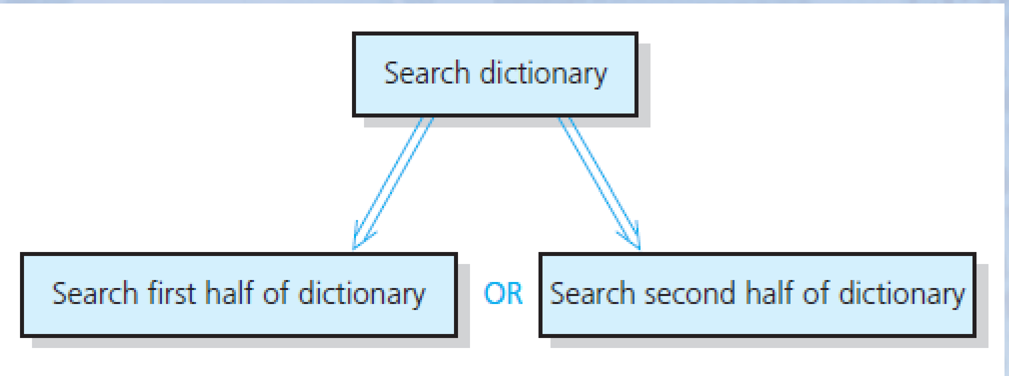
Recursion: The Mirrors

Chapter 2

Recursive Solutions

- Recursion breaks problem into smaller identical problems
  - An alternative to iteration
- FIGURE 2-1 A recursive solution



Recursive Solutions

- A recursive function calls itself
- Each recursive call solves an identical, but smaller, problem
- Test for base case enables recursive calls to stop
- Eventually, one of smaller problems must be the base case

Recursive Solutions

Questions for constructing recursive solutions

1. How to define the problem in terms of a smaller problem of same type?
2. How does each recursive call diminish the size of the problem?
3. What instance of problem can serve as base case?
4. As problem size diminishes, will you reach base case?

A Recursive Valued Function: The Factorial of *n*

- An iterative solution

$$factorial(n) = n \times (n-1) \times (n-2) \times \cdots \times 1 \quad \text{for an integer } n > 0$$
$$factorial(0) = 1$$

- A factorial solution

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

Note: Do not use recursion if a problem has a simple, efficient iterative solution

A Recursive Valued Function: The Factorial of *n*

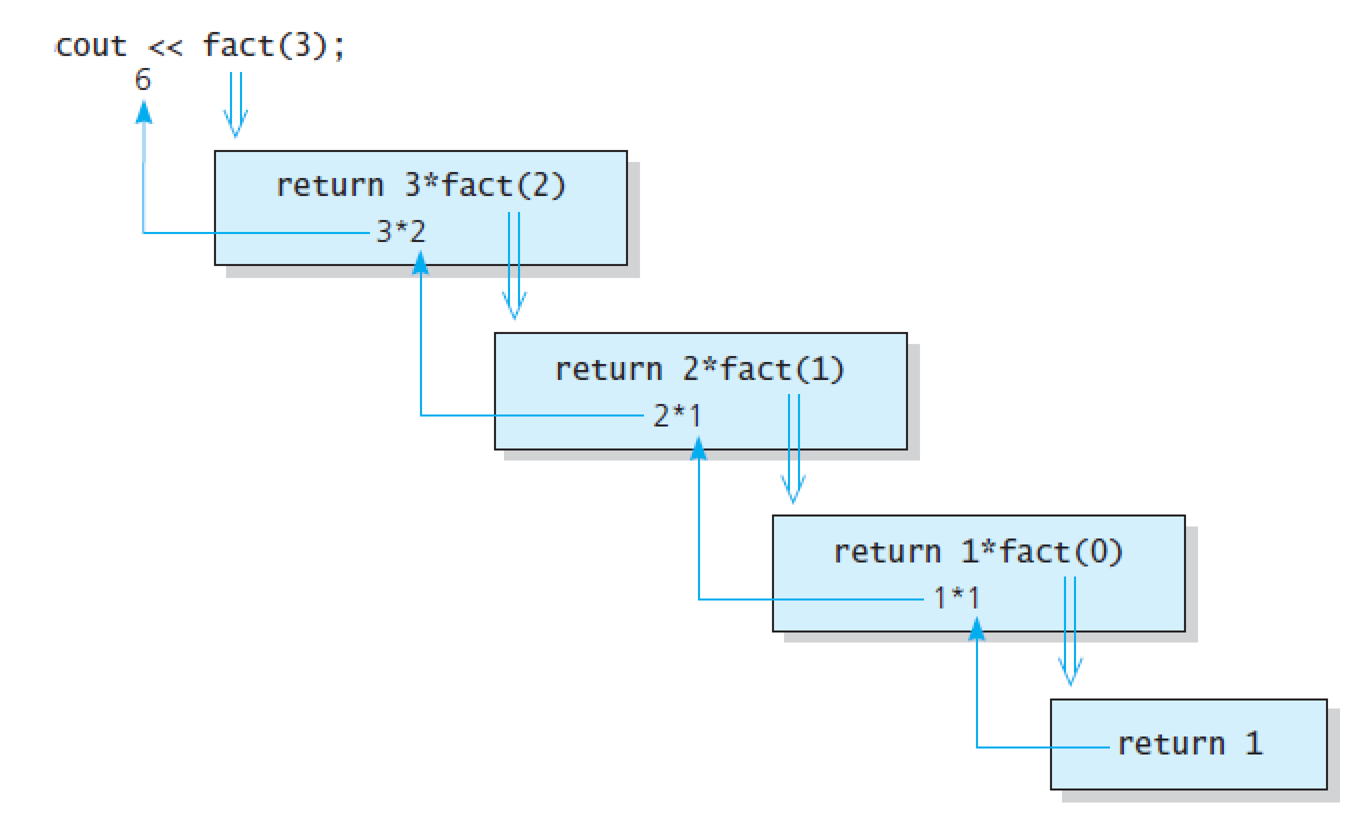


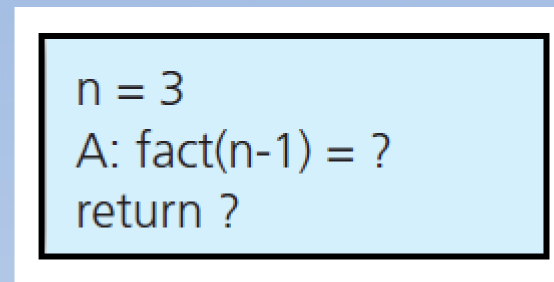
FIGURE 2-2 fact(3)

The Box Trace

1. Label each recursive call
2. Represent each call to function by a new box
3. Draw arrow from box that makes call to newly created box
4. After you create new box executing body of function
5. On exiting function, cross off current box and follow its arrow back

The Box Trace

FIGURE 2-3 A box



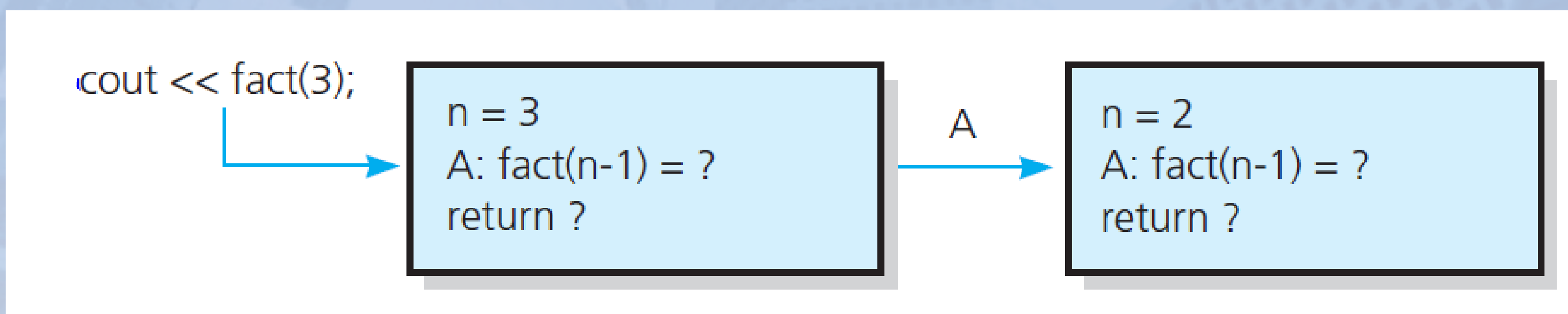


FIGURE 2-4 The beginning of the box trace

The Box Trace

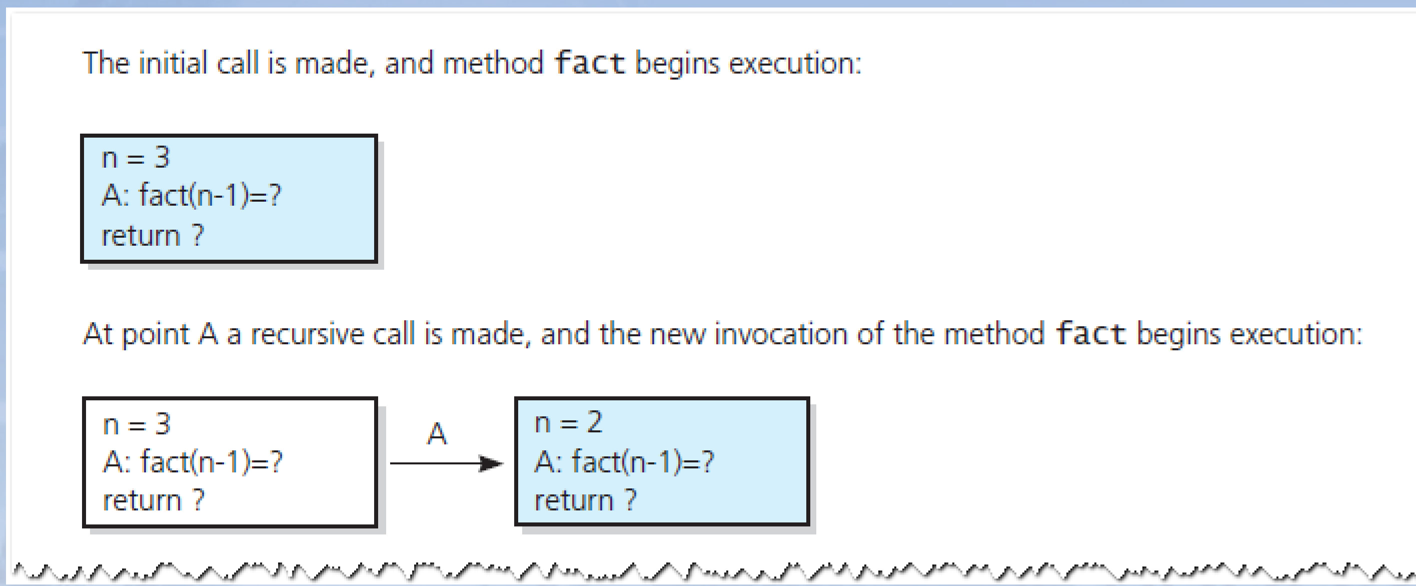


FIGURE 2-5 Box trace of fact(3)

The Box Trace

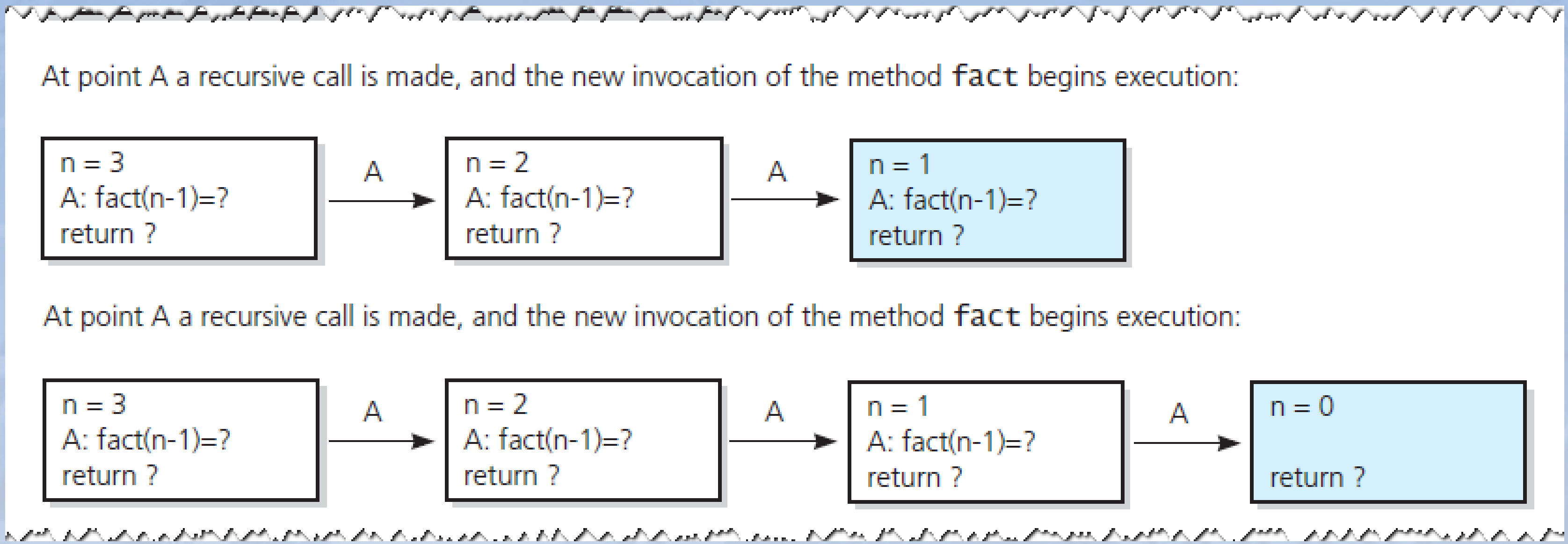


FIGURE 2-5 Box trace of fact(3)

The Box Trace

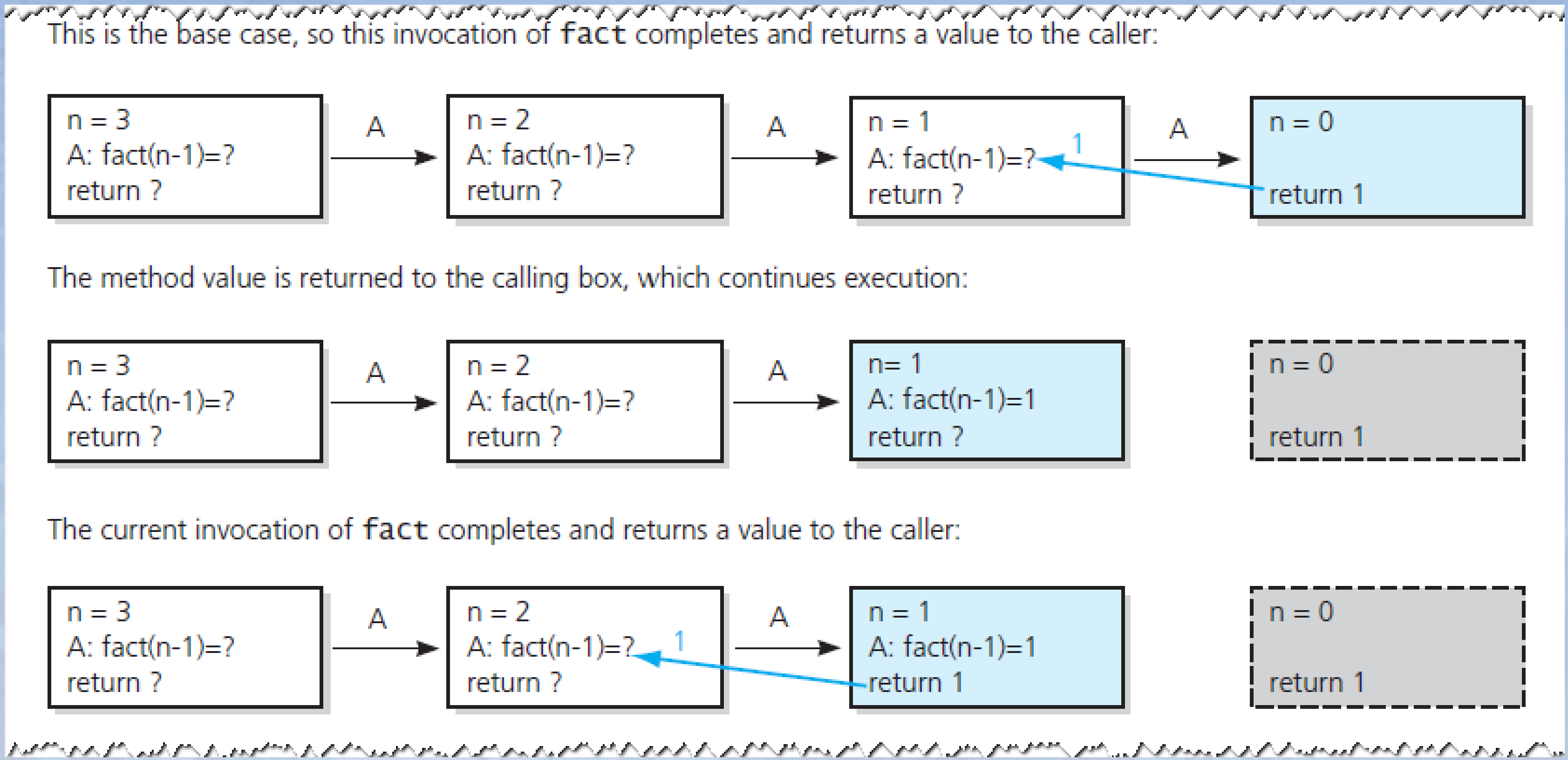


FIGURE 2-5 Box trace of fact(3)

The Box Trace

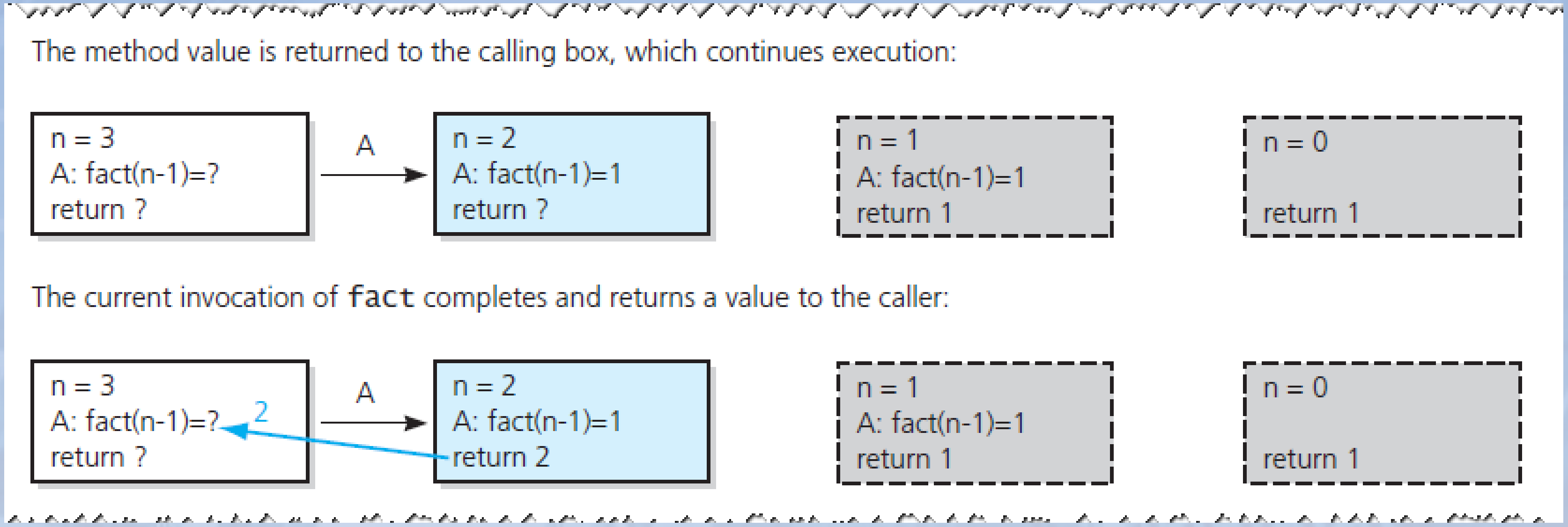


FIGURE 2-5 Box trace of fact(3)

The Box Trace

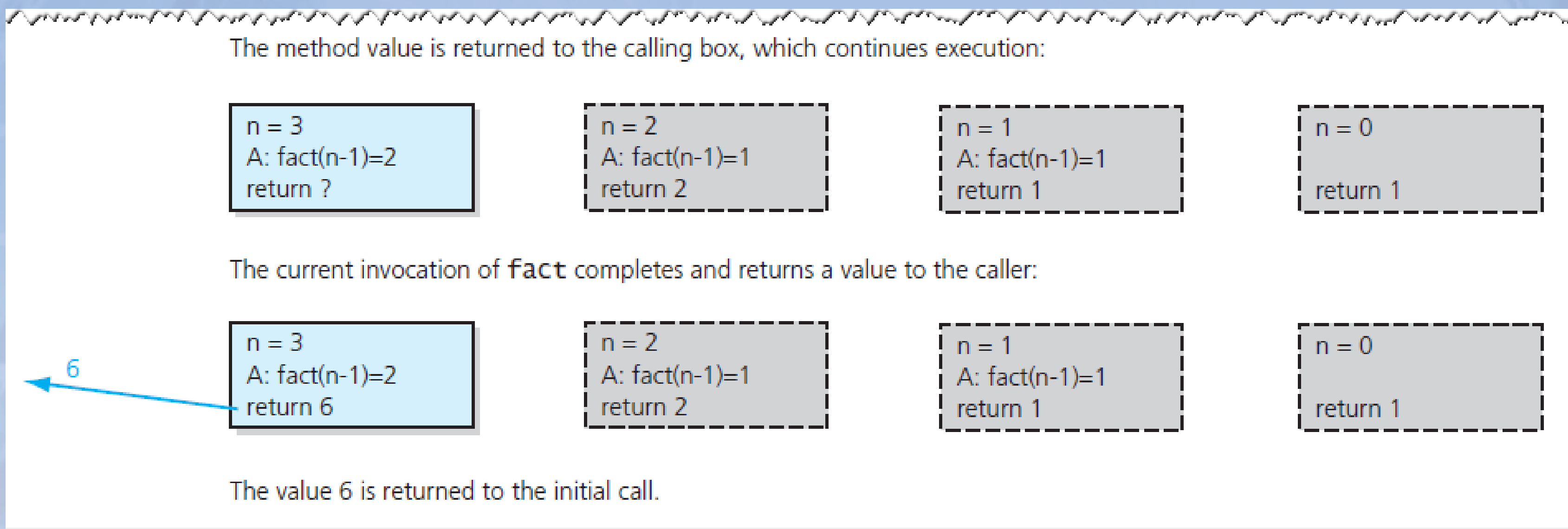


FIGURE 2-5 Box trace of fact(3)

A Recursive Void Function: Writing a String Backward

- Likely candidate for minor task is writing a single character.
  - Possible solution: strip away the last character

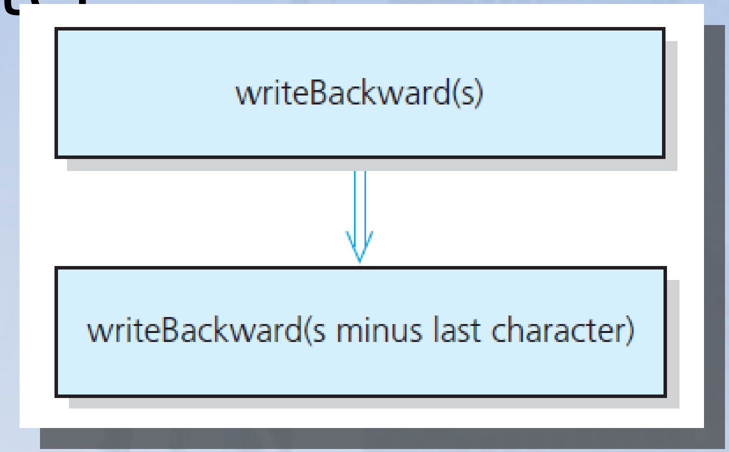


FIGURE 2-6 A recursive solution

A Recursive Void Function: Writing a String Backward

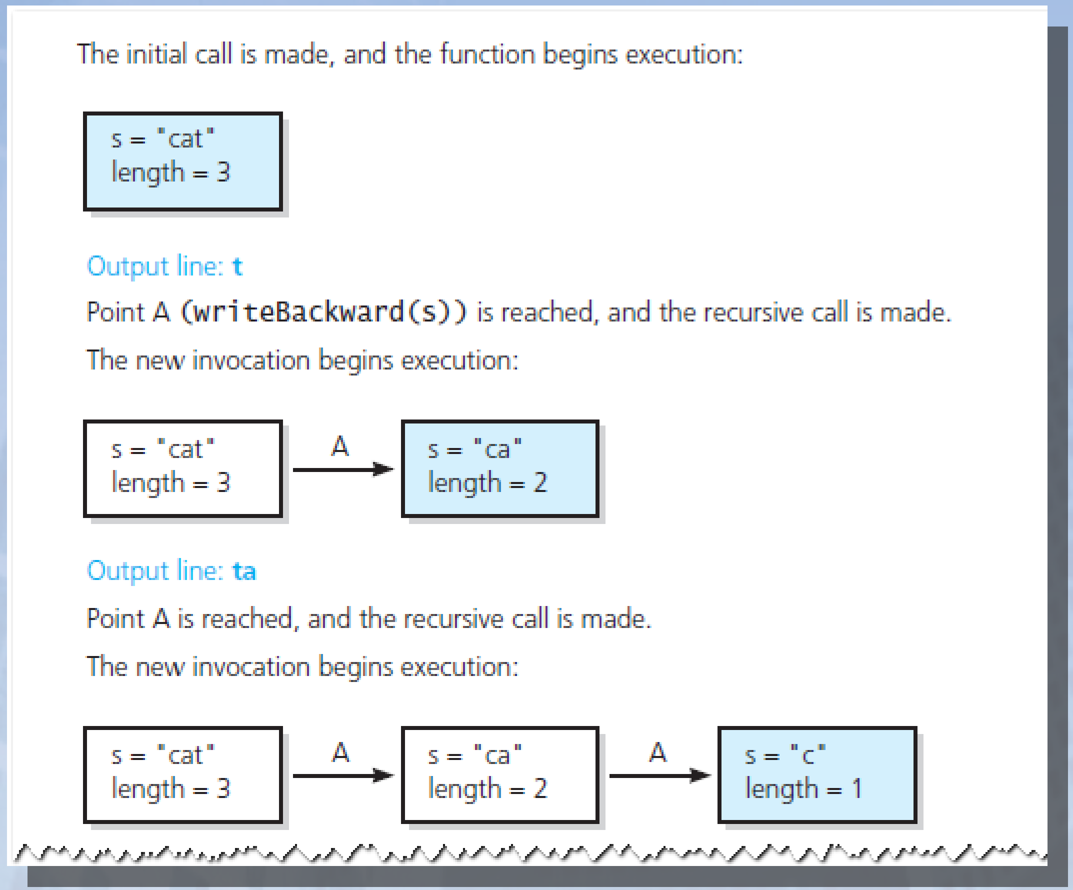


FIGURE 2-7 Box trace of writeBackward("cat")

A Recursive Void Function: Writing a String Backward

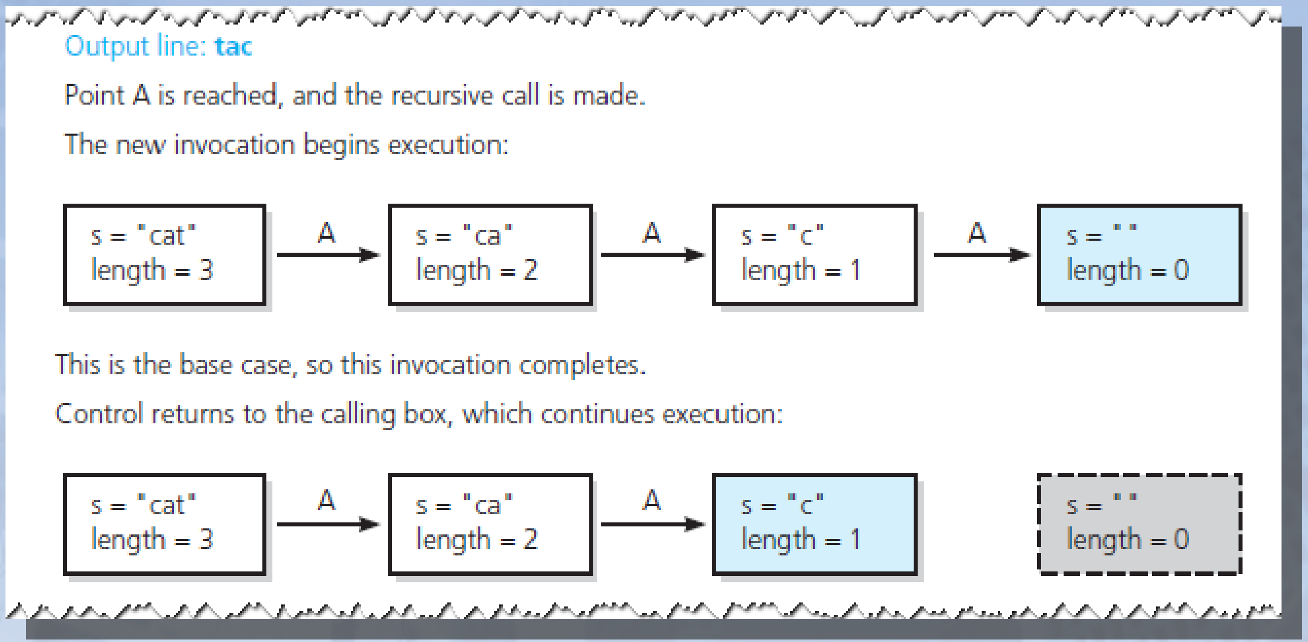


FIGURE 2-7 Box trace of writeBackward("cat")

A Recursive Void Function: Writing a String Backward

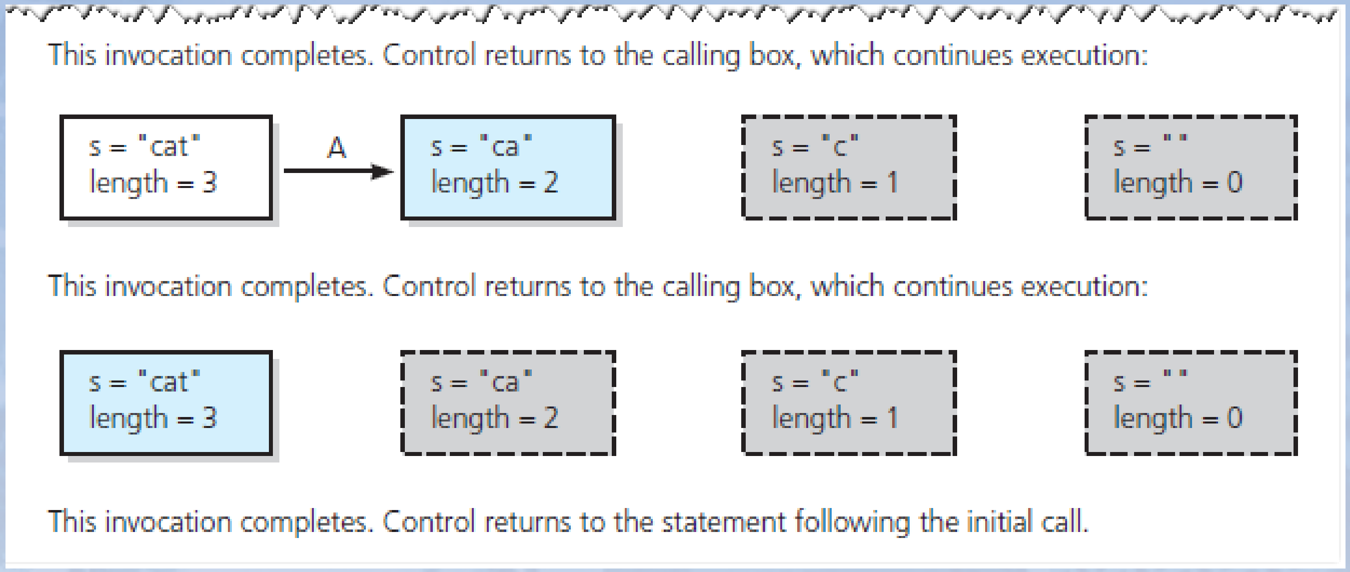


FIGURE 2-7 Box trace of writeBackward("cat")

A Recursive Void Function: Writing a String Backward

- Another possible solution
  - Strip away the first character

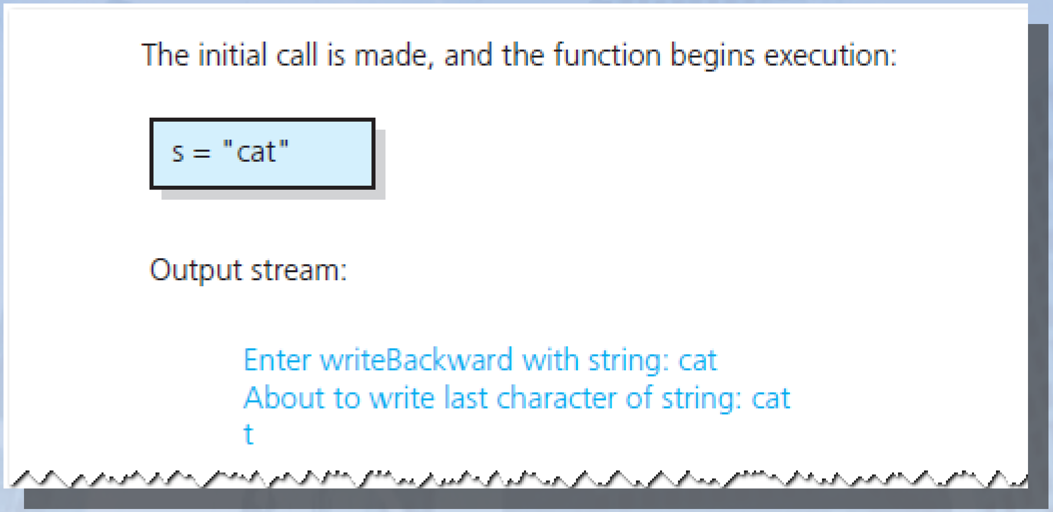


FIGURE 2-8 Box trace of writeBackward("cat") in pseudocode

A Recursive Void Function: Writing a String Backward

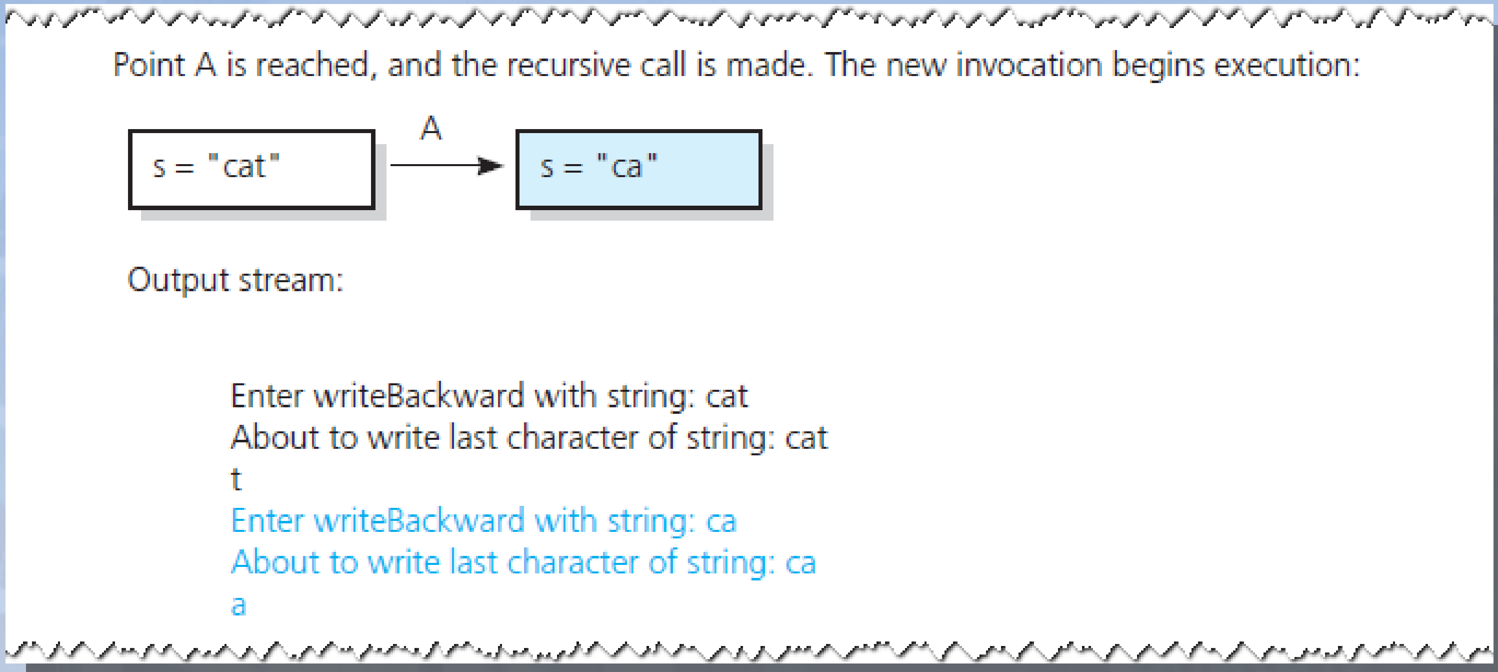


FIGURE 2-8 Box trace of writeBackward("cat") in pseudocode

A Recursive Void Function: Writing a String Backward

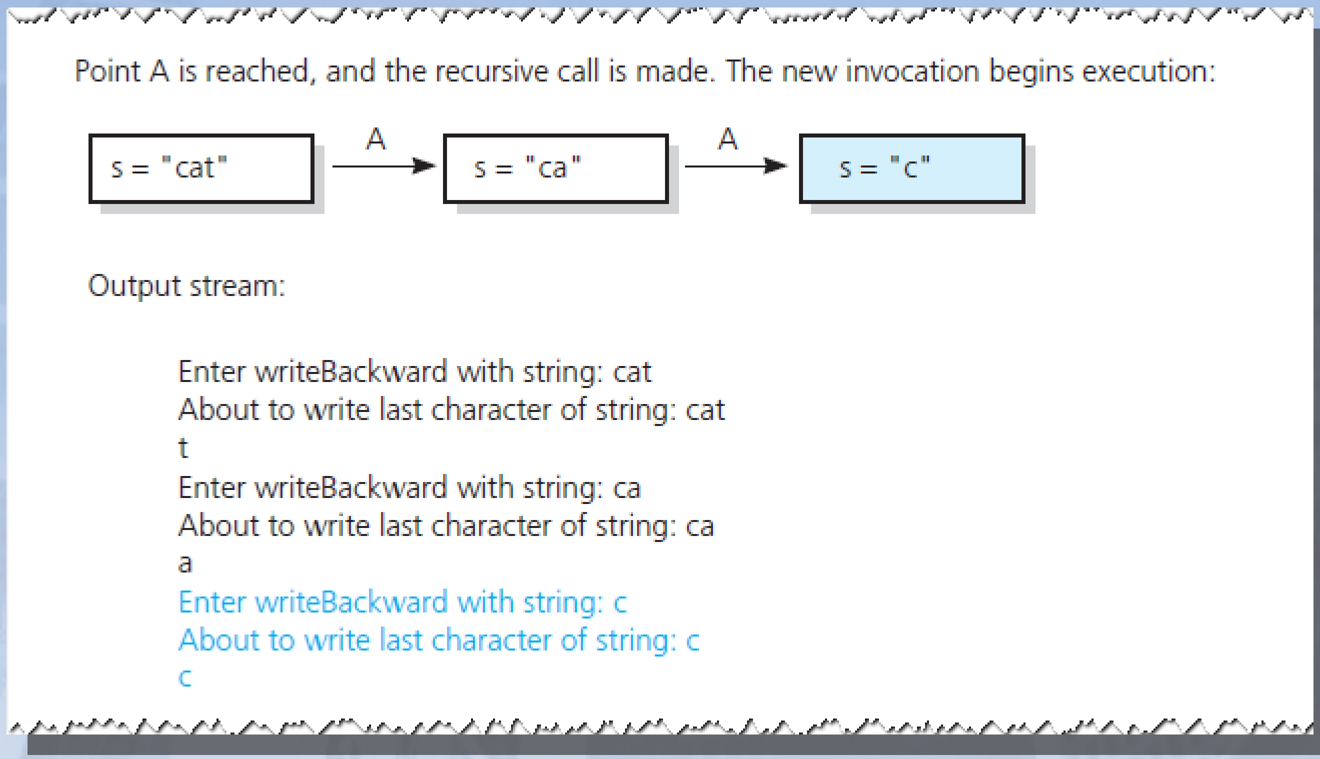


FIGURE 2-8 Box trace of writeBackward("cat") in pseudocode

A Recursive Void Function: Writing a String Backward

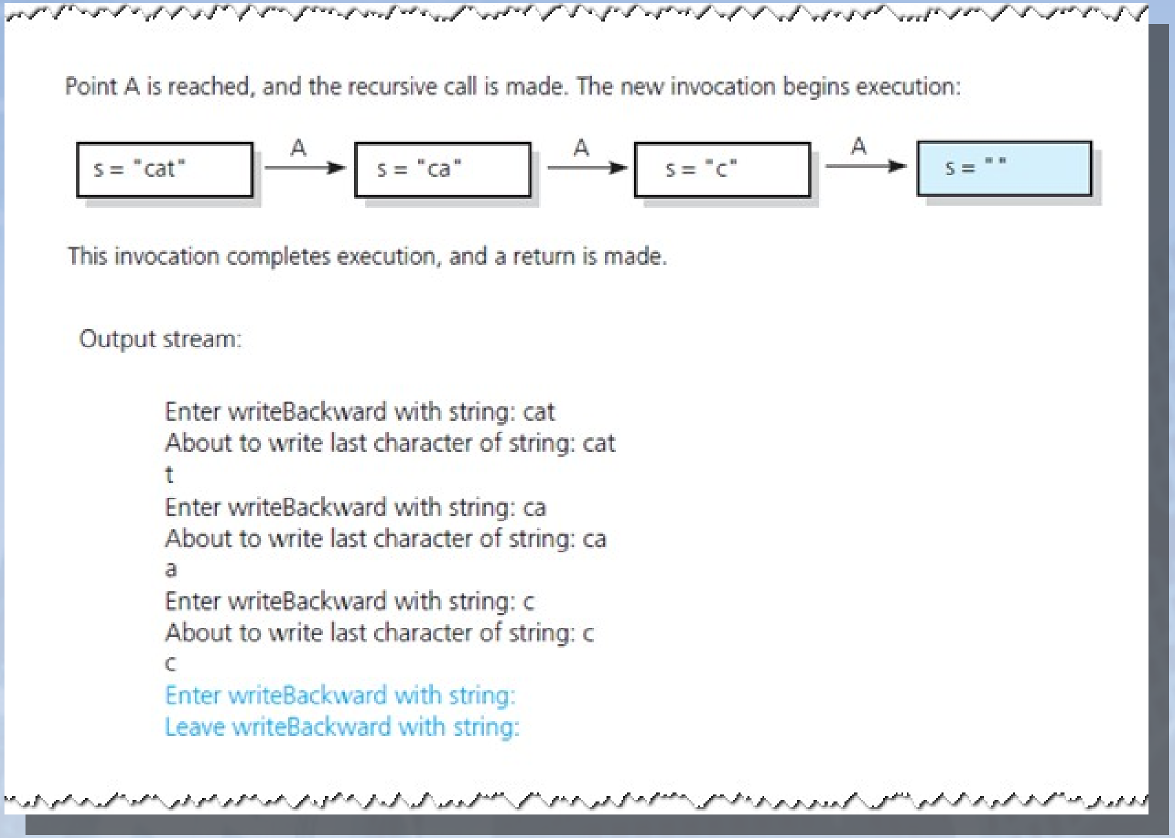
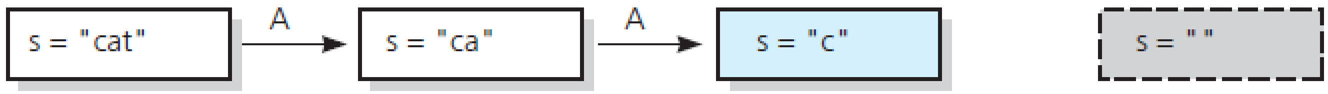


FIGURE 2-8 Box trace of writeBackward("cat") in pseudocode

A Recursive Void Function: Writing a String Backward



This invocation completes execution, and a return is made.

Output stream:

```
Enter writeBackward with string: cat
About to write last character of string: cat
t
Enter writeBackward with string: ca
About to write last character of string: ca
a
Enter writeBackward with string: c
About to write last character of string: c
c
Enter writeBackward with string:
Leave writeBackward with string:
Leave writeBackward with string: c
```

FIGURE 2-8 Box trace of writeBackward("cat") in pseudocode

A Recursive Void Function: Writing a String Backward

This invocation completes execution, and a return is made.

Output stream:

Enter writeBackward with string: cat
About to write last character of string: cat
t
Enter writeBackward with string: ca
About to write last character of string: ca
a
Enter writeBackward with string: c
About to write last character of string: c
c
Enter writeBackward with string:
Leave writeBackward with string:
Leave writeBackward with string: c
Leave writeBackward with string: ca

FIGURE 2-8 Box trace of writeBackward("cat") in pseudocode

A Recursive Void Function: Writing a String Backward

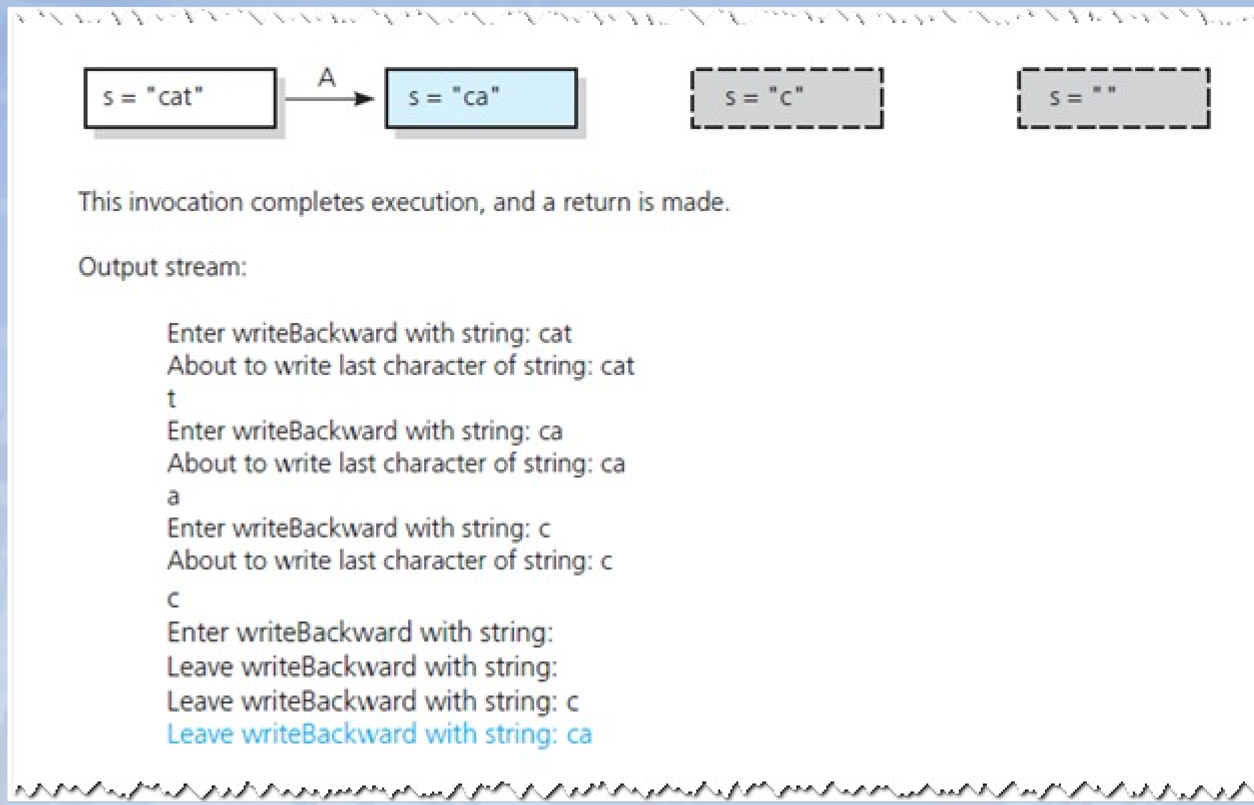


FIGURE 2-8 Box trace of writeBackward("cat") in pseudocode

A Recursive Void Function: Writing a String Backward

This invocation completes execution, and a return is made.

Output stream:

Enter writeBackward with string: cat
About to write last character of string: cat
t
Enter writeBackward with string: ca
About to write last character of string: ca
a
Enter writeBackward with string: c
About to write last character of string: c
c
Enter writeBackward with string:
Leave writeBackward with string:
Leave writeBackward with string: c
Leave writeBackward with string: ca
Leave writeBackward with string: cat

FIGURE 2-8 Box trace of writeBackward("cat") in pseudocode

A Recursive Void Function: Writing a String Backward

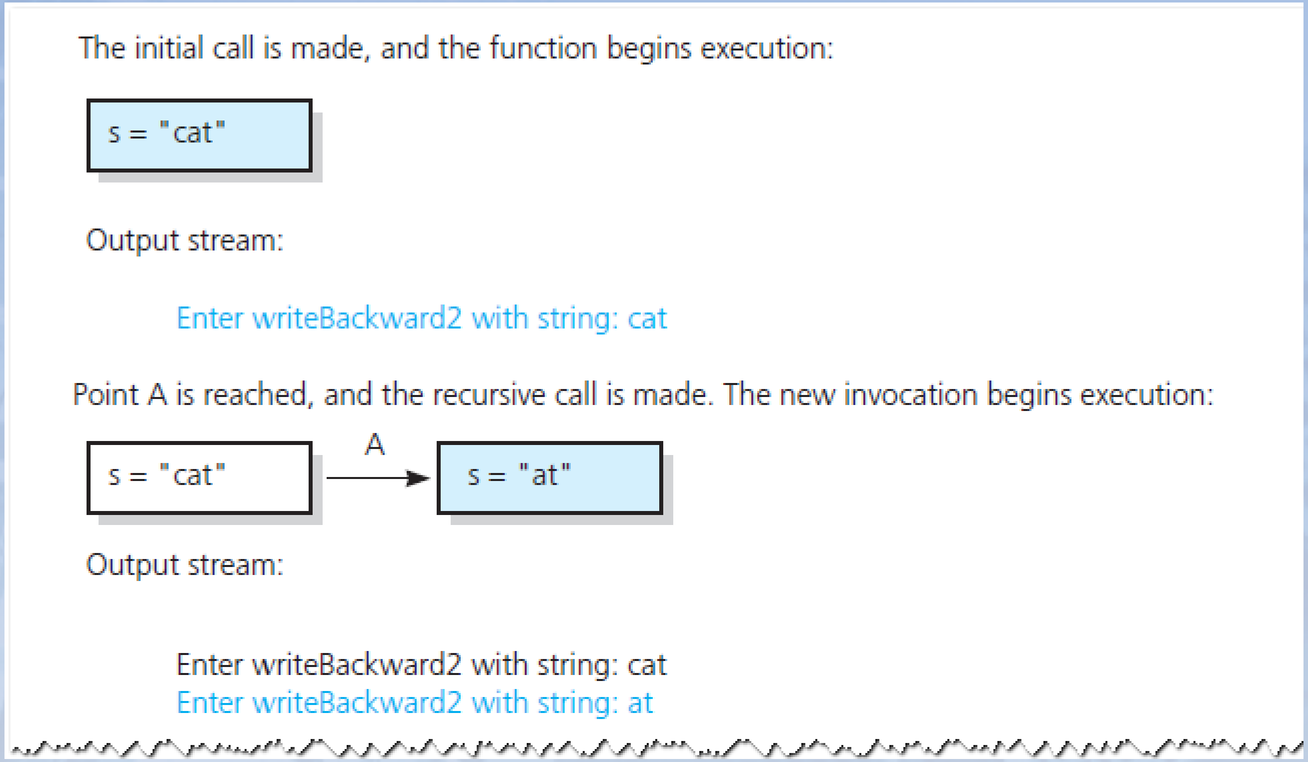


FIGURE 2-8 Box trace of writeBackward2("cat") in pseudocode

A Recursive Void Function: Writing a String Backward

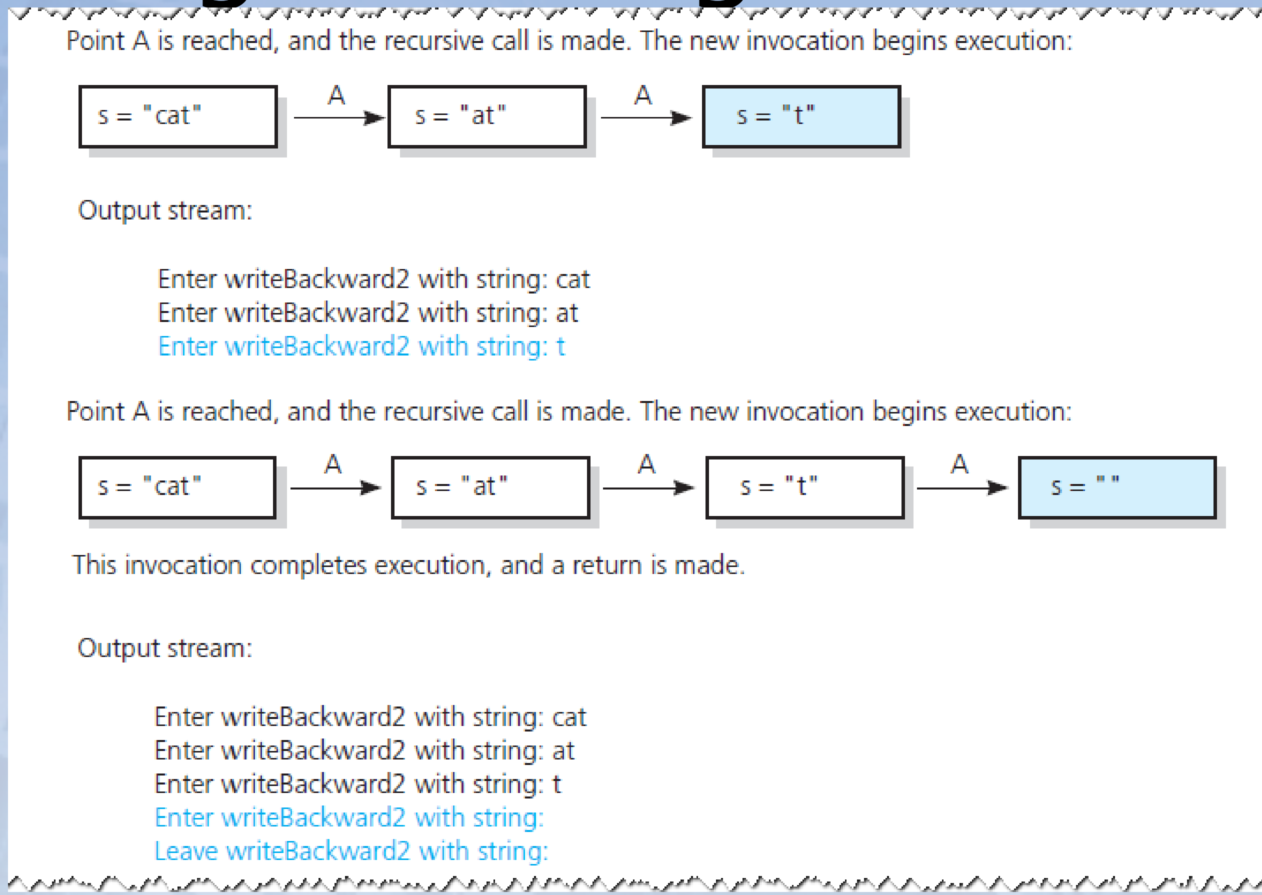


FIGURE 2-8 Box trace of writeBackward2("cat") in pseudocode

A Recursive Void Function: Writing a String Backward

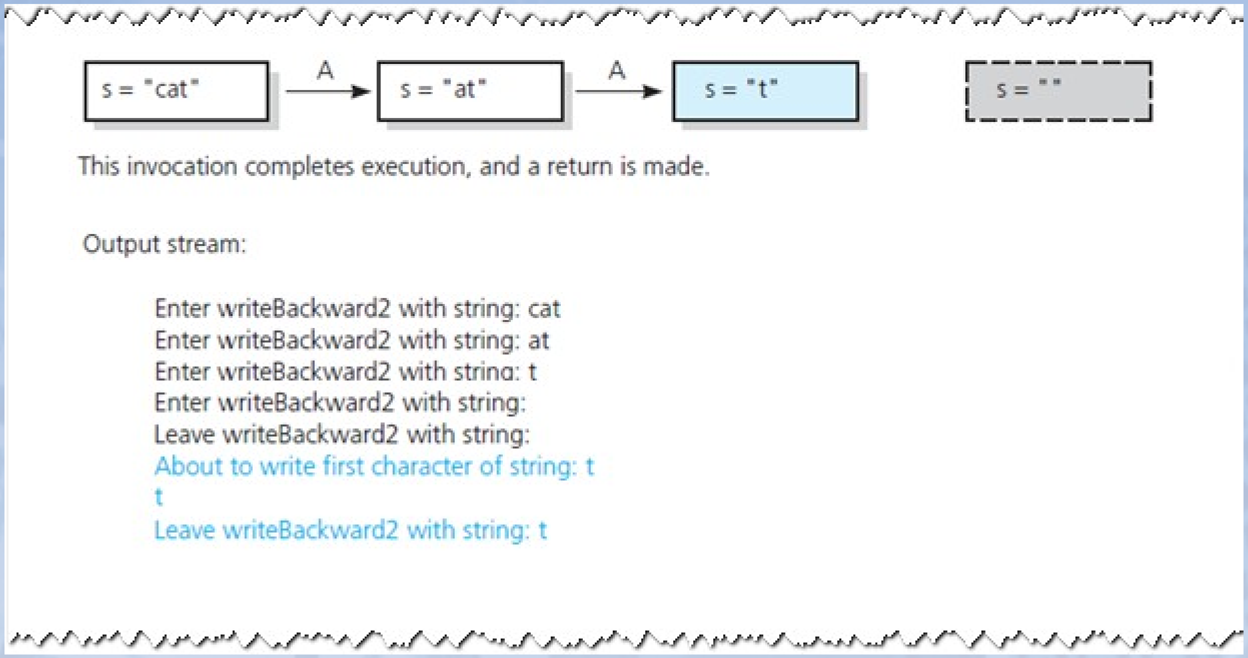


FIGURE 2-8 Box trace of writeBackward2("cat") in pseudocode

A Recursive Void Function: Writing a String Backward

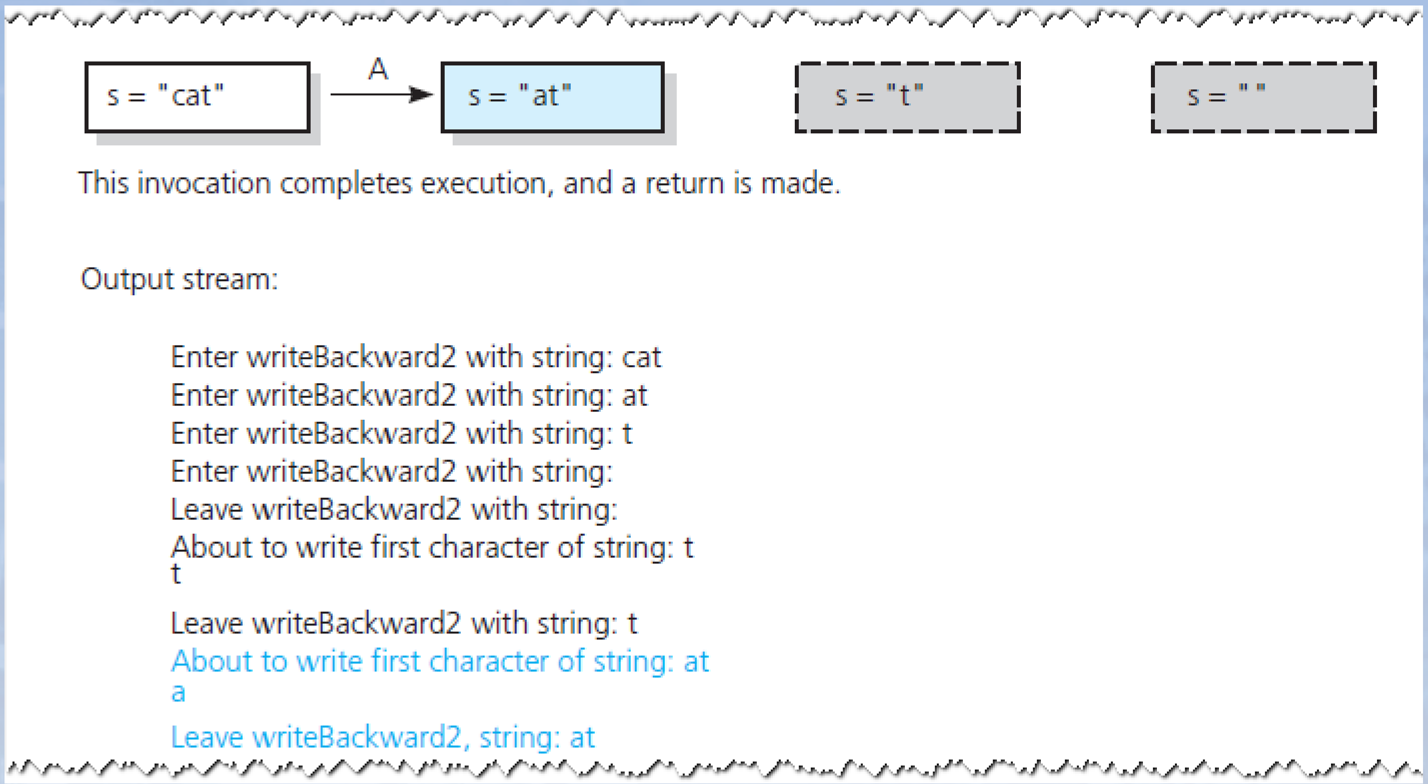
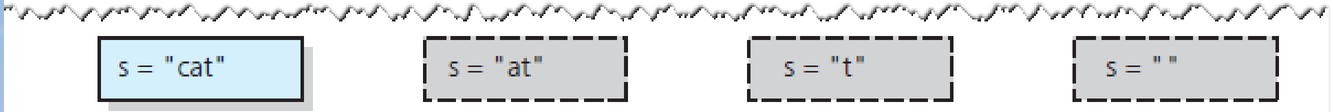


FIGURE 2-8 Box trace of writeBackward2("cat") in pseudocode

A Recursive Void Function: Writing a String Backward



This invocation completes execution, and a return is made.

Output stream:

Enter writeBackward2 with string: cat
Enter writeBackward2 with string: at
Enter writeBackward2 with string: t
Enter writeBackward2 with string:
Leave writeBackward2 with string:
About to write first character of string: t
t
Leave writeBackward2 with string: t
About to write first character of string: at
a
Leave writeBackward2 with string: at
About to write first character of string: cat
c
Leave writeBackward2 with string: cat

FIGURE 2-8 Box trace of writeBackward2("cat") in pseudocode

Writing an Array's Entries in Backward Order

```
/** Writes the characters in an array backward.
  @pre  The array anArray contains size characters, where size >= 0.
  @post  None.
  @param anArray  The array to write backward.
  @param first  The index of the first character in the array.
  @param last  The index of the last character in the array. */
void writeArrayBackward(const char anArray[], int first, int last)
{
    if (first <= last)
    {
        // Write the last character
        cout << anArray[last];

        // Write the rest of the array backward
        writeArrayBackward(anArray, first, last - 1);
    }  // end if

    // first > last is the base case - do nothing
}  // end writeArrayBackward
```

The function writeArrayBackward

The Binary Search

Consider details before implementing algorithm:

1. How to pass half of anArray to recursive calls of binarySearch ?
2. How to determine which half of array contains target?
3. What should base case(s) be?
4. How will binarySearch indicate result of search?

The Binary Search

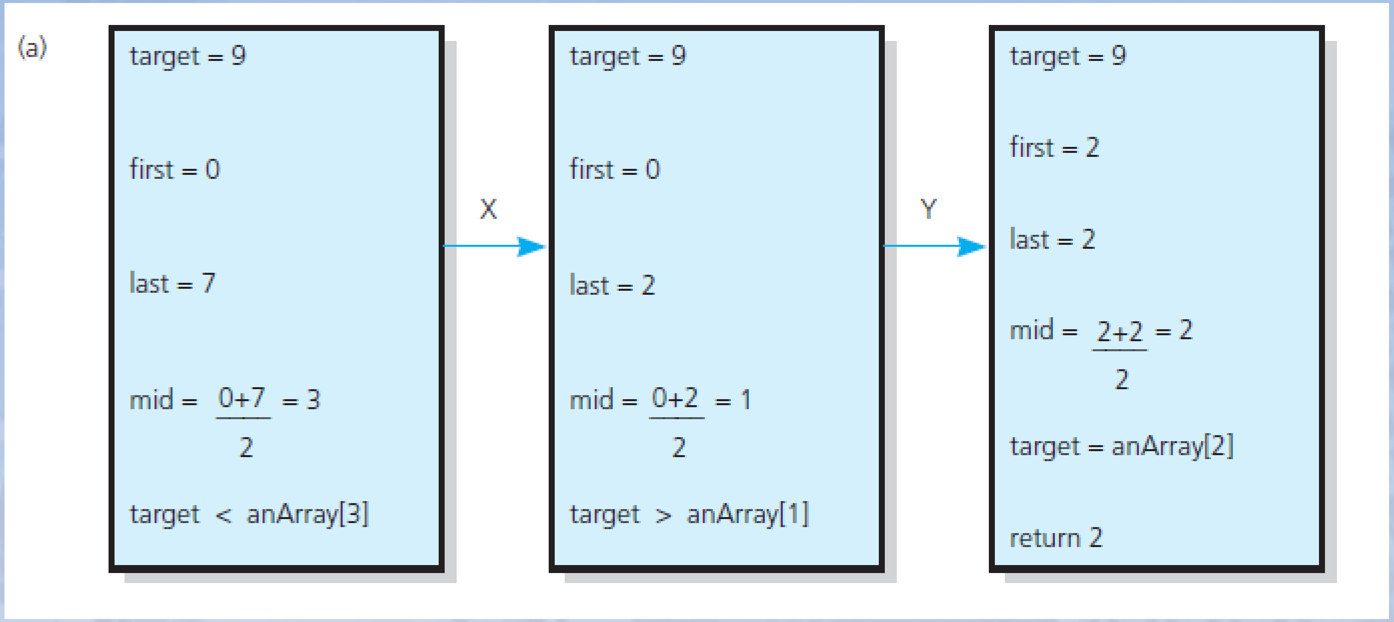


FIGURE 2-10 Box traces of binarySearch with anArray = <1, 5, 9, 12, 15, 21, 29, 31>: (a) a successful search for 9;

The Binary Search

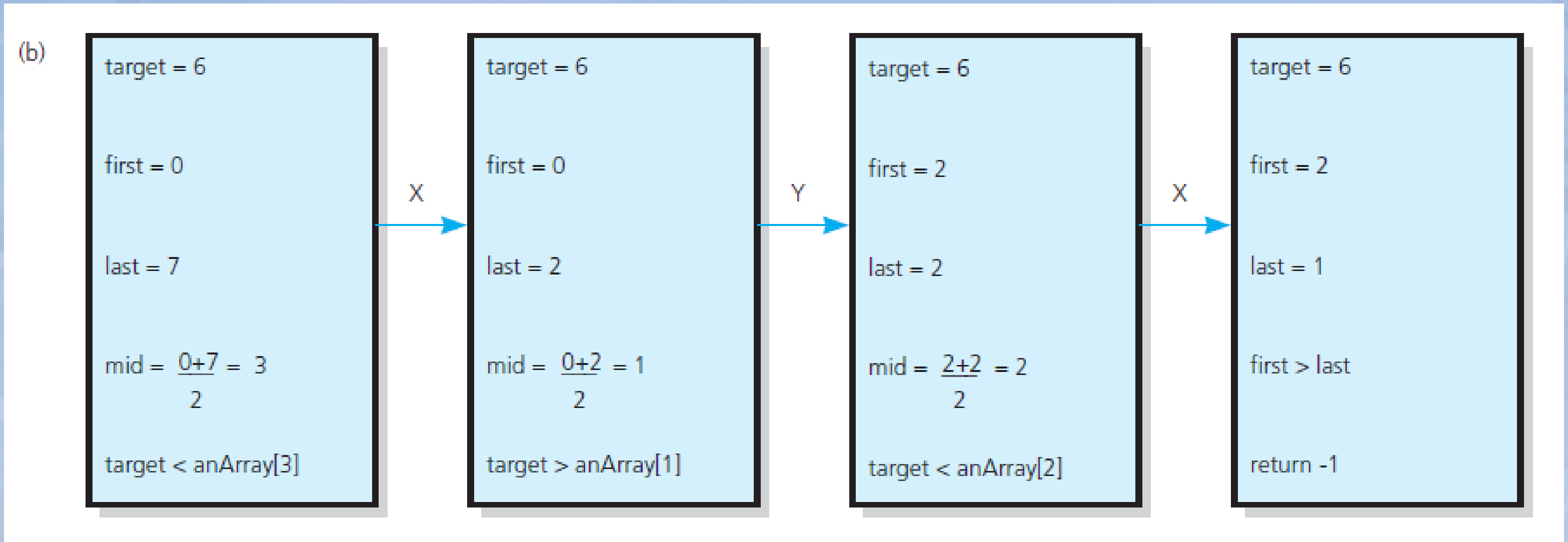


FIGURE 2-10 Box traces of binarySearch with anArray = <1, 5, 9, 12, 15, 21, 29, 31>: (b) an unsuccessful search for 6

The Binary Search

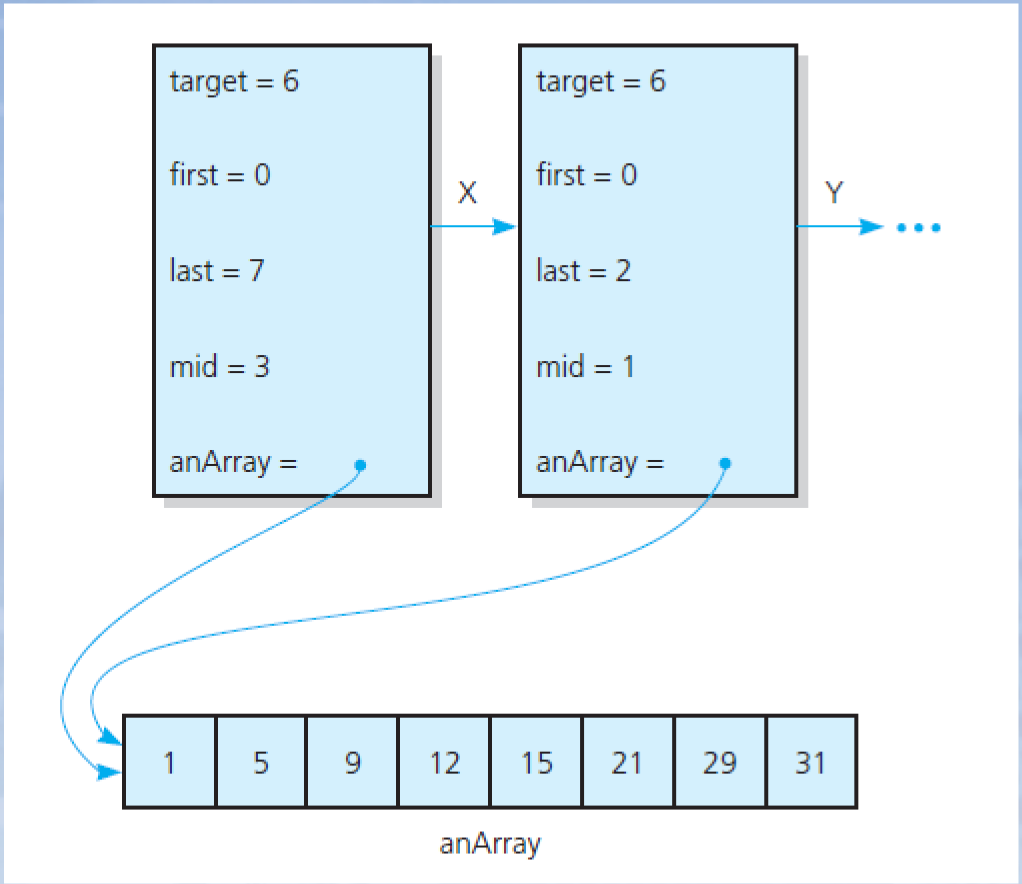


FIGURE 2-11 Box trace with a reference argument

Finding the Largest Value in an Array

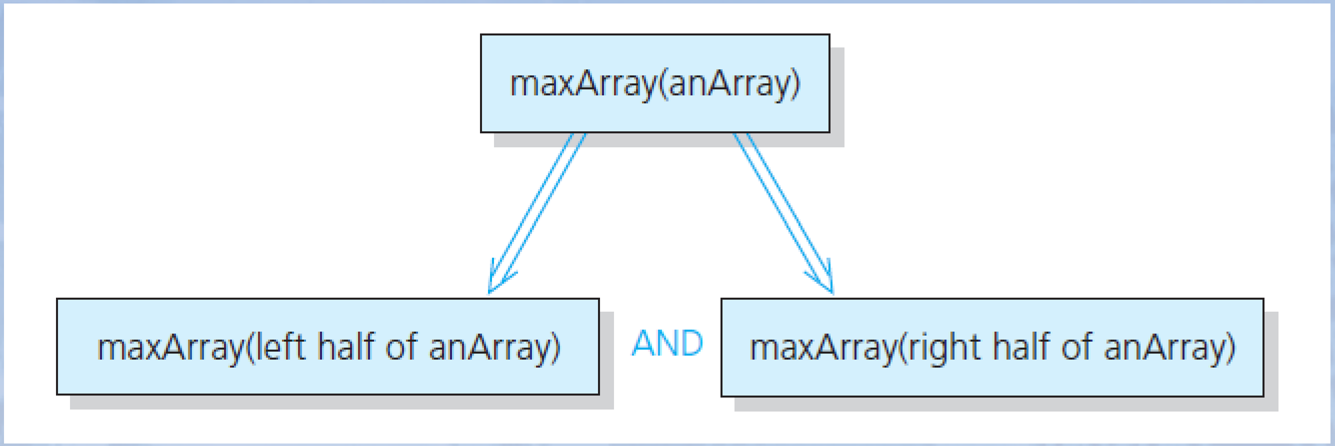


FIGURE 2-12 Recursive solution to the largest-value problem

Finding the Largest Value in an Array

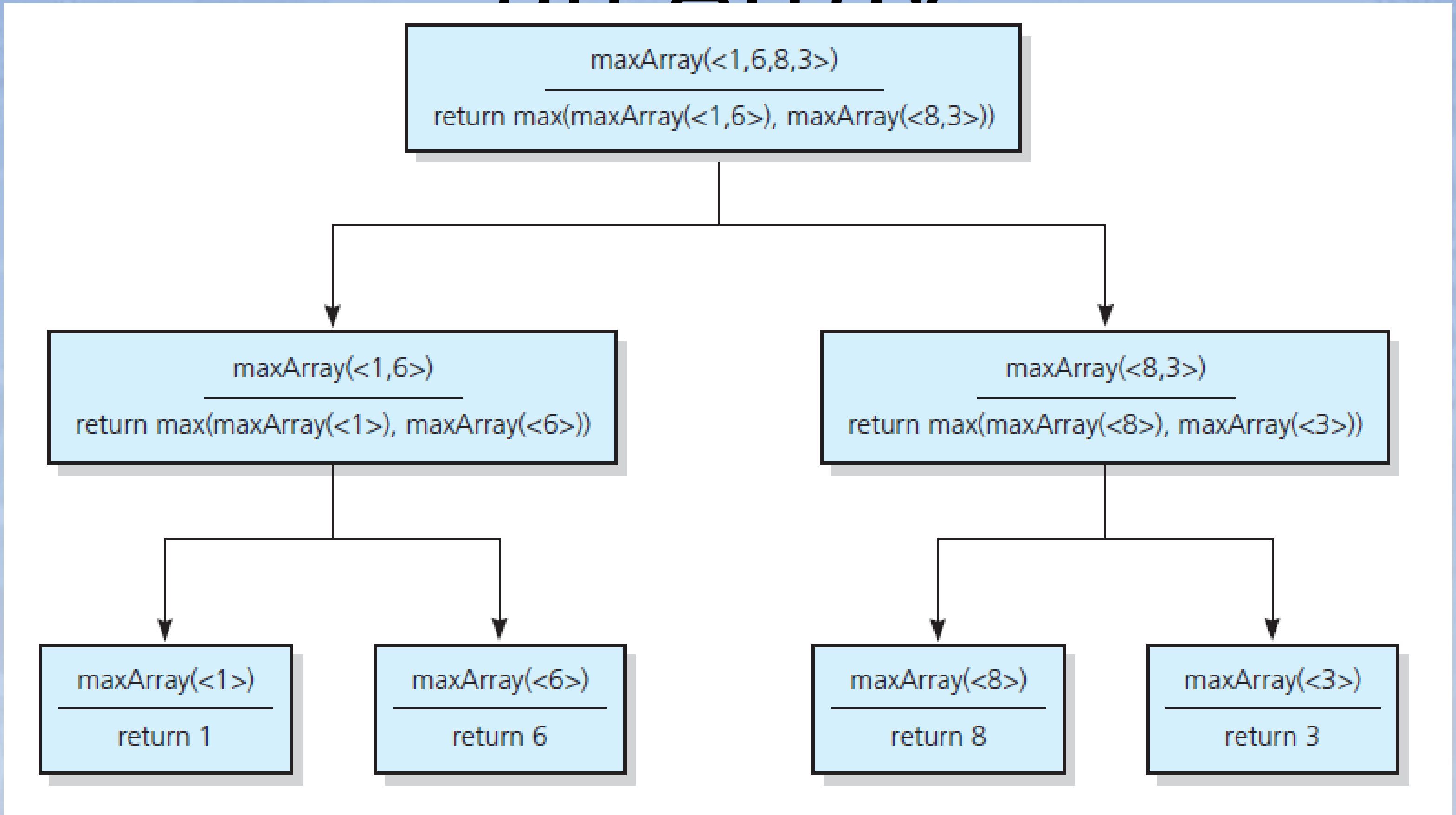


FIGURE 2-13 The recursive calls that maxArray(<1,6,8,3>) generates

Finding k^{th} Smallest Value of Array

Recursive solution proceeds by:

1. Selecting pivot value in array
2. Cleverly arranging/ partitioning values in array about pivot value
3. Recursively applying strategy to one of partitions

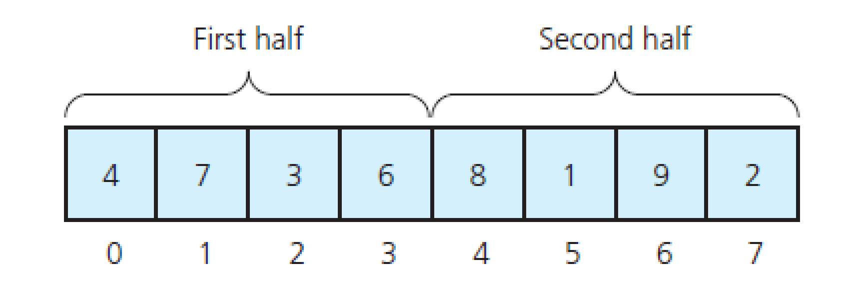


FIGURE 2-14 A sample array

Finding kth Smallest Value of Array

FIGURE 2-15 A partition about a pivot

The Towers of Hanoi

- The problem statement
  – Beginning with *n* disks on pole A and zero disks on poles B and C, solve towers(n, A, B, C) .
- Solution
  1. With all disks on A, solve towers(n – 1, A, C, B)
  2. With the largest disk on pole A and all others on pole C, solve towers(n – 1, A, B, C)
  3. With the largest disk on pole B and all the other disks on pole C, solve towers(n – 1, C, B, A)

The Towers of Hanoi

FIGURE 2-16 (

The Towers of Hanoi

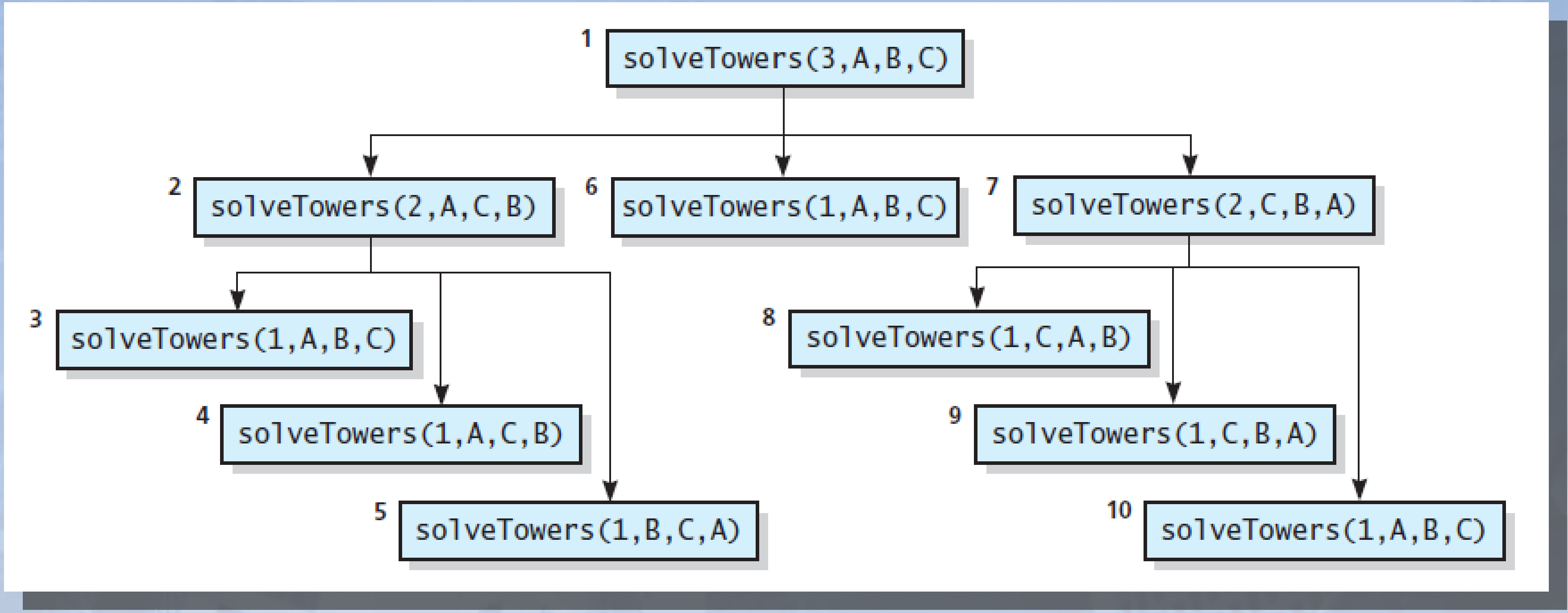


FIGURE 2-17 The order of recursive calls that results from solveTowers(3, A, B, C)

The Fibonacci Sequence (Multiplying Rabbits)

Assume the following “facts” …

- Rabbits never die.
- Rabbit reaches sexual maturity at beginning of third month of life.
- Rabbits always born in male-female pairs. At beginning of every month, each sexually mature male-female pair gives birth to exactly one male-female pair.

The Fibonacci Sequence (Multiplying Rabbits)

Monthly sequence

1. One pair, original two rabbits
2. One pair still
3. Two pairs (original pair, two newborns)
4. Three pairs (original pair, 1 month old, newborns)
5. Five pairs …
6. Eight pairs …

The Fibonacci Sequence (Multiplying Rabbits)

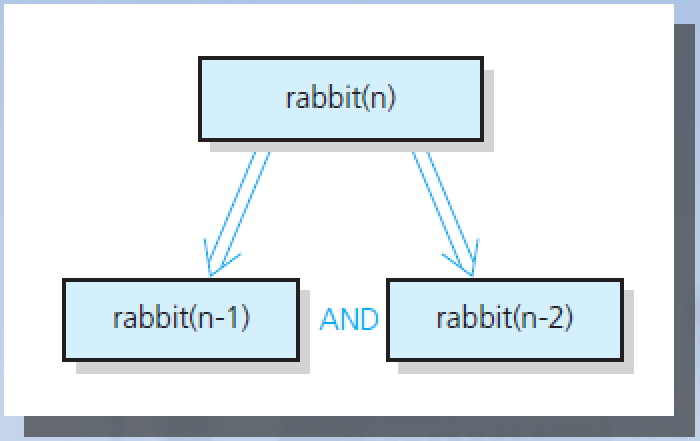
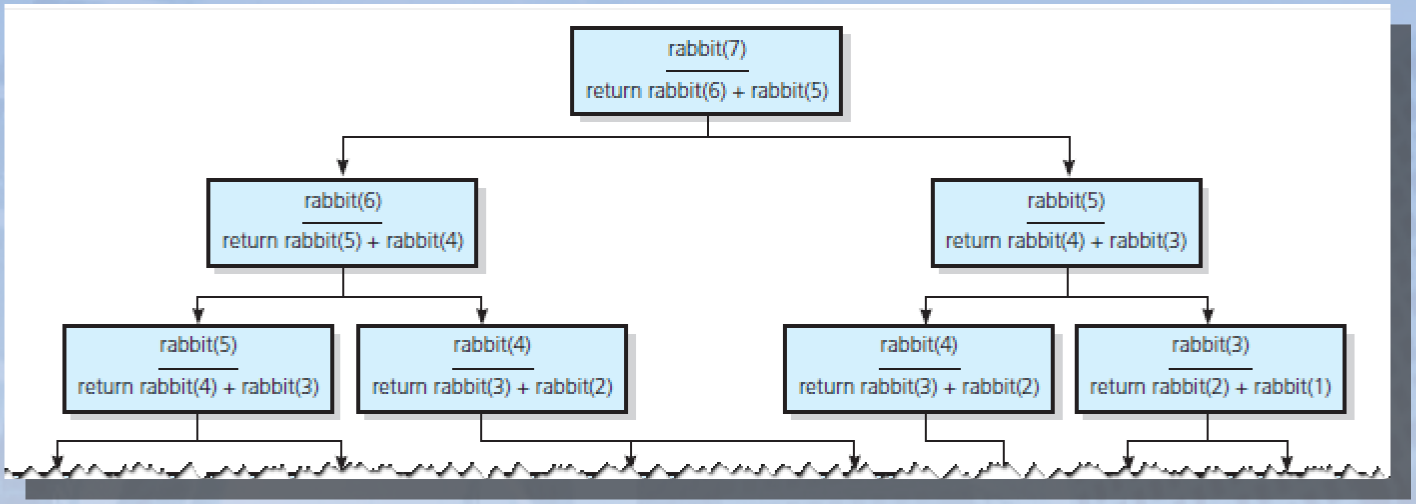


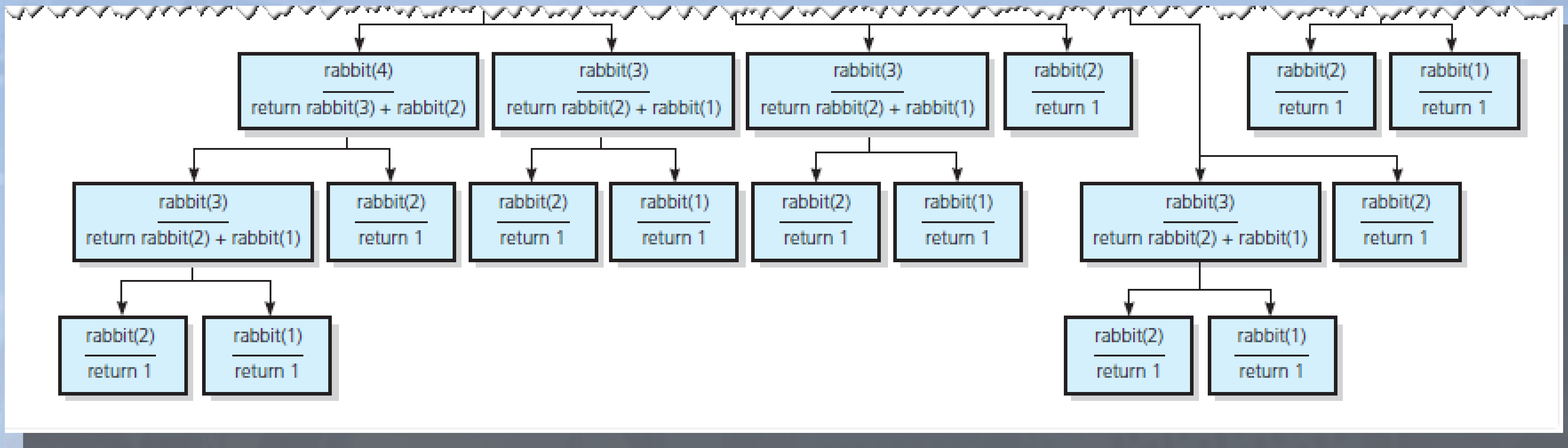
FIGURE 2-18 Recursive solution to the rabbit problem (number of pairs at month *n*)

The Fibonacci Sequence (Multiplying Rabbits)



F IGURE 2-19 The recursive calls that rabbit(7) generates

The Fibonacci Sequence (Multiplying Rabbits)



F IGURE 2-19 The recursive calls that rabbit(7) generates

Organizing a Parade

- Will consist of bands and floats in single line.
  - You are asked not to place one band immediately after another
- In how many ways can you organize a parade of length n ?
  - $P(n)$ = number of ways to organize parade of length n
  - $F(n)$ = number of parades of length n, end with a float
  - $B(n)$ = number of parades of length n, end with a band
- Then $P(n) = F(n) + B(n)$

Organizing a Parade

- Possible to see
  - $P(1) = 2$
  - $P(2) = 3$
  - $P(n) = P(n - 1) + P(n - 2)$ for $n > 2$
- Thus a recursive solution
  - Solve the problem by breaking up into cases

Choosing *k* Out of *n* Things

- Rock band wants to tour *k* out of *n* cities
  - Order not an issue
- Let *g(n, k)* be number of groups of *k* cities ch

$$g(n,k) = g(n-1,k-1) + g(n-1,k)$$

- *Base cases*

$$g(k,k) = 1$$
$$g(n,0) = 1$$

Choosing *k* Out of *n* Things

```
/** Computes the number of groups of k out of n things.
 @pre  n and k are nonnegative integers.
 @post  None.
 @param n  The given number of things.
 @param k  The given number to choose.
 @return  g(n, k). */
int getNumberOfGroups(int n, int k)
{
   if ( (k == 0) || (k == n) )
      return 1;
   else if (k > n)
      return  0;
   else
      return getNumberOfGroups(n - 1, k - 1) + getNumberOfGroups(n - 1, k);
}  // end getNumberOfGroups
```

Function for recursive solution.

Choosing *k* Out of *n* Things

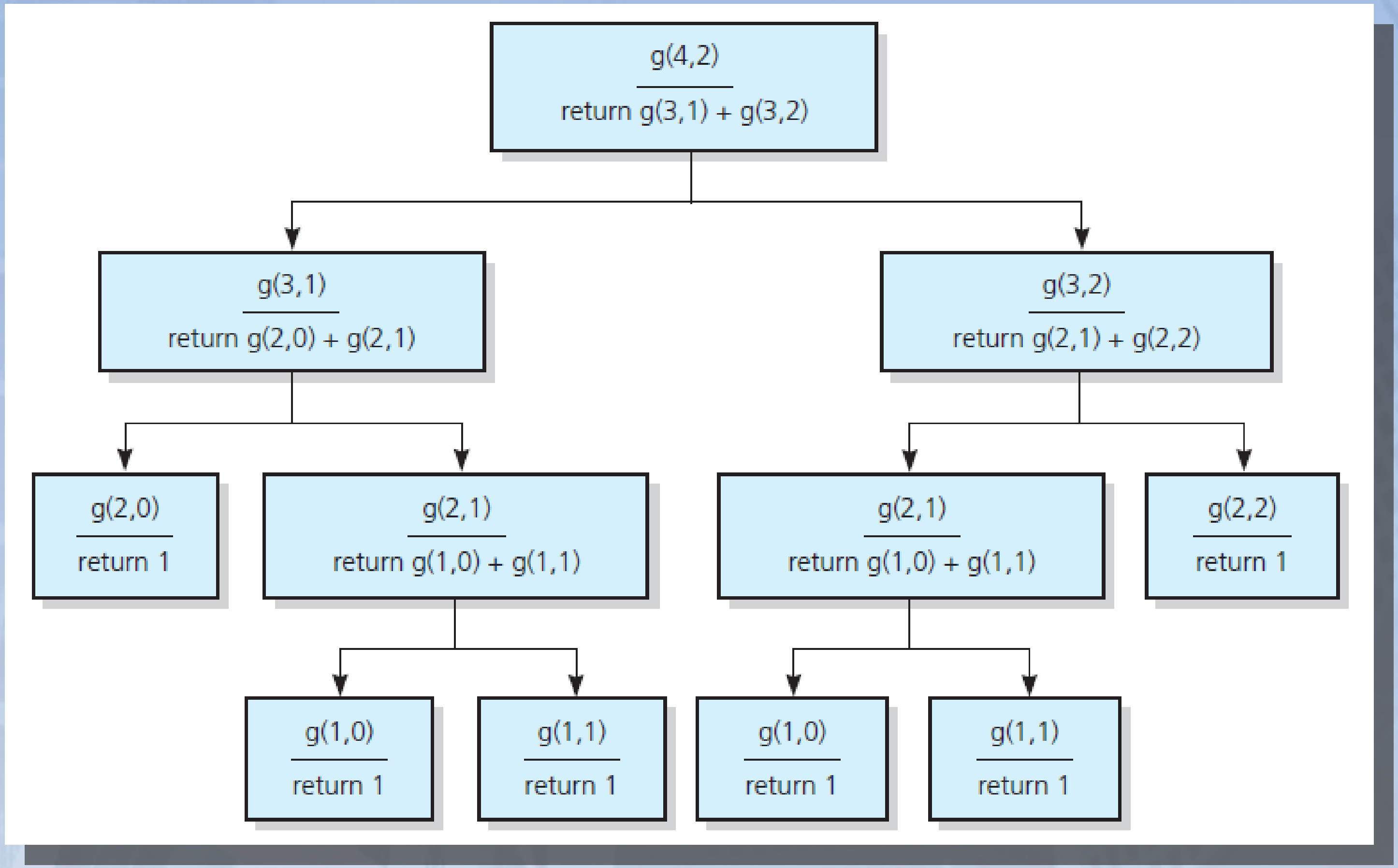


FIGURE 2-20 The recursive calls that g (4, 2) generates

Recursion and Efficiency

- Factors that contribute to inefficiency
  - Overhead associated with function calls
  - Some recursive algorithms inherently inefficient
- Keep in mind
  - Recursion can clarify complex solutions … but …
  - Clear, efficient iterative solution may be better

End

Chapter 2