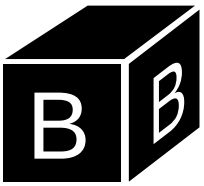
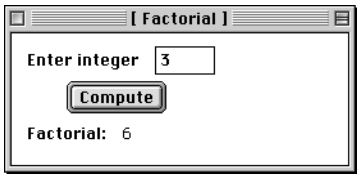


Chapter *19*

Recursion

$$\begin{cases} f(0) = 1 \\ f(n) = n\,f(n-1) & \text{for } n > 0 \end{cases}$$

A recursive definition of factorial



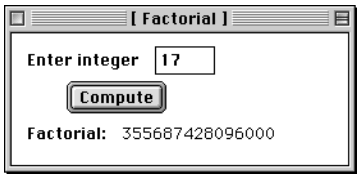


Figure 19.1
The dialog box for the factorial function of Figure 19.2.

```
MODULE Pbox19A;
    IMPORT Dialog;
    VAR
        d*: RECORD
            num*: INTEGER;
            factorial-: LONGINT
        END;

    PROCEDURE Factorial (n: INTEGER): LONGINT;
    BEGIN
        ASSERT((0 <= n) & (n <= 20), 20);
        IF n = 0 THEN
            RETURN 1
        ELSE
            RETURN n * Factorial(n - 1)  (* ra2 *)
        END
    END Factorial;
```

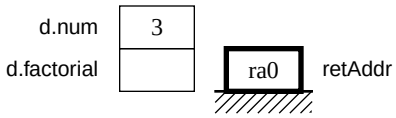
Figure 19.2
A program to compute the factorial recursively.

```
    PROCEDURE ComputeFactorial*;
    BEGIN
        IF d.num >= 0 THEN
            d.factorial := Factorial(d.num)  (* ra1 *)
        ELSE
            d.factorial := 0
        END;
        Dialog.Update(d)
    END ComputeFactorial;

BEGIN
    d.num := 0;
    d.factorial := 1
END Pbox19A.
```

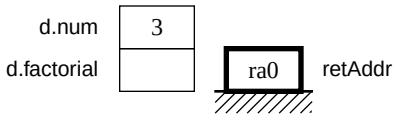
- Push storage for the returned value.
- Push the parameters.
- Push the return address.
- Push storage for the local variables.

Allocation for function procedures

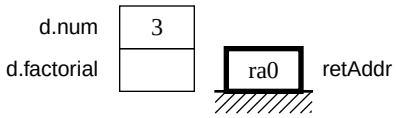
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



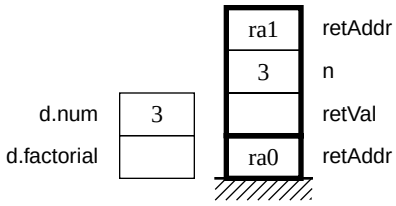
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



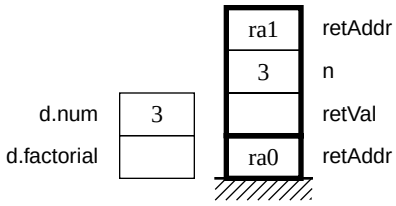
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



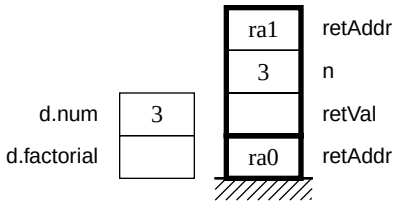
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



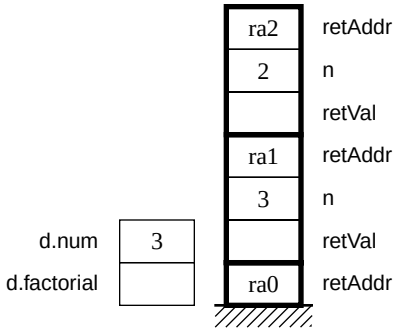
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



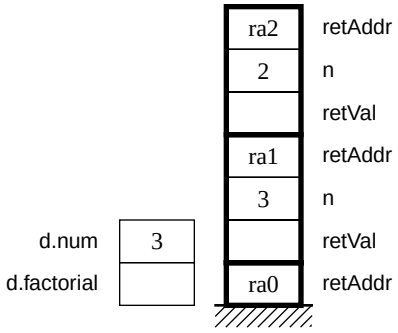
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



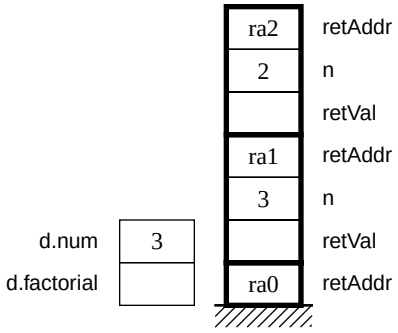
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



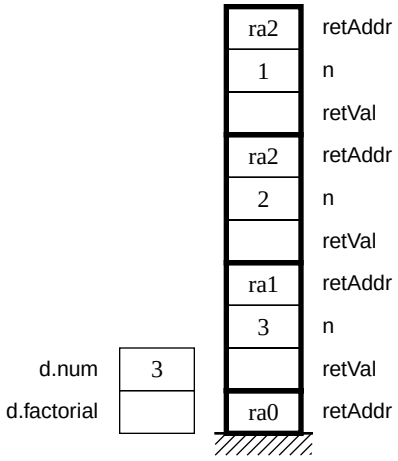
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



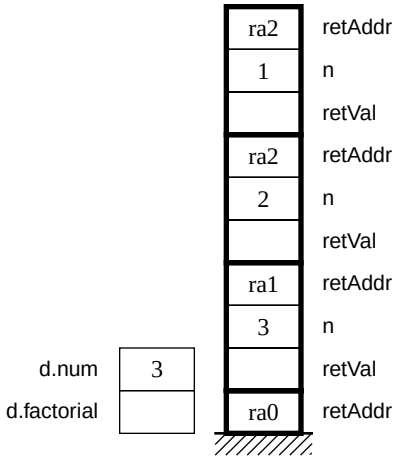
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



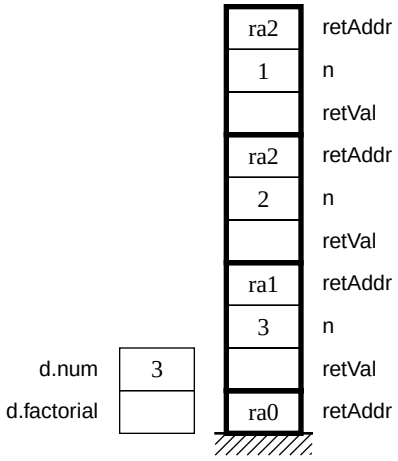
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



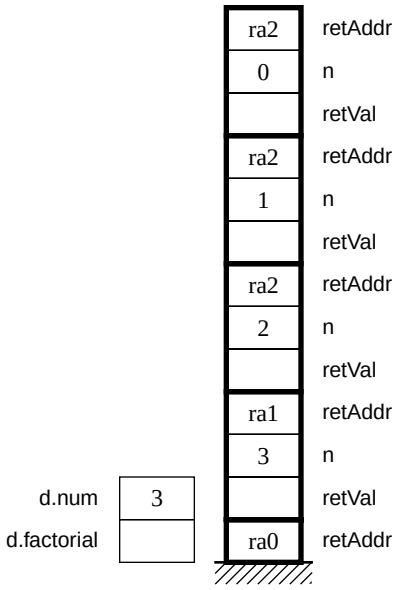
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



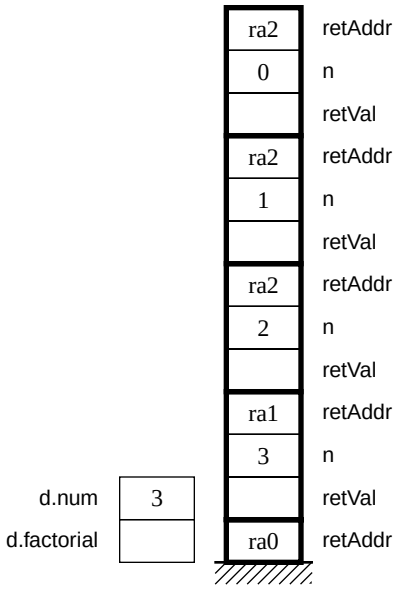
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



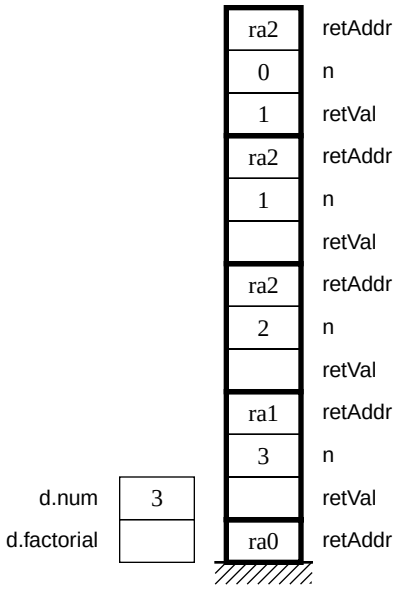
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



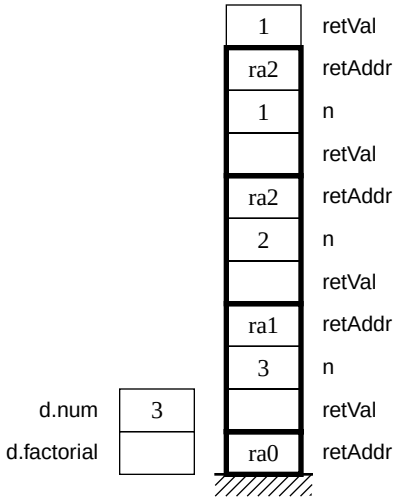
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



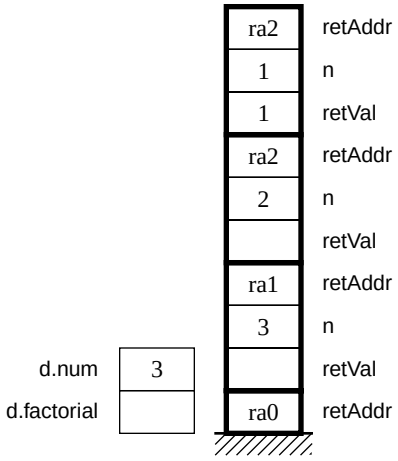
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



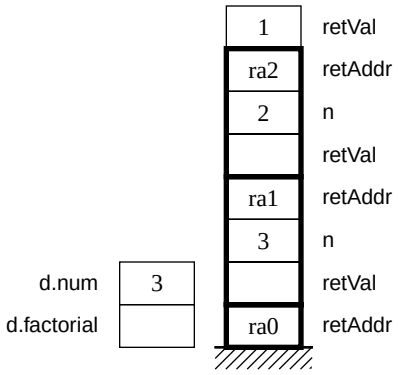
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



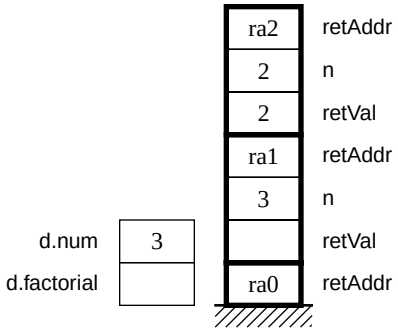
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



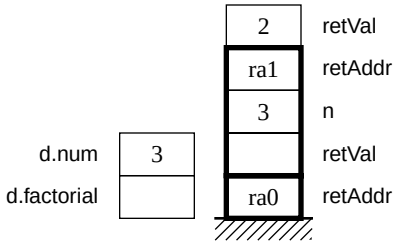
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



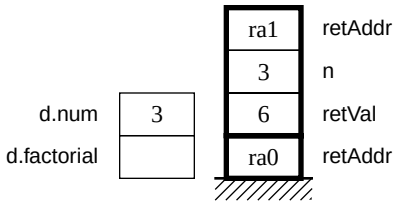
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



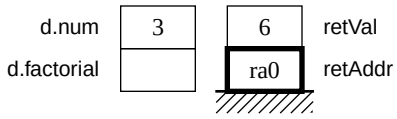
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



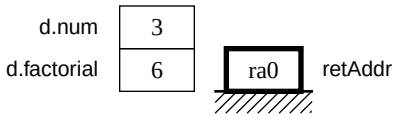
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
   ASSERT((0 <= n) & (n <= 20), 20);
   IF n = 0 THEN
      RETURN 1
   ELSE
      RETURN n * Factorial(n - 1)  (* ra2 *)
   END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
   IF d.num >= 0 THEN
      d.factorial := Factorial(d.num)  (* ra1 *)
   ELSE
      d.factorial := 0
   END;
   Dialog.Update(d)
END ComputeFactorial;
```



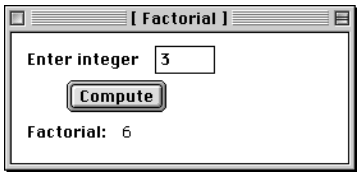
```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



```
PROCEDURE Factorial (n: INTEGER): LONGINT;
BEGIN
    ASSERT((0 <= n) & (n <= 20), 20);
    IF n = 0 THEN
        RETURN 1
    ELSE
        RETURN n * Factorial(n - 1)  (* ra2 *)
    END
END Factorial;

PROCEDURE ComputeFactorial*;
BEGIN
    IF d.num >= 0 THEN
        d.factorial := Factorial(d.num)  (* ra1 *)
    ELSE
        d.factorial := 0
    END;
    Dialog.Update(d)
END ComputeFactorial;
```



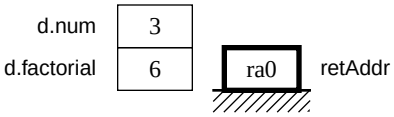
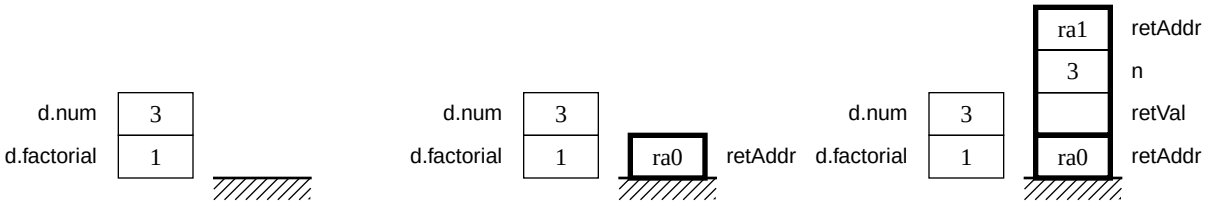
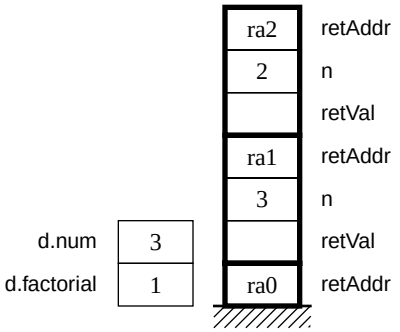


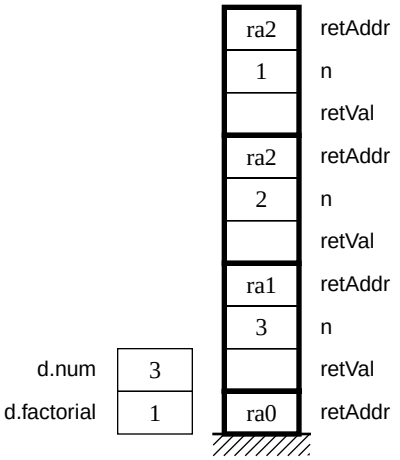
Figure 19.3
The run-time stack for Figure 19.2.



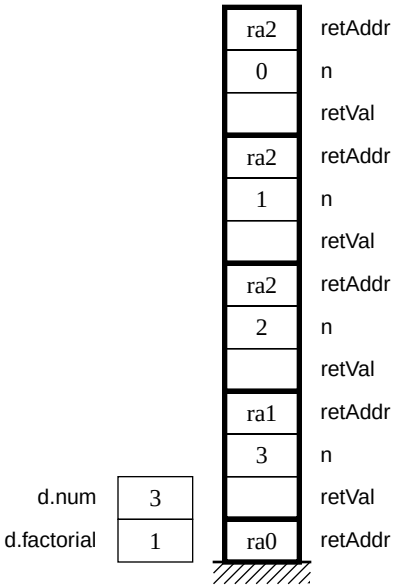
(a) After user enters d.num **(b)** Call ComputeFactorial) **(c)** Call Factorial(3)



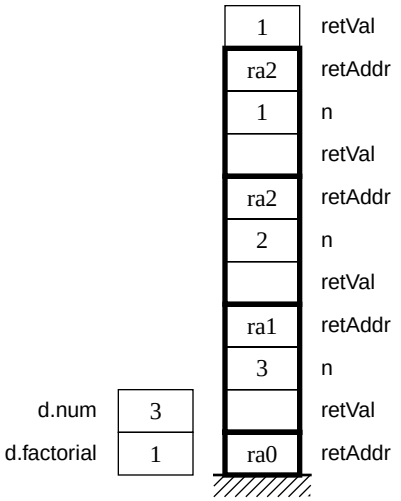
(d) Call Factorial(2)



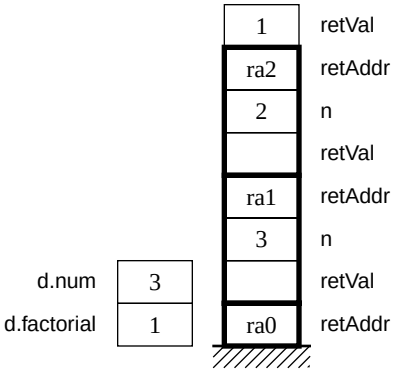
(e) Call Factorial(1)



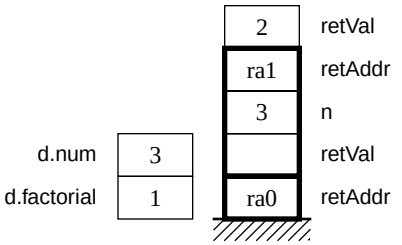
(f) Call Factorial(0)



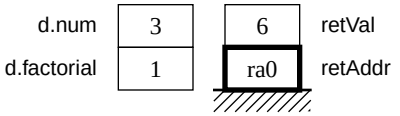
(g) Return



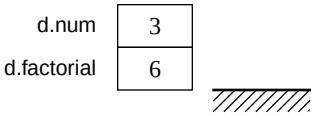
(h) Return



(i) Return



(j) Return



(k) Return

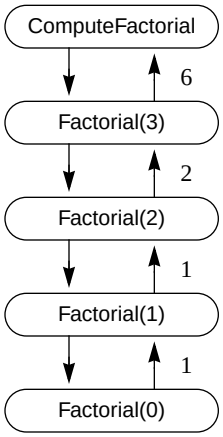


Figure 19.4
The calling sequence for Figure 19.2.

Figure 19.5
The input and output for the module in Figure 19.6.

```
PROCEDURE Sum (IN v: ARRAY OF INTEGER; n: INTEGER): INTEGER;
    (* Returns the sum of the first n elements of v *)
BEGIN
    ASSERT (n >= 0, 20);
```

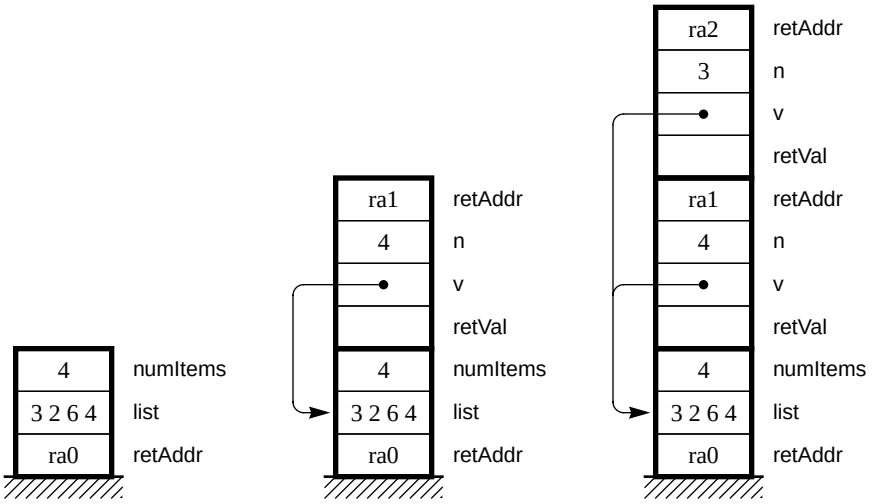
```
PROCEDURE Sum (IN v: ARRAY OF INTEGER; n: INTEGER): INTEGER;
   (* Returns the sum of the first n elements of v *)
BEGIN
   ASSERT (n >= 0, 20);
   IF n = 0 THEN
      RETURN 0
   ELSE
```

```
PROCEDURE Sum (IN v: ARRAY OF INTEGER; n: INTEGER): INTEGER;
    (* Returns the sum of the first n elements of v *)
BEGIN
    ASSERT (n >= 0, 20);
    IF n = 0 THEN
        RETURN 0
    ELSE
        RETURN v[n - 1] + Sum(v, n - 1)  (* ra2 *)
    END
END Sum;
```

```
MODULE Pbox19B;
    IMPORT TextControllers, PboxMappers, StdLog;

    PROCEDURE Sum (IN v: ARRAY OF INTEGER; n: INTEGER): INTEGER;
        (* Returns the sum of the first n elements of v *)
    BEGIN
        ASSERT (n >= 0, 20);
        IF n = 0 THEN
            RETURN 0
        ELSE
            RETURN v[n - 1] + Sum(v, n - 1)  (* ra2 *)
        END
    END Sum;
```

Figure 19.6
A recursive function that returns the sum of the first n numbers in an array.

```
    PROCEDURE ComputeSum*;
        VAR
            cn: TextControllers.Controller;
            sc: PboxMappers.Scanner;
            list: ARRAY 1024 OF INTEGER;
            numItems: INTEGER;
    BEGIN
        cn := TextControllers.Focus();
        IF cn # NIL THEN
            sc.ConnectTo(cn.text);
            sc.ScanIntVector(list, numItems);
            StdLog.String("Sum = ");
            StdLog.Int(Sum(list, numItems));  (* ra1 *)
            StdLog.Ln
        END
    END ComputeSum;

END Pbox19B.
```



(a) Call ComputeSum **(b)** Call Sum(v, 4) **(c)** Call Sum(v, 3)

Figure 19.7
The run-time stack for Figure 19.6.

$$gcd(m, n) = \begin{cases} m & \text{if } n = 0 \\ gcd(n, m \bmod n) & \text{if } n > 0 \end{cases}$$

- m div n is the quotient of $m \div n$.
- m mod n is the remainder of $m \div n$.

Let

$q = m \text{ div } n$
$r = m \bmod n$

Then the relationship between div and mod is expressed mathematically as

$$m = q \cdot n + r \qquad 0 \le r < n$$

$$(x+y)^1 = x+y$$
$$(x+y)^2 = x^2+2xy+y^2$$
$$(x+y)^3 = x^3+3x^2y+3xy^2+y^3$$
$$(x+y)^4 = x^4+4x^3y+6x^2y^2+4xy^3+y^4$$

$$(x+y)^1 = x + y$$
$$(x+y)^2 = x^2 + 2xy + y^2$$
$$(x+y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$$
$$(x+y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$$

	Term number, k							
Power, n	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 19.8
Pascal's triangle.

$$\begin{cases} b(n, 0) = 1 \\ b(k, k) = 1 \\ b(n, k) = b(n-1, k) + b(n-1, k-1) & 0 < k < n \end{cases}$$

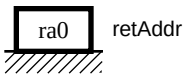
A recursive definition of the binomial coefficient

```
MODULE Pbox19C;
   IMPORT StdLog;

   PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
      VAR
         y1, y2: INTEGER;
   BEGIN
      ASSERT((0 <= k) & (k <= n), 20);
      IF (0 = k) OR (k = n) THEN
         RETURN 1
      ELSE
         y1 := BinomCoeff(n - 1, k);  (* ra2 *)
         y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
         RETURN y1 + y2
      END
   END BinomCoeff;

   PROCEDURE ComputeBinomCoeff*;
   BEGIN
      StdLog.String("BinomCoeff(3, 1) = ");
      StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
      StdLog.Ln
   END ComputeBinomCoeff;

END Pbox19C.
```

Figure 19.9
A recursive computation of the binomial coefficient.

```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

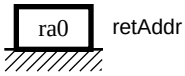
Log



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

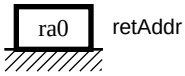
```
BinomCoeff(3, 1) =
```



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

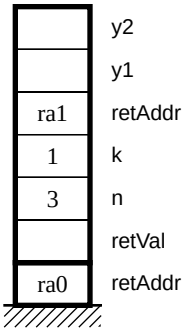
```
BinomCoeff(3, 1) =
```



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

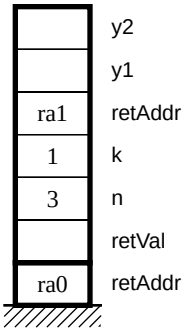
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

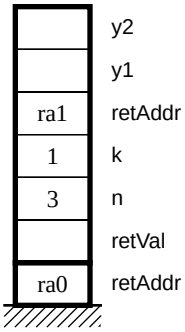
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

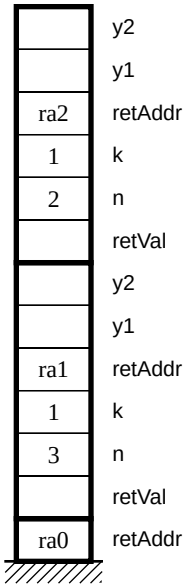
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

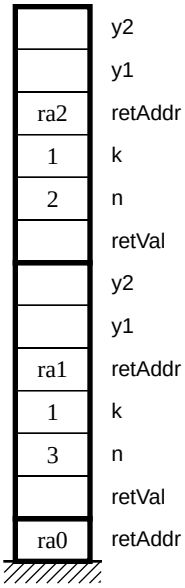
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

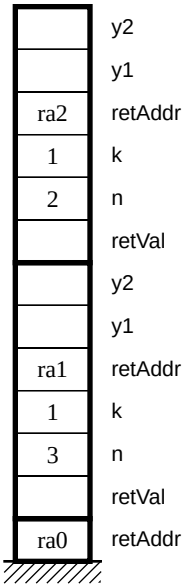
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

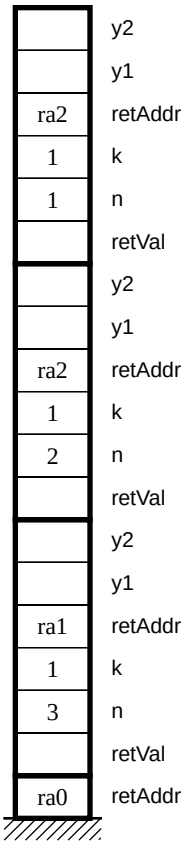
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
   VAR
      y1, y2: INTEGER;
BEGIN
   ASSERT((0 <= k) & (k <= n), 20);
   IF (0 = k) OR (k = n) THEN
      RETURN 1
   ELSE
      y1 := BinomCoeff(n - 1, k);  (* ra2 *)
      y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
      RETURN y1 + y2
   END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
   StdLog.String("BinomCoeff(3, 1) = ");
   StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
   StdLog.Ln
END ComputeBinomCoeff;
```

Log

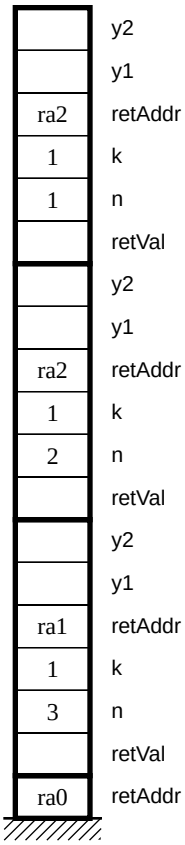
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
   VAR
      y1, y2: INTEGER;
BEGIN
   ASSERT((0 <= k) & (k <= n), 20);
   IF (0 = k) OR (k = n) THEN
      RETURN 1
   ELSE
      y1 := BinomCoeff(n - 1, k);  (* ra2 *)
      y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
      RETURN y1 + y2
   END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
   StdLog.String("BinomCoeff(3, 1) = ");
   StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
   StdLog.Ln
END ComputeBinomCoeff;
```

Log

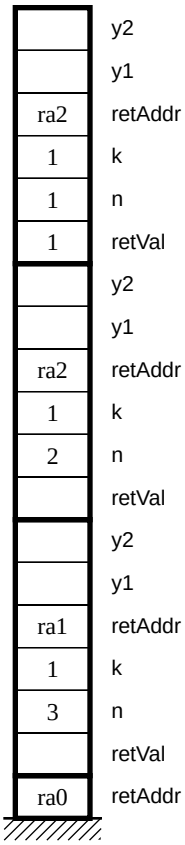
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

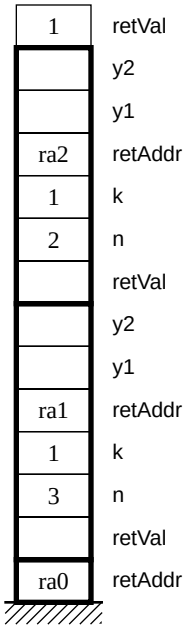
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

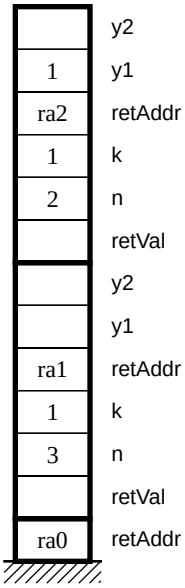
```
BinomCoeff(3, 1) =
```



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

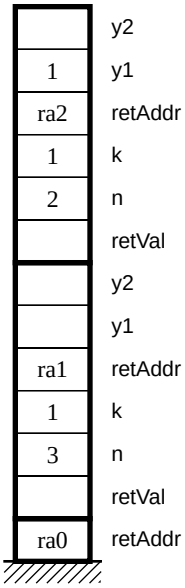
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

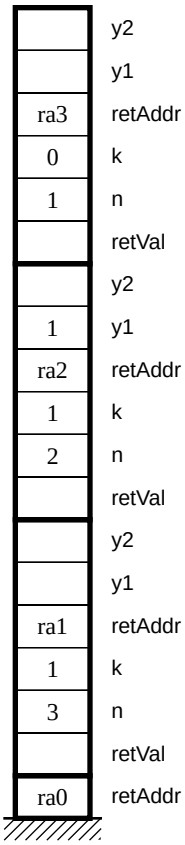
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
   VAR
      y1, y2: INTEGER;
BEGIN
   ASSERT((0 <= k) & (k <= n), 20);
   IF (0 = k) OR (k = n) THEN
      RETURN 1
   ELSE
      y1 := BinomCoeff(n - 1, k);  (* ra2 *)
      y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
      RETURN y1 + y2
   END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
   StdLog.String("BinomCoeff(3, 1) = ");
   StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
   StdLog.Ln
END ComputeBinomCoeff;
```

Log

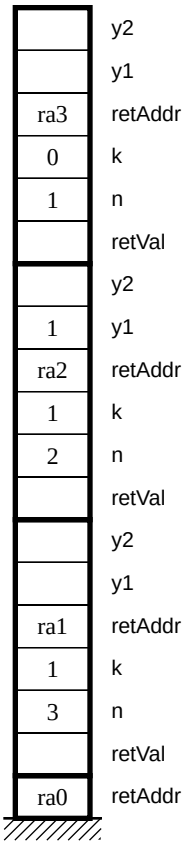
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
   VAR
      y1, y2: INTEGER;
BEGIN
   ASSERT((0 <= k) & (k <= n), 20);
   IF (0 = k) OR (k = n) THEN
      RETURN 1
   ELSE
      y1 := BinomCoeff(n - 1, k);  (* ra2 *)
      y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
      RETURN y1 + y2
   END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
   StdLog.String("BinomCoeff(3, 1) = ");
   StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
   StdLog.Ln
END ComputeBinomCoeff;
```

Log

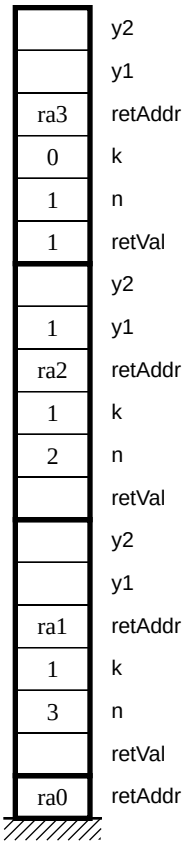
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
   VAR
      y1, y2: INTEGER;
BEGIN
   ASSERT((0 <= k) & (k <= n), 20);
   IF (0 = k) OR (k = n) THEN
      RETURN 1
   ELSE
      y1 := BinomCoeff(n - 1, k);  (* ra2 *)
      y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
      RETURN y1 + y2
   END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
   StdLog.String("BinomCoeff(3, 1) = ");
   StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
   StdLog.Ln
END ComputeBinomCoeff;
```

Log

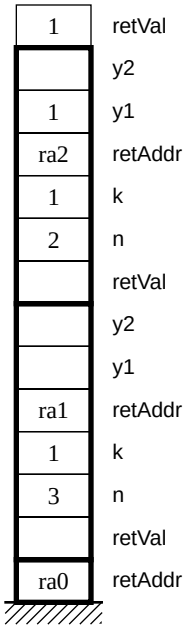
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

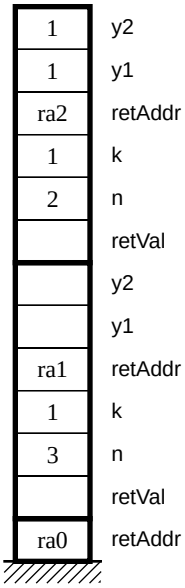
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

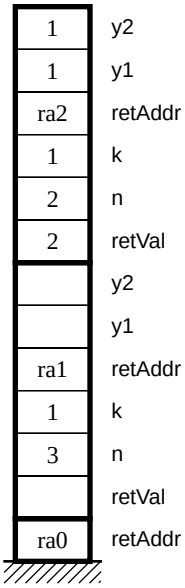
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

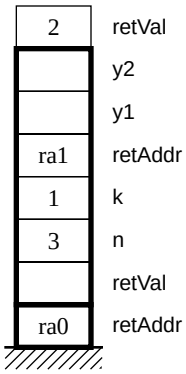
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

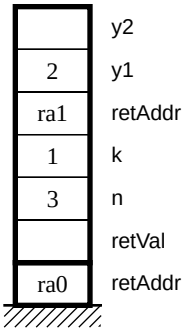
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

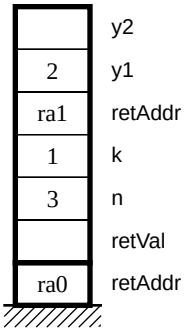
```
BinomCoeff(3, 1) =
```



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

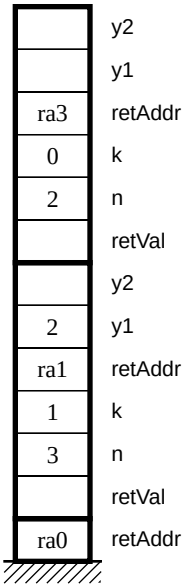
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

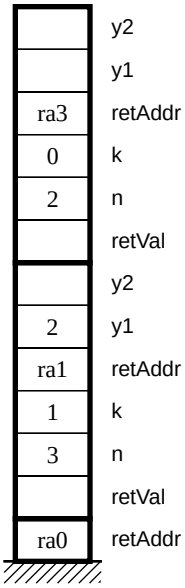
```
BinomCoeff(3, 1) =
```



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

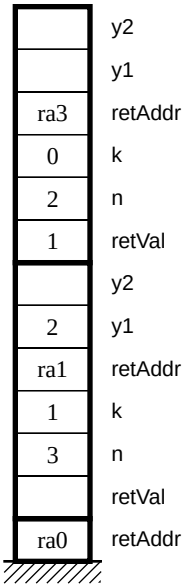
```
BinomCoeff(3, 1) =
```



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

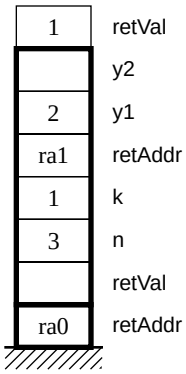
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

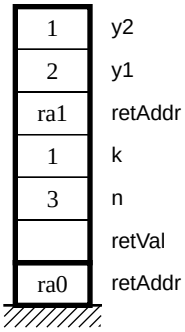
```
BinomCoeff(3, 1) =
```



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

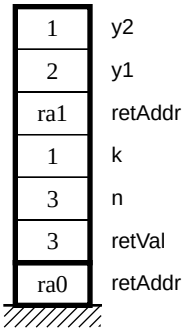
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

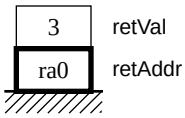
BinomCoeff(3, 1) =



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

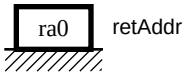
```
BinomCoeff(3, 1) =
```



```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

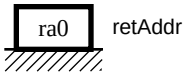
```
BinomCoeff(3, 1) = 3
```

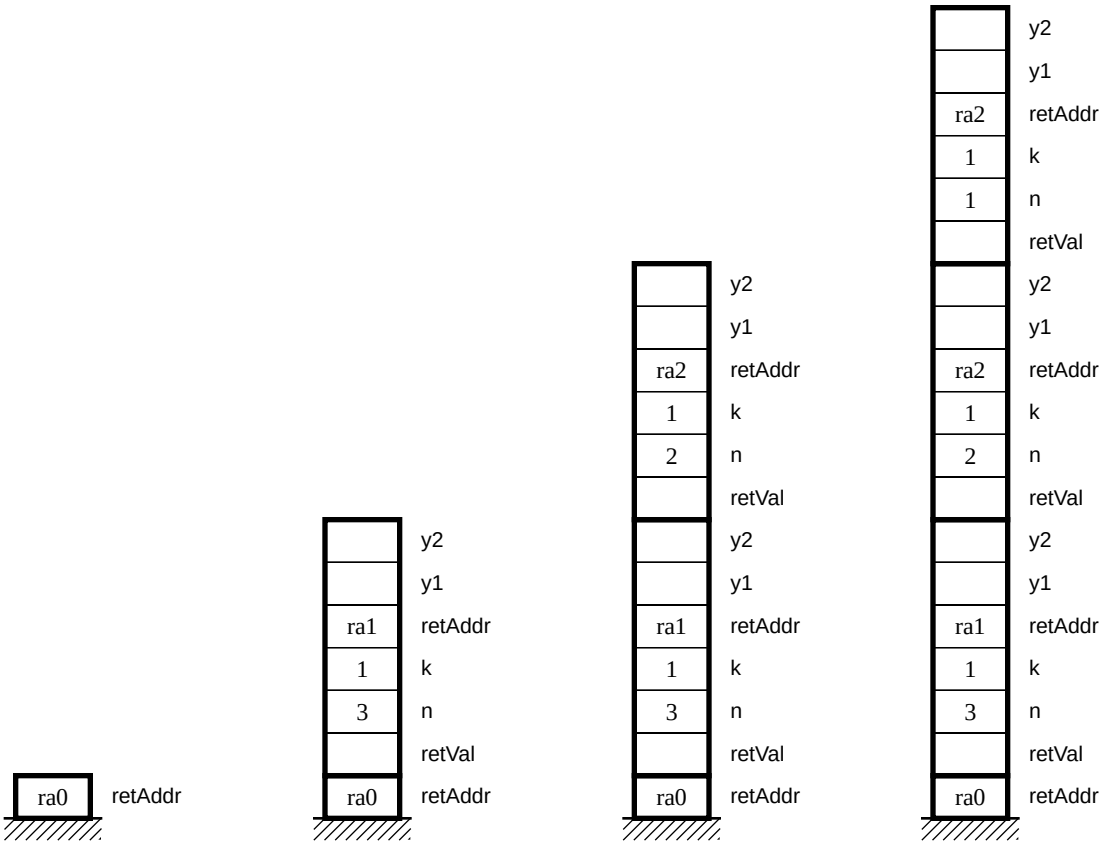


```
PROCEDURE BinomCoeff (n, k: INTEGER): INTEGER;
    VAR
        y1, y2: INTEGER;
BEGIN
    ASSERT((0 <= k) & (k <= n), 20);
    IF (0 = k) OR (k = n) THEN
        RETURN 1
    ELSE
        y1 := BinomCoeff(n - 1, k);  (* ra2 *)
        y2 := BinomCoeff(n - 1, k - 1);  (* ra3 *)
        RETURN y1 + y2
    END
END BinomCoeff;

PROCEDURE ComputeBinomCoeff*;
BEGIN
    StdLog.String("BinomCoeff(3, 1) = ");
    StdLog.Int(BinomCoeff(3, 1));  (* ra1 *)
    StdLog.Ln
END ComputeBinomCoeff;
```

Log

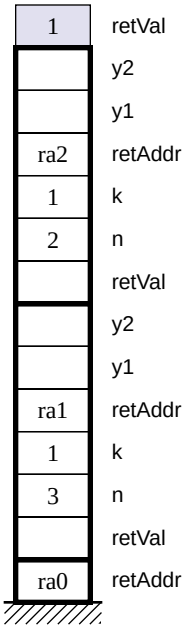
```
BinomCoeff(3, 1) = 3
```



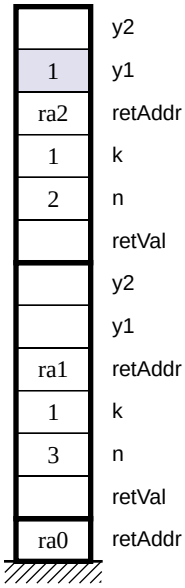


(a) Call ComputeBC **(b)** Call BC(3,1) **(c)** Call BC(2, 1) **(d)** Call BC(1, 1)

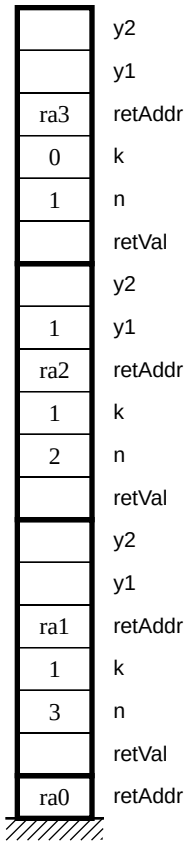
Figure 19.10
The run-time stack for Figure 19.9. BC stands for BinomCoeff.



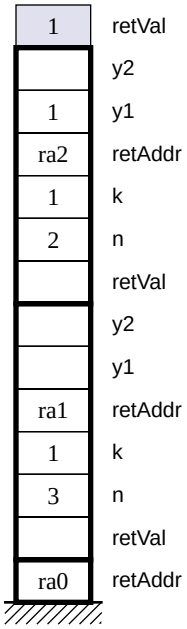
(e) Return



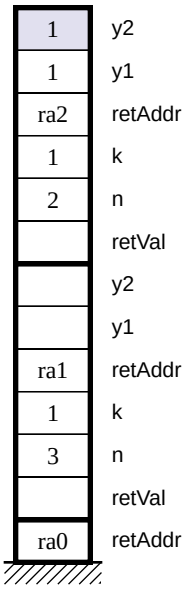
(f) y1 := BC(1, 1)



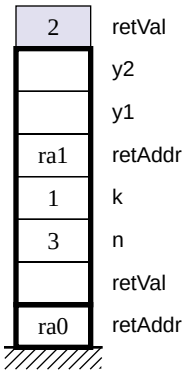
(g) Call BC(1, 0)



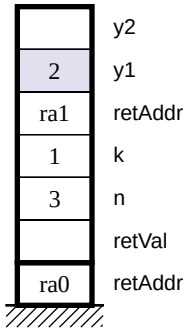
(h) Return



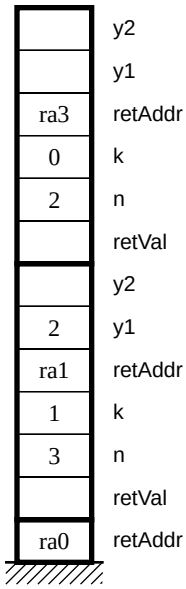
(i) y2 := BC(1, 0)



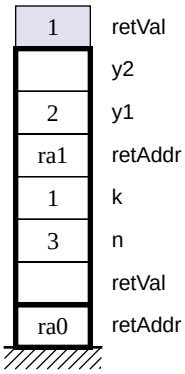
(j) Return



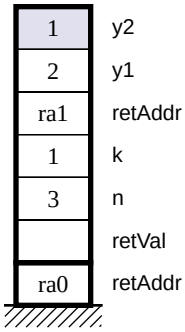
(k) y1 := BC(2, 1)



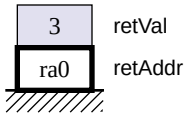
(l) Call BC(2, 0)



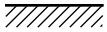
(m) Return



(n) y2 := BC(2, 0)



(o) Return



(p) Return

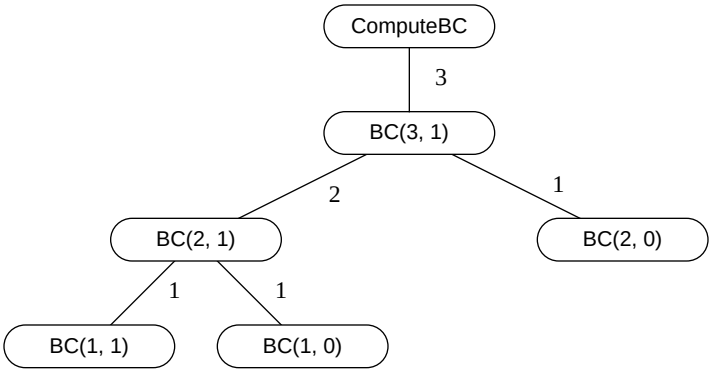


Figure 19.11
The call tree for Figure 19.9. The numbers next to the branches represent the value returned by the function.

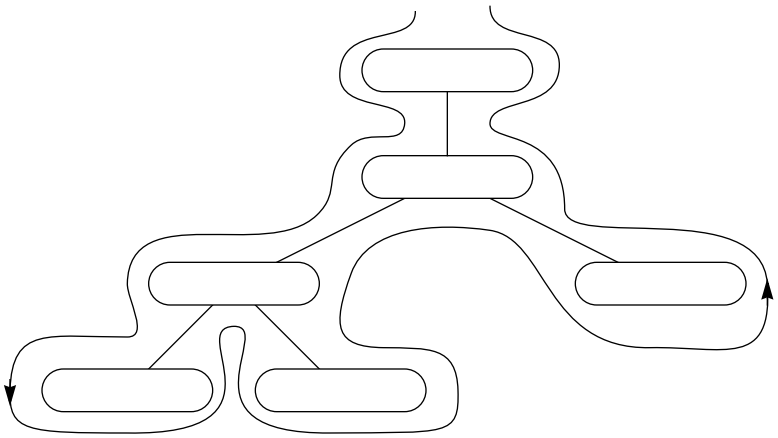


Figure 19.12
A visualization of the calling sequence for the call tree of Figure 19.11.

```
PROCEDURE Reverse (VAR str: ARRAY OF CHAR; first, last: INTEGER);
    (* Reverses the characters between str[first] and str[last] *)
    VAR
        temp: CHAR;
BEGIN
    ASSERT((0 <= first) & (last < LEN(str$)), 20);
```

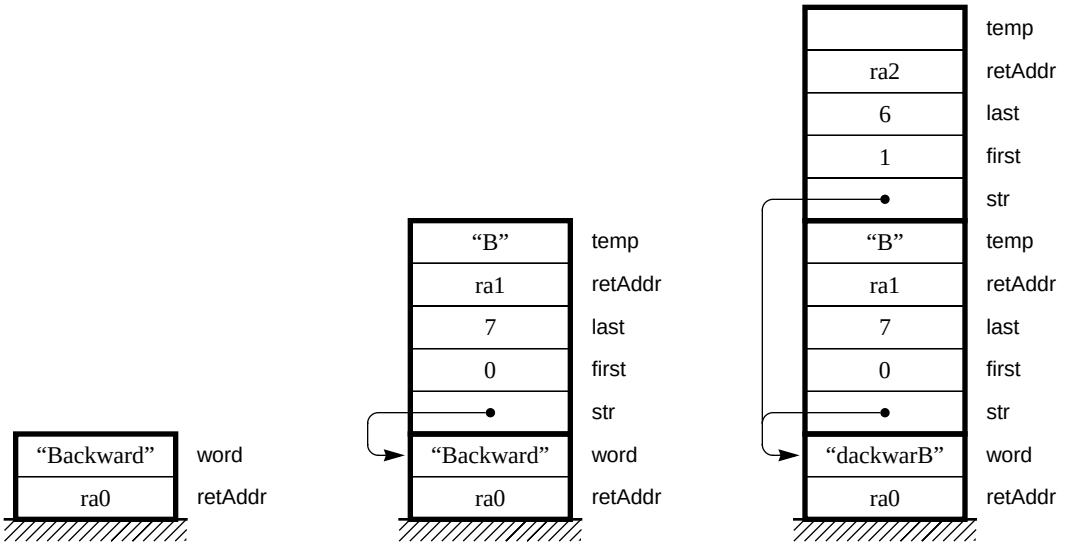
```
PROCEDURE Reverse (VAR str: ARRAY OF CHAR; first, last: INTEGER);
    (* Reverses the characters between str[first] and str[last] *)
    VAR
        temp: CHAR;
BEGIN
    ASSERT((0 <= first) & (last < LEN(str$)), 20);
    IF first < last THEN
```

```
PROCEDURE Reverse (VAR str: ARRAY OF CHAR; first, last: INTEGER);
    (* Reverses the characters between str[first] and str[last] *)
    VAR
        temp: CHAR;
BEGIN
    ASSERT((0 <= first) & (last < LEN(str$)), 20);
    IF first < last THEN
        temp := str[first];
        str[first] := str[last];
        str[last] := temp;
```

```
PROCEDURE Reverse (VAR str: ARRAY OF CHAR; first, last: INTEGER);
    (* Reverses the characters between str[first] and str[last] *)
    VAR
        temp: CHAR;
BEGIN
    ASSERT((0 <= first) & (last < LEN(str$)), 20);
    IF first < last THEN
        temp := str[first];
        str[first] := str[last];
        str[last] := temp;
        Reverse(str, first + 1, last - 1)
    END  (* ra2 *)
END Reverse;
```

```
MODULE Pbox19D;
   IMPORT StdLog;

   PROCEDURE Reverse (VAR str: ARRAY OF CHAR; first, last: INTEGER);
      (* Reverses the characters between str[first] and str[last] *)
      VAR
         temp: CHAR;
   BEGIN
      ASSERT((0 <= first) & (last < LEN(str$)), 20);
      IF first < last THEN
         temp := str[first];
         str[first] := str[last];
         str[last] := temp;
         Reverse(str, first + 1, last - 1)
      END  (* ra2 *)
   END Reverse;

   PROCEDURE ComputeReverse*;
      VAR
         word: ARRAY 16 OF CHAR;
   BEGIN
      word := "Backward";
      Reverse(word, 0, LEN(word$) - 1);
      StdLog.String(word);  (* ra1 *)
      StdLog.Ln
   END ComputeReverse;

END Pbox19D.
```

Figure 19.13
A recursive procedure to reverse the elements of an array.



(a) Before call Reverse

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

(c) Call Reverse(s, 1, 6)

Figure 19.14
The run-time stack for Figure 19.13.

abcd bacd cabd dabc
abdc badc cadb dacb
acbd bcad cbad dbac
acdb bcda cbda dbca
adbc bdac cdab dcab
adcb bdca cdba dcba

```
PROCEDURE Permute (str: ARRAY OF CHAR; first, last: INTEGER);
   (* Print the permutations of str between str[first] and str[last] *)
```

```
MODULE Pbox19E;
    IMPORT StdLog;

    PROCEDURE Exchange (VAR s: ARRAY OF CHAR; i, j: INTEGER);
        VAR
            temp: CHAR;
    BEGIN
        temp := s[i];
        s[i] := s[j];
        s[j] := temp
    END Exchange;
```

Figure 19.15
A recursive procedure that prints the permutations of the elements in an array.

```
    PROCEDURE Permute (str: ARRAY OF CHAR; first, last: INTEGER);
        (* Print the permutations of str between str[first] and str[last] *)
        VAR
            i: INTEGER;
    BEGIN
        ASSERT((0 <= first) & (first <= last) & (last < LEN(str$)), 20);
        IF first = last THEN
            StdLog.String(str); StdLog.Ln
        ELSE
            FOR i := first TO last DO
                Exchange(str, first, i);
                Permute(str, first + 1, last)
            END
        END
    END Permute;

    PROCEDURE ComputePermutation*;
    BEGIN
        Permute("abc", 0, 2);
    END ComputePermutation;

END Pbox19E.
```

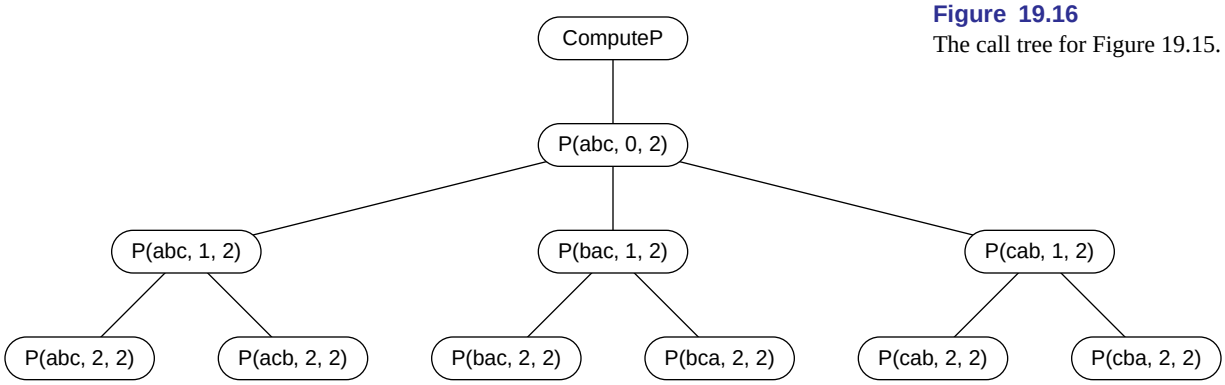


Figure 19.16
The call tree for Figure 19.15.

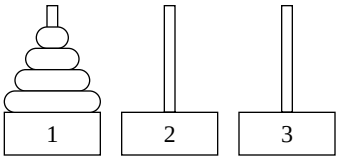
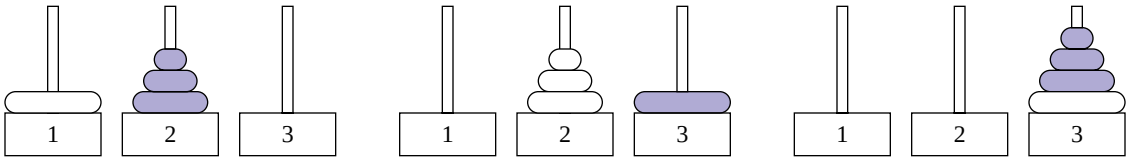


Figure 19.17
The Towers of Hanoi puzzle.

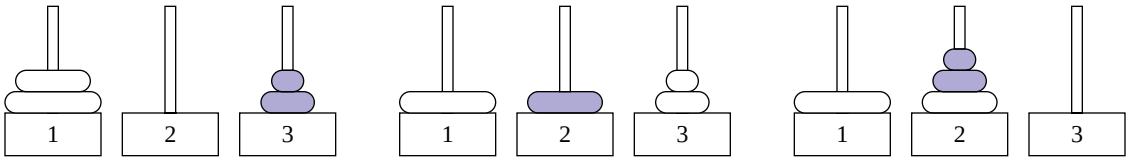


(a) Move three disks from peg 1 to peg 2.

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

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

Figure 19.18
The solution for moving four disks from peg 1 to peg 3.



(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 19.19
The solution for moving three disks from peg 1 to peg 2.

```
MODULE Alpha;
CONST, TYPE, VAR of Alpha

PROCEDURE^ A (x: SomeType);

PROCEDURE B (Y: SomeOtherType);
    Procedure body for B, including CONST, TYPE, VAR, etc.

PROCEDURE A (x: SomeType);
    Procedure body for A, including CONST, TYPE, VAR, etc.

BEGIN
    etc.
END Alpha.
```

Figure 19.20
The structure of a program with mutual recursion.