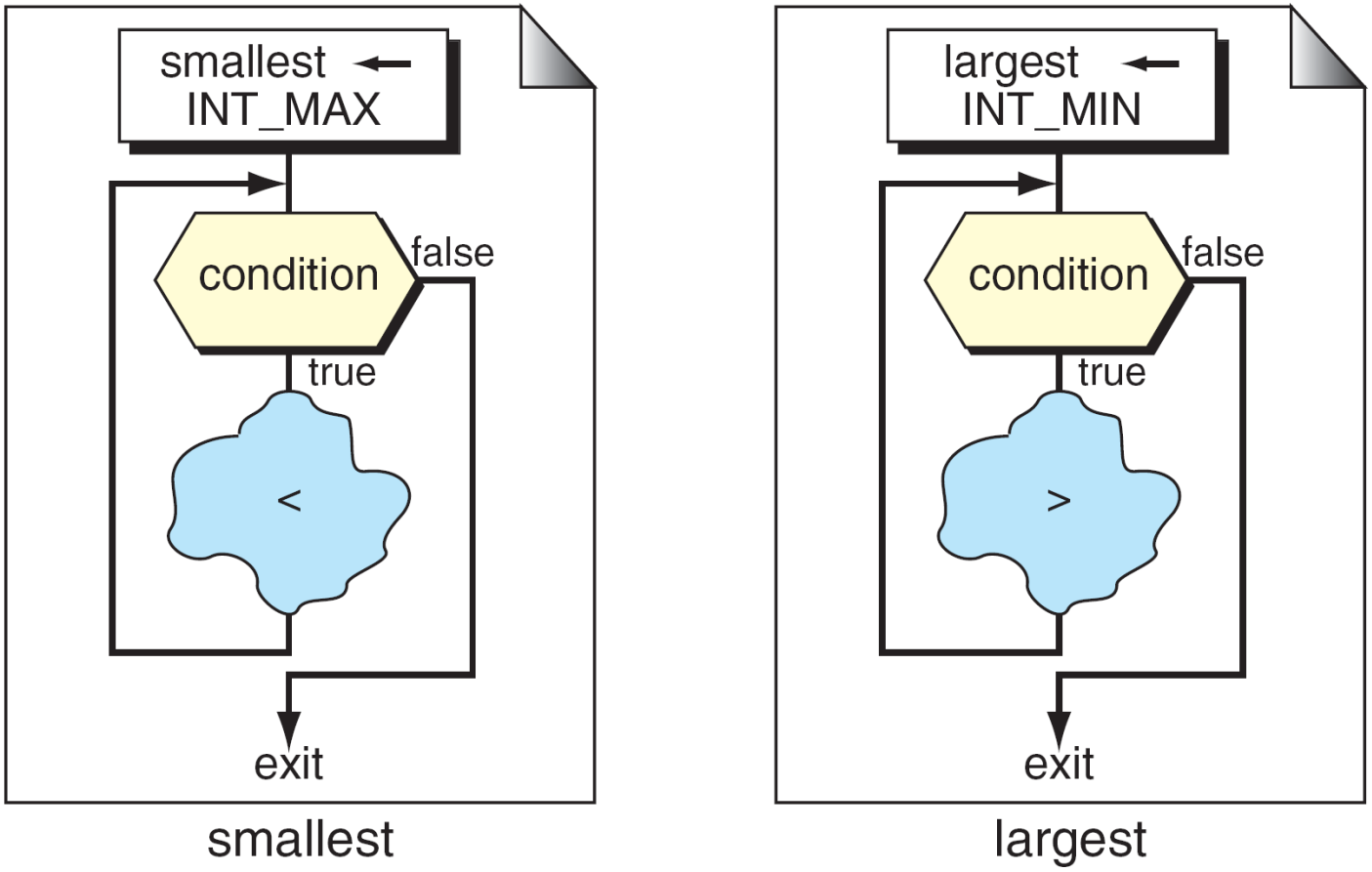


FIGURE 6-23 Smallest and Largest Loops

PROGRAM 6-21 Smallest to EOF Function

```
 1  /* Read a series of numbers, terminated by EOF, and
 2     pass the smallest to the calling program.
 3        Pre   nothing
 4        Post  data read and smallest returned
 5  */
 6  int smallestEOF (void)
 7  {
 8  // Local Declarations
 9     int numIn;
10     int smallest;
11
12  // Statements
13     smallest = INT_MAX;          // requires <limits.h>
14
15     printf("Please enter an integer: ");
16
17     while (scanf("%d", &numIn) != EOF)
```

PROGRAM 6-21 Smallest to EOF Function

```
18          {
19           if (numIn < smallest)
20                smallest = numIn;
21           printf("Enter next integer <EOF> to stop: ");
22          } // while
23       return smallest;
24   }    // smallestEOF
```

Note

To find the largest, we need to initialize the smallest variable to a very small number, such as INT_MIN.

To find the smallest, we need to initialize the result to a very large number, such as INT_MAX.

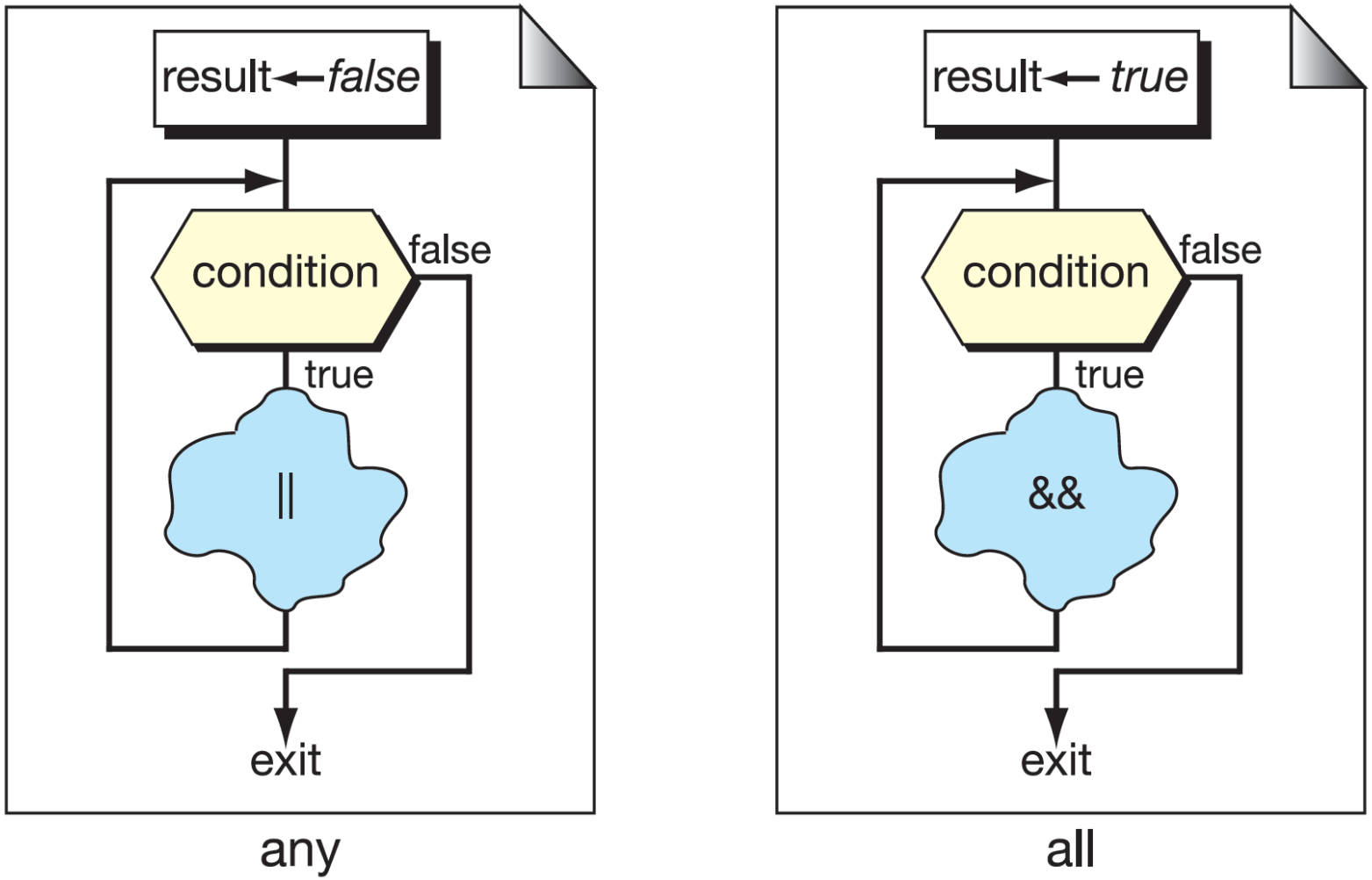


FIGURE 6-24 ***any*** **and** ***all*** **Inquiries**

PROGRAM 6-22 *any*Positive to EOF Function

```
 1  /* Read number series to determine if any positive.
 2        Pre  nothing
 3        Post return true if any number > zero
 4             return false if all numbers <= zero
 5  */
 6  bool anyPositiveEOF (void)
 7  {
 8  // Local Declarations
 9     bool anyPositive = false;
10     int  numIn;
11
12  // Statements
13     printf("Determine if any number are positive\n");
14     printf("Enter first number: ");
15     while (scanf("%d", &numIn) != EOF)
16        {
17         anyPositive = numIn > 0;
```

PROGRAM 6-22 *any*Positive to EOF Function

```
18          if (anyPositive)
19              return true;
20          printf("Enter next number:  ");
21        } // while
22     return false;
23  } // anyPositiveEOF
```

PROGRAM 6-23 *All* Positive Function

```
1  /* Read number series, and determine if all are positive.
2        Pre   nothing
3        Post  return true if all numbers > zero
4              return false if any numbers <= zero
5  */
6  bool allPositiveEOF (void)
7  {
8  // Local Declarations
9     bool allPositive = true;
10    int  numIn;
11
12 // Statements
13    printf("Determine if all numbers are positive\n");
14    printf("Enter first number: ");
15    while (allPositive && (scanf("%d", &numIn) != EOF))
16       {
```

PROGRAM 6-22 *any*Positive to EOF Function

```
17          allPositive =  numIn > 0;
18          if (!allPositive)
19              return false;
20          printf("Enter next number:  ");
21        } // while
22     return true;
23  } // allPositiveEOF
```

6-9 Recursion

In general, programmers use two approaches to writing repetitive algorithms. One approach uses loops; the other uses recursion. Recursion is a repetitive process in which a function calls itself.

Topics discussed in this section:

Iterative and Recursive Definition
Iterative and Recursive Solution
Designing Recursive Functions
Fibonacci Numbers
Limitations of Recursion
The Towers Of Hanoi

$$\text{factorial}\,(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1) * (n-2) \ldots 3 * 2 * 1 & \text{if } n > 0 \end{cases}$$

FORMULA 6-1 Iterative Factorial Definition

$$\text{factorial}\,(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * \text{factorial}\,(n-1) & \text{if } n > 0 \end{cases}$$

FORMULA 6-2 Recursive Factorial Definition

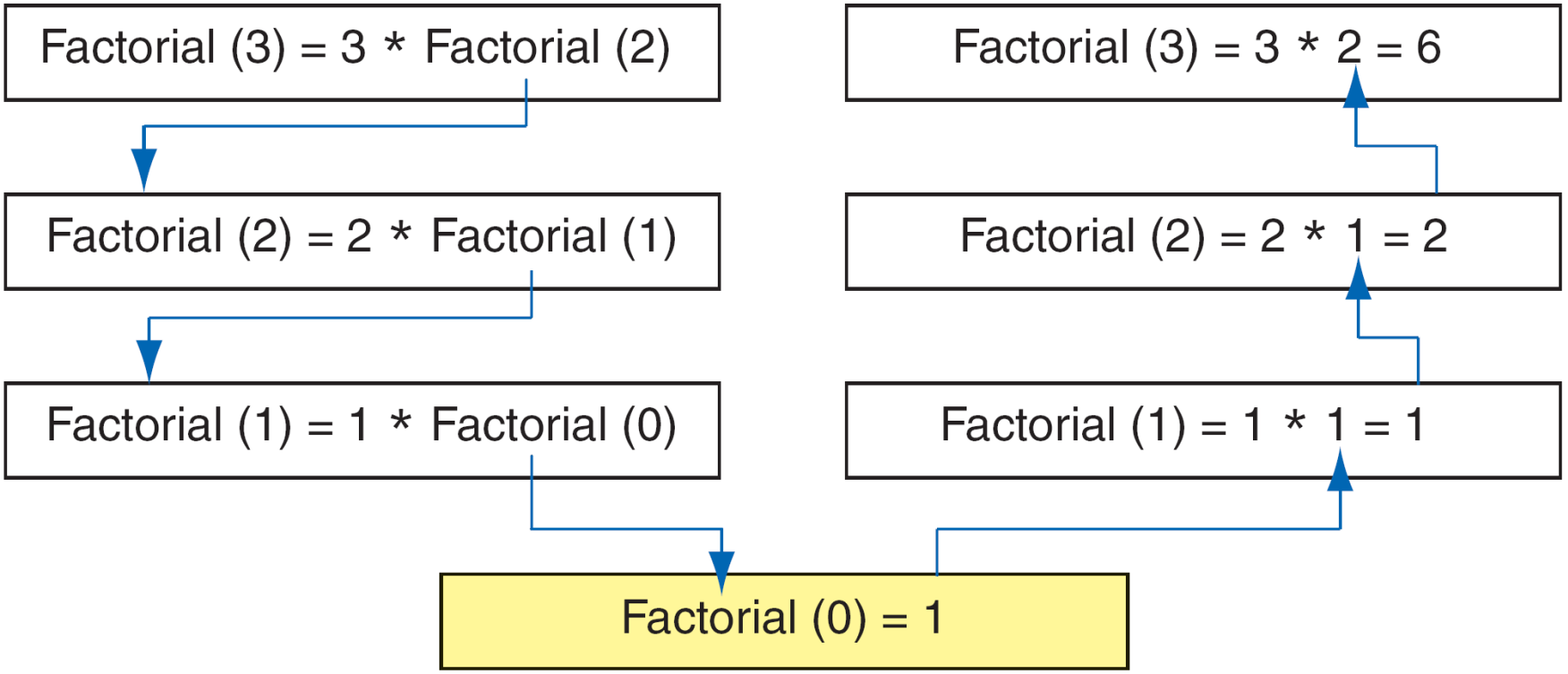


FIGURE 6-25 Factorial (3) Recursively

PROGRAM 6-24 Iterative Factorial Function

```
1  /* Calculate the factorial of a number using a loop.
2     There is no test that the result fits in a long.
3        Pre   n is the number to be raised factorially
4        Post  result is returned
5  */
6  long factorial (int n)
7  {
8  // Local Declarations
9     long factN = 1;
10
11 // Statements
12    for (int i = 1; i <= n; i++)
13        factN = factN * i;
14    return factN;
15 } // factorial
```

PROGRAM 6-25 Recursive Factorial Function

```
 1  /* Calculate factorial of a number using recursion.
 2     There is no test that the result fits in a long.
 3        Pre   n is the number being raised factorially
 4        Post  result is returned
 5  */
 6  long factorial (int n)
 7  {
 8  // Statements
 9     if (n == 0)
10        return 1;
11     else
12        return (n * factorial (n - 1));
13  } // factorial
```

Designing Recursive Functions

- Requirements of a recursive function
  - Call to itself with a reduced data size
  - Termination condition
- Example
  - Factorial

Note

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

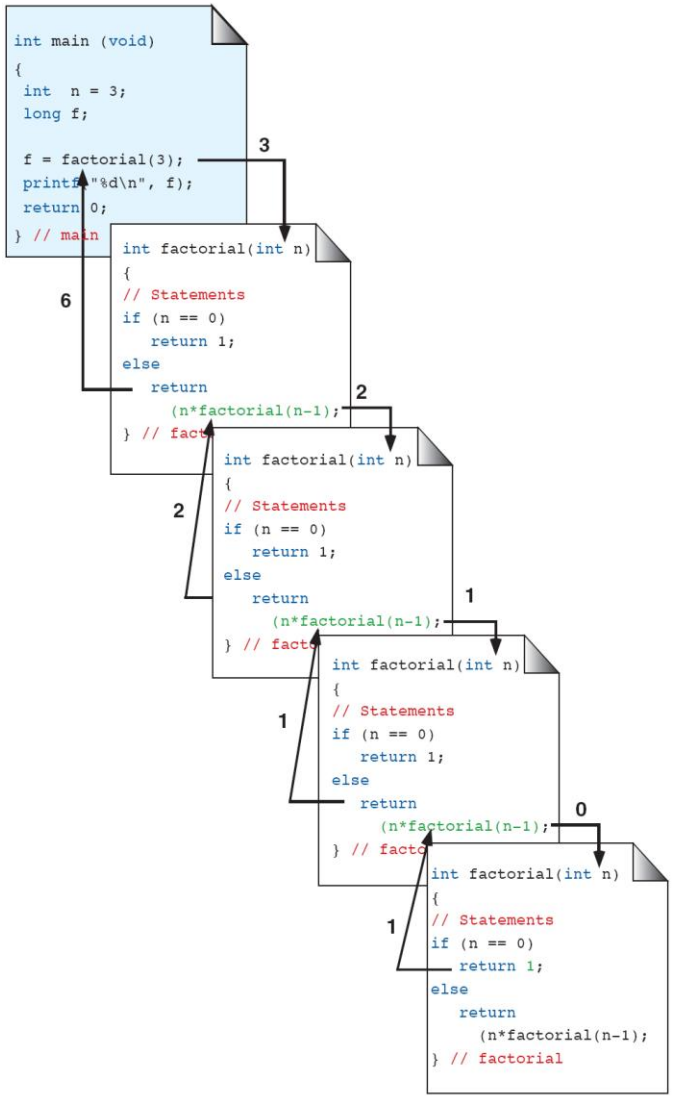


FIGURE 6-26 Calling a Recursive Function

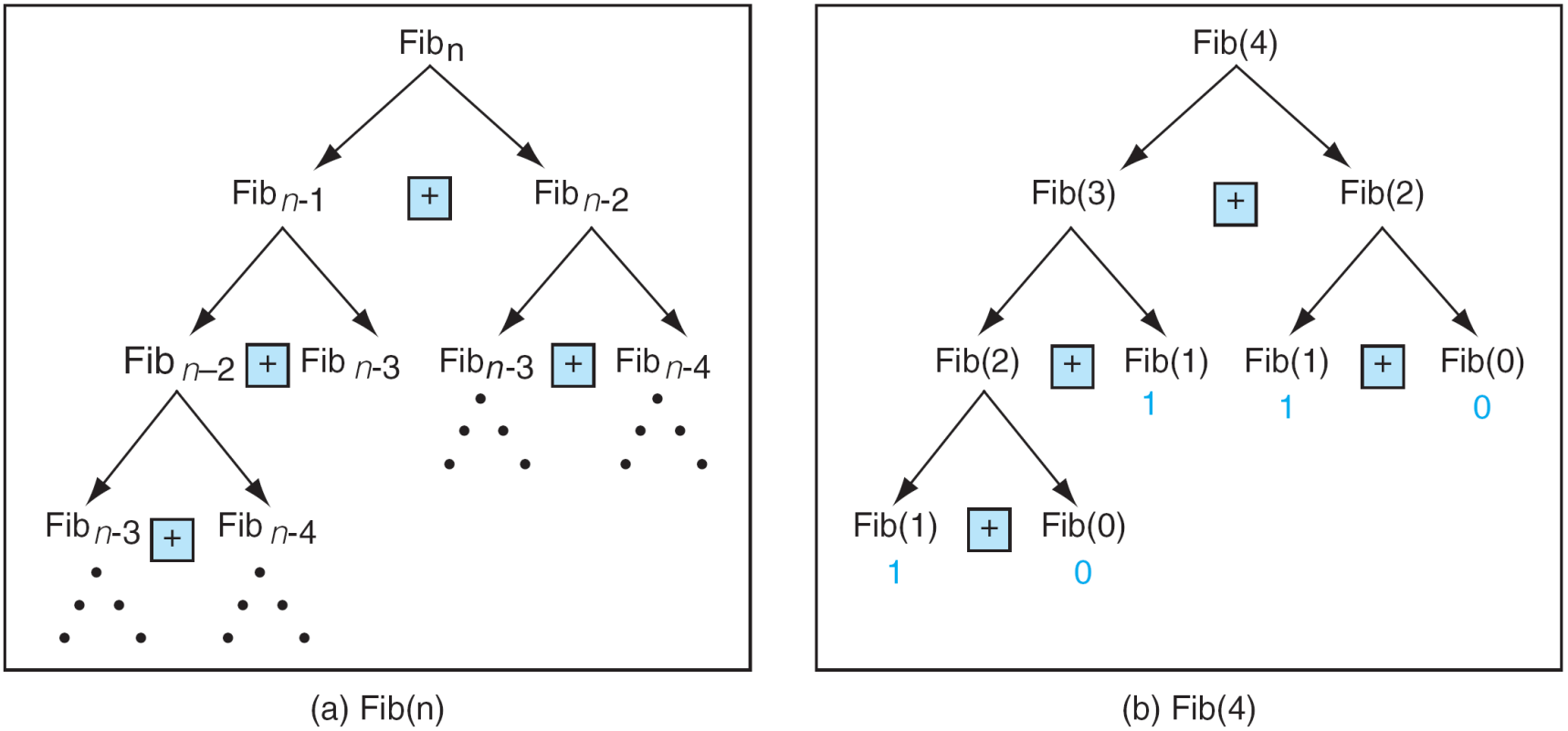


FIGURE 6-27 Fibonacci Numbers

PROGRAM 6-26 Recursive Fibonacci

```
 1  /* This program prints out a Fibonacci series.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  // Function Declaration
 8     long fib (long num);
 9
10  int main (void)
11  {
12  // Local Declarations
13     int seriesSize;
14
15  // Statements
16     printf("This program prints a Fibonacci series.\n");
17     printf("How many numbers do you want? ");
18     scanf ("%d", &seriesSize);
19     if (seriesSize < 2)
20        seriesSize = 2;
```

PROGRAM 6-26 Recursive Fibonacci

```
21
22      printf("First %d Fibonacci numbers: \n", seriesSize);
23      for (int looper = 0; looper < seriesSize; looper++)
24         {
25          if (looper % 5)
26              printf(", %8ld", fib(looper));
27          else
28              printf("\n%8ld", fib(looper));
29         }
30      printf("\n");
31      return 0;
32  }   // main
33
34  /* ================ fib ===================
35      Calculates the nth Fibonacci number.
36         Pre   num identifies Fibonacci number
37         Post  returns nth Fibonacci number
38  */
```

PROGRAM 6-26 Recursive Fibonacci

```
39  long fib (long num)
40  {
41  // Statements
42     if (num == 0 || num == 1)
43        // Base Case
44        return num;
45
46     return (fib (num - 1) + fib (num - 2));
47  }  // fib
```

```
Results:
This program prints a Fibonacci series.
How many numbers do you want? 30
First 30 Fibonacci numbers:
       0,        1,        1,        2,        3
       5,        8,       13,       21,       34
      55,       89,      144,      233,      377
     610,      987,     1597,     2584,     4181
    6765,    10946,    17711,    28657,    46368
   75025,   121393,   196418,   317811,   514229
```

No.	Calls	No.	Calls
1	1	11	287
2	3	12	465
3	5	13	753
4	9	14	1,219
5	15	15	1,973
6	25	20	21,891
7	41	25	242,785
8	67	30	2,692,573
9	109	35	29,860,703
10	177	40	331,160,281

Table 6-2 Fibonacci Run Time

Limitations of Recursion

- Recursive solutions may involve extensive overhead because of function calls
- Each recursive call uses up some of memory allocation
- Possibly duplicate computation, but not always

Recursive vs. Iterative Solutions

- Many algorithms are easier to implement recursively, while providing efficiency
- Example
  - Towers of Hanoi

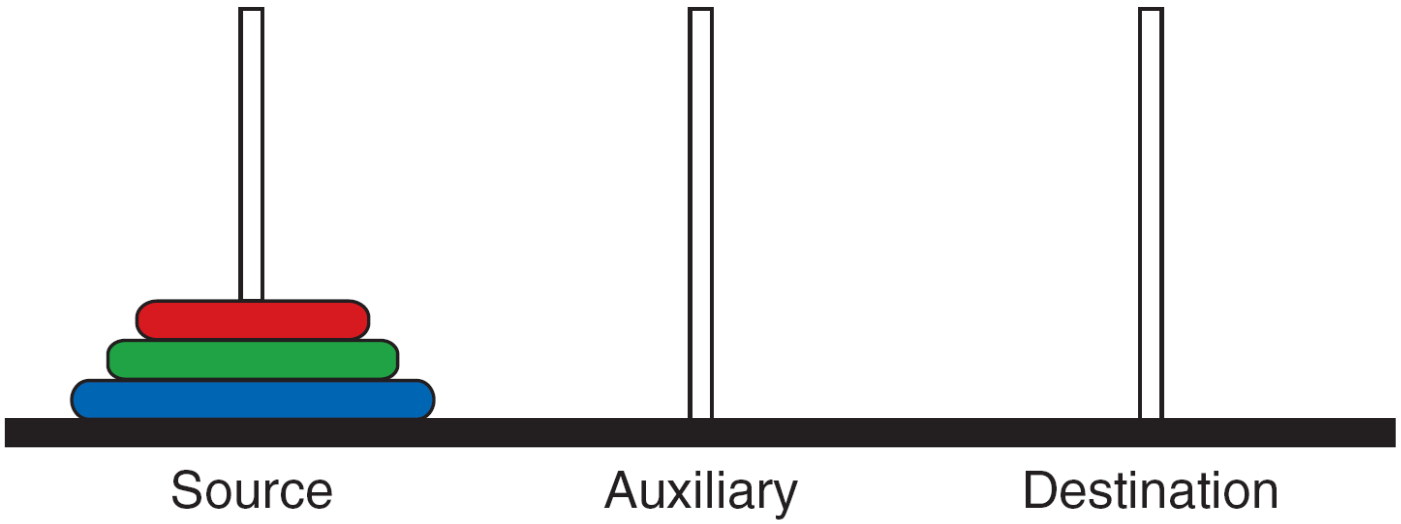


FIGURE 6-28 Towers of Hanoi—Start Position

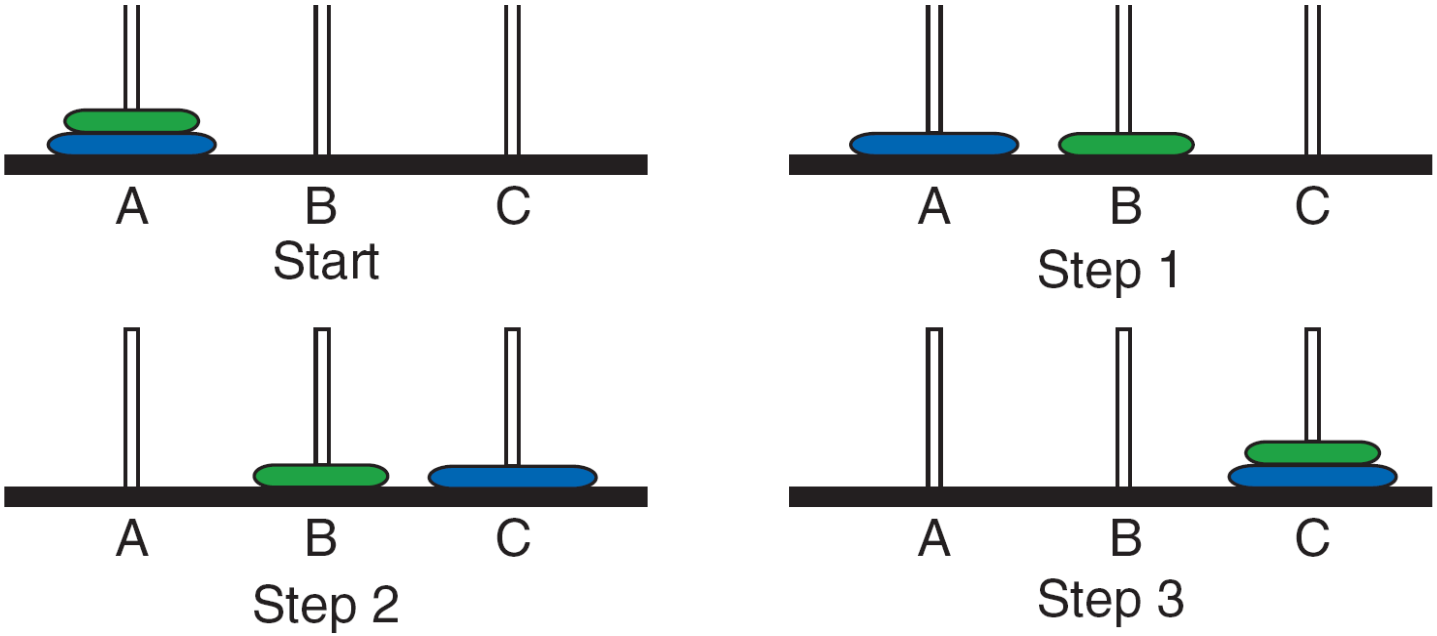


FIGURE 6-29 Towers Solution for Two Disks

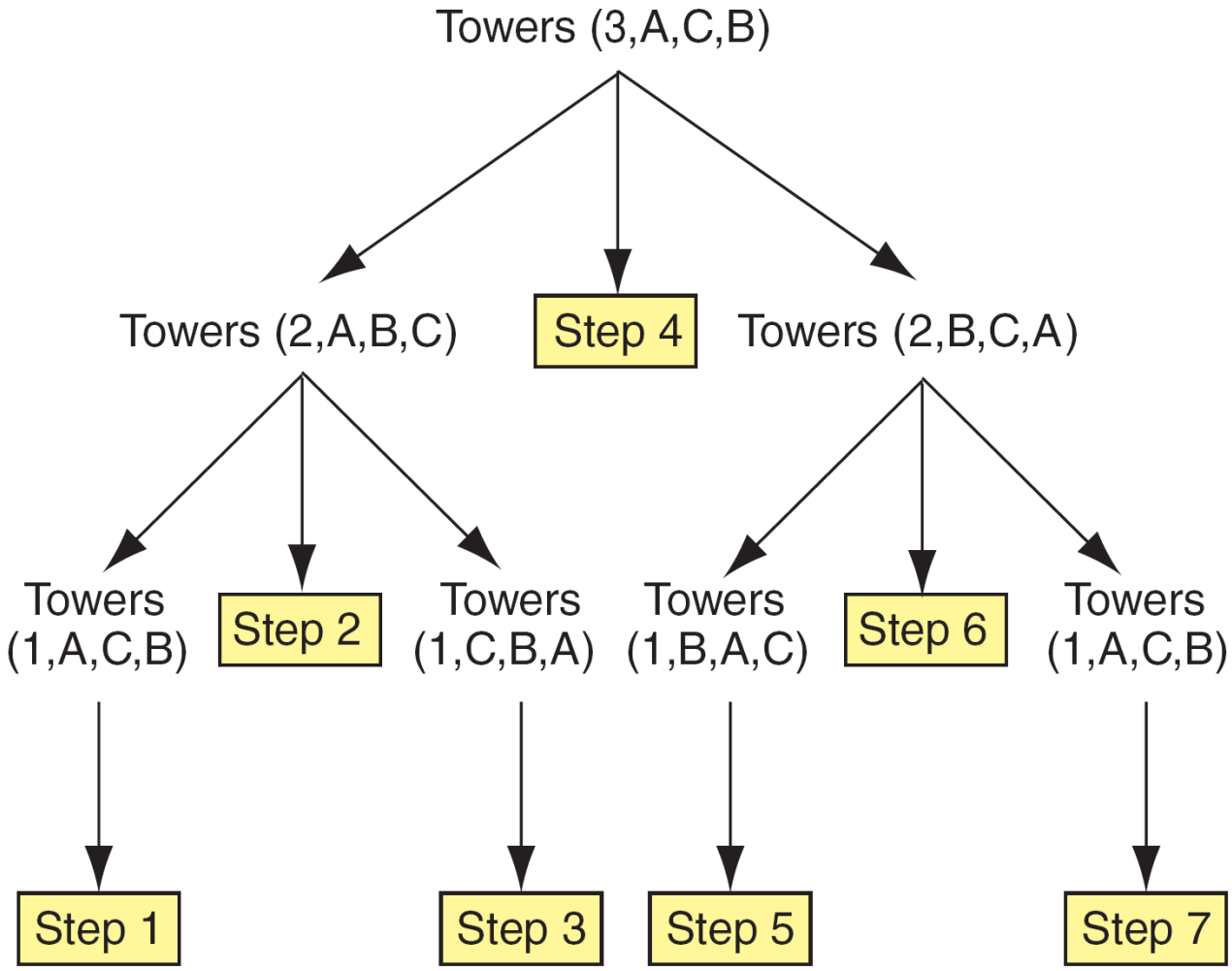


FIGURE 6-30 Towers of Hanoi Solution for Three Disks (Part I)

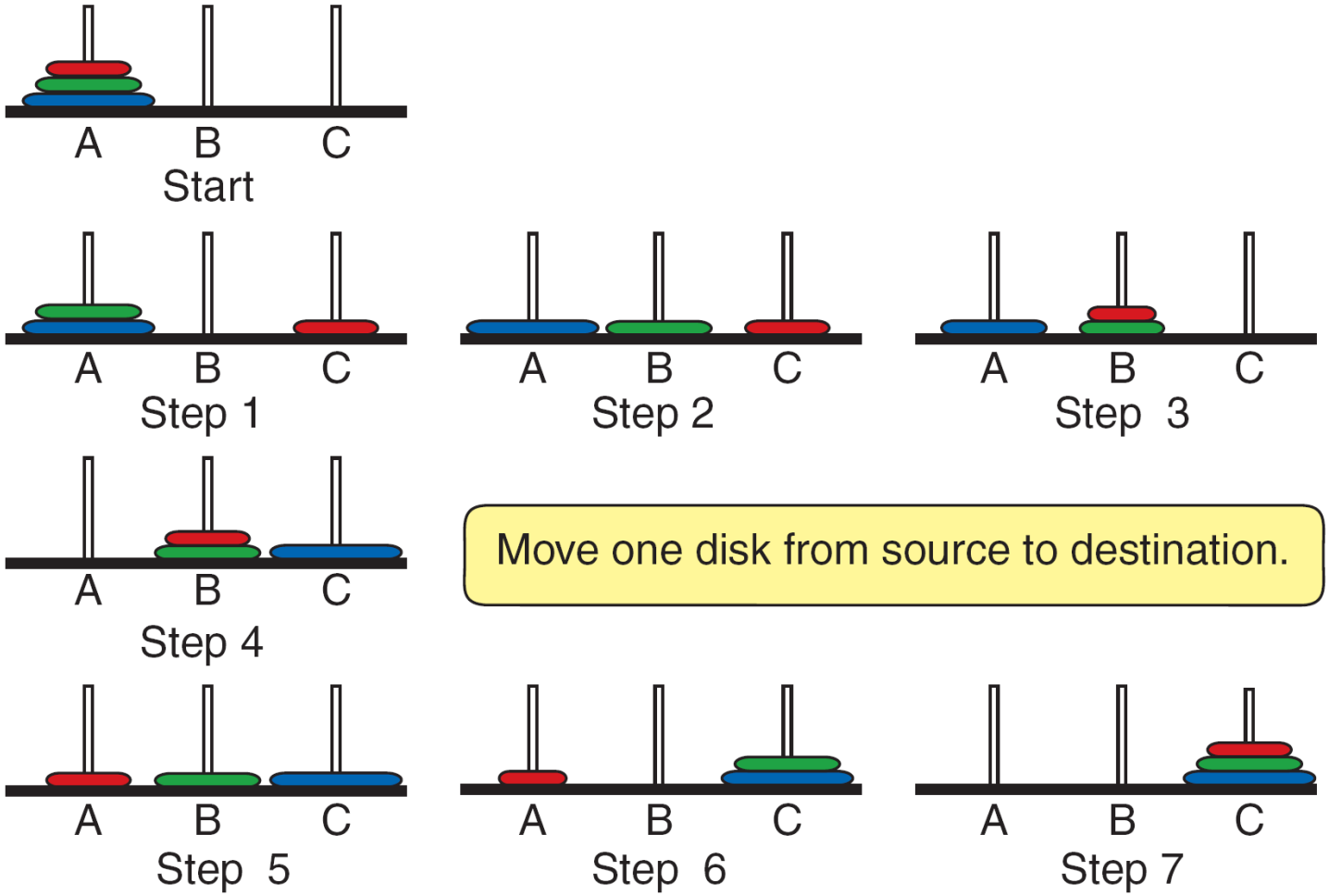


FIGURE 6-30 Towers of Hanoi Solution for Three Disks (Part II)

PROGRAM 6-27 Towers of Hanoi

```
 1  /* Move one disk from source to destination through
 2     the use of recursion.
 3        Pre   The tower consists of n disks--source,
 4              destination, & auxiliary towers given
 5        Post  Steps for moves printed
 6  */
 7  void towers (int  n,    char source,
 8               char dest, char auxiliary)
 9  {
10  // Local Declarations
11     static int step = 0;
12
13  // Statements
14     printf("Towers (%d, %c, %c, %c)\n",
15                       n, source, dest, auxiliary);
16     if (n == 1)
17        printf("\t\t\t\tStep %3d: Move from %c to %c\n",
18                ++step, source, dest);
19     else
```

PROGRAM 6-27 Towers of Hanoi

```
20          {
21           towers (n - 1, source, auxiliary, dest);
22           printf("\t\t\t\tStep %3d: Move from %c to %c\n",
23                  ++step, source, dest);
24           towers (n - 1, auxiliary, dest, source);
25          } // if … else
26      return;
27  }   // towers
```

Calls	Output
`Towers (3, A, C, B)`	
`Towers (2, A, B, C)`	
`Towers (1, A, C, B)`	
	`Step 1: Move from A to C`
	`Step 2: Move from A to B`
`Towers (1, C, B, A)`	
	`Step 3: Move from C to B`
	`Step 4: Move from A to C`
`Towers (2, B, C, A)`	
`Towers (1, B, A, C)`	
	`Step 5: Move from B to A`
	`Step 6: Move from B to C`
`Towers (1, A, C, B)`	
	`Step 7: Move from A to C`

Table 6-3 Tracing of Program 6-27, Towers of Hanoi

6-10 Programming Example—The Calculator Program

Let's look at our calculator program one more time. In Chapter 5, we gave users the capability of selecting one of four options: add, subtract, multiply, or divide. However, if users needed to make two calculations, they had to run the program twice. We now add a loop that allows users to make as many calculations as needed.

PROGRAM 6-28 The Complete Calculator

```
 1  /* This program adds, subtracts, multiplies, and divides
 2     two integers.
 3        Written by:
 4        Date:
 5  */
 6  #include <stdio.h>
 7  #include <stdlib.h>
 8
 9  // Function Declaration
10  int   getOption (void);
11  void  getData   (int* a,       int* b);
12  float calc      (int option,  int  num1, int num2);
13
14  float add  (float num1, float num2);
15  float sub  (float num1, float num2);
16  float mul  (float num1, float num2);
17  float divn (float num1, float num2);
18
19  void  printResult (float num1,   float num2,
20                     float result, int   option);
21
```

PROGRAM 6-28 The Complete Calculator

```
22  int main (void)
23  {
24  // Local Declarations
25     int    done = 0;
26     int    option;
27     float  num1;
28     float  num2;
29     float  result;
30
31  // Statements
32     while (!done)
33        {
34         option = getOption();
35         if (option ==  5)
36            done = 1;
37         else
38           {
39            do
40              {
41               printf("\n\nEnter two numbers: ");
```

PROGRAM 6-28 The Complete Calculator

```
42                scanf("%f %f", &num1, &num2);
43                if (option ==  4 && num2 == 0)
44                   {
45                    printf("\a\n *** Error *** ");
46                    printf("Second number cannot be 0\n");
47                   } // if
48              } while (option ==  4 && num2 ==  0);
49
50            switch (option)
51              {
52               case 1: result = add (num1, num2);
53                       break;
54               case 2: result = sub (num1, num2);
55                       break;
56               case 3: result = mul (num1, num2);
57                       break;
58               case 4: result = divn (num1, num2);
59              } // switch
60
61            printResult (num1, num2, result, option);
62           } // else option != 5
63       } // while
```

PROGRAM 6-28 The Complete Calculator

```
64      printf("\nThank you for using Calculator.\n");
65      return 0;
66   }  // main
67
68   /* ==================== getOption ====================
69      This function shows a menu and reads the user option.
70         Pre   nothing
71         Post  returns a valid option
72   */
73   int getOption (void)
74   {
75   // Local Declarations
76      int option;
77
78   // Statements
79      do
80         {
81          printf("\n**********************");
82          printf("\n*        MENU        *");
83          printf("\n*                    *");
84          printf("\n*  1. ADD            *");
85          printf("\n*  2. SUBTRACT       *");
```

PROGRAM 6-28 The Complete Calculator

```
 86          printf("\n*  3. MULTIPLY    *");
 87          printf("\n*  4. DIVIDE      *");
 88          printf("\n*  5. QUIT        *");
 89          printf("\n*                 *");
 90          printf("\n*******************");
 91
 92          printf("\n\n\nPlease type your choice ");
 93          printf("and press the return key : ");
 94          scanf("%d", &option);
 95
 96          if (option < 1 || option > 5)
 97             printf("Invalid option. Please re-enter.\n");
 98
 99       } while (option < 1 || option > 5);
100       return option;
101    }  // getOption
```

6-11 Software Engineering

In this section, we discuss some software engineering issues related to loops.

Topics discussed in this section:

Loops in Structure Charts
Determining Algorithm Efficiency
Linear Loops
Logarithmic Loops
Nested Loops
Big-O Notation
Standard Measures of Efficiency

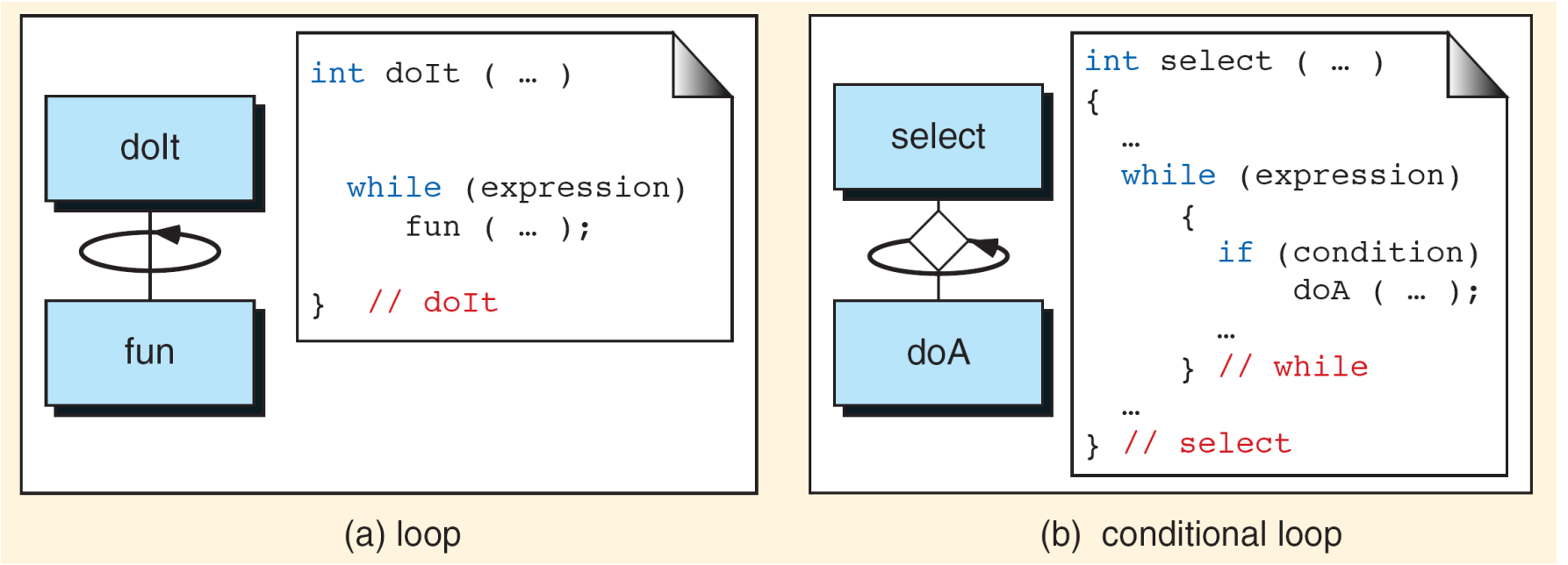


FIGURE 6-31 Structure Chart Symbols for Loops

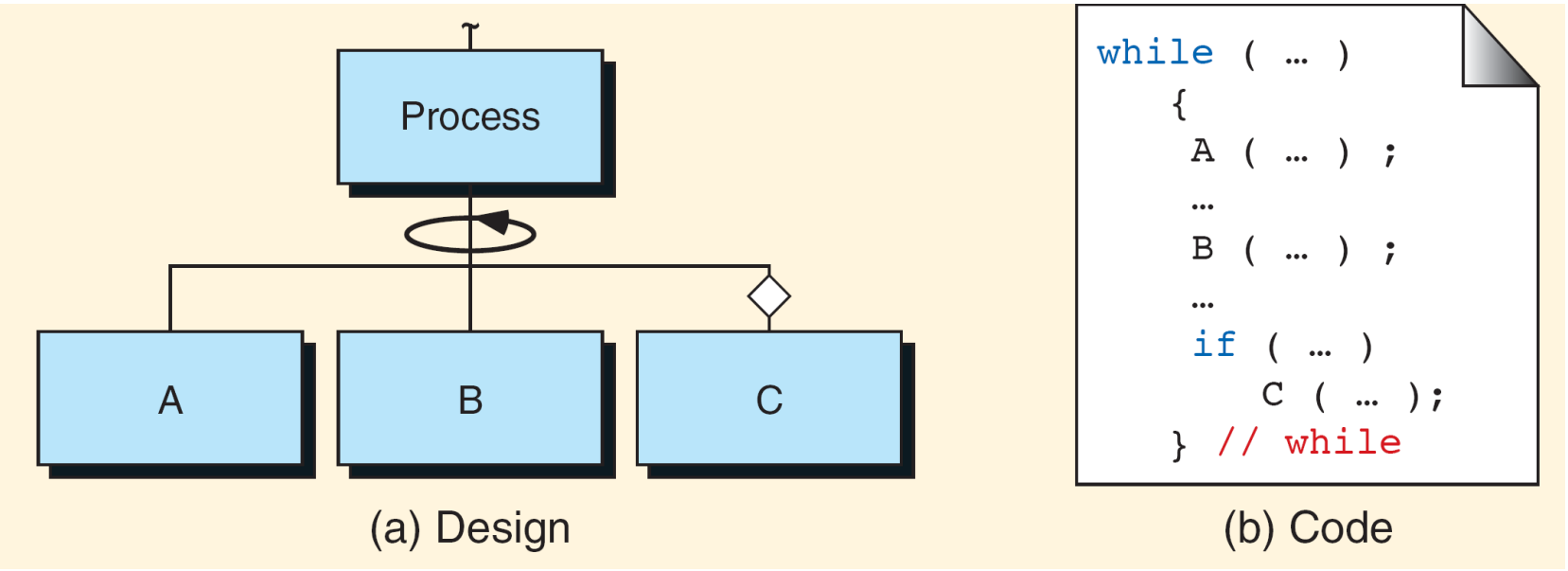


FIGURE 6-32 Structure Chart for Process

Determining Algorithm Efficiency

- Linear loops

```
for (int i = 0; i < 1000; i++) {
    BODY;
}
```

- Logarithmic loops

```
i = 1;
while (i < 1000) {
    BODY;
    i *= 2;
}
```

Multiply		Divide	
Iteration	i	Iteration	i
1	1	1	1000
2	2	2	500
3	4	3	250
4	8	4	125
5	16	5	62
6	32	6	31
7	64	7	15
8	128	8	7
9	256	9	3
10	512	10	1
(exit)	1024	(exit)	0

Table 6-4 Analysis of Multiply / Divide Loops

Determining Algorithm Efficiency

- Nested loops
  - Quadratic
  - Dependent quadratic
  - Linear logarithmic

Big-O Notation

- Keep the largest term in the function, and discard the others
- $f(n) = n^2 + n \rightarrow O(f(n)) = O(n^2)$
- Measure of efficiency
  - Count of the most frequent major operations
  - Big-O notation

Efficiency	Big-O	Iterations
logarithmic	$O(\log n)$	14
linear	$O(n)$	10,000
linear logarithmic	$O(n(\log n)$	140,000
quadratic	$O(n^2)$	$10{,}000^2$
polynomial	$O(n^k)$	$10{,}000^k$
exponential	$O(c^n)$	$2^{10{,}000}$
factorial	$O(n!)$	10,000!

Table 6-5 Measures of Efficiency