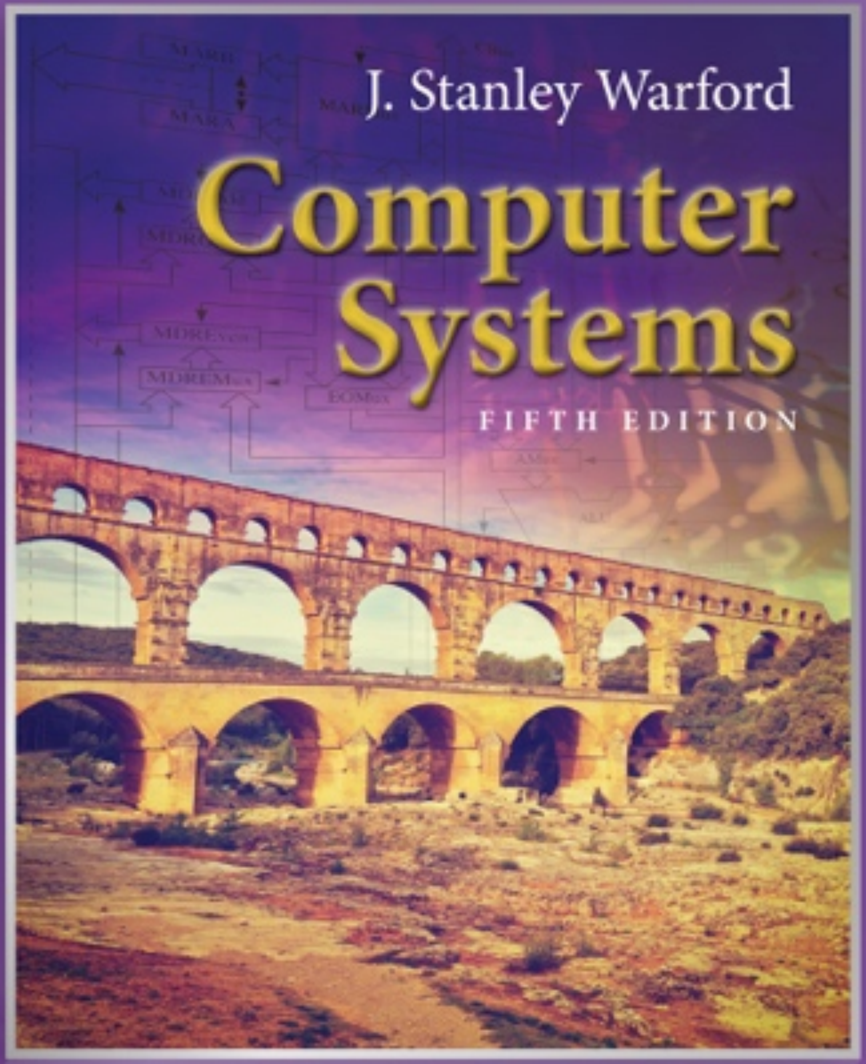
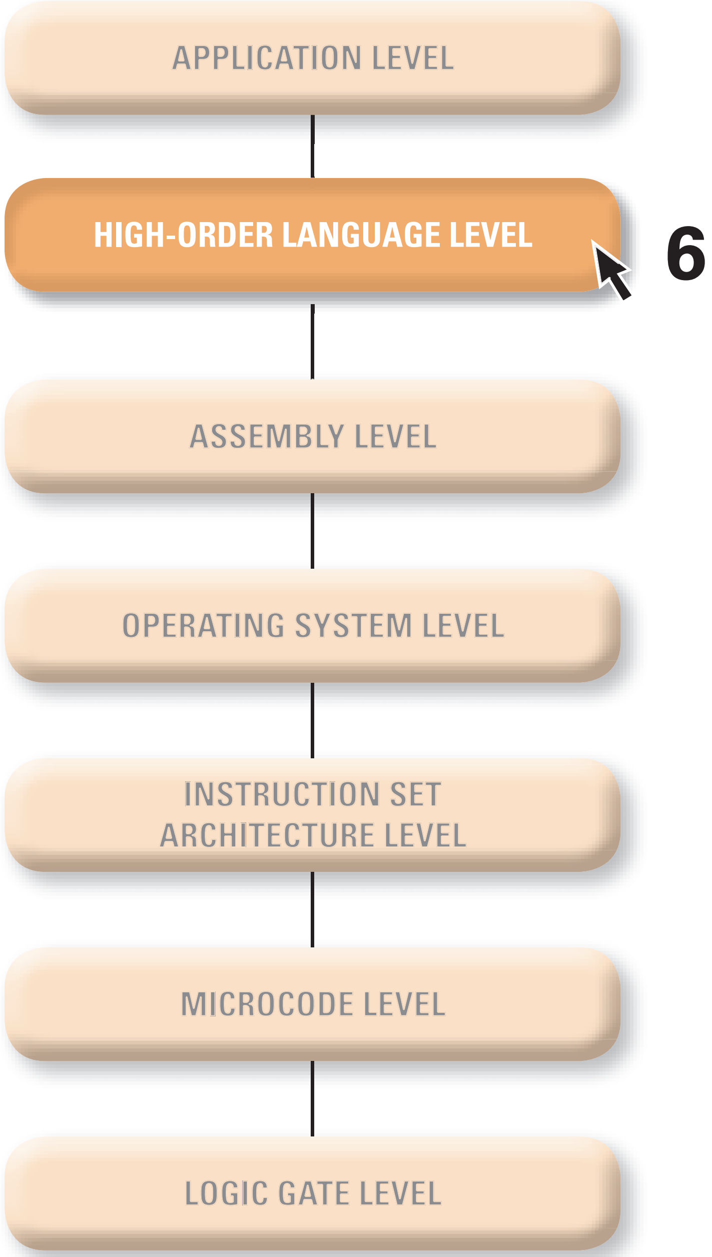
Chapter 2

C



High-Order Language



Computer Systems FIFTH EDITION

Figure 2.1

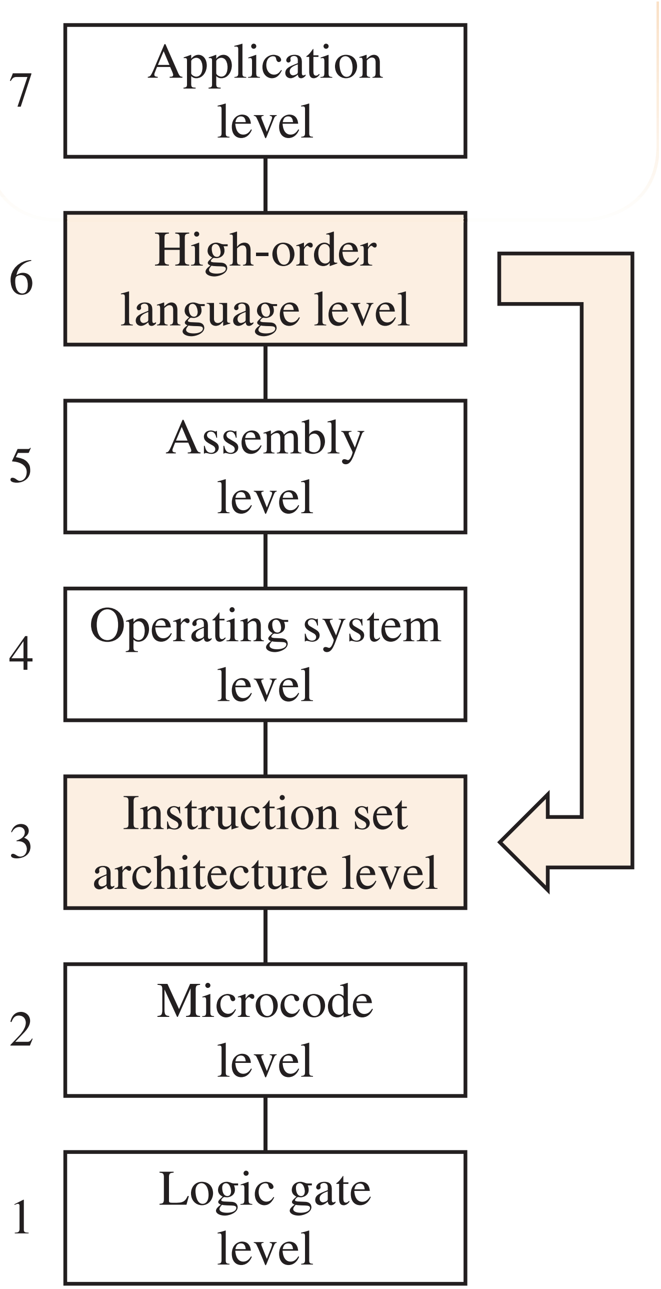


Figure 2.2

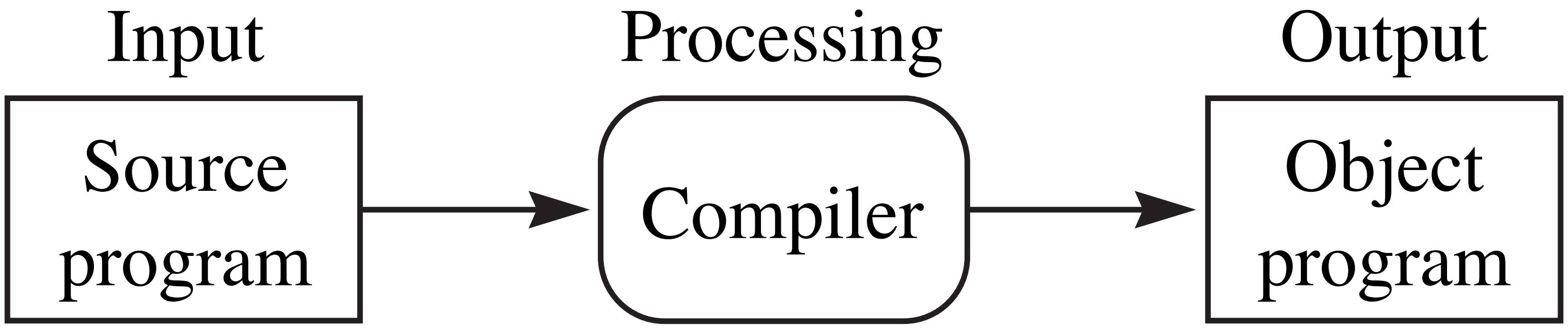
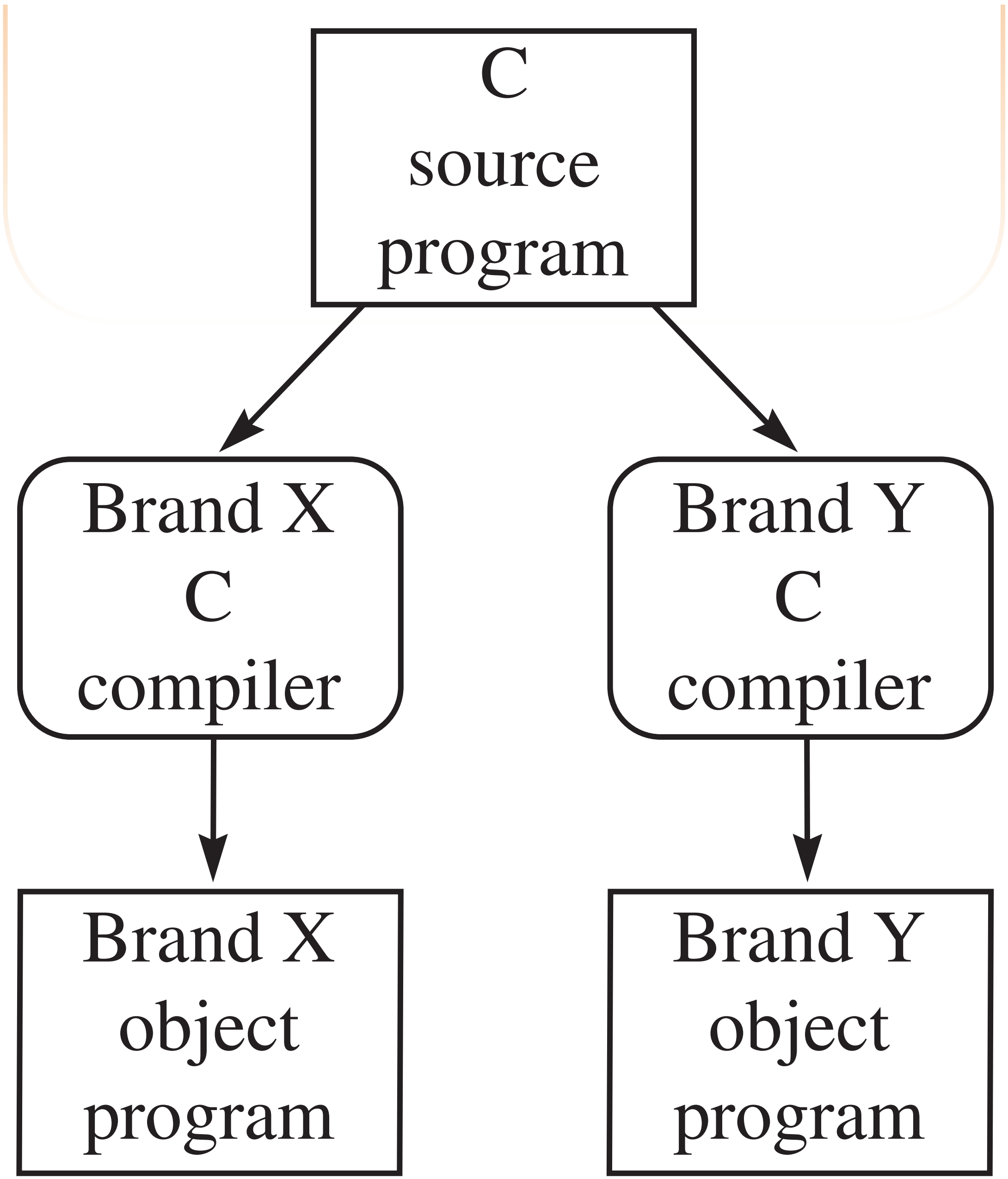

Input
Processing
Output
Source program
Compiler
Object program

Figure 2.3


C source program
Brand X C compiler
Brand Y C compiler
Brand X object program
Brand Y object program

The C
memory model

The C memory model

- Global variables – fixed location in memory

The C memory model

- Global variables – fixed location in memory
- Local variables and parameters – run-time stack

The C
memory model

- Global variables – fixed location in memory
- Local variables and parameters – run-time stack
- Dynamically allocated variables – heap

Function call

Function call

- Push storage for the return value

Function call

- Push storage for the return value
- Push the actual parameters

Function call

- Push storage for the return value
- Push the actual parameters
- Push the return address

Function call

- Push storage for the return value
- Push the actual parameters
- Push the return address
- Push storage for the local variables

Function return

- Pop the local variables
- Pop the return address
- Pop the parameters
- Pop the return value

Three attributes of a C variable

- Name
- Type
- Value

Figure 2.4

```
// Stan Warford
// A nonsense program to illustrate global variables.

#include <stdio.h>

char ch;
int j;

int main() {
   scanf("%c %d", &ch, &j);
   j += 5;
   ch++;
   printf("%c\n%d\n", ch, j);
   return 0;
}
```

Input

```
M 419
```

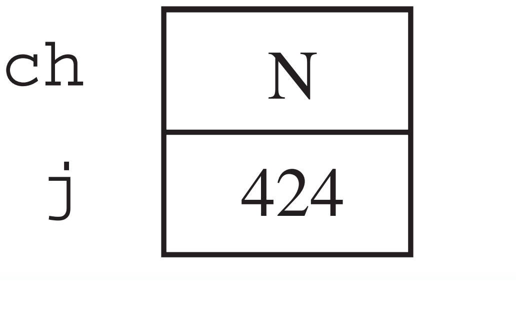
Output

```
N
424
```

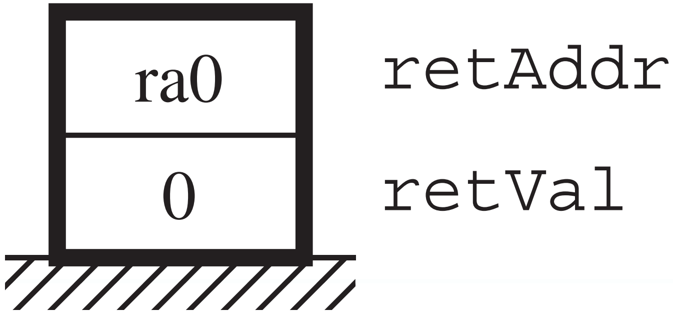
Variables

- Global – Declared outside of `main()`
- Local – Declared within `main()`

Figure 2.5



(**a**) Fixed location.



(**b**) Run-time stack.

```c
#include <stdio.h>

int main() {
   const int bonus = 10;
   int exam1;
   int exam2;
   int score;
   scanf("%d %d", &exam1, &exam2);
   score = (exam1 + exam2) / 2 + bonus;
   printf("score = %d\n", score);
   return 0;
}
```

Input

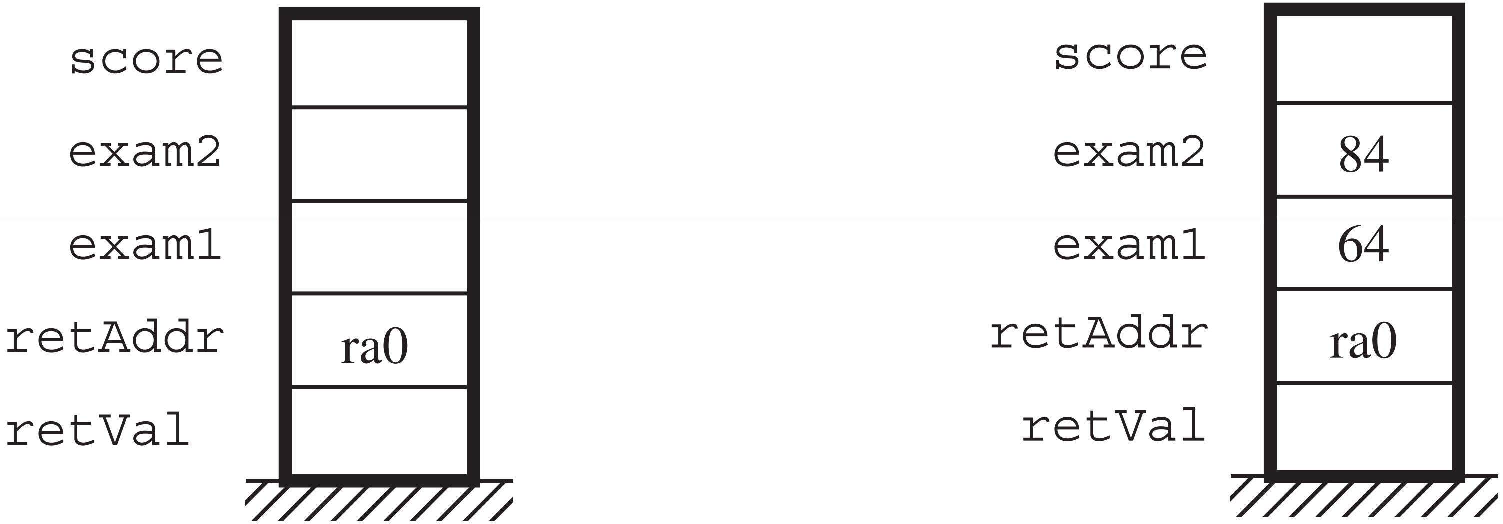
```
68 84
```

Output

```
score = 86
```

Expression	Value
15 / 3	5
14 / 3	4
13 / 3	4
12 / 3	4
11 / 3	3

Expression	Value
15 % 3	0
14 % 3	2
13 % 3	1
12 % 3	0
11 % 3	2



(a) Before the input statement executes.

(b) After the input statement executes.

Figure 2.9

Operator	Meaning
==	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
!=	Not equal to

Figure 2.10

```
#include <stdio.h>

int main() {
   const int limit = 100;
   int num;
   scanf("%d", &num);
   if (num >= limit) {
      printf("high\n");
   }
   else {
      printf("low\n");
   }
   return 0;
}
```

Input

```
75
```

Output

```
low
```

Figure 2.11

Symbol	Meaning
&&	AND
||	OR
!	NOT

Figure 2.12

```
#include <stdio.h>

int main() {
   int guess;
   printf("Pick a number 0..3: ");
   scanf("%d", &guess);
   switch (guess) {
      case 0: printf("Not close\n"); break;
      case 1: printf("Close\n"); break;
      case 2: printf("Right on\n"); break;
      case 3: printf("Too high\n");
   }
   return 0;
}
```

Interactive Input/Output

```
Pick a number 0..3: 1
Close
```

```
#include <stdio.h>

char letter;

int main() {
   scanf("%c", &letter);
   while (letter != '*') {
      if (letter == ' ') {
         printf("\n");
      }
      else {
         printf("%c", letter);
      }
      scanf("%c", &letter);
   }
   return 0;
}
```

Input

```
Hello, world!*
```

Output

```
Hello,
world!
```

Figure 2.14

```
#include <stdio.h>

int cop;
int driver;

int main() {
   cop = 0;
   driver = 40;
   do {
      cop += 25;
      driver += 20;
   }
   while (cop < driver);
   printf("%d", cop);
   return 0;
}
```

Output

```
200
```

Figure 2.15

```
#include <stdio.h>

int vector[4];
int j;

int main() {
   for (j = 0; j < 4; j++) {
      scanf("%d", &vector[j]);
   }
   for (j = 3; j >= 0; j--) {
      printf("%d %d\n", j, vector[j]);
   }
   return 0;
}
```

Input

```
2  26 -3 9
```

Output

```
3 9
2 -3
1 26
0 2
```

Allocation process for a void function

- Push the actual parameters
- Push the return address
- Push storage for the local variables

Deallocation process for a void function

- Pop the local variables
- Pop the return address
- Pop the parameters

Figure 2.16

```
#include <stdio.h>

int numPts;
int value;
int j;

void printBar(int n) {
   int k;
   for (k = 1; k <= n; k++) {
      printf("*");
   }
   printf("\n");
}

int main() {
   scanf("%d", &numPts);
   for (j = 1; j <= numPts; j++) {
      scanf("%d", &value);
      printBar(value);
      //ra1
   }
   return 0;
}
```

Figure 2.16 (continued)

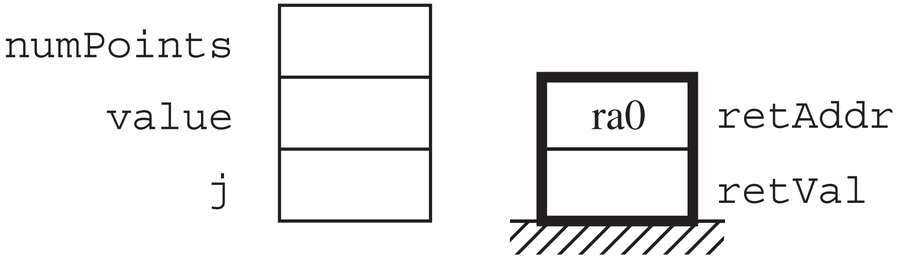
Input

```
12  3 13 17 34 27 23 25 29 16 10 0 2
```

Output

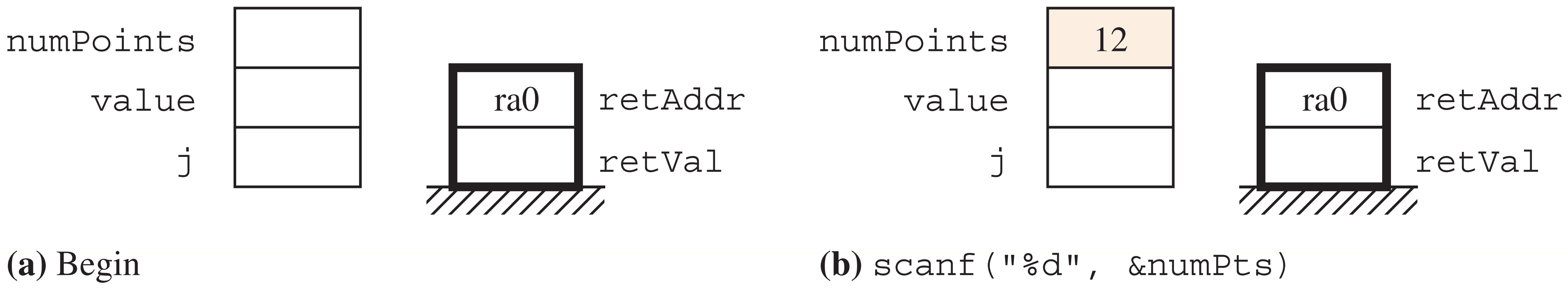
```
***
*************
*****************
**********************************
***************************
***********************
*************************
*****************************
****************
**********

**
```

Figure 2.17



(**a**) Begin

Figure 2.17

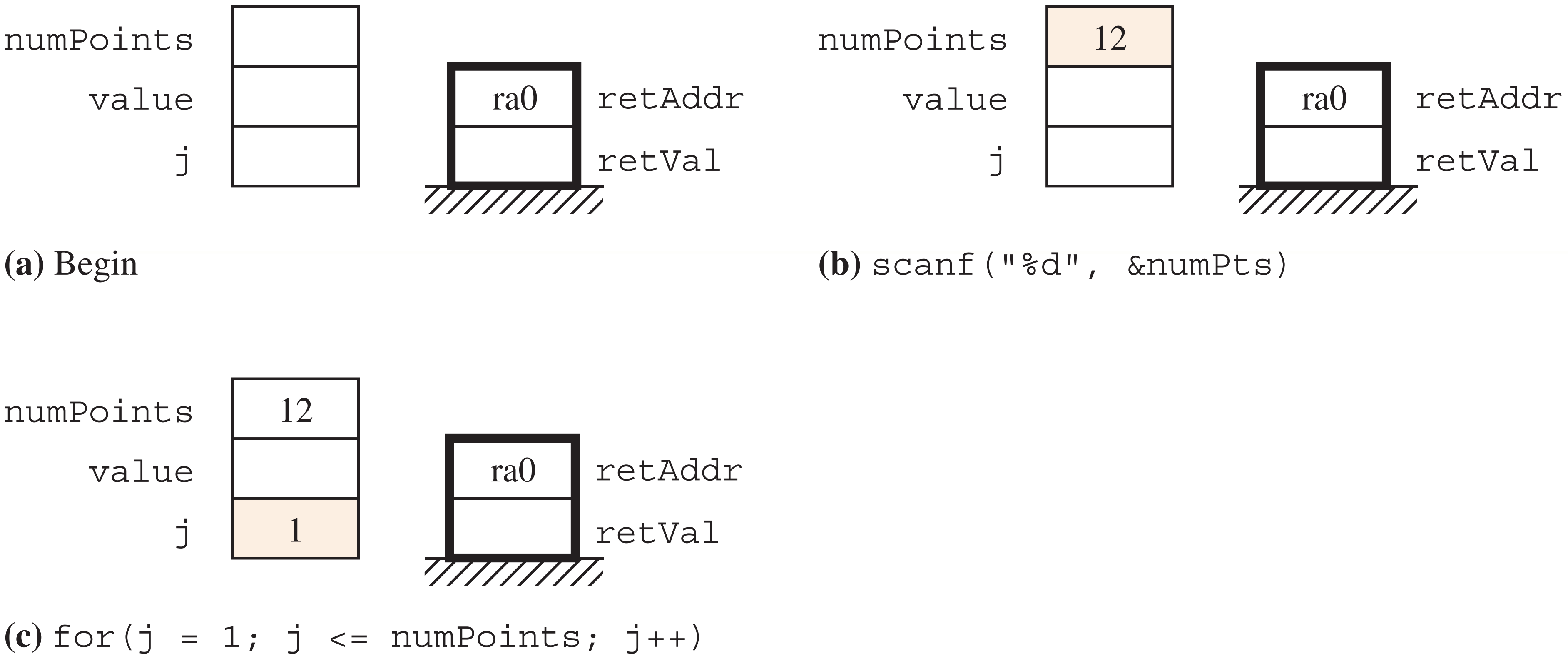


(a) Begin

(b) `scanf("%d", &numPts)`

y

Figure 2.17



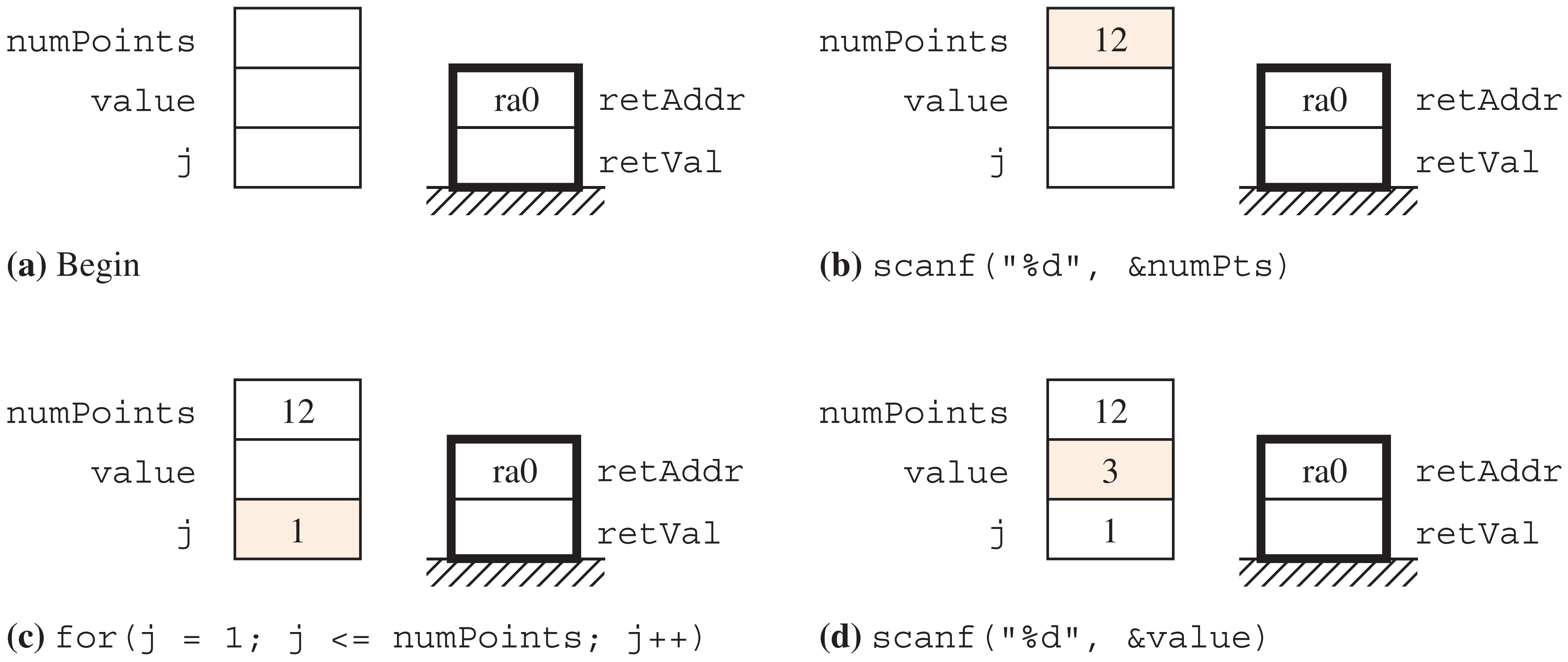
(a) Begin

(b) `scanf("%d", &numPts)`

(c) `for(j = 1; j <= numPoints; j++)`

Computer Systems FIFTH EDITION

Figure 2.17



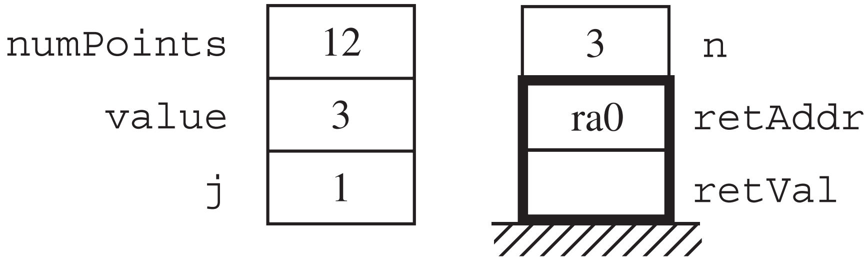
(a) Begin

(b) `scanf("%d", &numPts)`

(c) `for(j = 1; j <= numPoints; j++)`

(d) `scanf("%d", &value)`

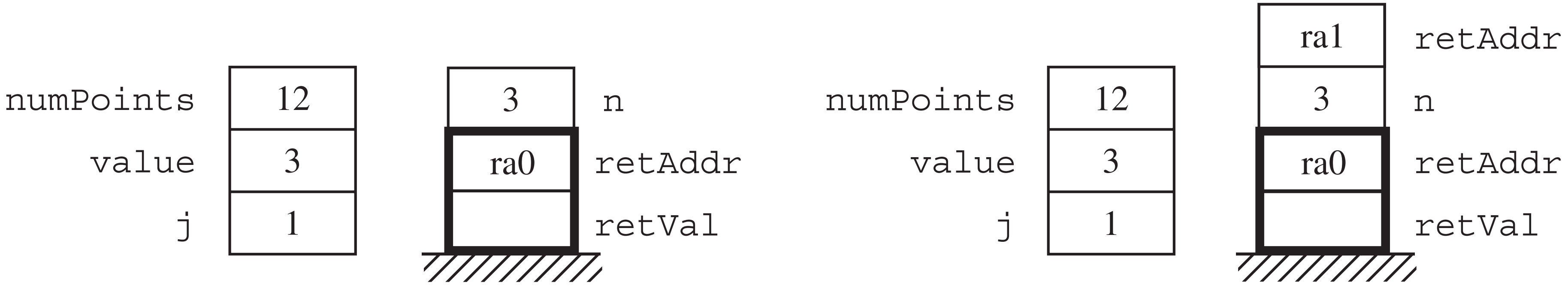
Figure 2.17 (continued)



(e) Push formal parameter

y

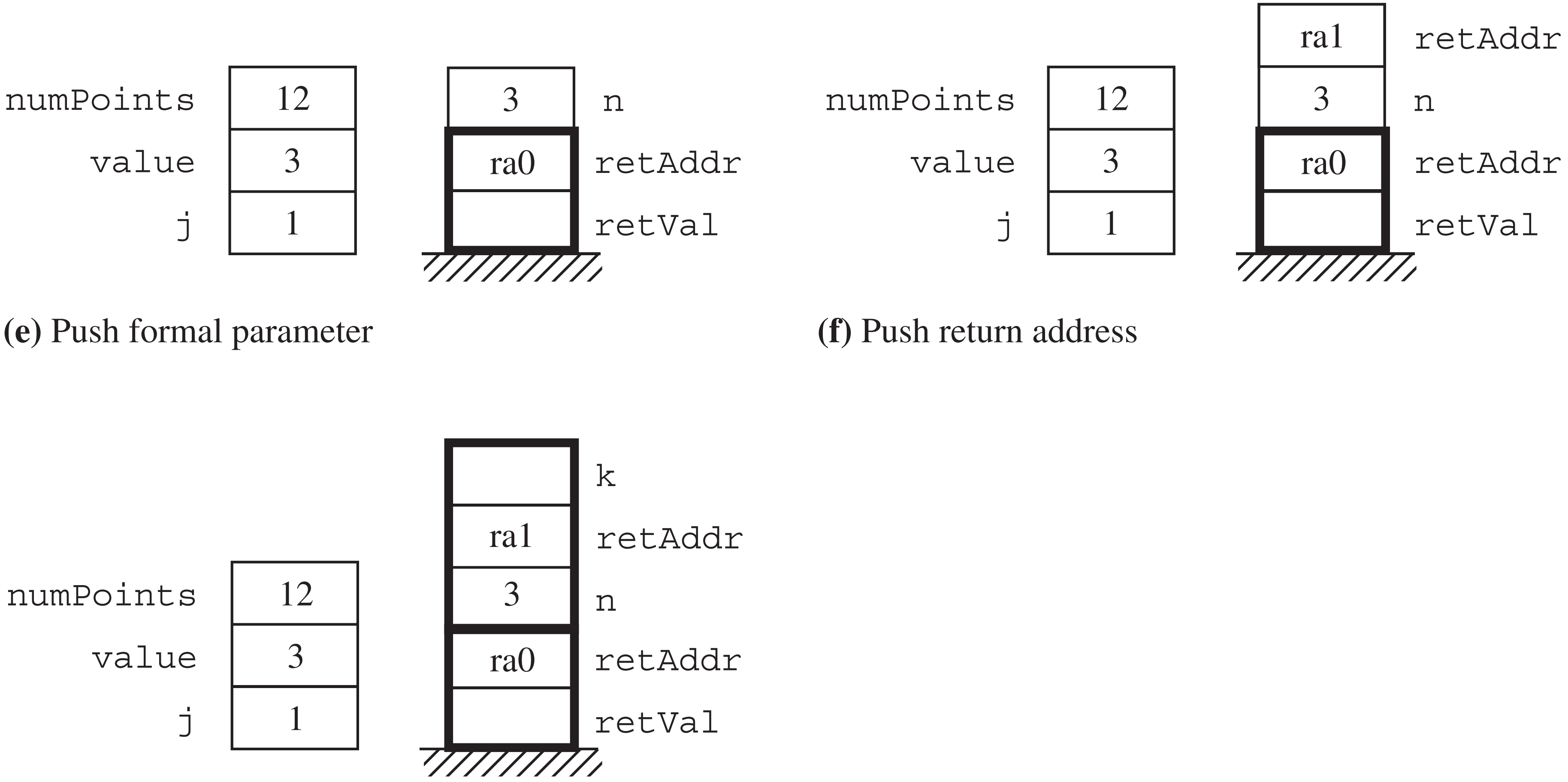
Figure 2.17 (continued)



(e) Push formal parameter

(f) Push return address

Figure 2.17 (continued)



(e) Push formal parameter

(f) Push return address

(g) Push storage for local variable k

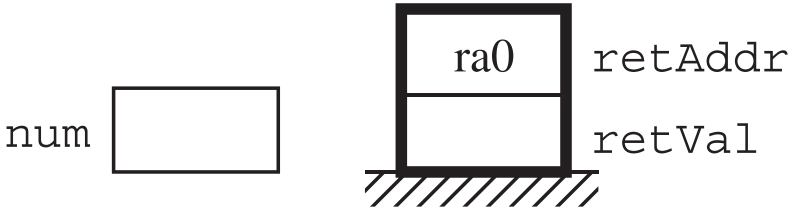
Figure 2.18

```
#include <stdio.h>

int num;

int fact(int n) {
   int f, j;
   f = 1;
   for (j = 1; j <= n; j++) {
      f *= j;
   }
   return f;
}

int main() {
   printf("Enter a small integer: ");
   scanf("%d", &num);
   printf("Its factorial is: %d\n", fact(num)); // ra1
   return 0;
}
```

Interactive Input/Output

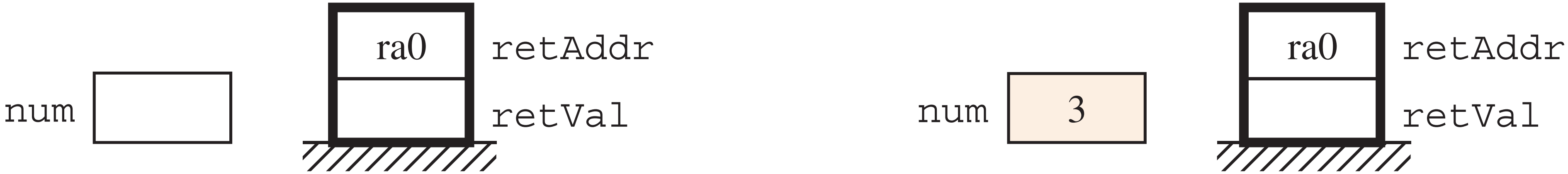
```
Enter a small integer: 3
Its factorial is: 6
```

Figure 2.19



(a) Begin

Figure 2.19

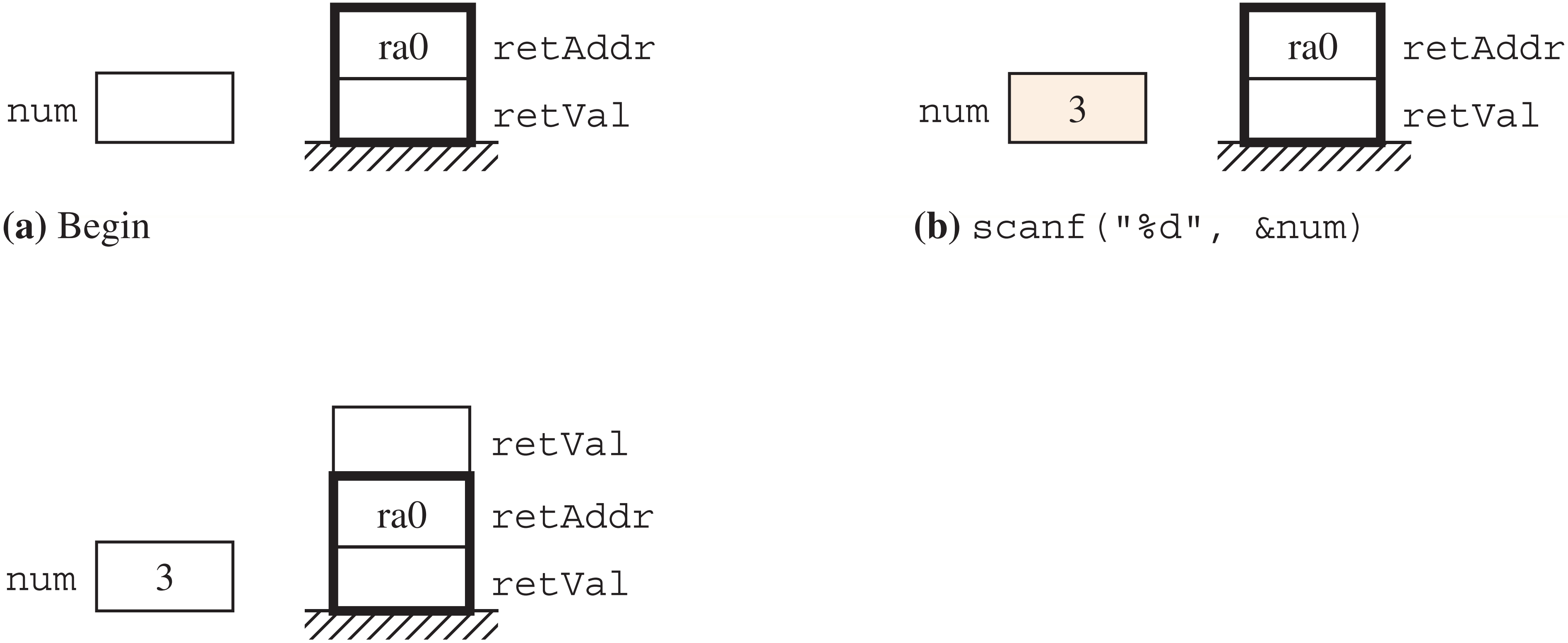


(a) Begin

(b) `scanf("%d", &num)`

Computer Systems FIFTH EDITION

Figure 2.19



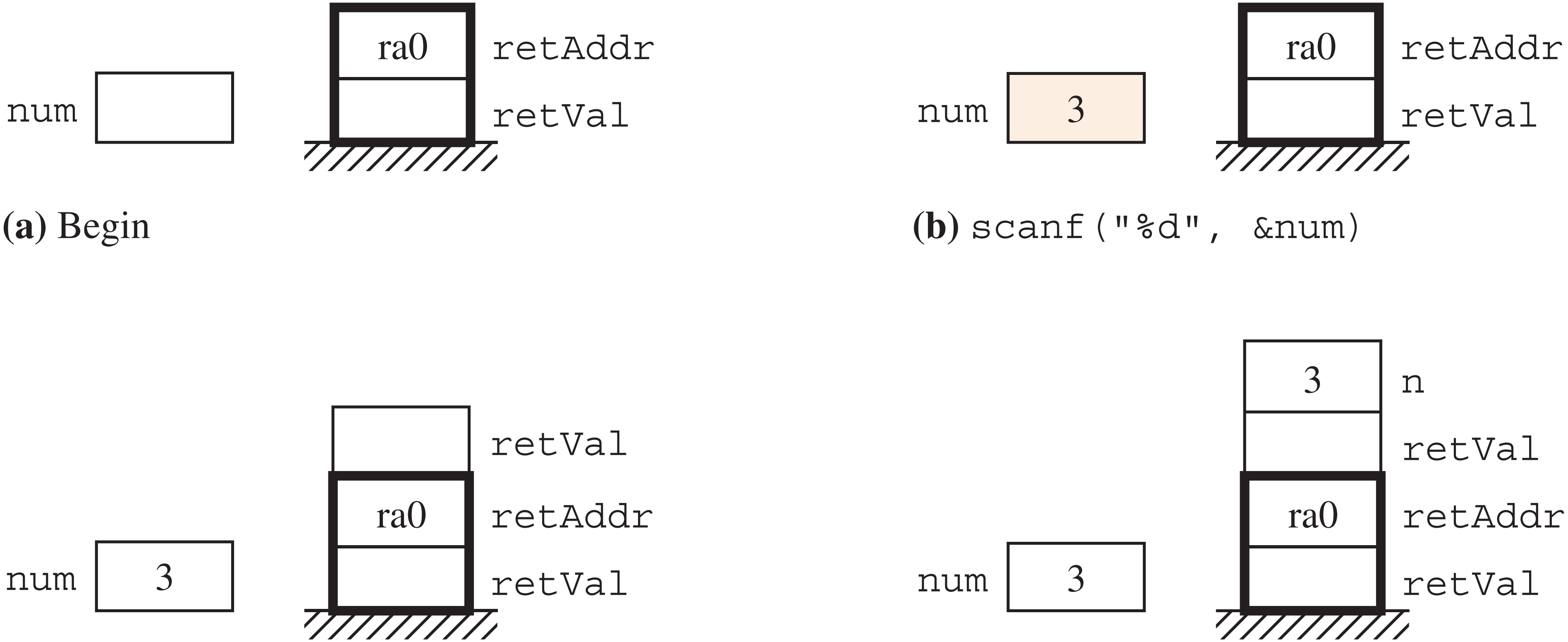
(a) Begin

(b) `scanf("%d", &num)`

(c) Push storage for return value `i`

y

Figure 2.19



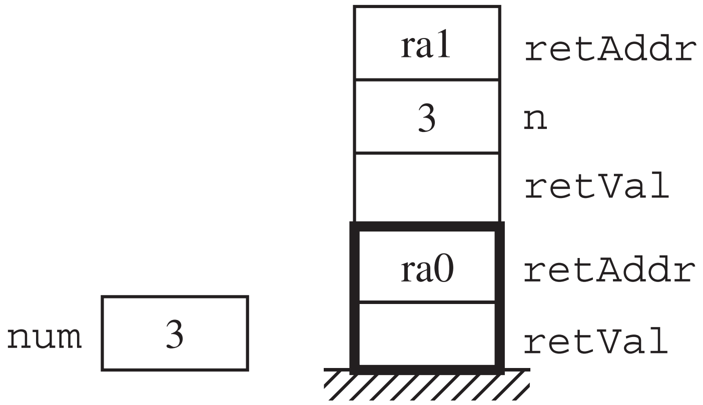
(a) Begin

(b) `scanf("%d", &num)`

(c) Push storage for return value `i`

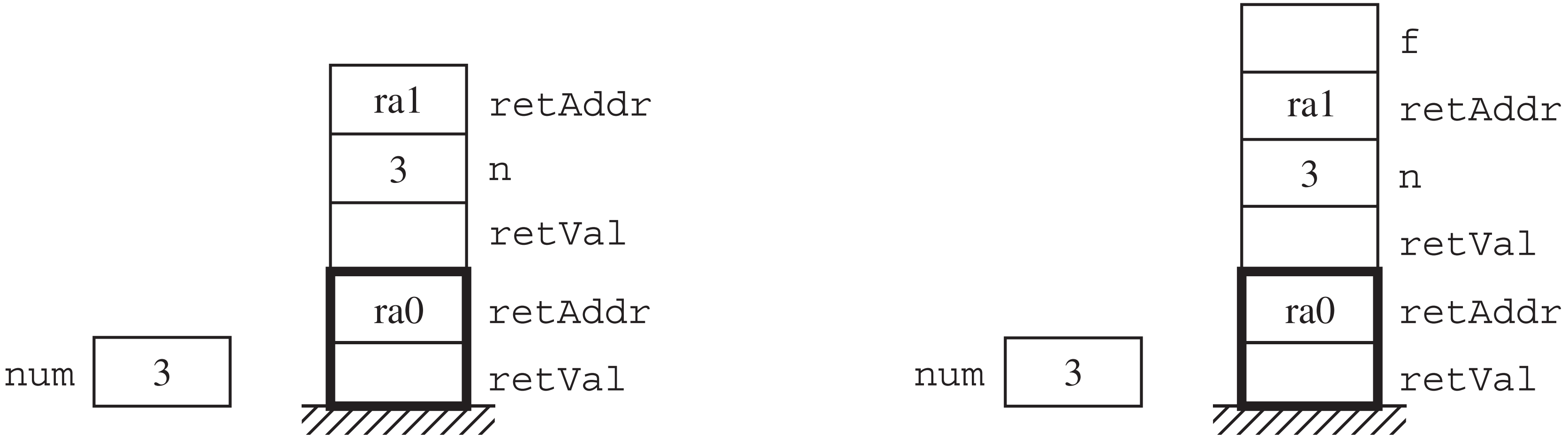
(d) Push actual parameter

Figure 2.19 (continued)



(e) Push return address

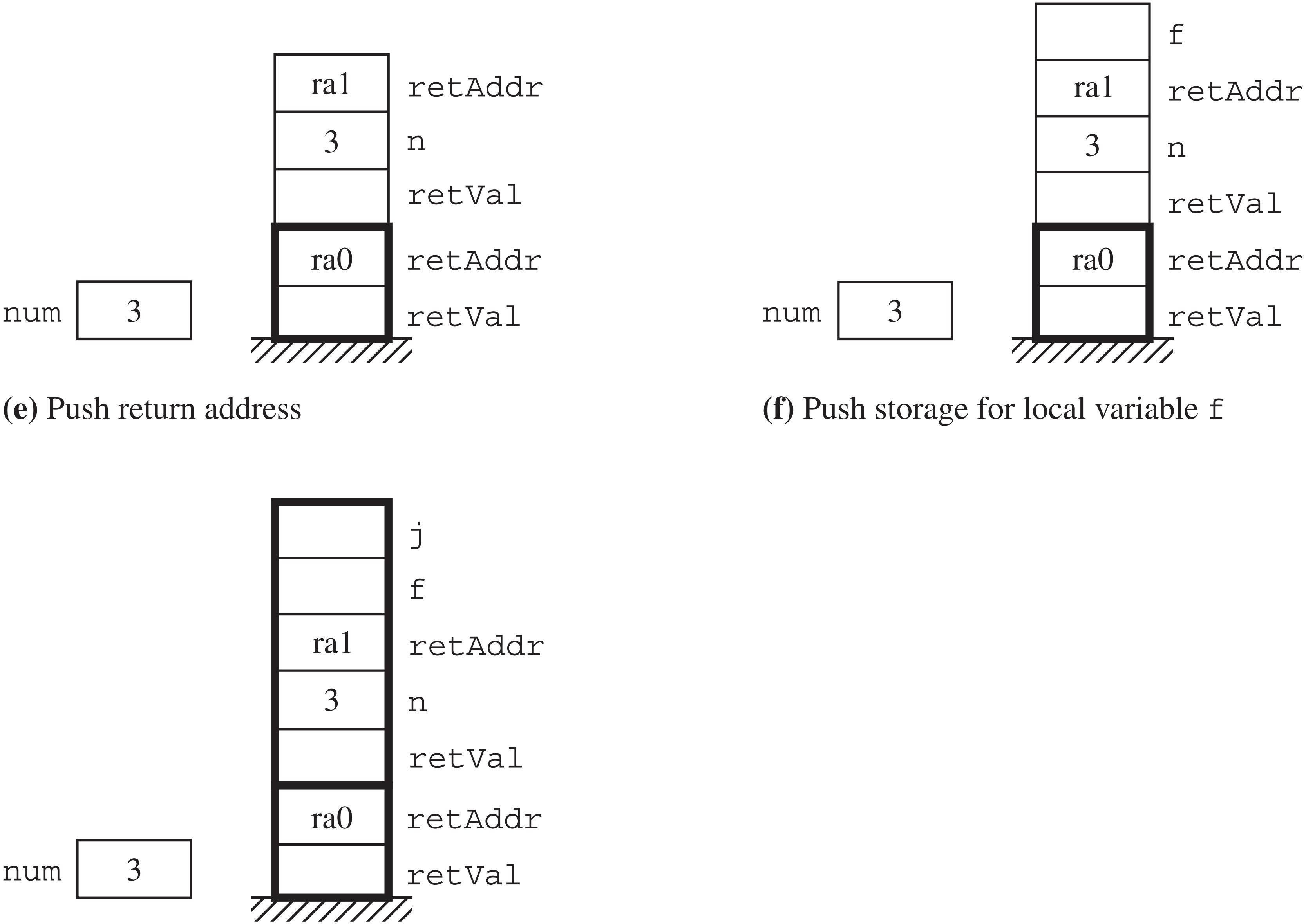
Figure 2.19 (continued)



(e) Push return address

(f) Push storage for local variable `f`

Figure 2.19 (continued)

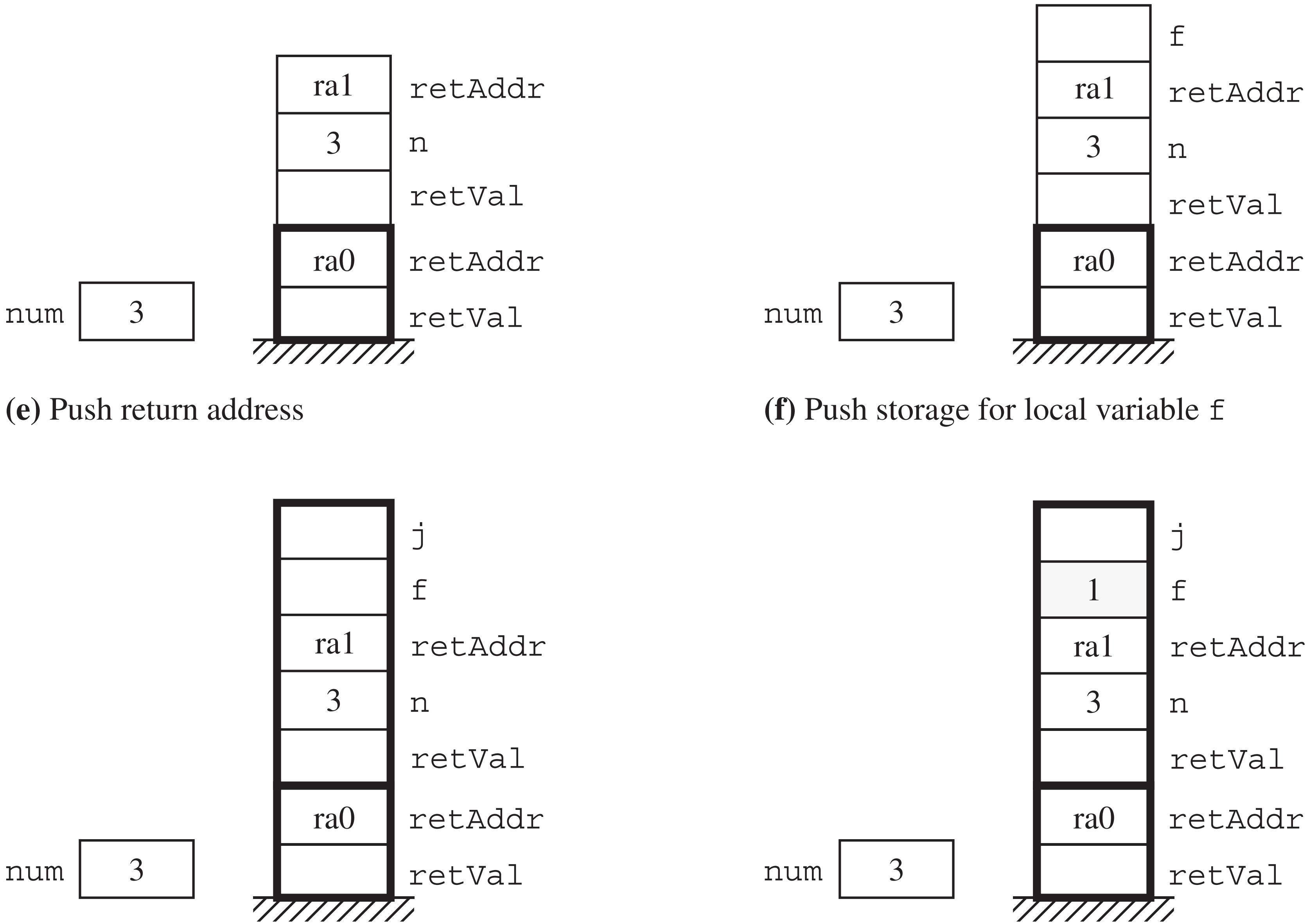


(e) Push return address

(f) Push storage for local variable f

(g) Push storage for local variable j

Figure 2.19 (continued)



(e) Push return address

(f) Push storage for local variable `f`

(g) Push storage for local variable `j`

(h) `f = 1`

Call by reference

- In call by *value*, the formal parameter gets the *value of* the actual parameter.
    - If the formal parameter changes, the actual parameter does *not* change.
- In call by *reference*, the formal parameter gets *a reference to* the actual parameter.
    - If the formal parameter changes, the actual parameter *does* change.

Figure 2.20

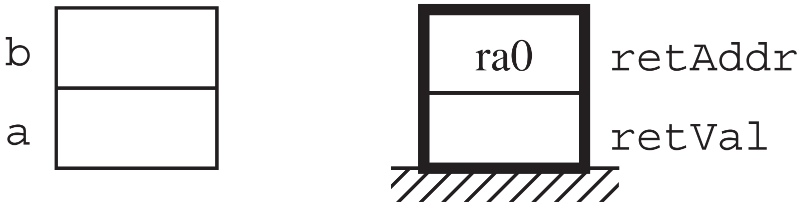
```
#include <stdio.h>

int a, b;

void swap(int *r, int *s) {
   int temp;
   temp = *r;
   *r = *s;
   *s = temp;
}

void order(int *x, int *y) {
   if (*x > *y) {
      swap (x, y);
   }  // ra2
}

int main() {
   printf("Enter an integer: ");
   scanf("%d", &a);
   printf("Enter an integer: ");
   scanf("%d", &b);
   order (&a, &b);
   printf("Ordered they are: %d, %d\n", a ,b); // ra1
   return 0;
}
```

Figure 2.20 (continued)

Interactive Input/Output

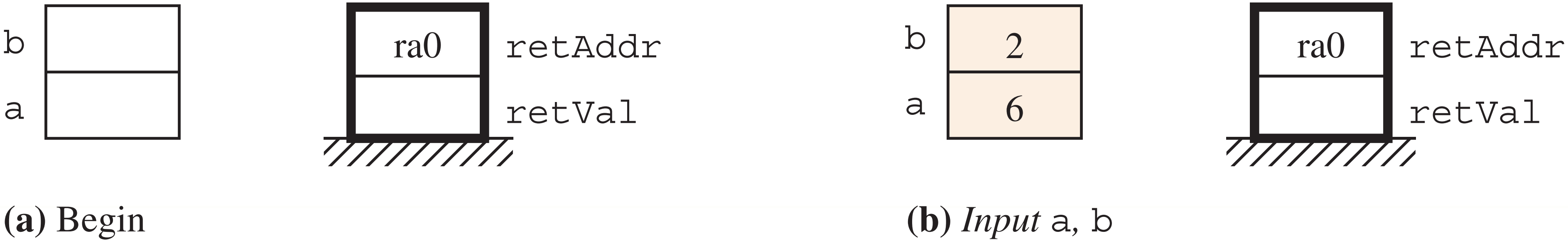
```
Enter an integer: 6
Enter an integer: 2
Ordered they are: 2, 6
```

Figure 2.21



(a) Begin

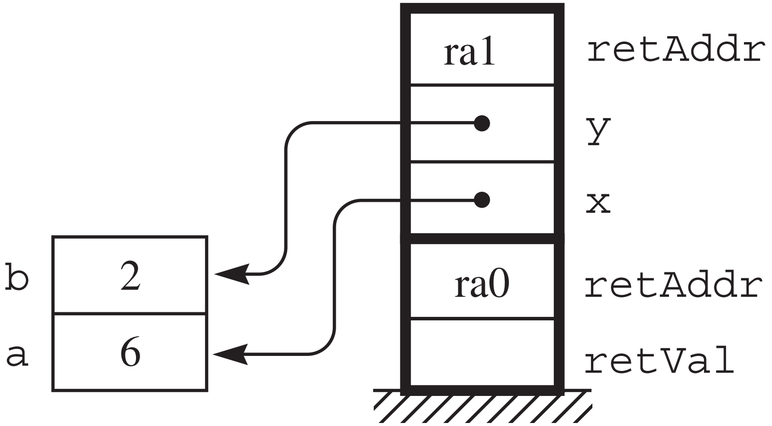
Figure 2.21



(a) Begin

(b) *Input* a, b

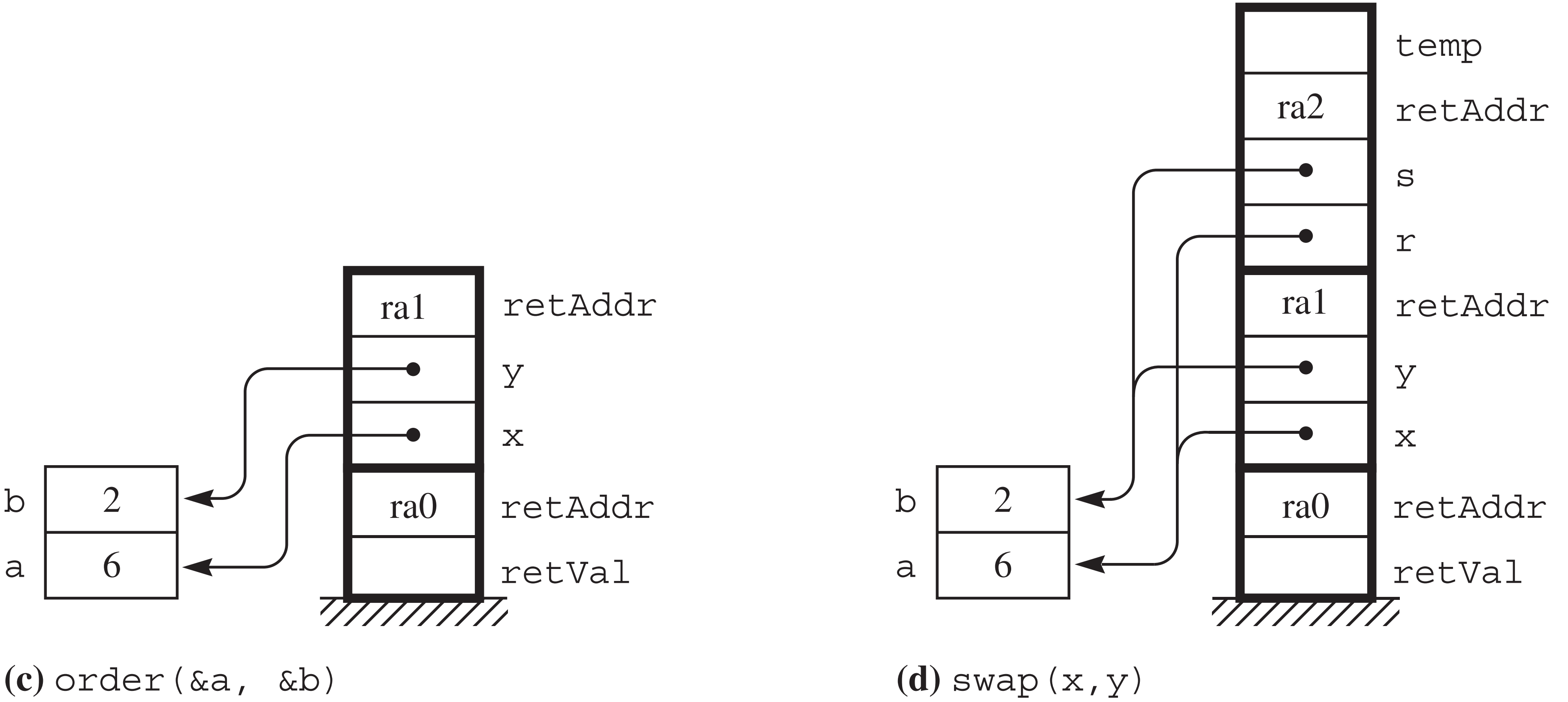
Figure 2.21 (continued)



(c) order(&a, &b)

Computer Systems FIFTH EDITION

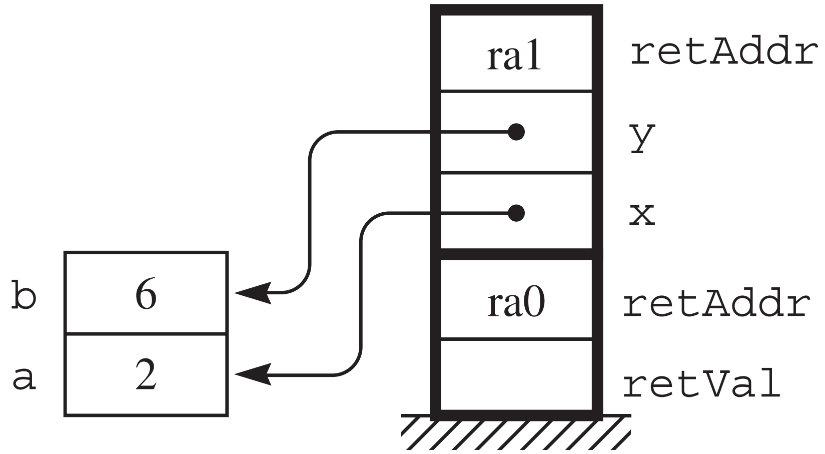
Figure 2.21 (continued)



(c) `order(&a, &b)`

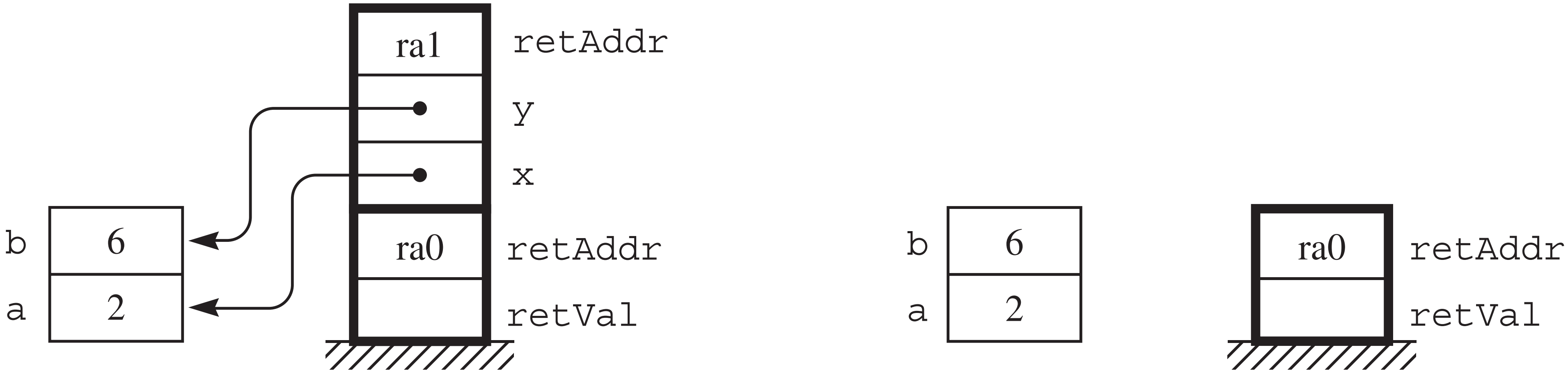
(d) `swap(x,y)`

Figure 2.21 (continued)



(e) Return from `swap()`

Figure 2.21 (continued)



(e) Return from `swap()`

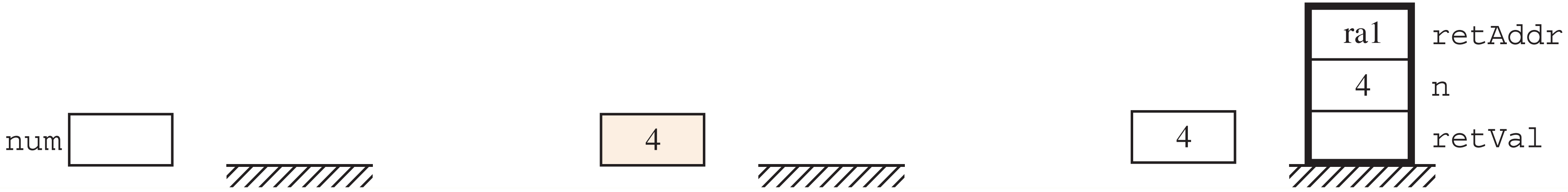
(f) Return from `order()`

Figure 2.22

```
#include <stdio.h>

int num;

int fact(int n) {
   if (n <= 1) {
      return 1;
   }
   else {
      return n * fact(n - 1); // ra2
   }
}

int main() {
   printf("Enter a small integer: ");
   scanf("%d", &num);
   printf("Its factorial is: %d\n", fact(num)); // ra1
   return 0;
}
```

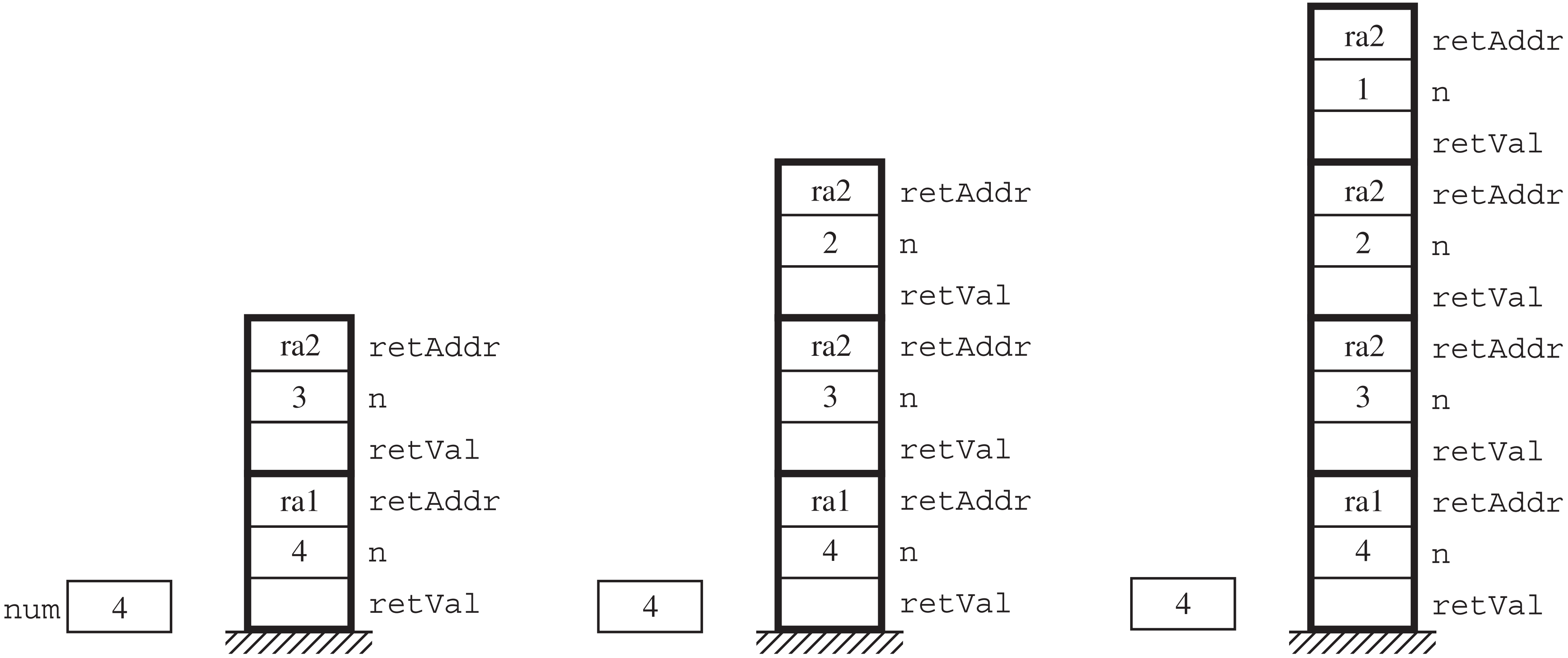
Interactive Input/Output

```
Enter a small integer: 4
Its factorial is: 24
```



(a) Begin **(b)** `scanf("%d", &num)` **(c)** Call `fact` (4)

Figure 2.23 (continued)



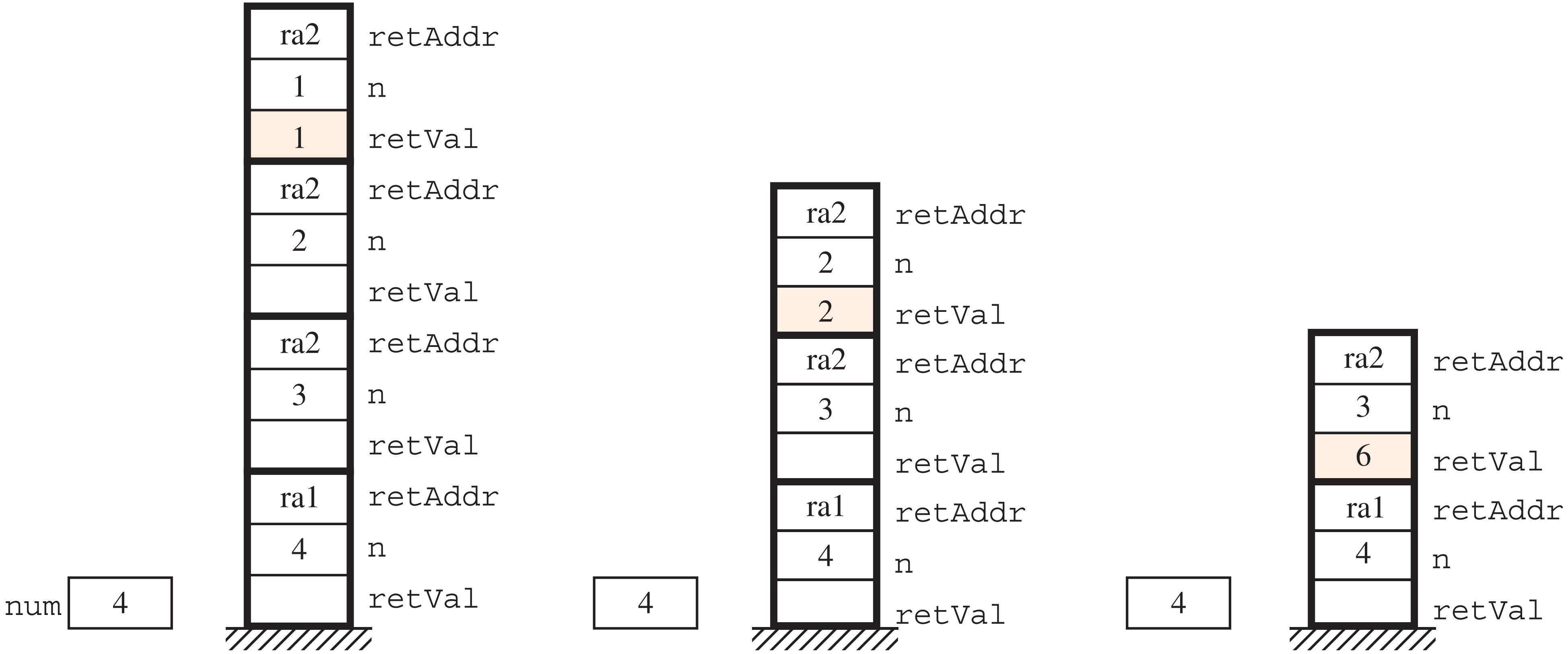
(d) Call `fact` (3)

(e) Call `fact` (2)

(f) Call `fact` (1)

Computer Systems FIFTH EDITION

Figure 2.23 (continued)

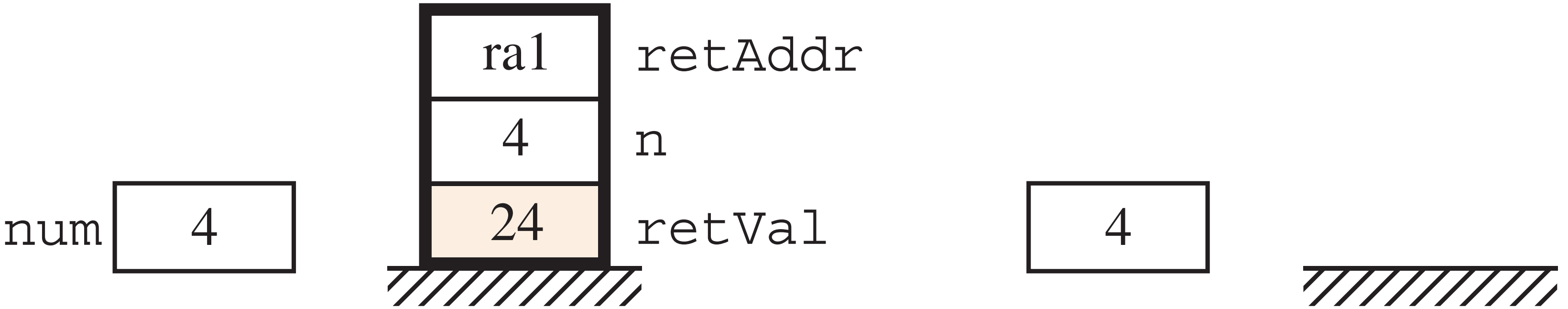


(g) Compute `retVal`

(h) Return

(i) Return

Figure 2.23 (continued)



(j) Return

(k) Return

Figure 2.24

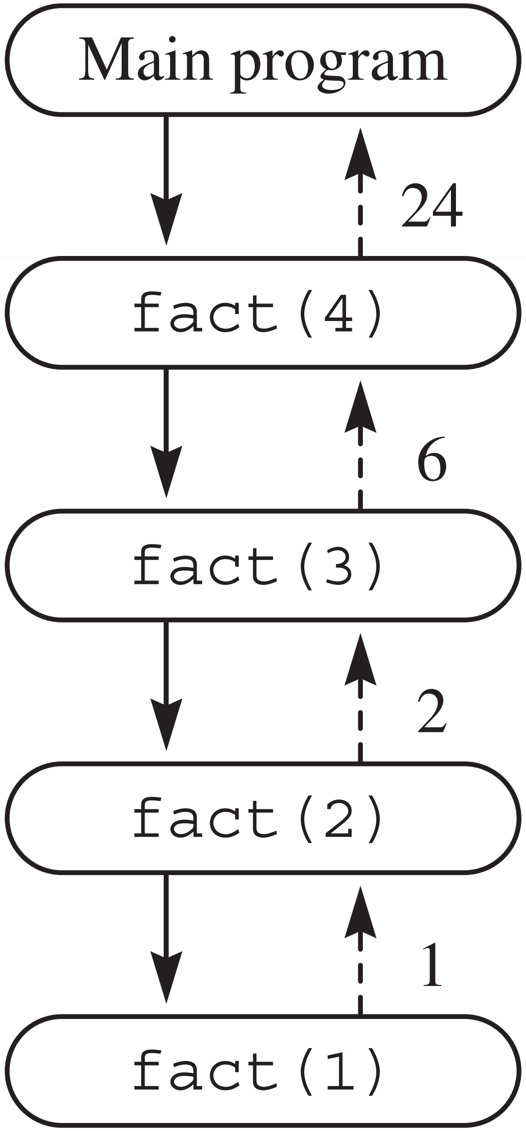


Figure 2.25

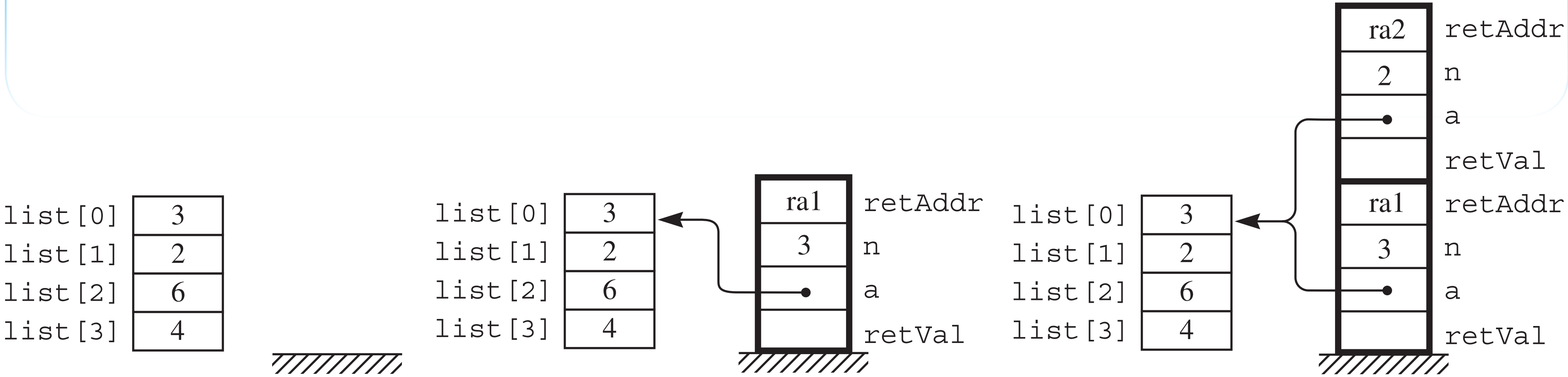
```
#include <stdio.h>

int list[4];

int sum(int a[], int n) {
// Returns the sum of the elements of a between a[0] and a[n].
   if (n == 0) {
      return a[0];
   }
   else {
      return a[n] + sum(a, n - 1); // ra2
   }
}

int main() {
   printf("Enter four integers: ");
   scanf("%d %d %d %d", &list[0], &list[1], &list[2], &list[3]);
   printf("Their sum is: %d\n", sum(list, 3));
   return 0;
}
```

Interactive Input/Output

```
Enter four integers: 3 2 6 4
Their sum is: 15
```

Computer Systems FIFTH EDITION

Figure 2.26



(a) *Input* list

(b) Call sum (list, 3)

(c) Call sum (list, 2)

Power, n	Term number, k							
	0	1	2	3	4	5	6	7
1	1	1						
2	1	2	1					
3	1	3	3	1				
4	1	4	6	4	1			
5	1	5	10	10	5	1		
6	1	6	15	20	15	6	1	
7	1	7	21	35	35	21	7	1

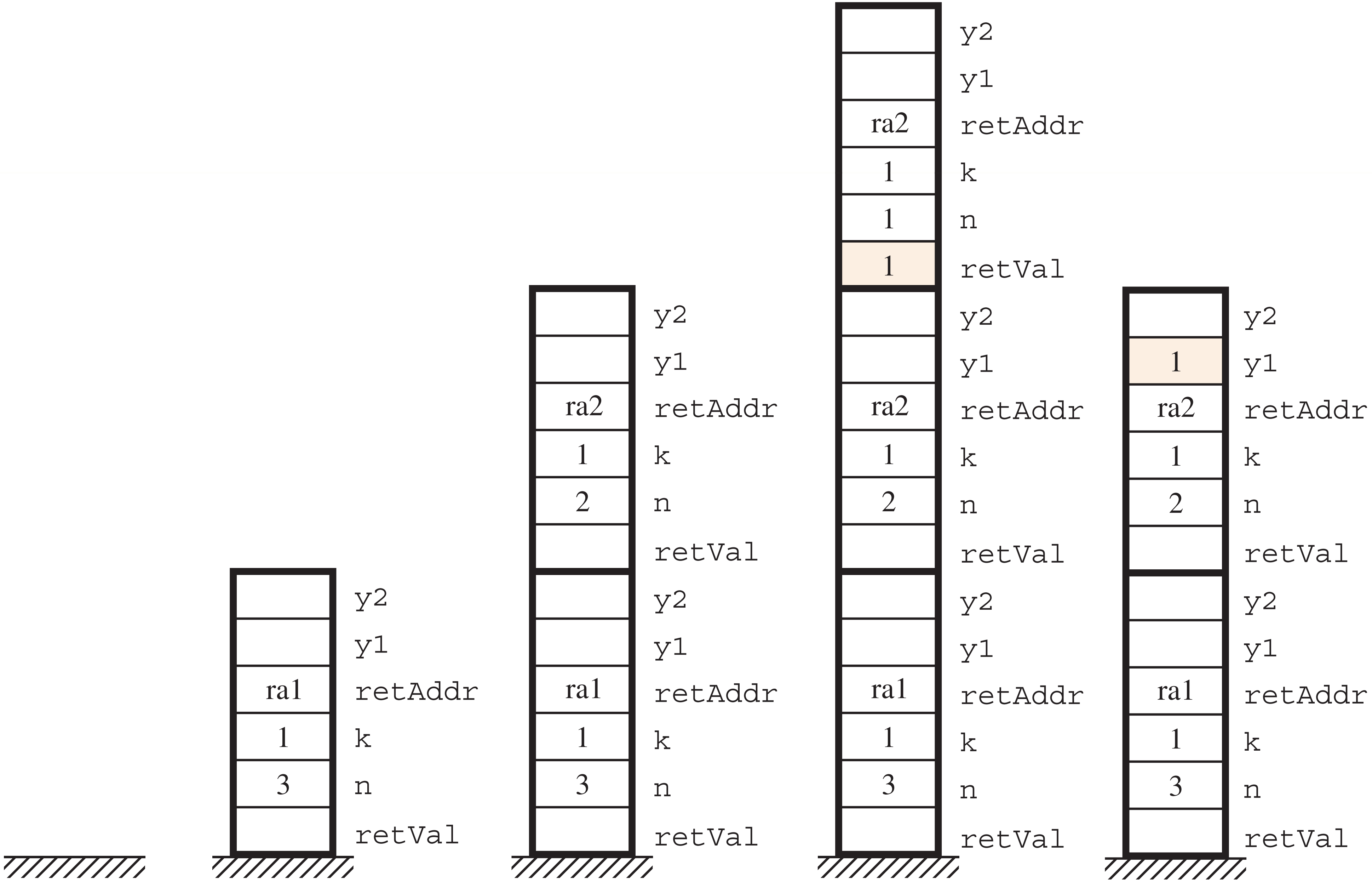
Figure 2.28

```c
#include <stdio.h>

int binCoeff(int n, int k) {
   int y1, y2;
   if ((k == 0) || (n == k)) {
      return 1;
   }
   else {
      y1 = binCoeff(n - 1, k); // ra2
      y2 = binCoeff(n - 1, k - 1); // ra3
      return y1 + y2;
   }
}

int main() {
   printf("binCoeff(3, 1) = %d\n", binCoeff(3, 1)); // ra1
   return 0;
}
```

Output

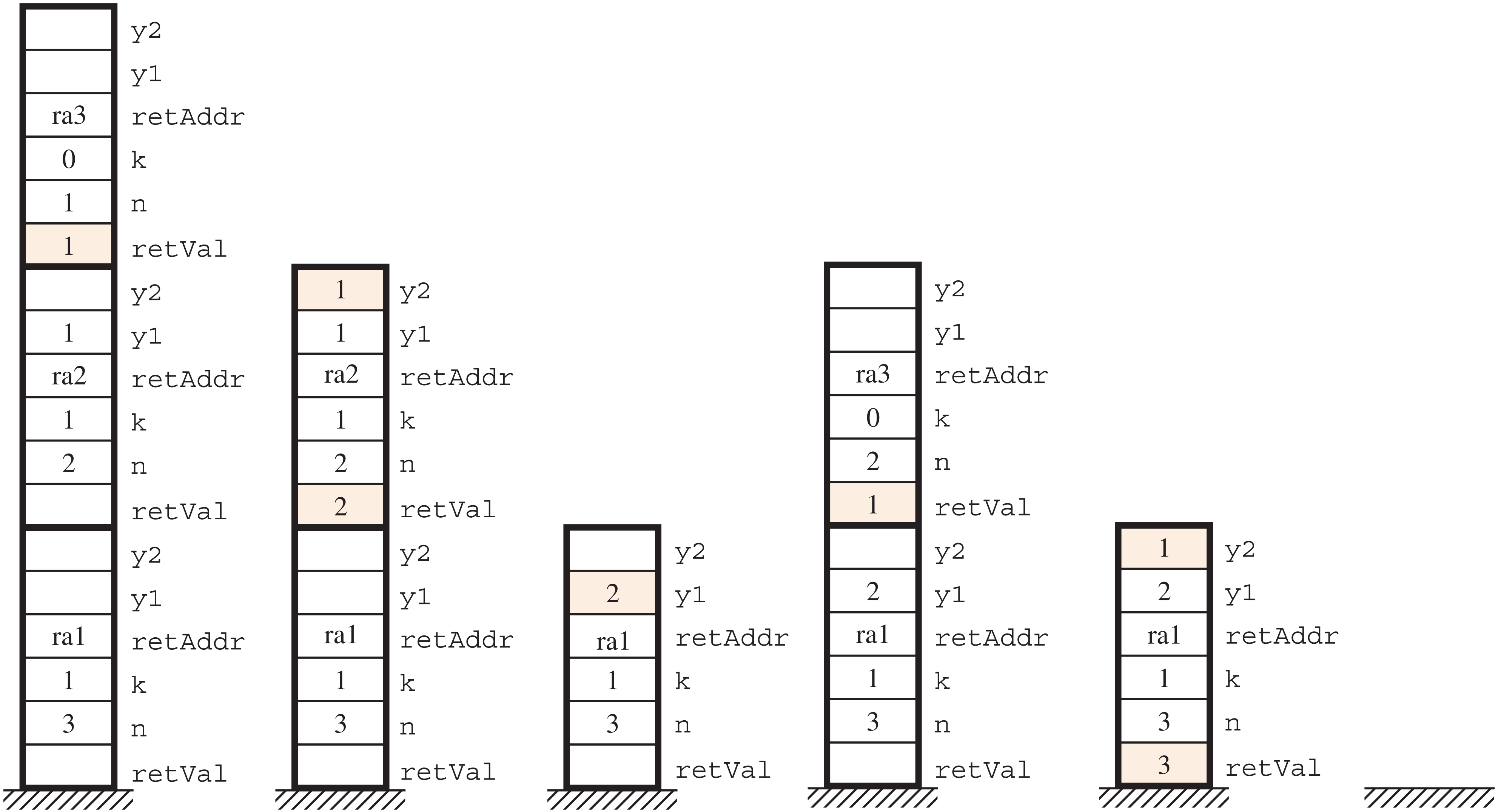
```
binCoeff(3, 1) = 3
```

Figure 2.29



(a) Begin **(b)** Call BC (3, 1) **(c)** Call BC (2, 1) **(d)** Call BC (1, 1) **(e)** Return

Figure 2.29 (continued)



(f) Call BC (1, 0)

	y2
	y1
ra3	retAddr
0	k
1	n
1	retVal
	y2
1	y1
ra2	retAddr
1	k
2	n
	retVal
	y2
	y1
ra1	retAddr
1	k
3	n
	retVal

(g) Return

1	y2
1	y1
ra2	retAddr
1	k
2	n
2	retVal
	y2
	y1
ra1	retAddr
1	k
3	n
	retVal

(h) Return

	y2
2	y1
ra1	retAddr
1	k
3	n
	retVal

(i) Call BC (2, 0)

	y2
	y1
ra3	retAddr
0	k
2	n
1	retVal
	y2
2	y1
ra1	retAddr
1	k
3	n
	retVal

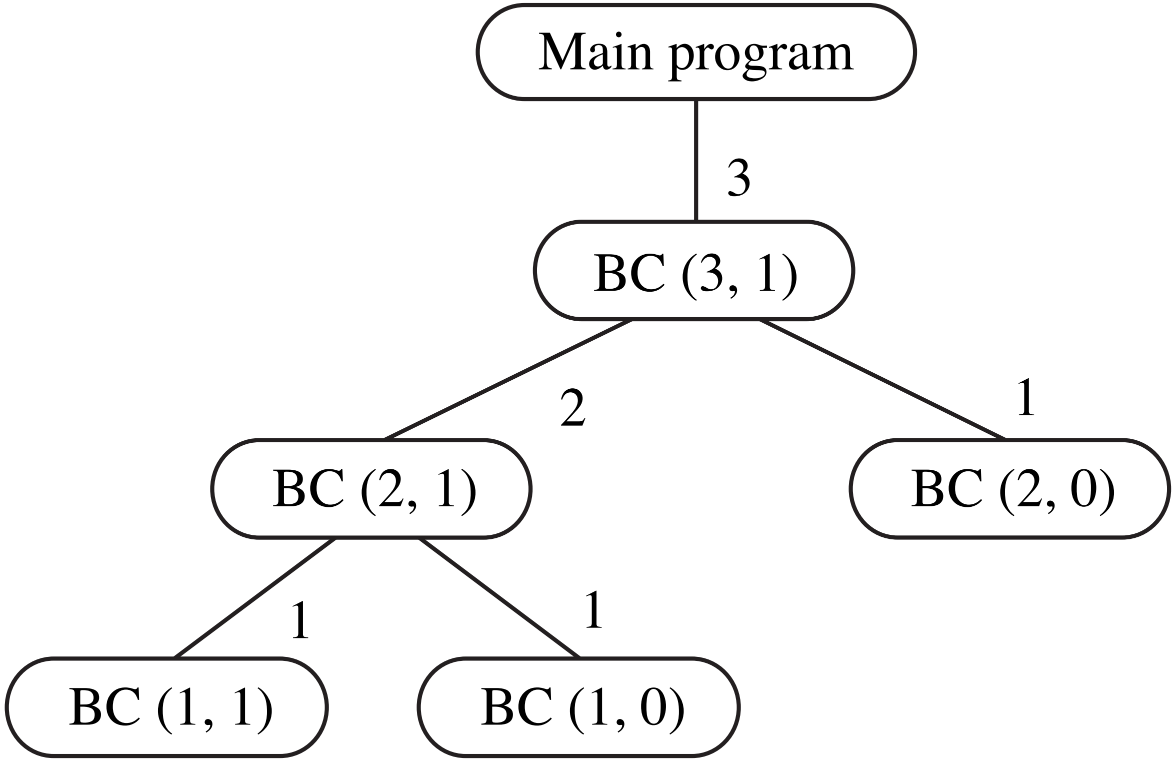
(j) Return

1	y2
2	y1
ra1	retAddr
1	k
3	n
3	retVal

(k) Return

Figure 2.30

```
Main program
    Call BC(3,1)
        Call BC(2,1)
            Call BC(1,1)
        Return to BC(2,1)
            Call BC(1,0)
        Return to BC(2,1)
    Return to BC(3,1)
        Call BC(2,0)
    Return to BC(3,1)
Return to main program
```



Computer Systems FIFTH EDITION

Figure 2.31

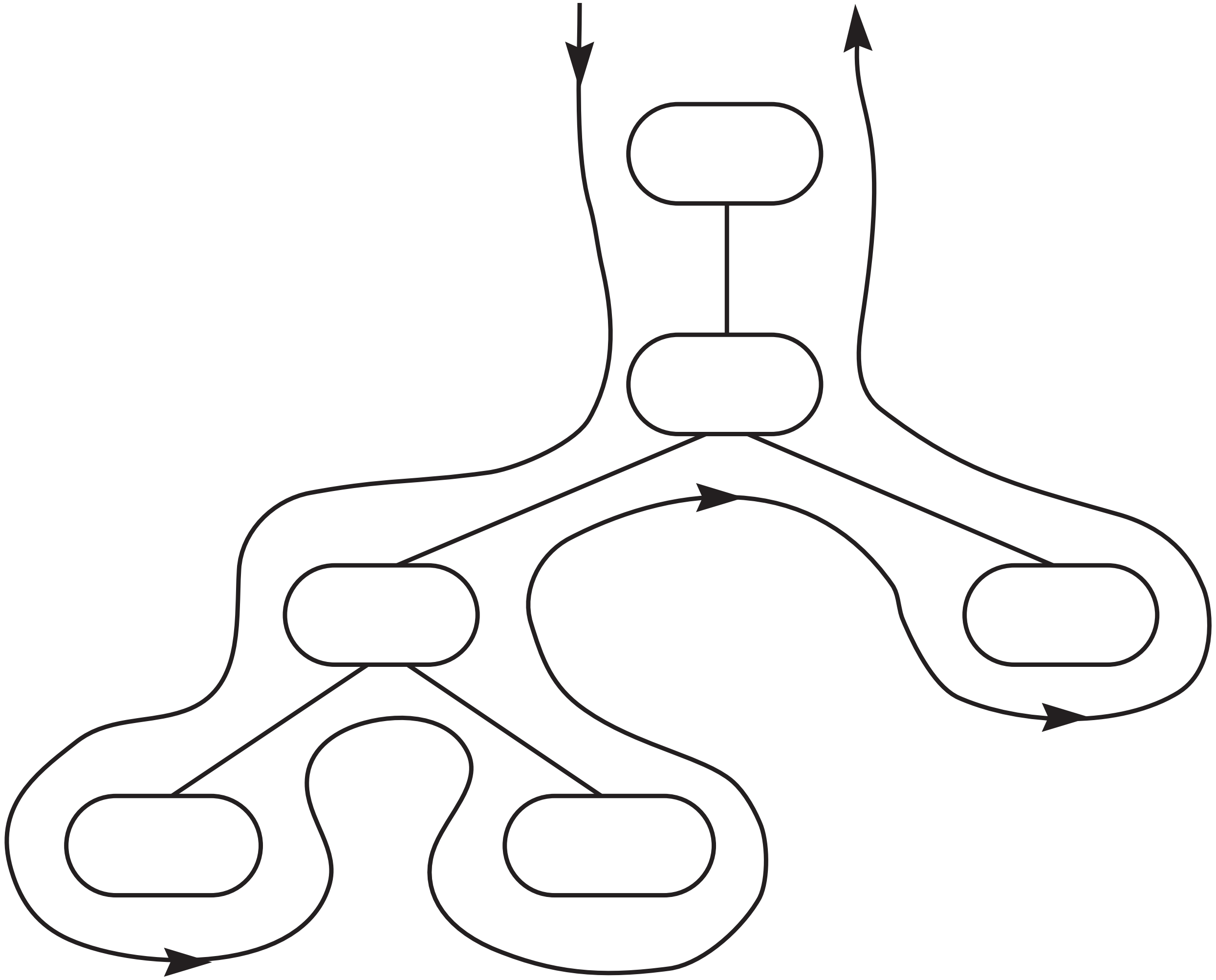


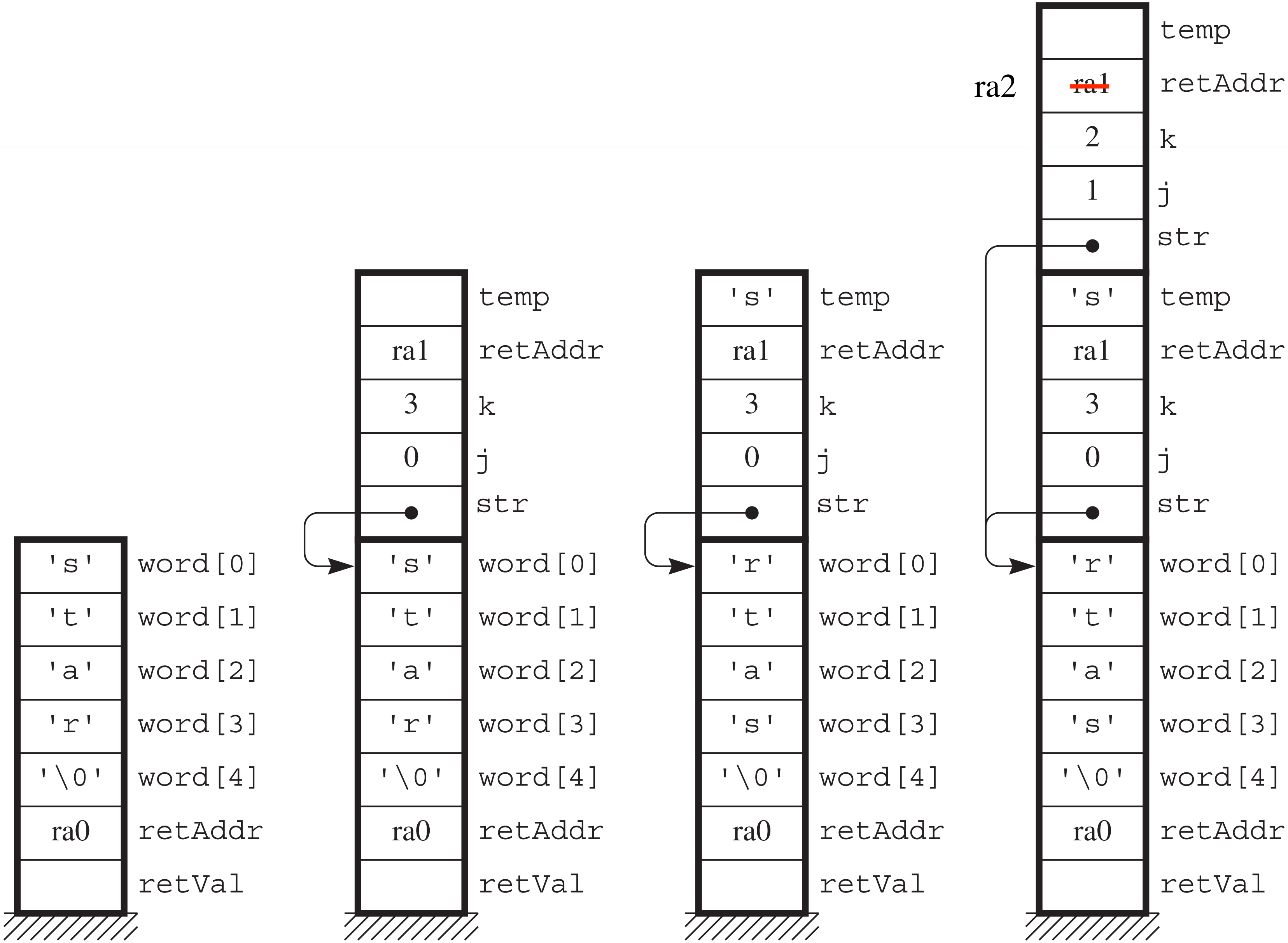
Figure 2.32

```
#include <stdio.h>

void reverse(char *str, int j, int k) {
   char temp;
   if (j < k) {
      temp = str[j];
      str[j] = str[k];
      str[k] = temp;
      reverse(str, j + 1, k - 1);
   } // ra2
}

int main() {
   char word[5] = "star";
   printf("%s\n", word);
   reverse(word, 0, 3);
   printf("%s\n", word); // ra1
   return 0;
}
```

Output

```
star
rats
```

Figure 2.33



(a) char word[5]
= "star"

(b) Call reverse
(word, 0, 3)

(c) Switch s and r

(d) Call reverse
(word, 1, 2)

Figure 2.34

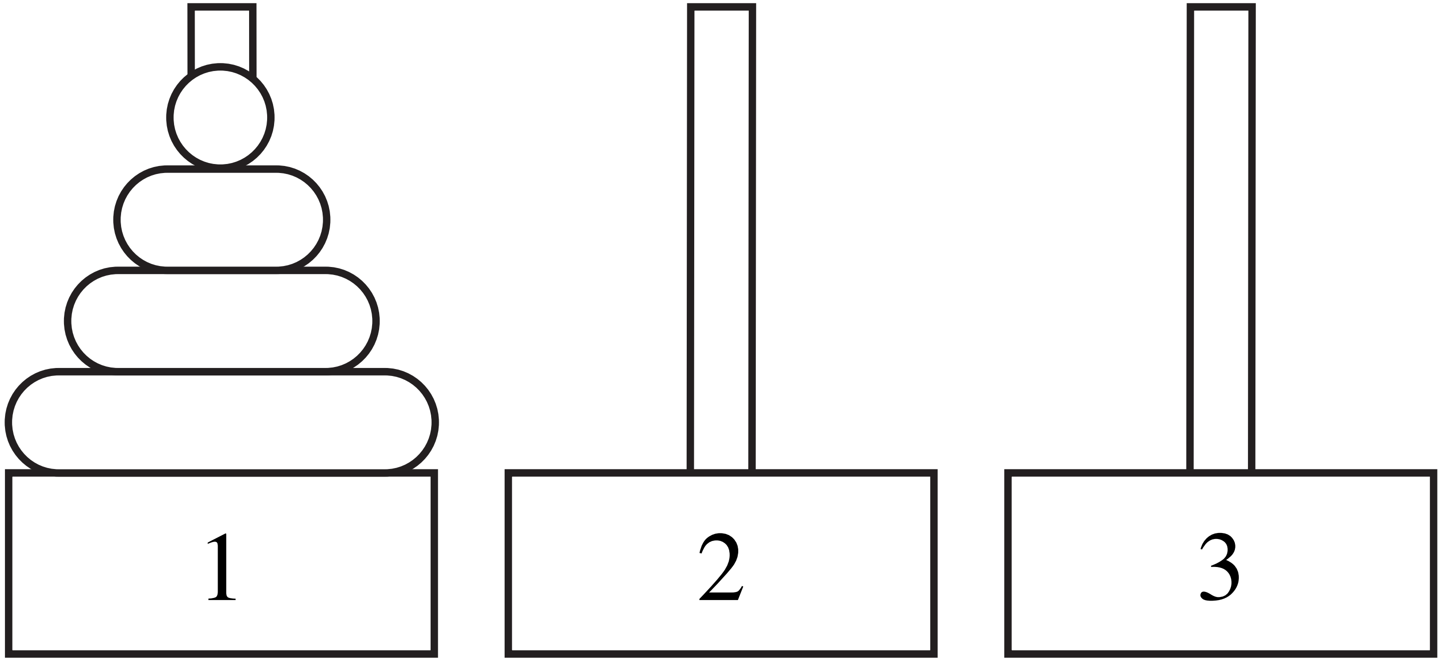
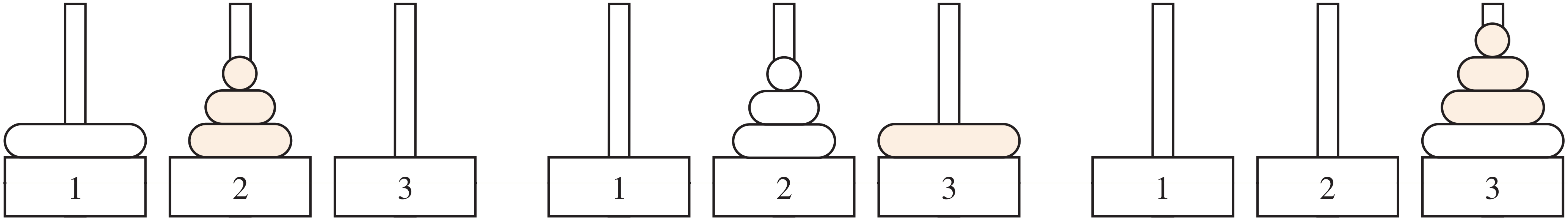


Figure 2.35

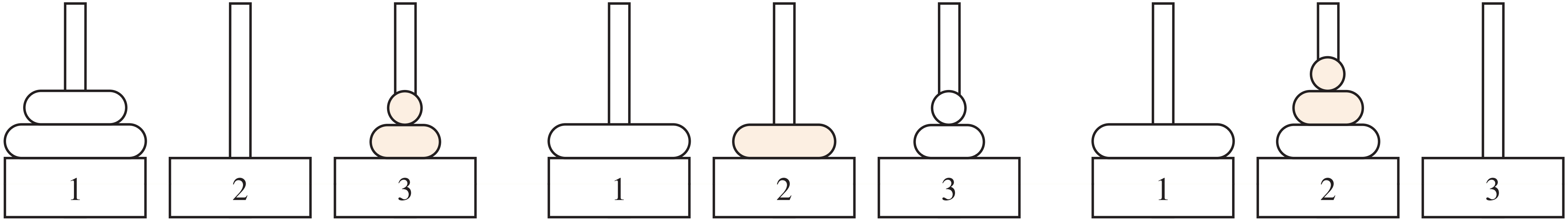


(a) Move three disks from peg 1 to peg 2.

(b) Move one disk from peg 1 to peg 3.

(c) Move three disks from peg 2 to peg 3.

Figure 2.36

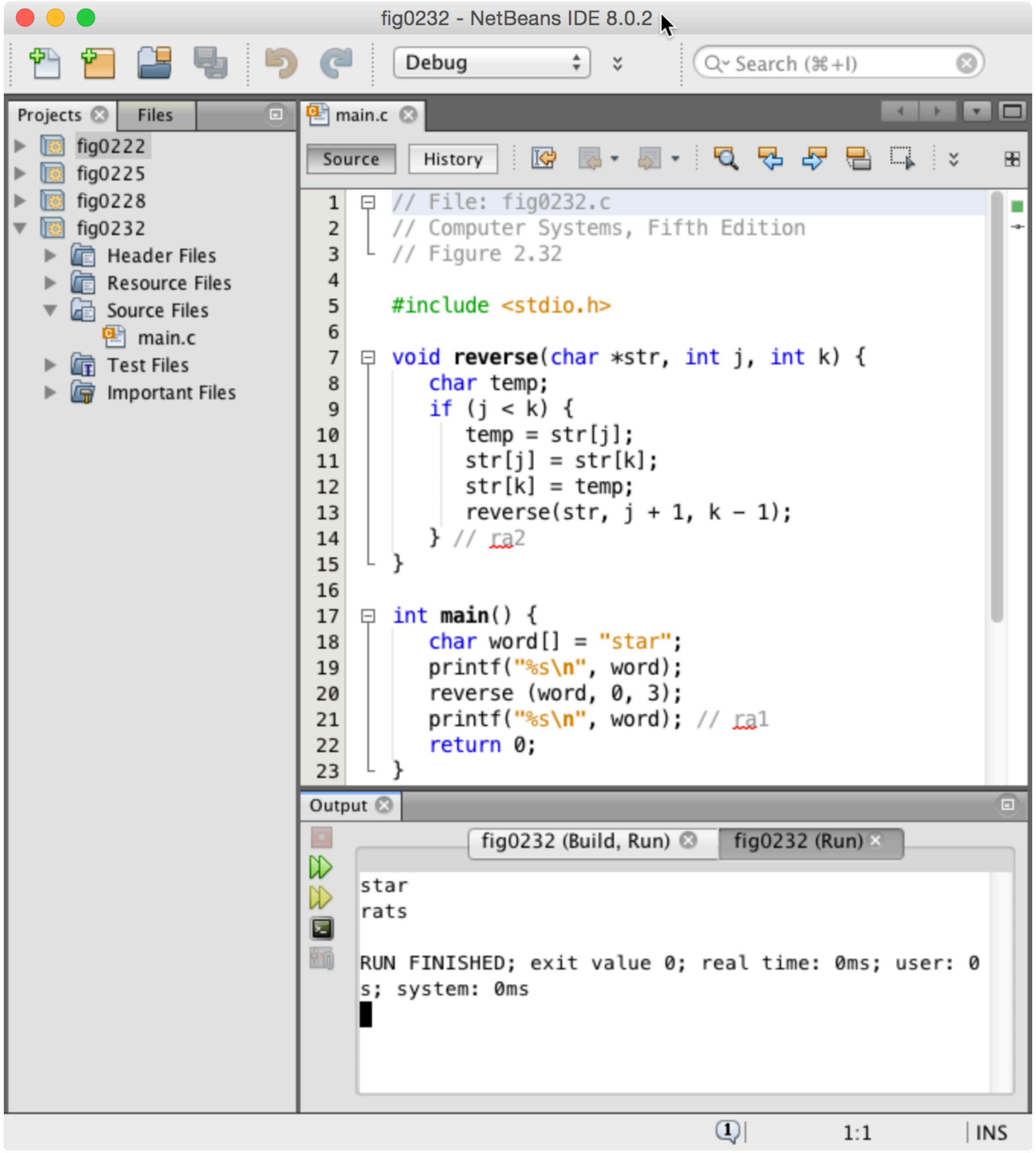


(a) Move two disks from peg 1 to peg 3.

(b) Move one disk from peg 1 to peg 2.

(c) Move two disks from peg 3 to peg 2.

Figure 2.37



```
// File: fig0232.c
// Computer Systems, Fifth Edition
// Figure 2.32

#include <stdio.h>

void reverse(char *str, int j, int k) {
    char temp;
    if (j < k) {
        temp = str[j];
        str[j] = str[k];
        str[k] = temp;
        reverse(str, j + 1, k - 1);
    } // ra2
}

int main() {
    char word[] = "star";
    printf("%s\n", word);
    reverse (word, 0, 3);
    printf("%s\n", word); // ra1
    return 0;
}
```

```
star
rats

RUN FINISHED; exit value 0; real time: 0ms; user: 0
s; system: 0ms
```

The C memory model

- Global variables – fixed location in memory
- Local variables and parameters – run-time stack
- Dynamically allocated variables – heap

Two operators for dynamic memory allocation

- `malloc()`, to allocate from the heap
- `free()`, to deallocate from the heap

Two actions of the `malloc()` function

- It allocates a memory cell from the heap large enough to hold a value of the type that is on its right-hand side.
- It returns a pointer to the newly allocated storage.

The pointer assignment rule

- If p and q are pointers, the assignment

  ```
  p = q
  ```

  makes p point to the same cell to which q points.

Figure 2.38

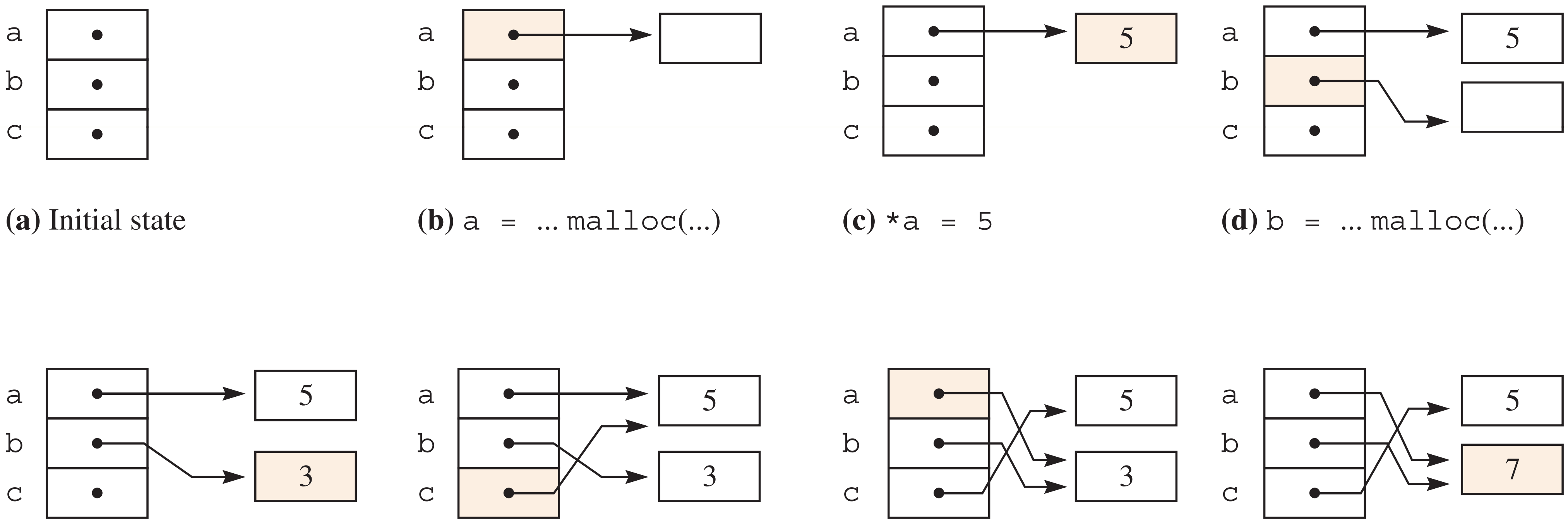
```
#include <stdio.h>
#include <stdlib.h>

int *a, *b, *c;

int main() {
   a = (int *) malloc(sizeof(int));
   *a = 5;
   b = (int *) malloc(sizeof(int));
   *b = 3;
   c = a;
   a = b;
   *a = 2 + *c;
   printf("*a = %d\n", *a);
   printf("*b = %d\n", *b);
   printf("*c = %d\n", *c);
   return 0;
}
```

Output

```
*a = 7
*b = 7
*c = 5
```

Computer Systems FIFTH EDITION

Figure 2.39



(a) Initial state

(b) `a = ... malloc(...)`

(c) `*a = 5`

(d) `b = ... malloc(...)`

(e) `*b = 3`

(f) `c = a`

(g) `a = b`

(h) `*a = 2 + *c`

Figure 2.40

```
#include <stdio.h>

struct person {
   char first;
   char last;
   int age;
   char gender;
};
struct person bill;

int main() {
   scanf("%c%c%d %c", &bill.first, &bill.last, &bill.age, &bill.gender);
   printf("Initials: %c%c\n", bill.first, bill.last);
   printf("Age: %d\n", bill.age);
   printf("Gender: ");
   if (bill.gender == 'm') {
      printf("male\n");
   }
   else {
      printf("female\n");
   }
   return 0;
}
```

Figure 2.40 (continued)

Input

```
bj 32 m
```

Output

```
Initials: bj
Age: 32
Gender: male
```

Figure 2.41

```
#include <stdio.h>
#include <stdlib.h>

struct node {
   int data;
   struct node *next;
};

int main() {
   struct node *first, *p;
   int value;
   first = 0;
   scanf("%d", &value);
   while (value != -9999) {
      p = first;
      first = (struct node *) malloc(sizeof(struct node));
      first->data = value;
      first->next = p;
      scanf("%d", &value);
   }
   for (p = first; p != 0; p = p->next) {
      printf("%d ", p->data);
   }
   return 0;
}
```

Figure 2.41 (continued)

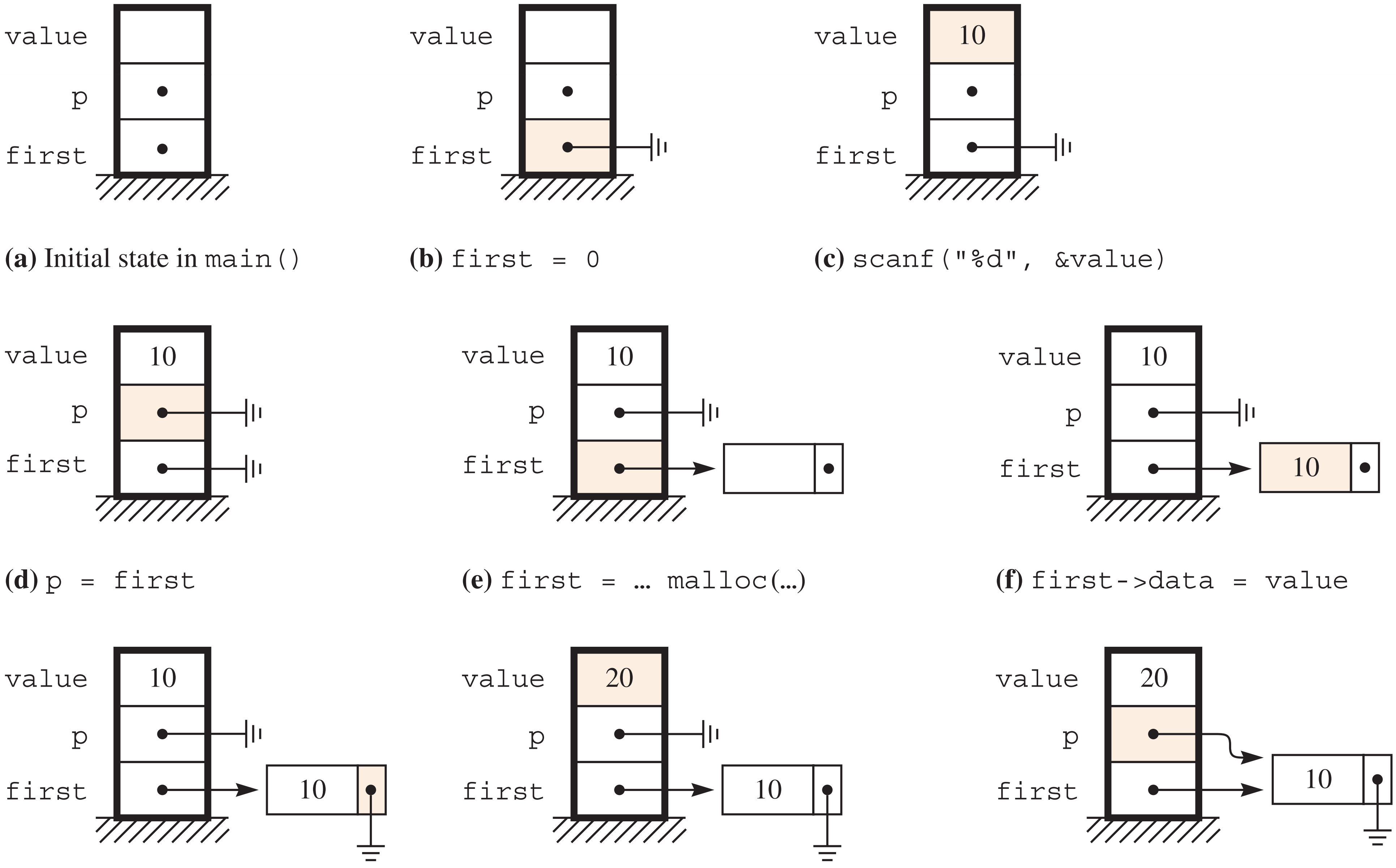
Input

```
10 20 30 40 -9999
```

Output

```
40 30 20 10
```

Figure 2.42



(a) Initial state in `main()`

(b) `first = 0`

(c) `scanf("%d", &value)`

(d) `p = first`

(e) `first = … malloc(…)`

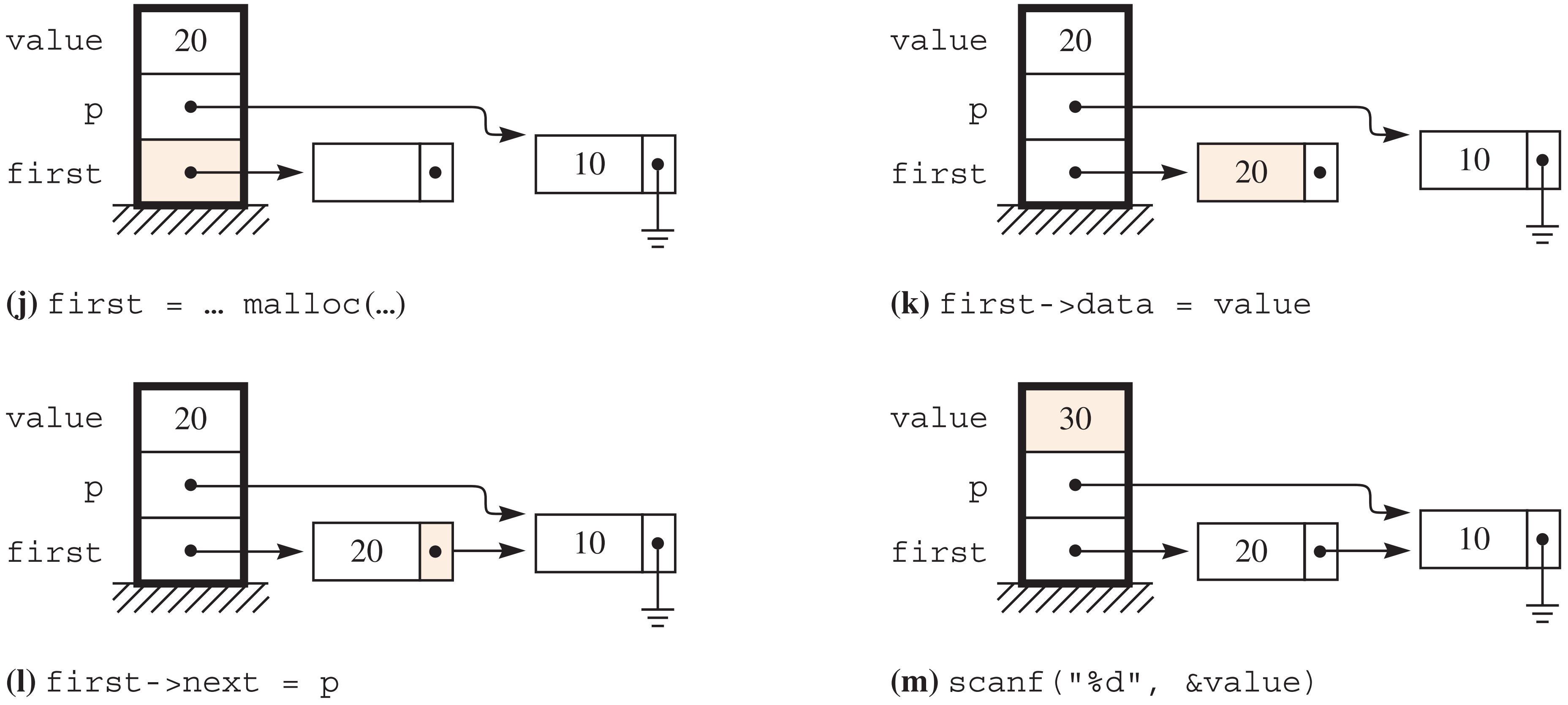
(f) `first->data = value`

(g) `first->next = p`

(h) `scanf("%d", &value)`

(i) `p = first`

Figure 2.42 (continued)



(j) `first = … malloc(…)`

(k) `first->data = value`

(l) `first->next = p`

(m) `scanf("%d", &value)`