C++

Figure 2.1

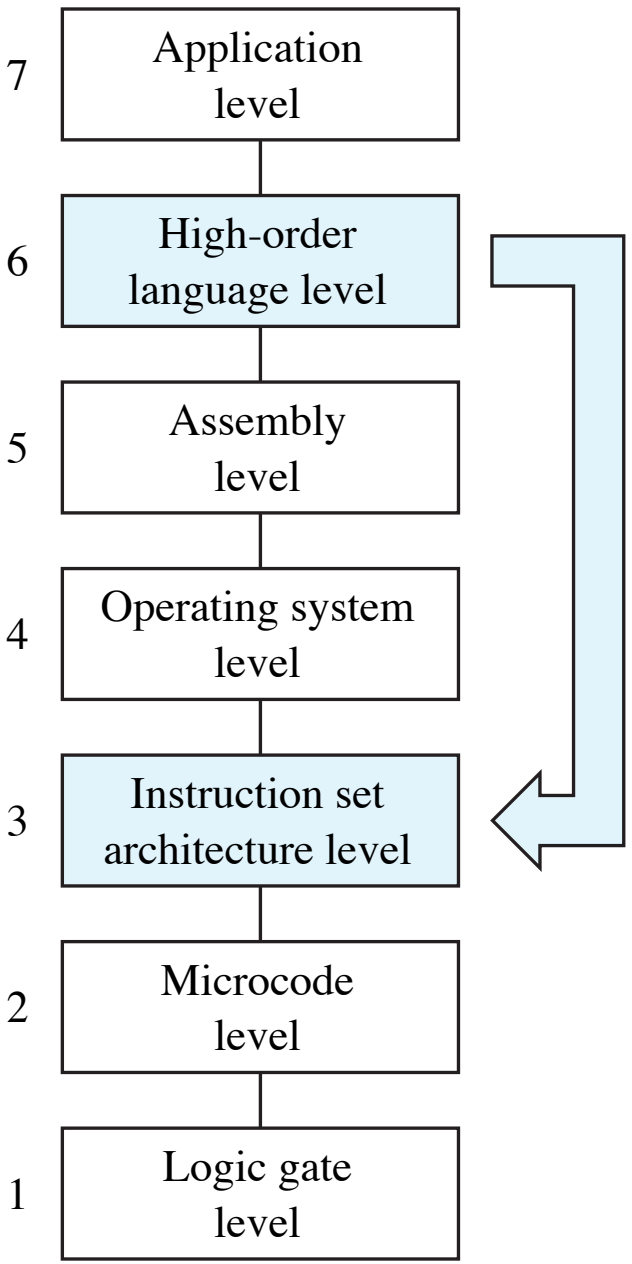


Figure 2.2

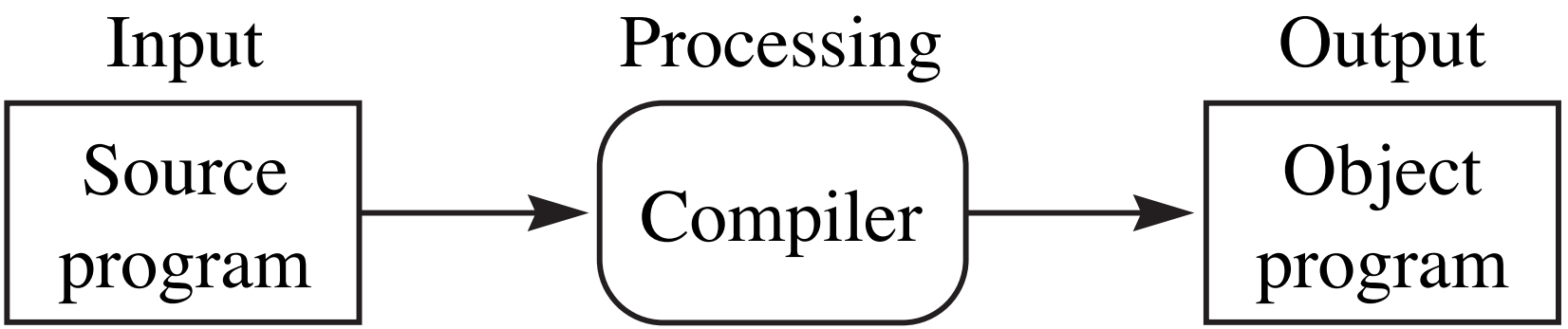
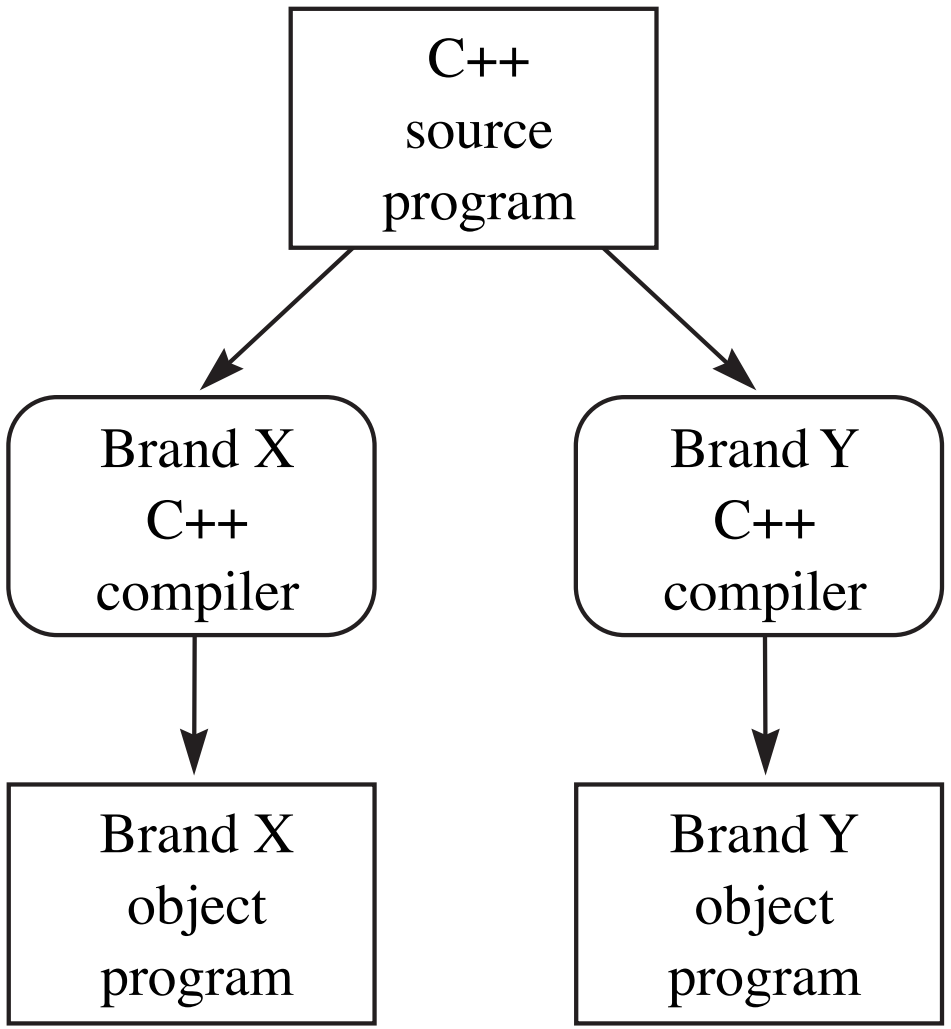


Figure 2.3



The C++ memory model

- Global variables, fixed location
- Local variables, run-time stack
- Dynamically allocated variables, heap

Function call

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

Function return

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

Three attributes of a C++ variable

- Name
- Type
- Value

Figure 2.4

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

#include <iostream>
using namespace std;

char ch;
int j;

int main () {
   cin >> ch >> j;
   j += 5;
   ch++;
   cout << ch << endl << j << endl;
   return 0;
}
```

Input

```
M 419
```

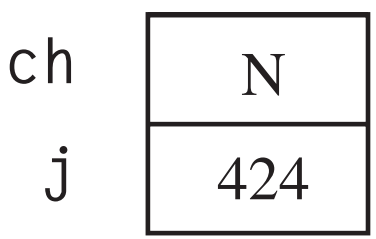
Output

```
N
424
```

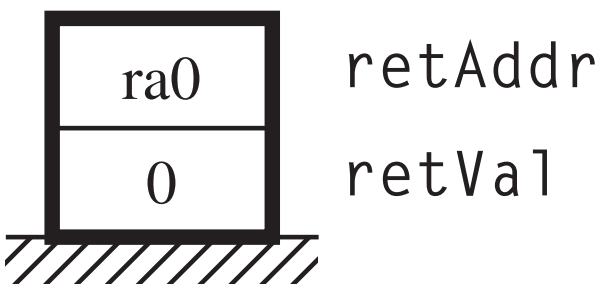
Variables

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

Figure 2.5



(a) Fixed location.



(b) Run-time stack.

Figure 2.6

```
#include <iostream>
using namespace std;

int main () {
   const int bonus = 5;
   int exam1;
   int exam2;
   int score;
   cin >> exam1 >> exam2;
   score = (exam1 + exam2) / 2 + bonus;
   cout << "score = " << score << endl;
   return 0;
}
```

Input

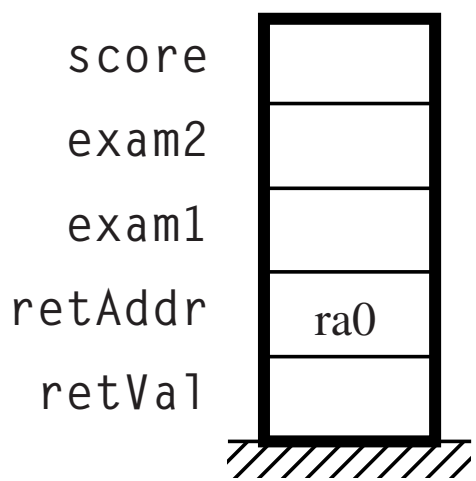
```
68 84
```

Output

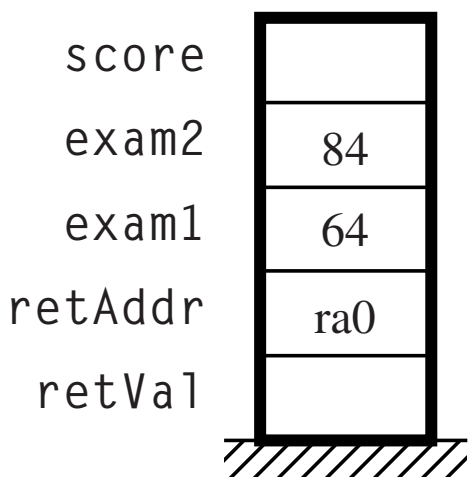
```
score = 81
```

Figure 2.7

Expression	Value	Expression	Value
15 / 3	5	15 % 3	0
14 / 3	4	14 % 3	2
13 / 3	4	13 % 3	1
12 / 3	4	12 % 3	0
11 / 3	3	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 <iostream>
using namespace std;

int main () {
   const int limit = 100;
   int num;
   cin >> num;
   if (num >= limit) {
      cout << "high";
   }
   else {
      cout << "low";
   }
   return 0;
}
```

Input

```
75
```

Output

```
low
```

Figure 2.11

Meaning	Symbol
AND	&&
OR	||
NOT	!

Figure 2.12

```
#include <iostream>
using namespace std;

int main () {
   int guess;
   cout << "Pick a number 0..3: ";
   cin >> guess;
   switch (guess) {
      case 0: cout << "Not close"; break;
      case 1: cout << "Close"; break;
      case 2: cout << "Right on"; break;
      case 3: cout << "Too high";
   }
   cout << endl;
   return 0;
}
```

Interactive Input/Output

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

Figure 2.13

```
#include <iostream>
using namespace std;

char letter;

int main () {
   cin >> letter;
   while (letter != '*') {
      cout << letter;
      cin >> letter;
   }
   return 0;
}
```

Input

```
happy*
```

Output

```
happy
```

Figure 2.14

```
#include <iostream>
using namespace std;

int cop;
int driver;

int main () {
   cop = 0;
   driver = 40;
   do {
      cop += 25;
      driver += 20;
   }
   while (cop < driver);
   cout << cop;
   return 0;
}
```

Output

```
200
```

Figure 2.15

```
#include <iostream>
using namespace std;

int main () {
   int vector[4];
   int j;
   for (j = 0; j < 4; j++) {
      cin >> vector[j];
   }
   for (j = 3; j >= 0; j--) {
      cout << j  << ' ' << vector[j] << endl;
   }
   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

- Deallocate storage for the local variables
- Pop the return address
- Deallocate the actual parameters

```
#include <iostream>
using namespace std;

int numPts;
int value;
int j;

void printBar (int n) {
   int k;
   for (k = 1; k <= n; k++) {
      cout << '*';
   }
   cout << endl;
}

int main () {
   cin >> numPts;
   for (j = 1; j <= numPts; j++) {
      cin >> value;
      printBar (value);
   }  // ra1
   return 0;
}
```

Figure 2.16
(Continued)

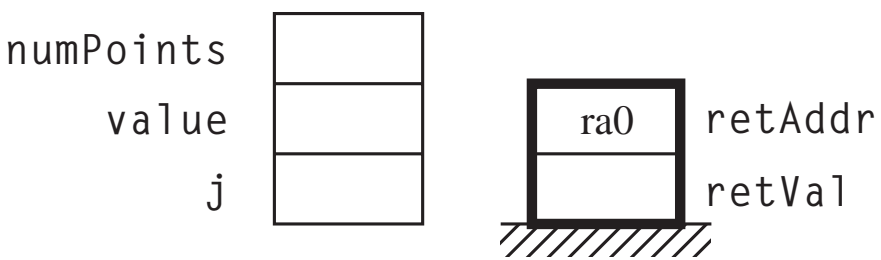
Input

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

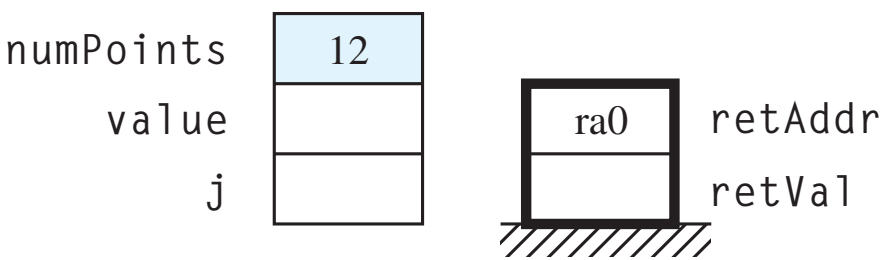
Output

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

**
```

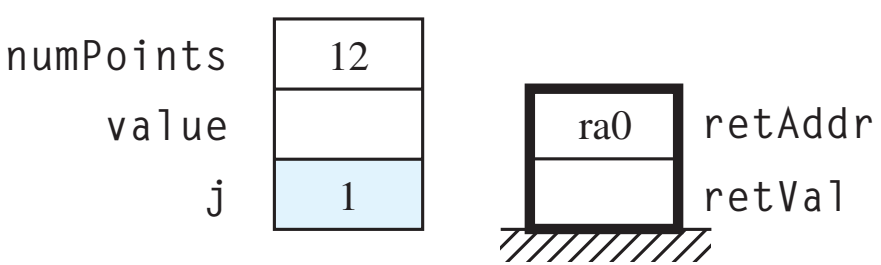
Figure 2.17



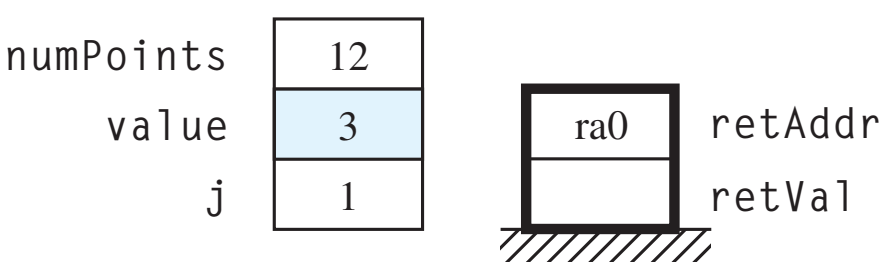
(a) Begin



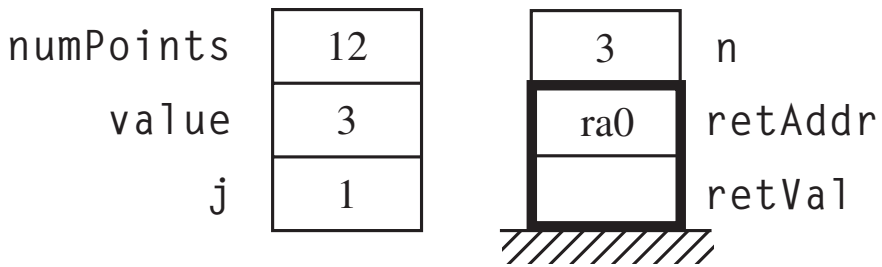
(b) `cin >> numPoints`



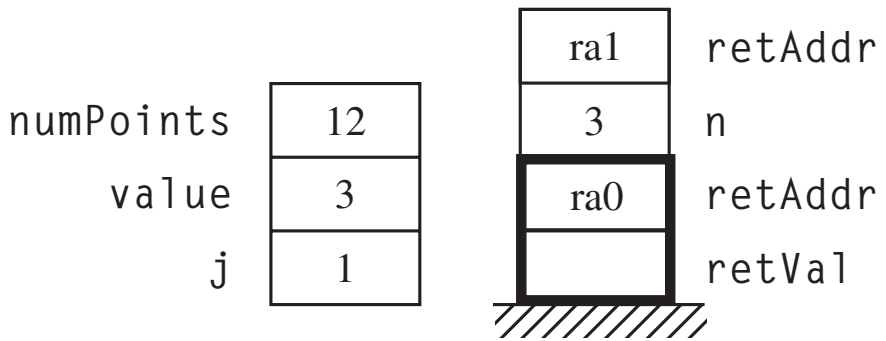
(c) `for(j = 1; j <= numPoints; j++)`



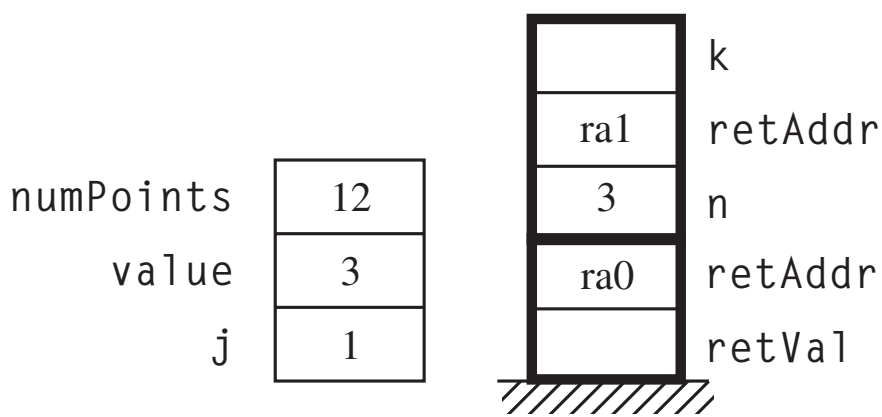
(d) `cin >> value`



(e) Push formal parameter



(f) Push return address



(g) Push storage for local variable k

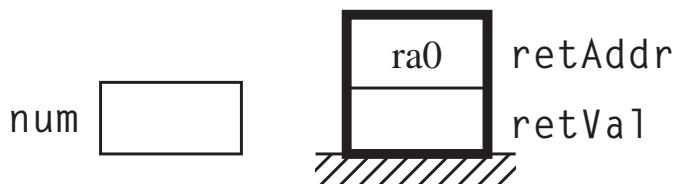
Figure 2.18

```
#include <iostream>
using namespace std;

int num;

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

int main () {
   cout << "Enter a small integer: ";
   cin >> num;
   cout << "Its factorial is: " << fact (num) << endl; // ra1
   return 0;
}
```

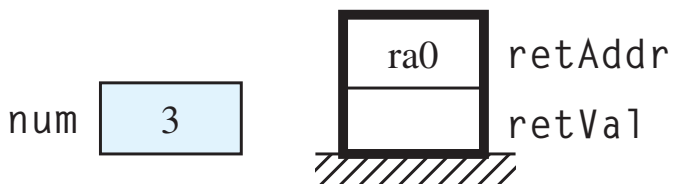
Interactive Input/Output

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

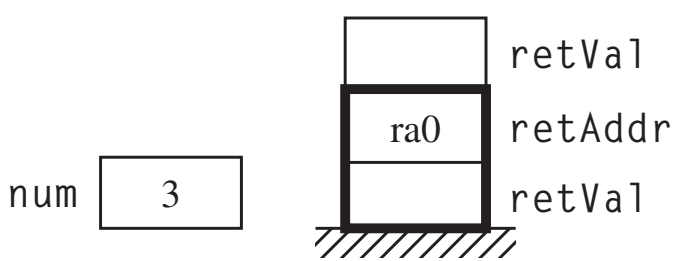
Figure 2.19



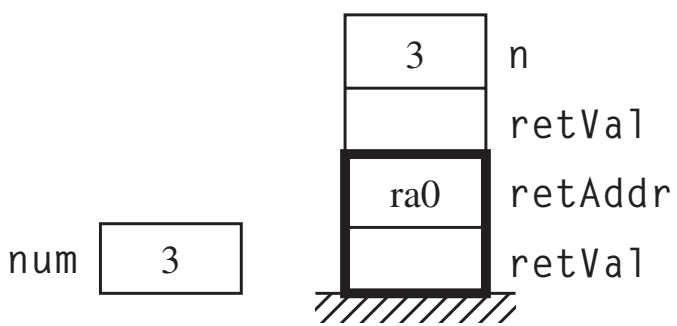
(a) Begin



(b) `cin >> num`

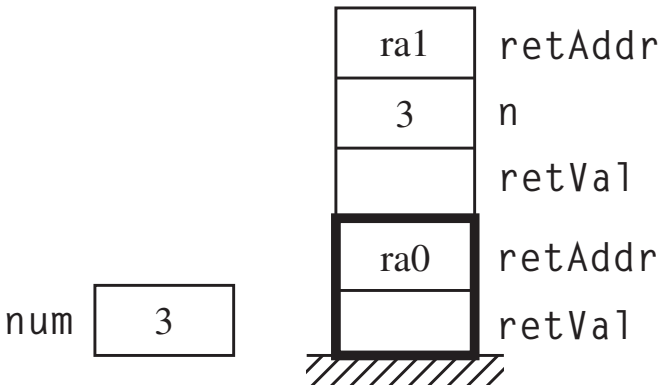


(c) Push storage for return value `i`

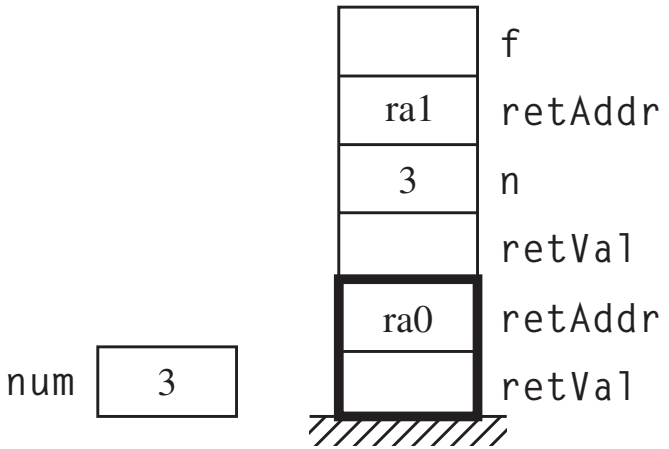


(d) Push actual parameter

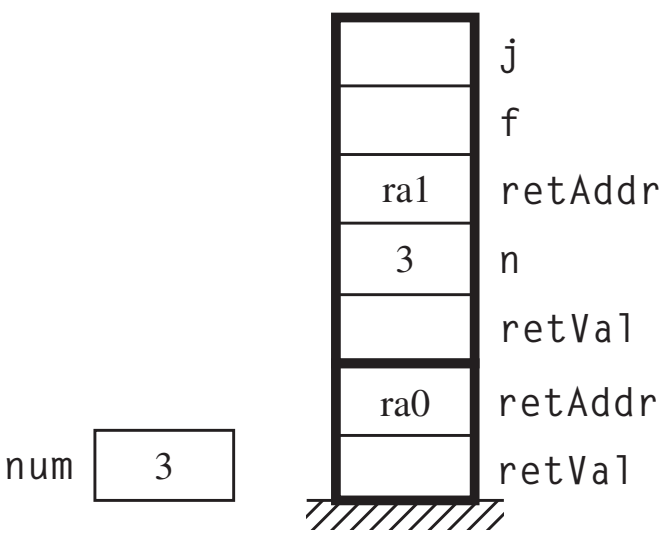
Figure 2.19
(Continued)



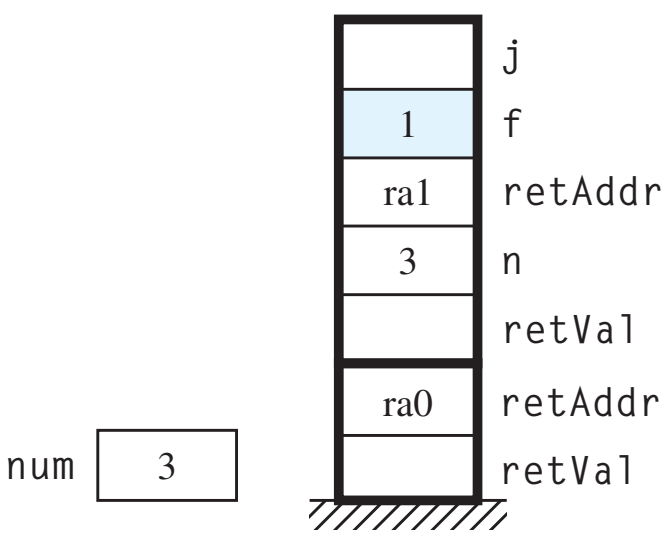
(e) Push return address



(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.

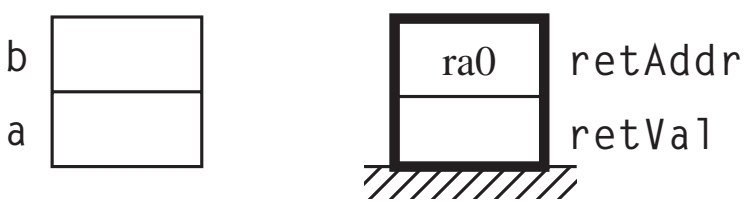
Figure 2.20

```
#include <iostream>
using namespace std;

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 () {
   cout << "Enter an integer: ";
   cin >> a;
   cout << "Enter an integer: ";
   cin >> b;
   order (a, b);
   cout << "Ordered they are: " << a << ", " << b << endl; // ra1
   return 0;
}
```

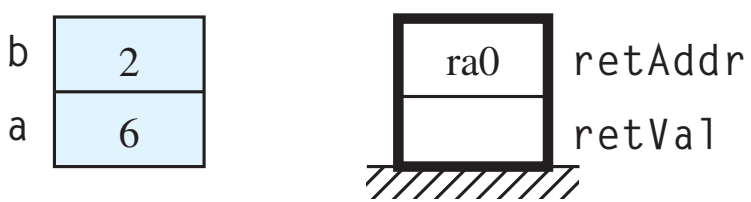
Figure 2.19
(Continued)

Interactive Input/Output

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

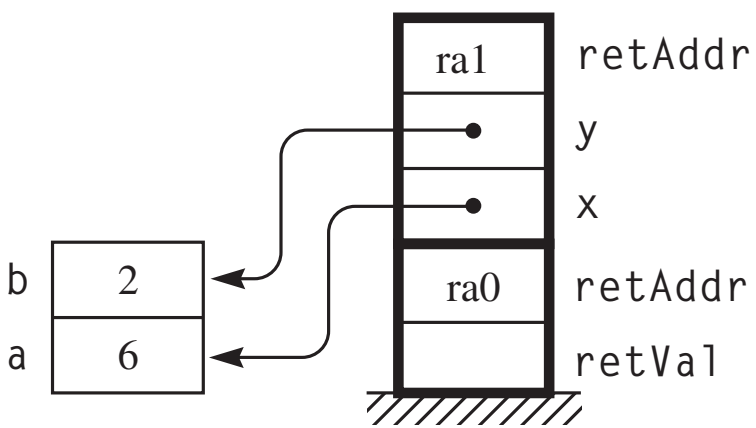


(a) Begin

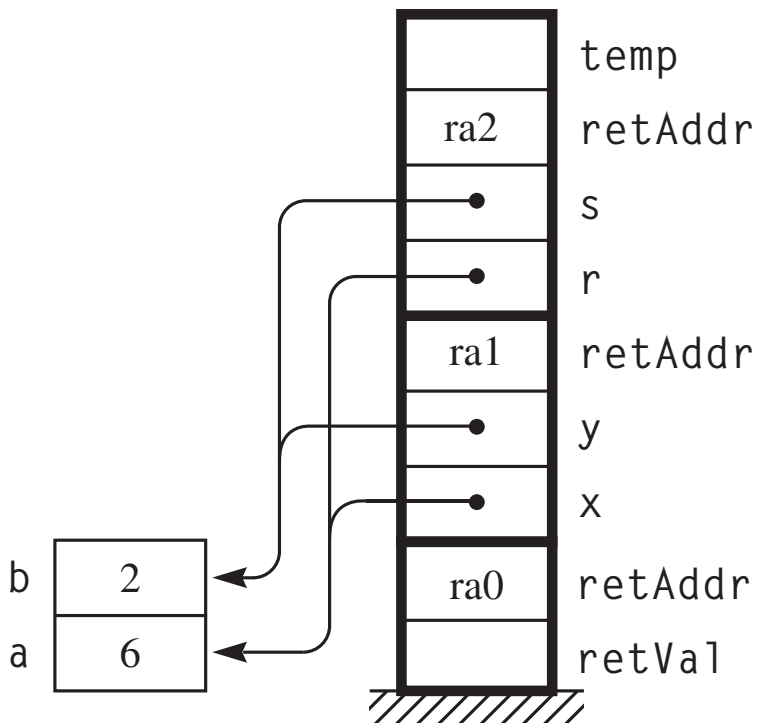


(b) *Input* a, b

Figure 2.21
(Continued)

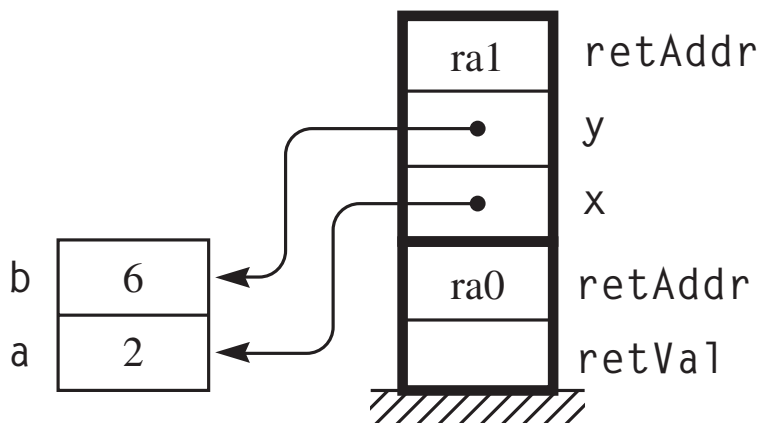


(c) `order(a,b)`

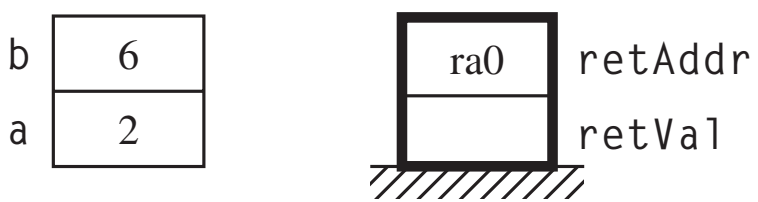


(d) `swap(x,y)`

Figure 2.21
(Continued)



(e) Return from `swap`



(f) Return from `order`

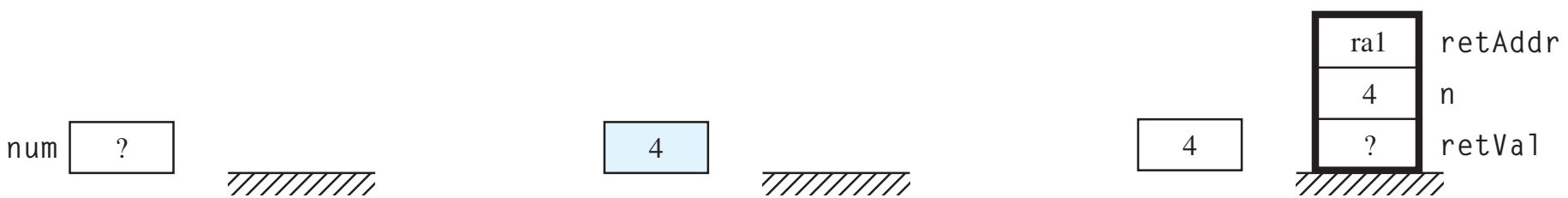
Figure 2.22

```
#include <iostream>
using namespace std;

int num;

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

int main () {
   cout << "Enter a small integer: ";
   cin >> num;
   cout << "Its factorial is: " << fact (num) << endl; // ra1
   return 0;
}
```

Interactive Input/Output

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

Figure 2.23

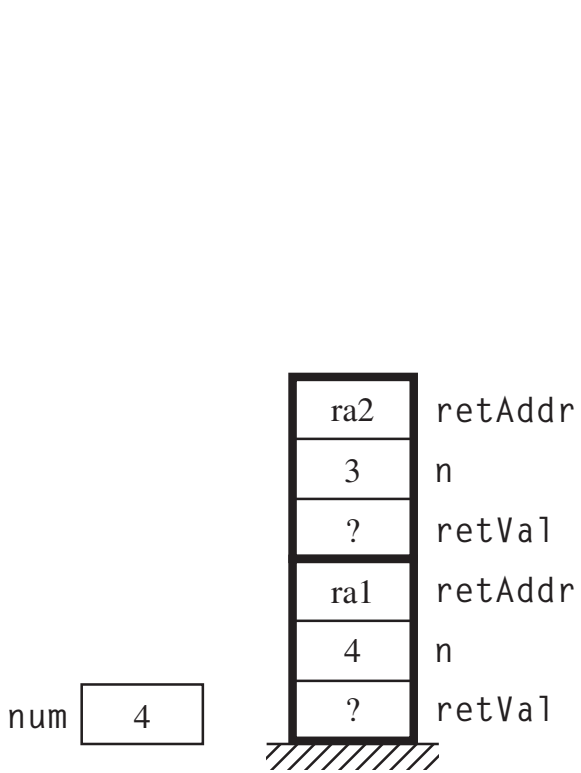


(a) Begin

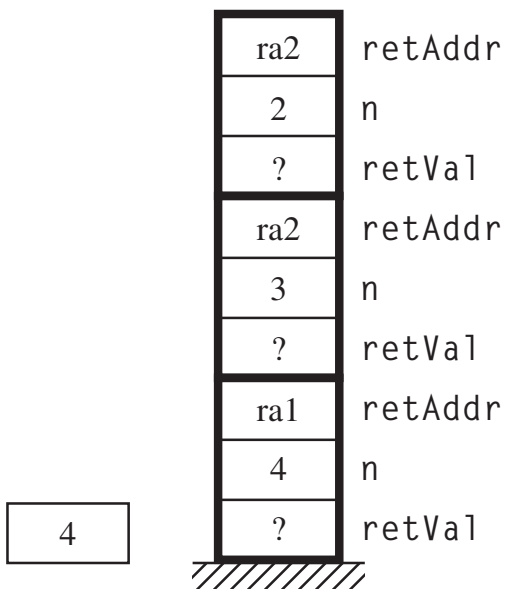
(b) `cin >> num`

(c) Call `fact` (4)

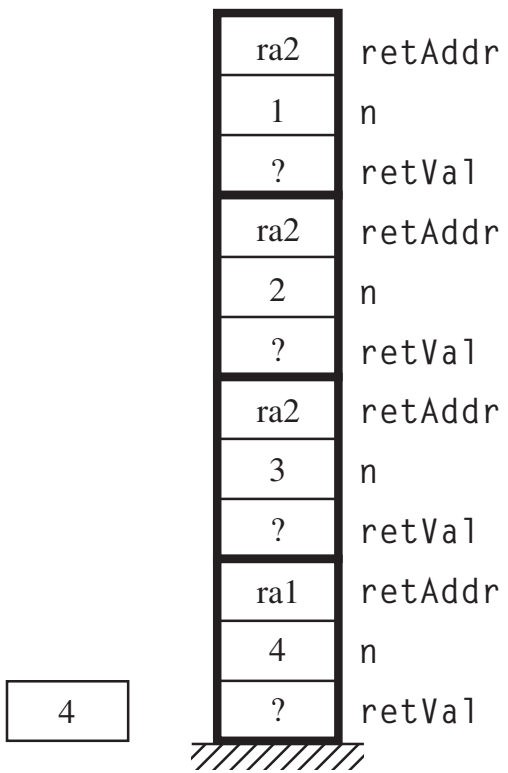
Figure 2.23
(Continued)



(d) Call `fact` (3)

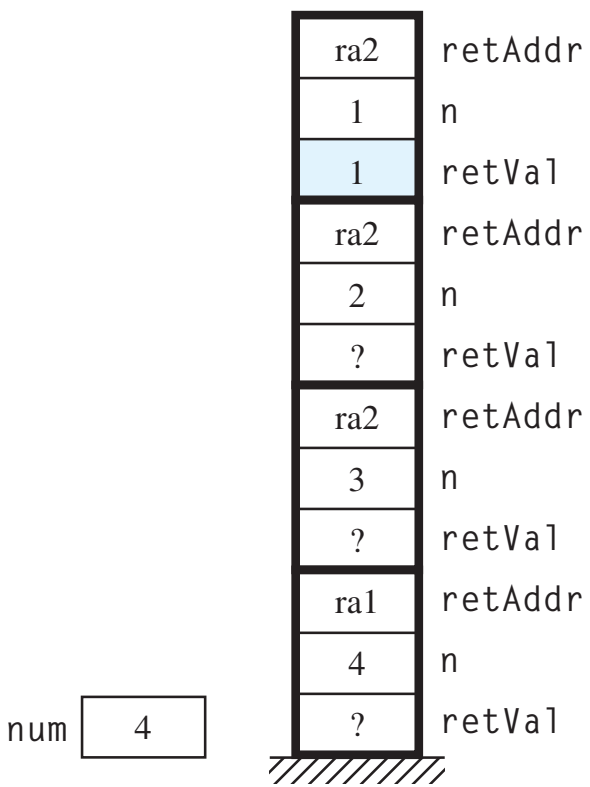


(e) Call `fact` (2)

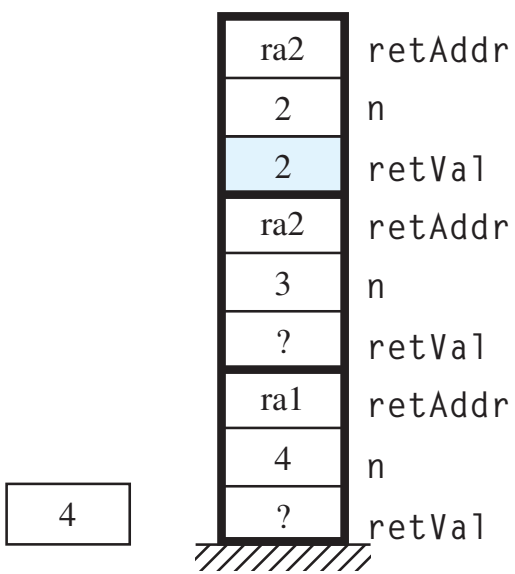


(f) Call `fact` (1)

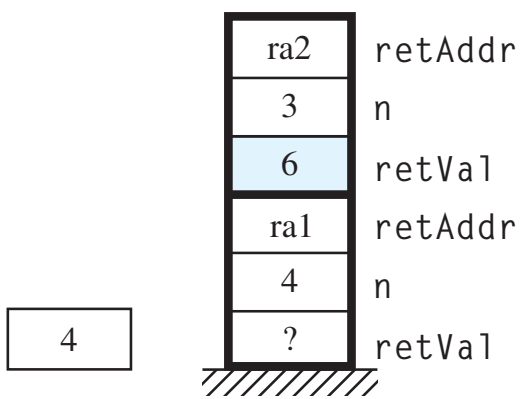
Figure 2.23
(Continued)



(g) Compute `retVal`

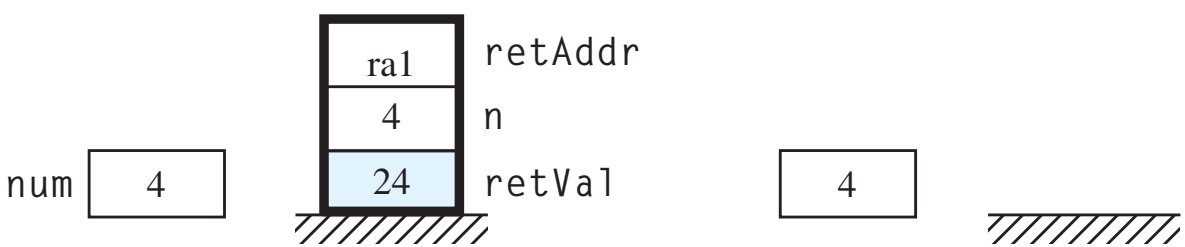


(h) Return



(i) Return

Figure 2.23
(Continued)



(j) Return **(k)** Return

Figure 2.24

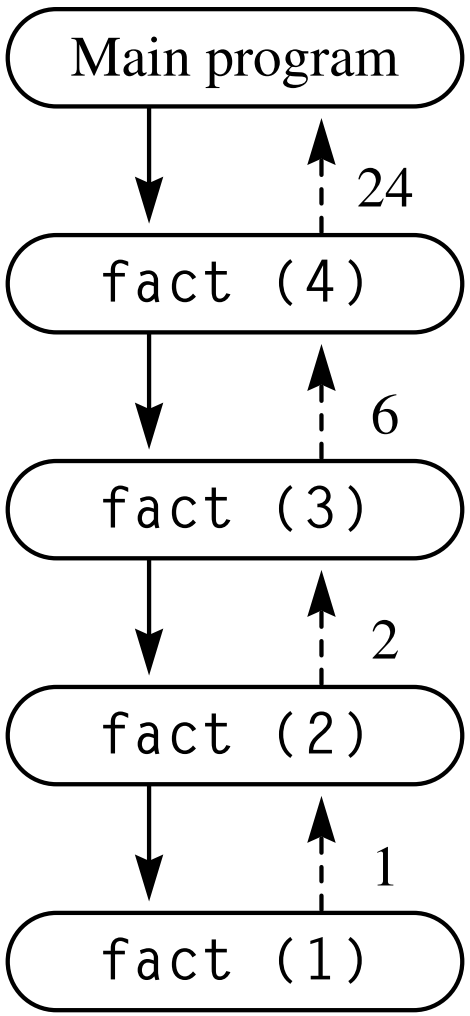


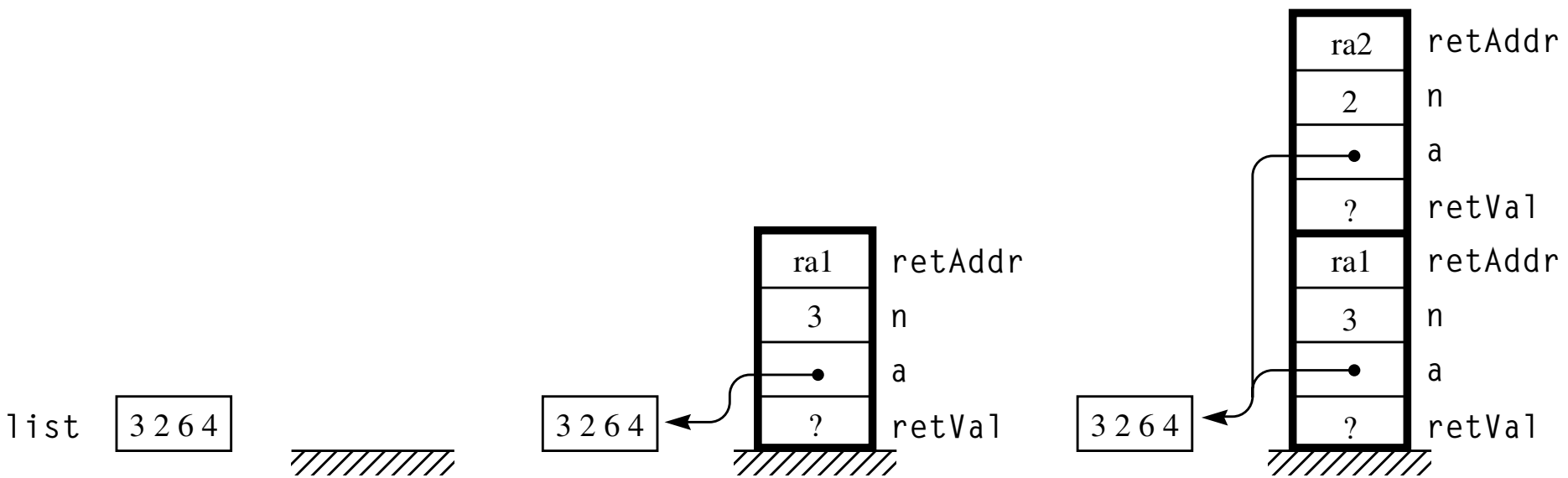
Figure 2.25

```
#include <iostream>
using namespace std;

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 () {
   cout << "Enter four integers: ";
   cin >> list[0] >> list[1] >> list[2] >> list[3];
   cout << "Their sum is: " << sum(list, 3) << endl; // ra1
   return 0;
}
```

Interactive Input/Output

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

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

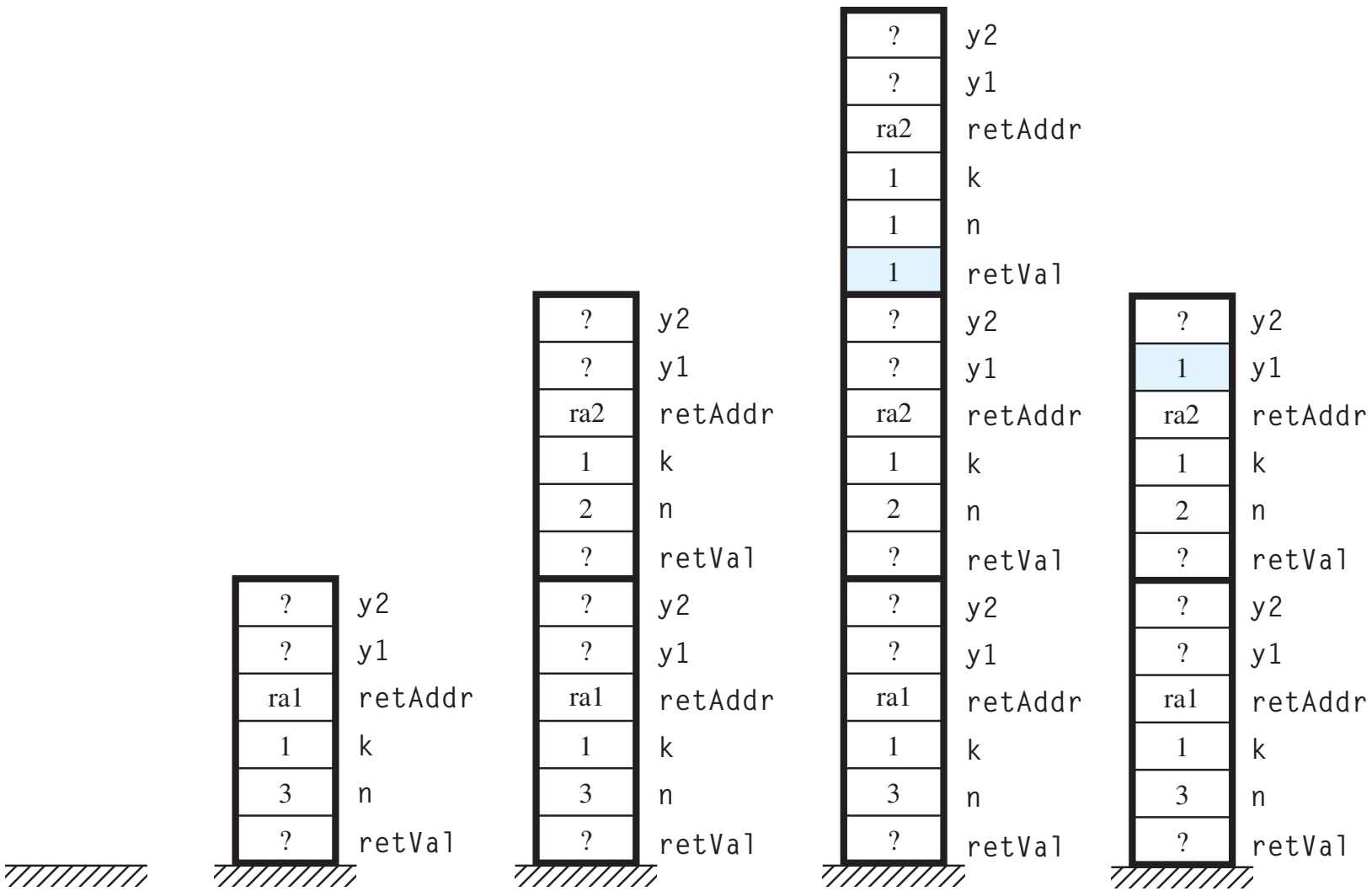
Figure 2.28

```
#include <iostream>
using namespace std;

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 () {
   cout << "binCoeff (3, 1) = " << binCoeff (3, 1); // ra1
   cout << endl;
   return 0;
}
```

Output

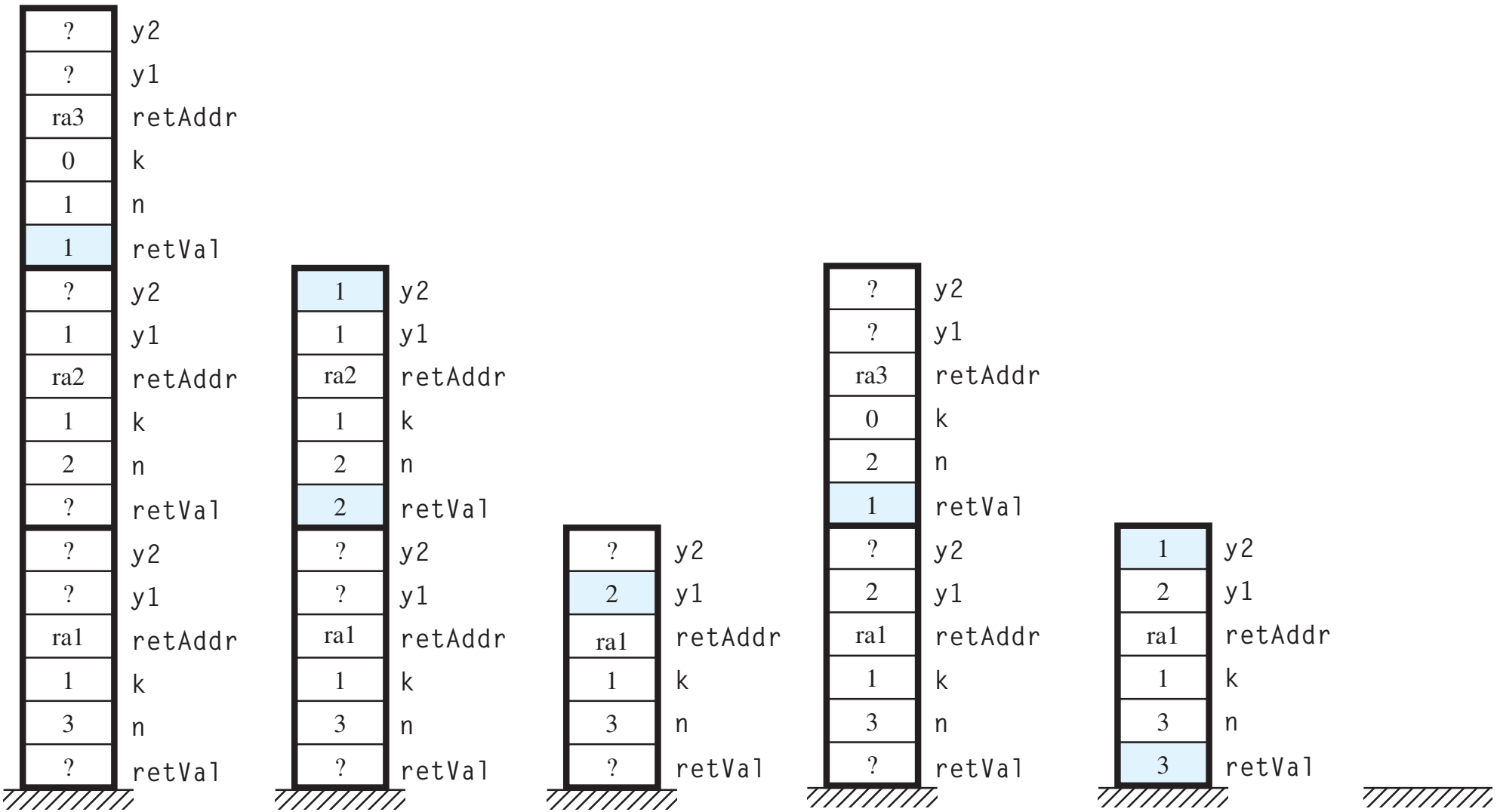
```
binCoeff(3, 1) = 3
```

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) **(g)** Return **(h)** Return **(i)** Call BC (2, 0) **(j)** Return **(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

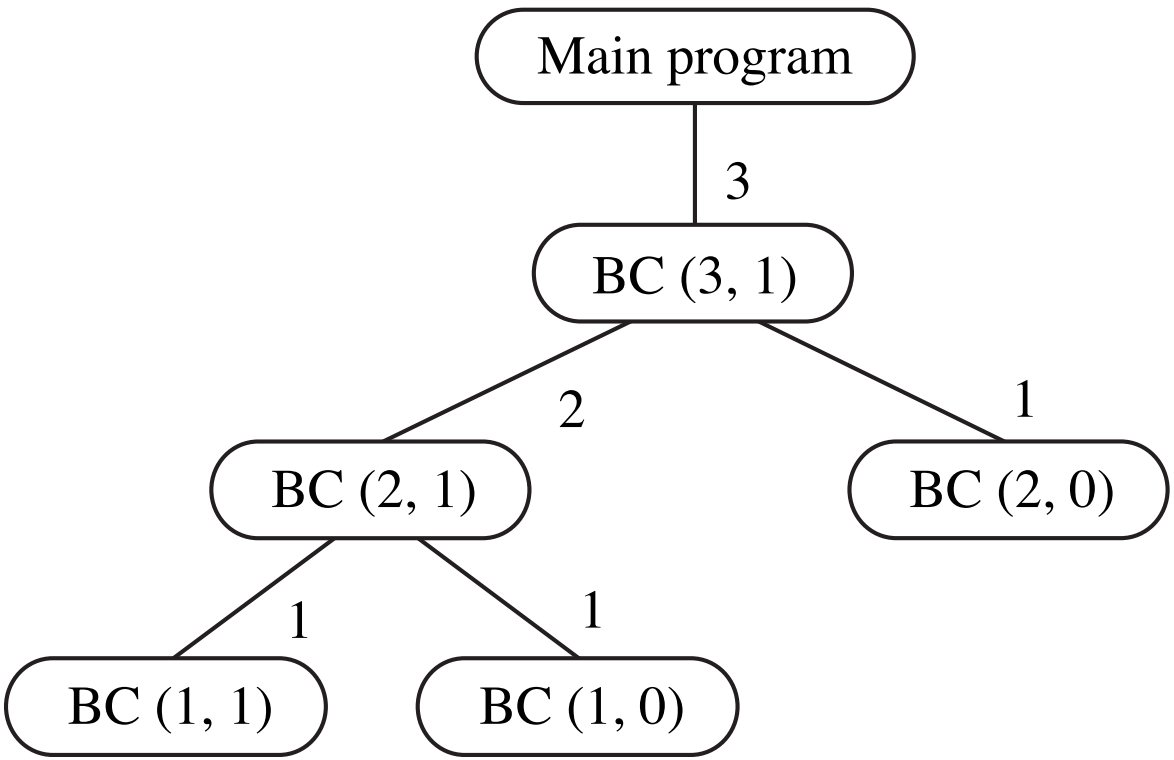


Figure 2.31

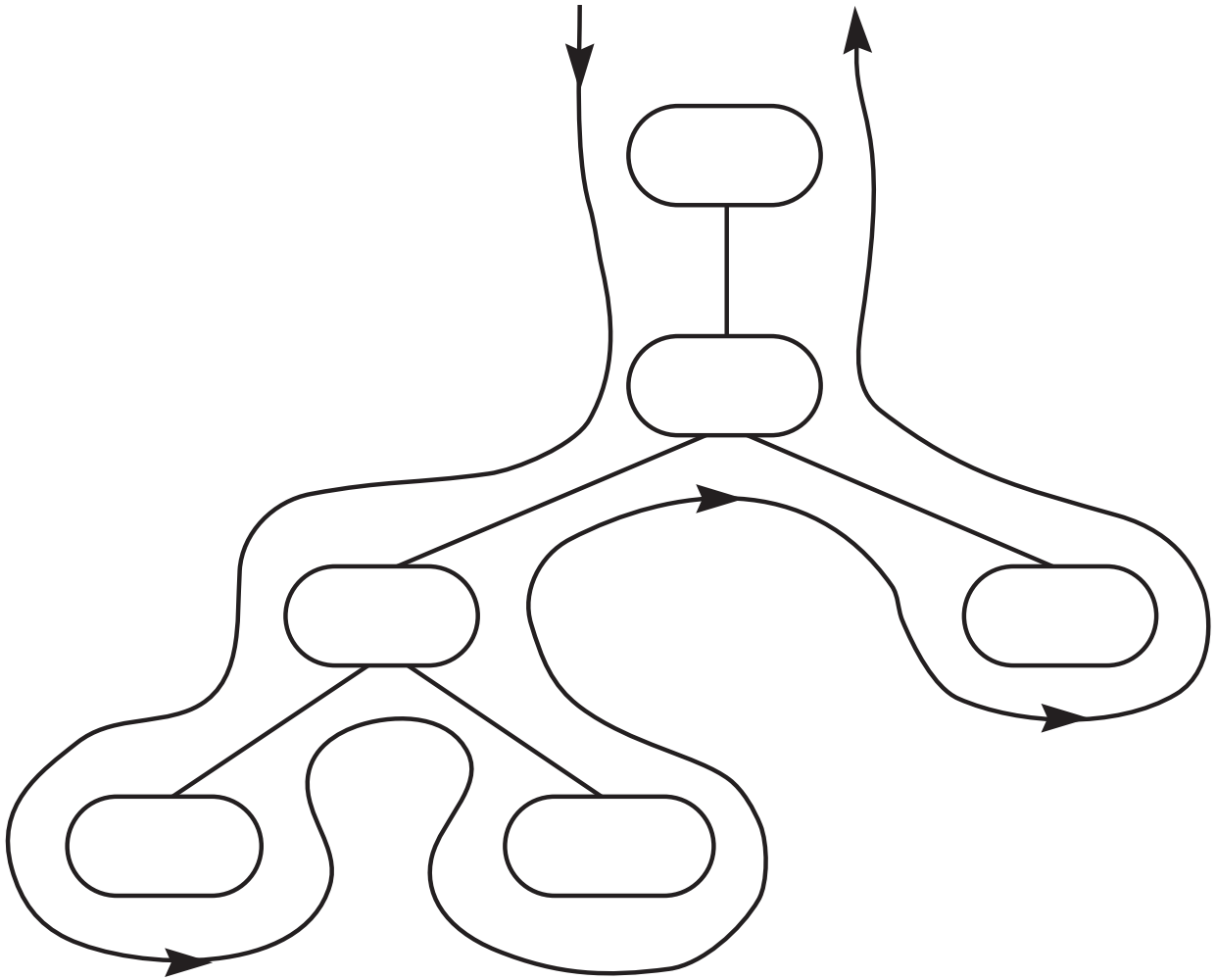


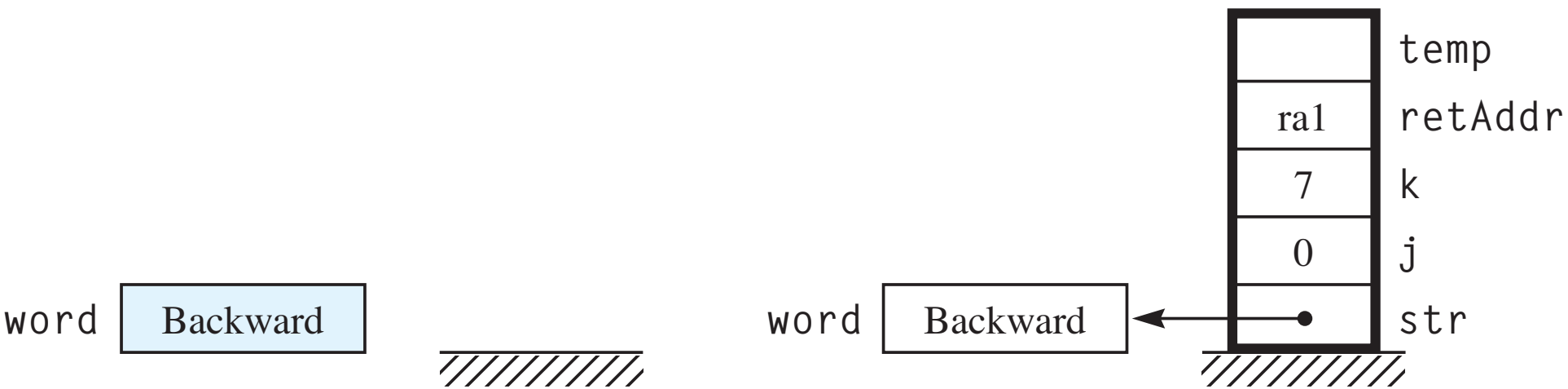
Figure 2.32

```
#include <iostream>
using namespace std;

char word[32] = "Backward";

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 () {
   reverse (word, 0, 7);
   cout << word << endl; // ra1
   return 0;
}
```

Output

```
drawkcaB
```

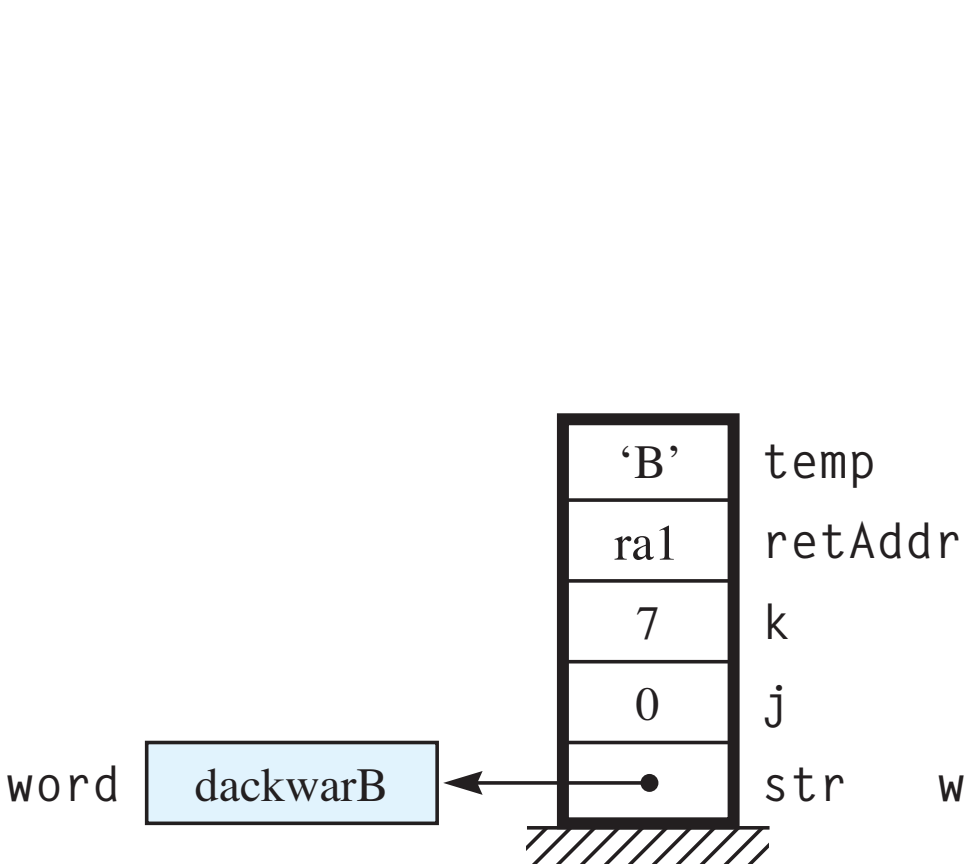
Figure 2.33



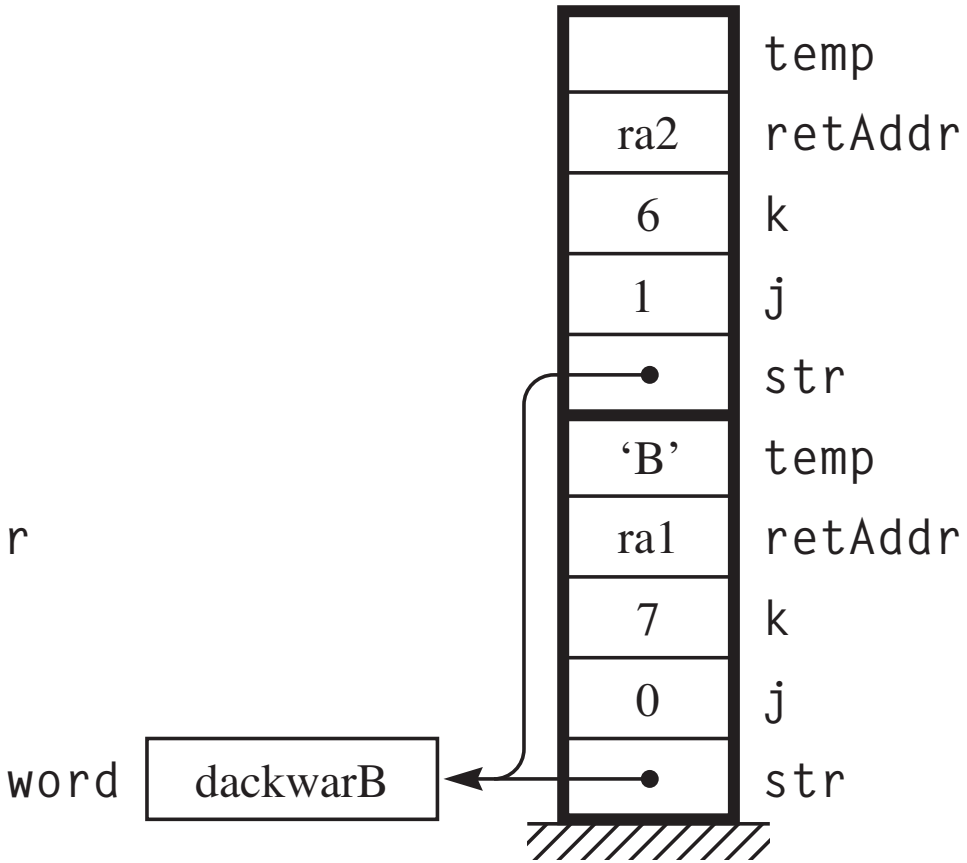
(a) `char word [32] = 'Backward'`

(b) Call reverse (word, 0, 7)

Figure 2.33
(Continued)



(c) Switch 'd' and 'B'



(d) Call reverse (word, 1, 6)

Figure 2.34

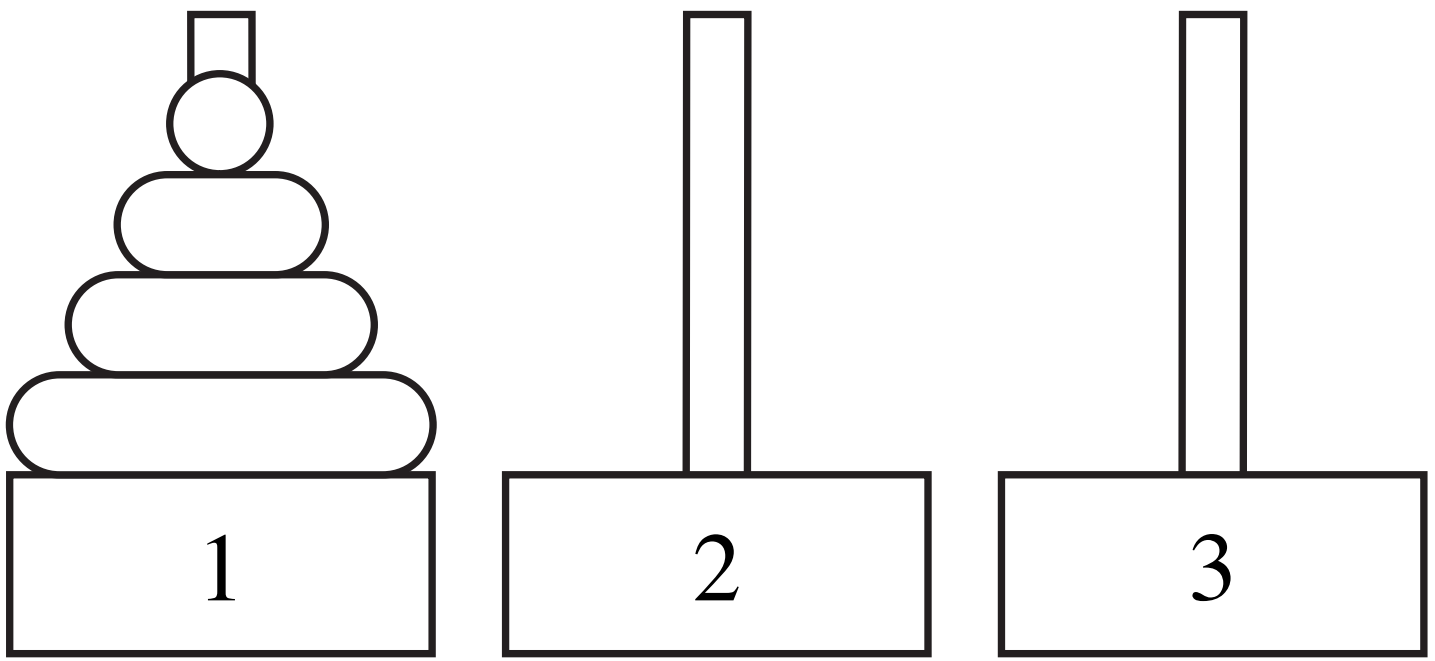
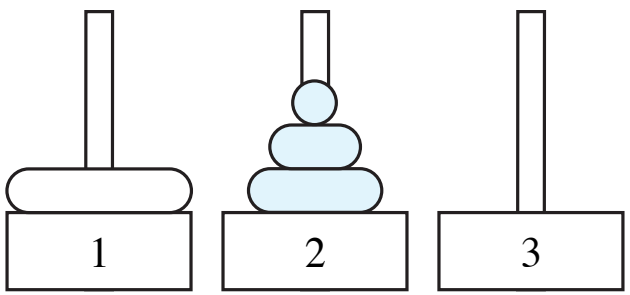
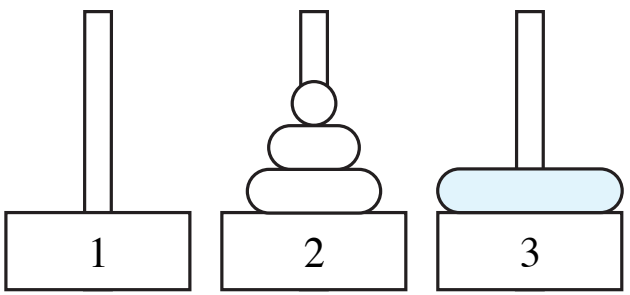


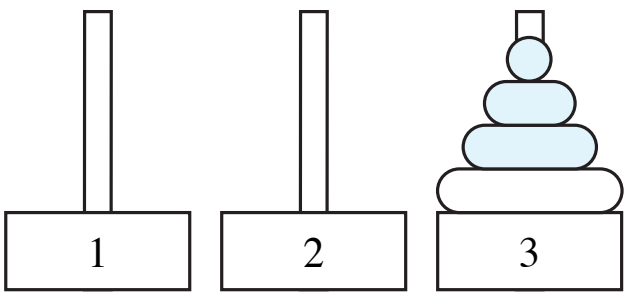
Figure 2.35



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

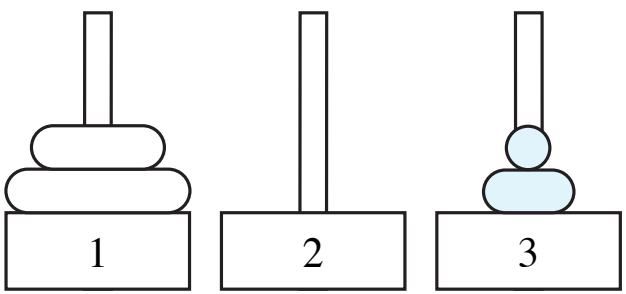


(b) Move one disk from peg 1 to peg 3.

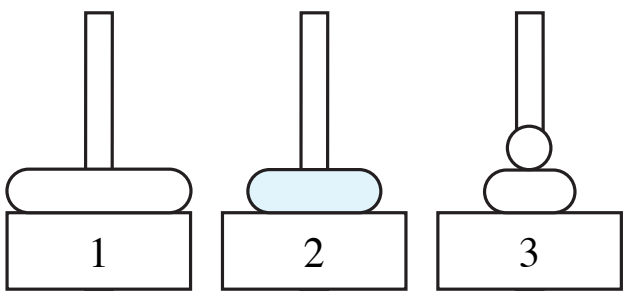


(c) Move three disks from peg 2 to peg 3.

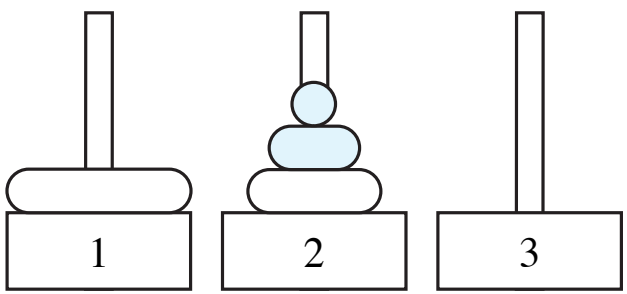
Figure 2.36



(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.

The C++ memory model

- Fixed locations in memory for global variables
- The run-time stack for local variables
- The heap for dynamically allocated variables

Two operators for dynamic memory allocation

- `new`, to allocate from the heap
- `delete`, to deallocate from the heap

Two actions of the new operator

- 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.

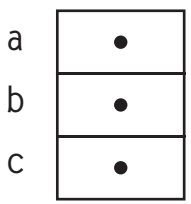
Figure 2.37

```
#include <iostream>
using namespace std;

int *a, *b, *c;

int main () {
   a = new int;
   *a = 5;
   b = new int;
   *b = 3;
   c = a;
   a = b;
   *a = 2 + *c;
   cout << "*a = " << *a << endl;
   cout << "*b = " << *b << endl;
   cout << "*c = " << *c << endl;
   return 0;
}
```

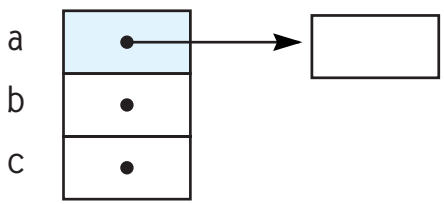
Output

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

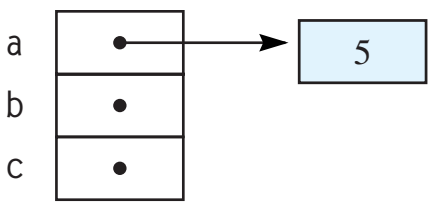
Figure 2.38



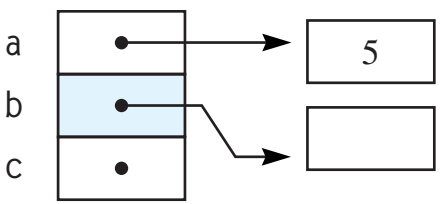
(a) Initial state.



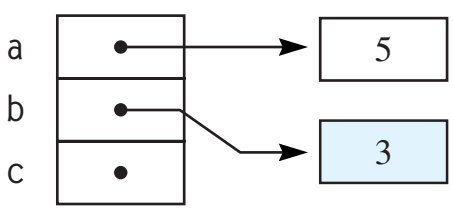
(b) `a = new int;`



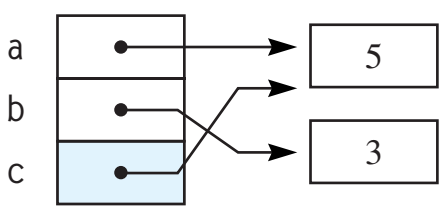
(c) `*a = 5;`



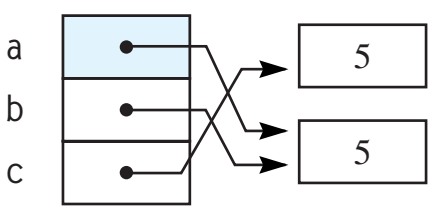
(d) `b = new int;`



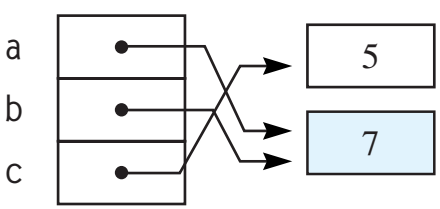
(e) `*b = 3`



(f) `c = a`



(g) `a = b;`



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

Figure 2.39

```
#include <iostream>
using namespace std;

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

int main () {
   cin >> bill.first >> bill.last >> bill.age >> bill.gender;
   cout << "Initials: " << bill.first << bill.last << endl;
   cout << "Age: " << bill.age << endl;
   cout << "Gender: ";
   if (bill.gender == 'm') {
      cout << "male\n";
   }
   else {
      cout << "female\n";
   }
   return 0;
}
```

Figure 2.39
(Continued)

Input

```
bj 32 m
```

Output

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

Figure 2.40

```
#include <iostream>
using namespace std;

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

int main () {
   node *first, *p;
   int value;
   first = 0;
   cin >> value;
   while (value != -9999) {
      p = first;
      first = new node;
      first->data = value;
      first->next = p;
      cin >> value;
   }
   for (p = first; p != 0; p = p->next) {
      cout << p->data << ' ';
   }
   return 0;
}
```

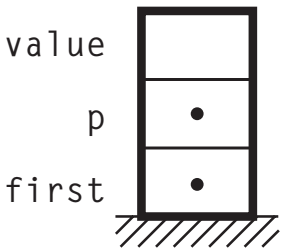
Figure 2.40
(Continued)

Input

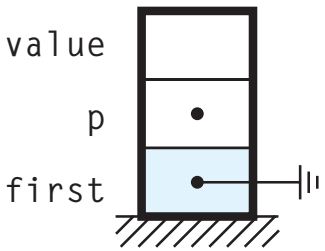
```
10 20 30 40 -9999
```

Output

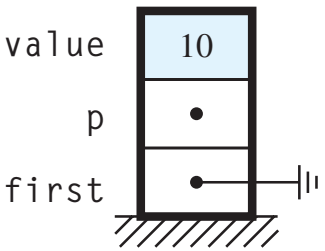
```
40 30 20 10
```



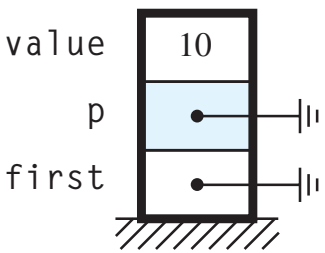
(a) Initial state in `main ( )`.



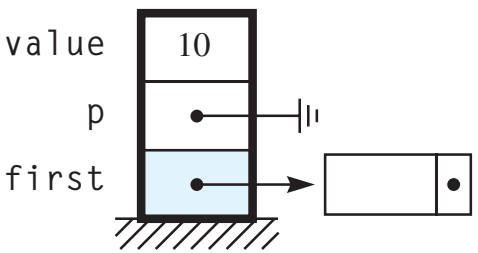
(b) `first = 0;`



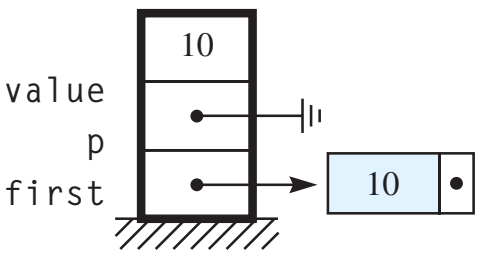
(c) `cin >> value;`



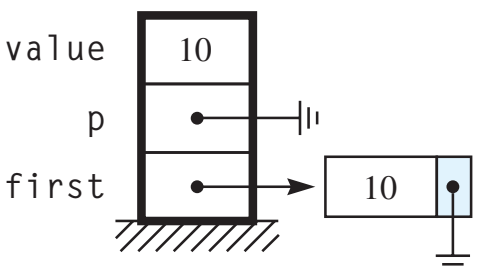
(d) `p = first;`



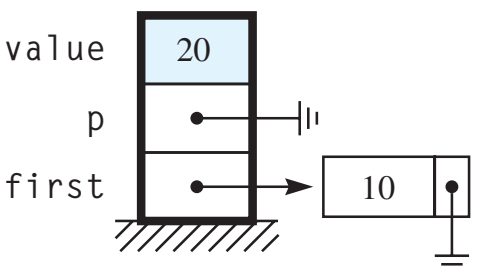
(e) `first = new node;`



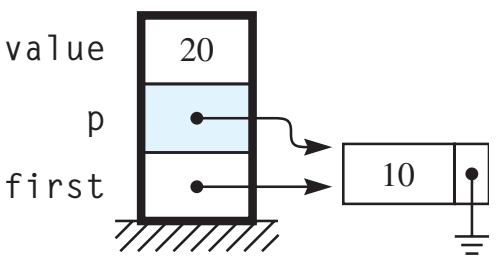
(f) `first->data = value;`



(g) `first->next = p;`

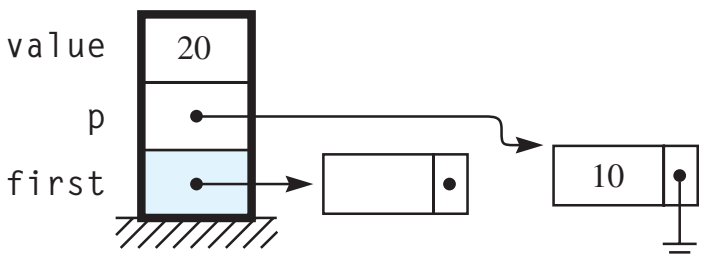


(h) `cin >> value;`

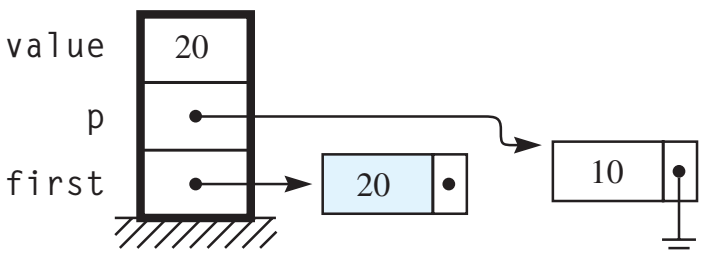


(i) `p = first;`

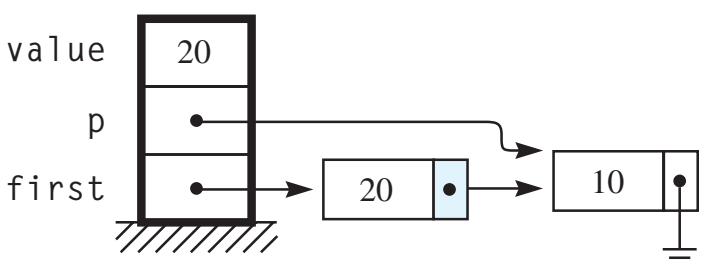
Figure 2.41
(Continued)



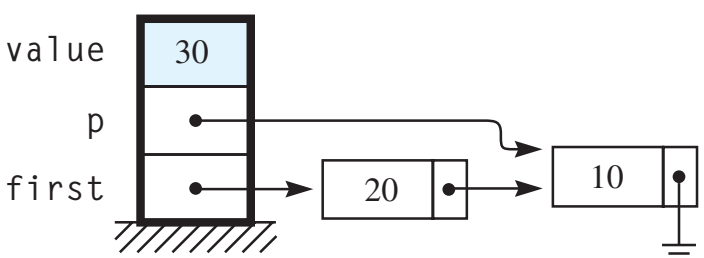
(j) `first = node node;`



(k) `first->data = value;`



(l) `first->next = p;`



(m) `cin >> value;`