Recursion

Mirza Mohammad Lutfe Elahi

Outline

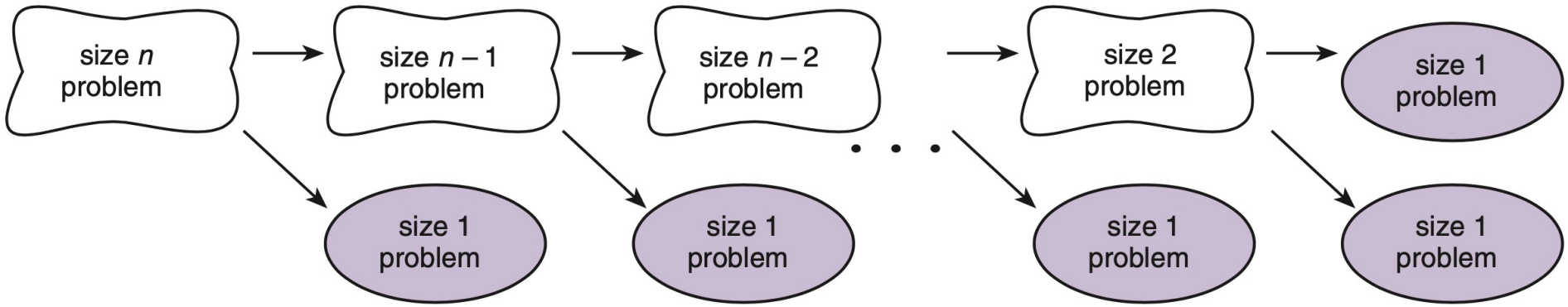
- To understand how recursion is used as a problem solving tool
- To learn how to write and trace recursive function
- To learn how to implement mathematical functions with recursive definition as C functions

Nature of Recursion

- One or more simple cases of the problem have a straightforward, nonrecursive solution
- The other cases can be redefined in terms of problems that are closer to the simple cases.
- By applying this redefinition process every time the recursive function is called, eventually the problem is reduced entirely to simple cases, which are relatively easy to solve.

Recursive Algorithm

if this is a simple case
    solve it
else
    redefine the problem using recursion



Problem of Multiplying 6 by 3

1. Multiply 6 by 2
2. Add 6 to the result of problem 1

1. Multiply 6 by 2
   1.1 Multiply 6 by 1
   1.2 Add 6 to the result of problem 1.1
2. Add 6 to the result of problem 1

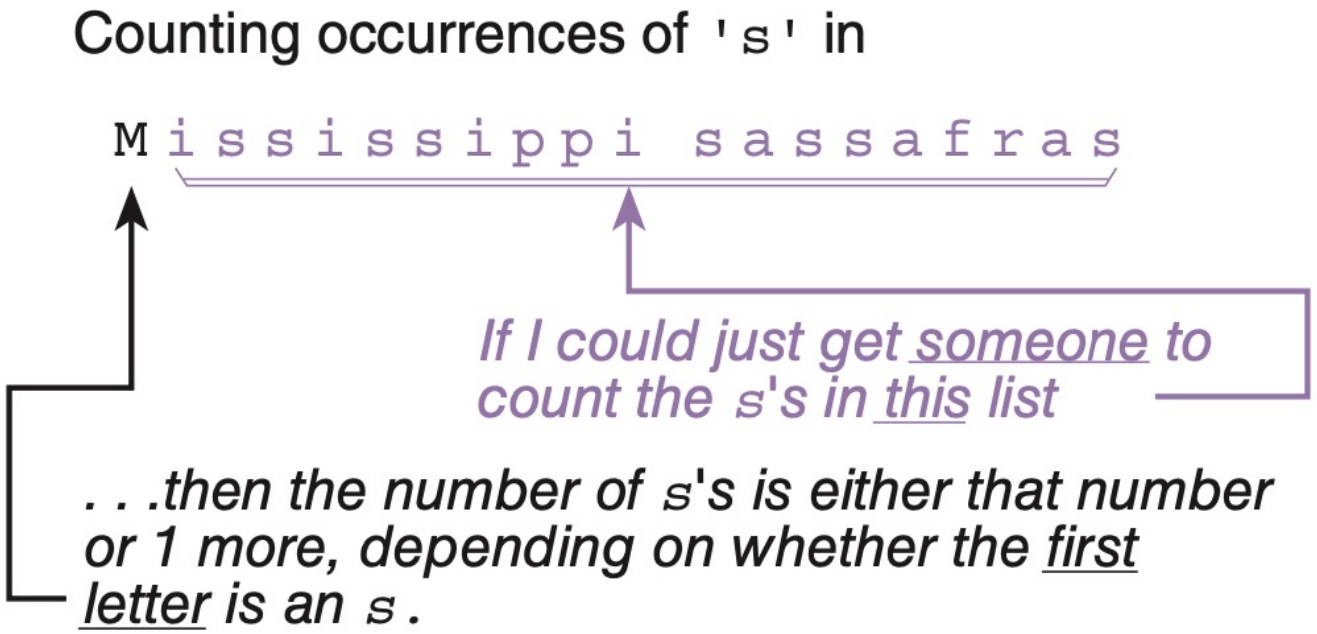
Problem of Multiplying 6 by 3

FIGURE 9.2 Recursive Function Multiply

```
1.  /*
2.   * Performs integer multiplication using + operator.
3.   * Pre:   m and n are defined and n > 0
4.   * Post:  returns m * n
5.   */
6.  int
7.  multiply(int m, int n)
8.  {
9.      int ans;
10.
11.     if (n == 1)
12.         ans = m;     /* simple case */
13.     else
14.         ans = m + multiply(m, n - 1); /* recursive step */
15.
16.     return (ans);
17. }
```

Count Character Appearance

- Develop a function to count the no. of times a particular character appears in a string



Problem of Multiplying 6 by 3

FIGURE 9.4 Counting Occurrences of a Character in a String

```
1.  /*
2.   * Counting occurrences of a letter in a string.
3.   */
4.
5.  #include <stdio.h>
6.
7.  int count(char ch, const char *str);
8.
9.  int
10. main(void)
11. {
12.       char str[80];         /* string to be processed */
13.       char target;          /* character counted */
14.       int my_count;
15.
16.       printf("Enter up to 79 characters.\n");
17.       gets(str);            /* read in the string */
18.
19.       printf("Enter the character you want to count: ");
20.       scanf("%c", &target);
21.
22.       my_count = count(target, str);
23.       printf("The number of occurrences of %c in\n\"%s\"\nis %d\n",
24.              target, str, my_count);
25.
```

FIGURE 9.4 (continued)

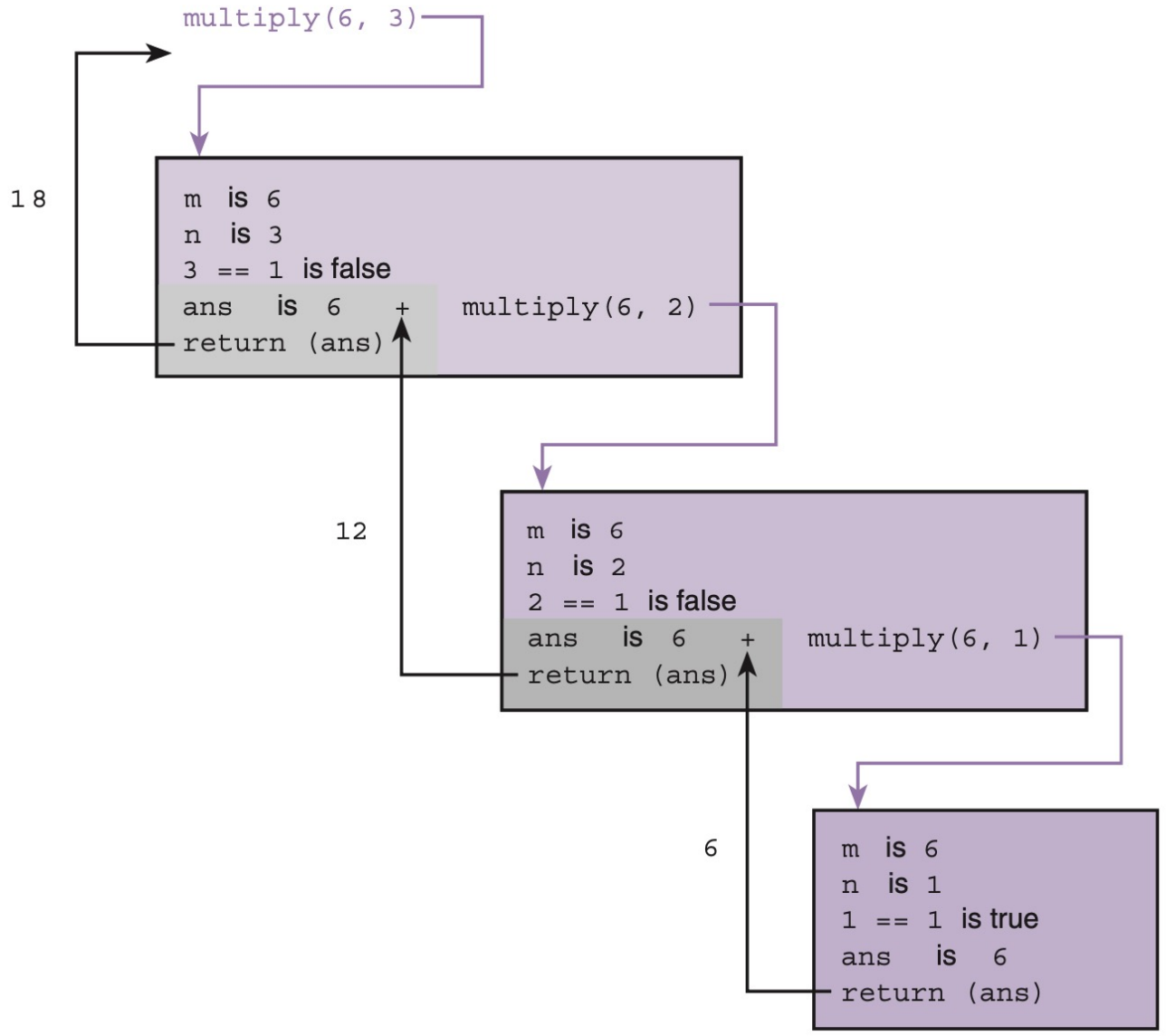
```
26.      return (0);
27.  }
28.
29.  /*
30.   * Counts the number of times ch occurs in string str.
31.   * Pre:  Letter ch and string str are defined.
32.   */
33.  int
34.  count(char ch, const char *str)
35.  {
36.        int ans;
37.
38.        if (str[0] == '\0')                          /* simple case */
39.            ans = 0;
40.        else                    /* redefine problem using recursion */
41.           if (ch == str[0]) /* first character must be counted */
42.               ans = 1 + count(ch, &str[1]);
43.           else                  /* first character is not counted */
44.               ans = count(ch, &str[1]);
45.
46.        return (ans);
47.  }
48.

Enter up to 79 characters.
this is the string I am testing
Enter the character you want to count: t
The number of occurrences of t in
"this is the string I am testing" is 5
```

Tracing a Recursive Function

- Hand tracing an algorithm's execution provides valuable insight into how that algorithm works
- Recursive function that returns a value
- Recursive function that returns no value

Tracing a Recursive Function - Returns



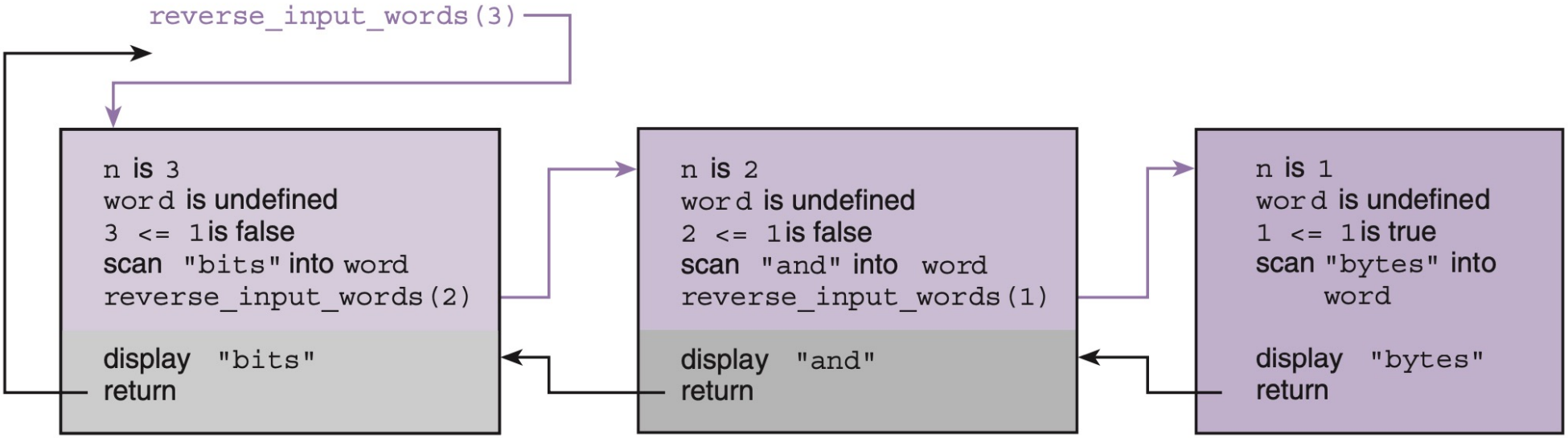
Tracing a Recursive Function - Void

FIGURE 9.6 Function reverse_input_words

```
1.  /*
2.   * Take n words as input and print them in reverse order on separate lines.
3.   * Pre: n > 0
4.   */
5.  void
6.  reverse_input_words(int n)
7.  {
8.        char word[WORDSIZ]; /* local variable for storing one word          */
9.
10.     if (n <= 1) { /* simple case: just one word to get and print          */
11.
12.           scanf("%s", word);
13.           printf("%s\n", word);
14.
15.     } else { /* get this word; get and print the rest of the words in
16.                 reverse order; then print this word                       */
17.
18.           scanf("%s", word);
19.           reverse_input_words(n - 1);
20.           printf("%s\n", word);
21.     }
22. }
```

Tracing a Recursive Function - Void

- Trace of **reverse_input_words(3)** When the Words Entered Are "bits" "and" "bytes"



Parameter and Local variable stacks

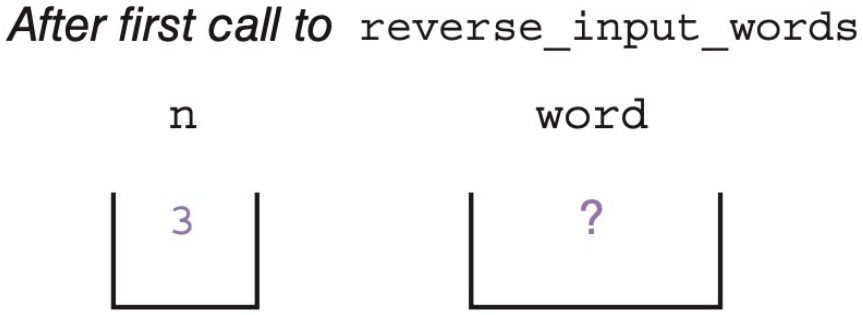
- C uses the stack data structure to track recursive calls
- In this data structure
  - we add data items (the push operation) and
  - remove them (the pop operation) from the same end of the list, so the last item stored is the first processed.

Executing a Call to reverse_input_words

- System pushes the parameter value associated with the call on top of the parameter stack, and pushes a new undefined cell on top of the stack maintained for the local variable word.
- A return from reverse_input_words pops each stack, removing the top value.

Executing a Call to reverse_input_words

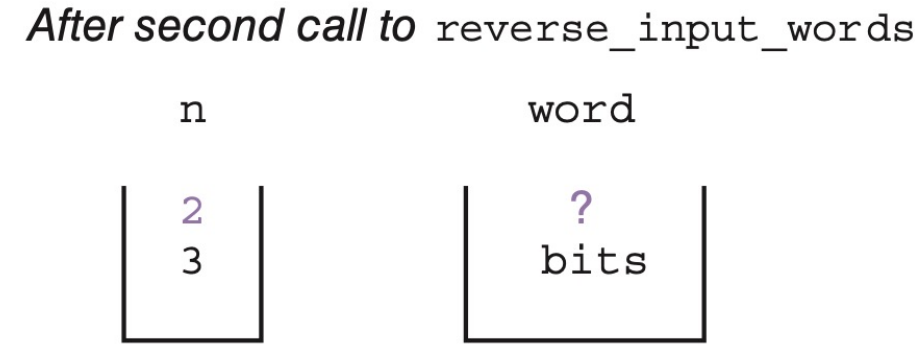
- two stacks right after the first call to reverse_input_words.



- The word "bits" is stored in word just before the second call to reverse_input_ words.

Executing a Call to reverse_input_words

- After the second call to reverse_input_words, the number 2 is pushed on the stack for n,

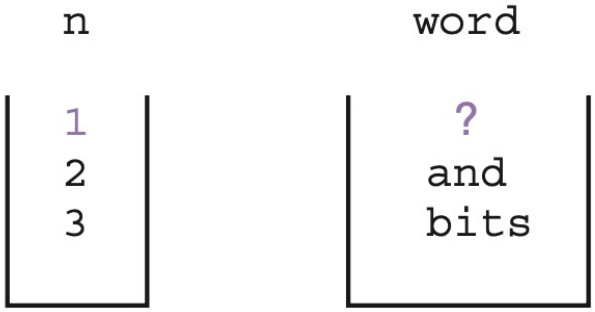


- The word "and" is scanned and stored in word just before the third call to reverse_input_words.

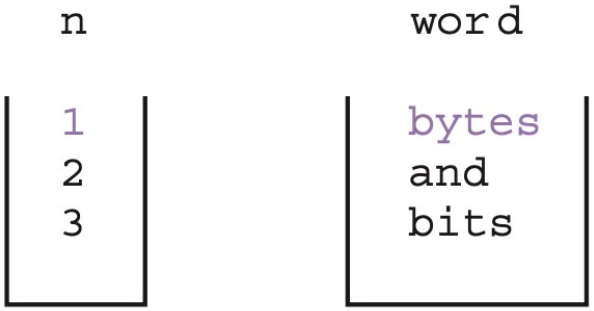
Executing a Call to reverse_input_words

- word becomes undefined again right after the third call.

After third call to `reverse_input_words`



- the word "bytes" is scanned and stored in word, and "bytes" is echo printed immediately because n is 1 (a simple case).



Executing a Call to reverse_input_words

- The function return pops both stacks

After first return

After second return

Trace Recursive Functions - When & How

- Doing a trace by hand of multiple calls to a recursive function is helpful
- Less useful when trying to develop a recursive algorithm
- Best to trace a specific case simply
- Hand trace can correct result

Trace Recursive Functions – Self-tracing

FIGURE 9.9 Recursive Function multiply with Print Statements to Create Tra from multiply(8, 3)

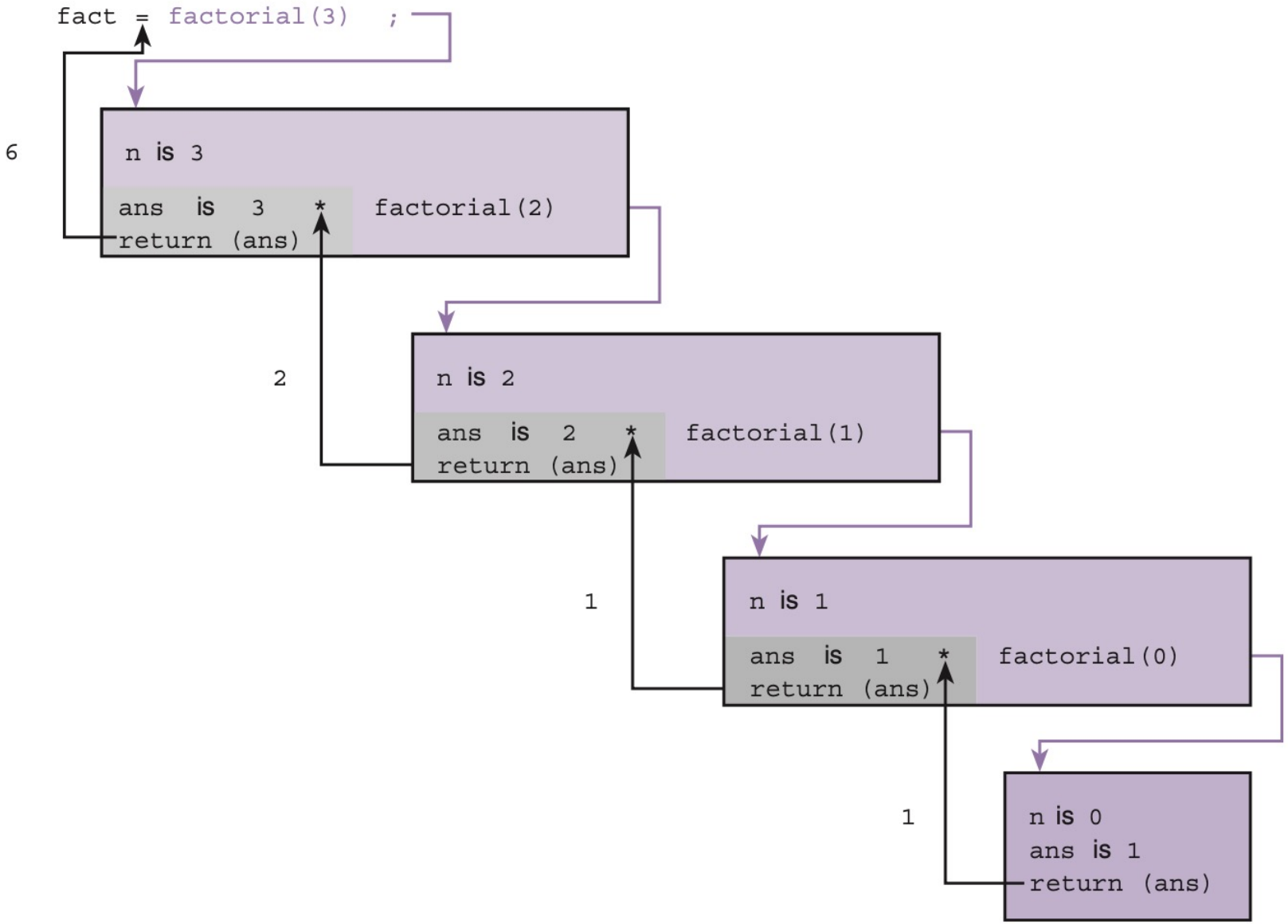
```
1.  /*
2.   * *** Includes calls to printf to trace execution ***
3.   * Performs integer multiplication using + operator.
4.   * Pre: m and n are defined and n > 0
5.   * Post: returns m * n
6.   */
7.  int
8.  multiply(int m, int n)
9.  {
10.       int ans;
11.
12.    printf("Entering multiply with m = %d, n = %d\n", m, n);
13.
14.       if (n == 1)
15.             ans = m; /* simple case */
16.       else
17.             ans = m + multiply(m, n - 1); /* recursive step */
18.    printf("multiply(%d, %d) returning %d\n", m, n, ans);
19.
20.       return (ans);
21. }
22.
23. Entering multiply with m = 8, n = 3
24. Entering multiply with m = 8, n = 2
25. Entering multiply with m = 8, n = 1
26. multiply(8, 1) returning 8
27. multiply(8, 2) returning 16
28. multiply(8, 3) returning 24
```

Recursive Mathematical Functions

FIGURE 9.10 Recursive Factorial Function

```
1.  /*
2.   * Compute n! using a recursive definition
3.   * Pre: n >= 0
4.   */
5.  int
6.  factorial(int n)
7.  {
8.        int ans;
9.
10.       if (n == 0)
11.             ans = 1;
12.       else
13.             ans = n * factorial(n - 1);
14.
15.       return (ans);
16. }
```

Recursive Mathematical Functions



Recursive Mathematical Functions

FIGURE 9.13 Recursive Function fibonacci

```
1.  /*
2.   * Computes the nth Fibonacci number
3.   * Pre: n > 0
4.   */
5.  int
6.  fibonacci(int n)
7.  {
8.        int ans;
9.
10.       if (n == 1 || n == 2)
11.             ans = 1;
12.       else
13.             ans = fibonacci(n - 2) + fibonacci(n - 1);
14.
15.       return (ans);
16. }
```

Recursive Mathematical Functions

FIGURE 9.14 Program Using Recursive Function gcd

```
1.  /*
2.   * Displays the greatest common divisor of two integers
3.   */
4.
5.  #include <stdio.h>
6.
7.  /*
8.   * Finds the greatest common divisor of m and n
9.   * Pre: m and n are both > 0
10.  */
11. int
12. gcd(int m, int n)
13. {
14.         int ans;
15.
16.         if (m % n == 0)
17.                 ans = n;
18.         else
19.                 ans = gcd(n, m % n);
20.
21.         return (ans);
22. }
```

Recursive Mathematical Functions

```
23. int
24. main(void)
25. {
26.         int n1, n2;
27.
28.         printf("Enter two positive integers separated by a space> ");
29.         scanf("%d%d", &n1, &n2);
30.         printf("Their greatest common divisor is %d\n", gcd(n1, n2));
31.
32.         return (0);
33. }
34.
35. Enter two positive integers separated by a space> 24 84
36. Their greatest common divisor is 12
```