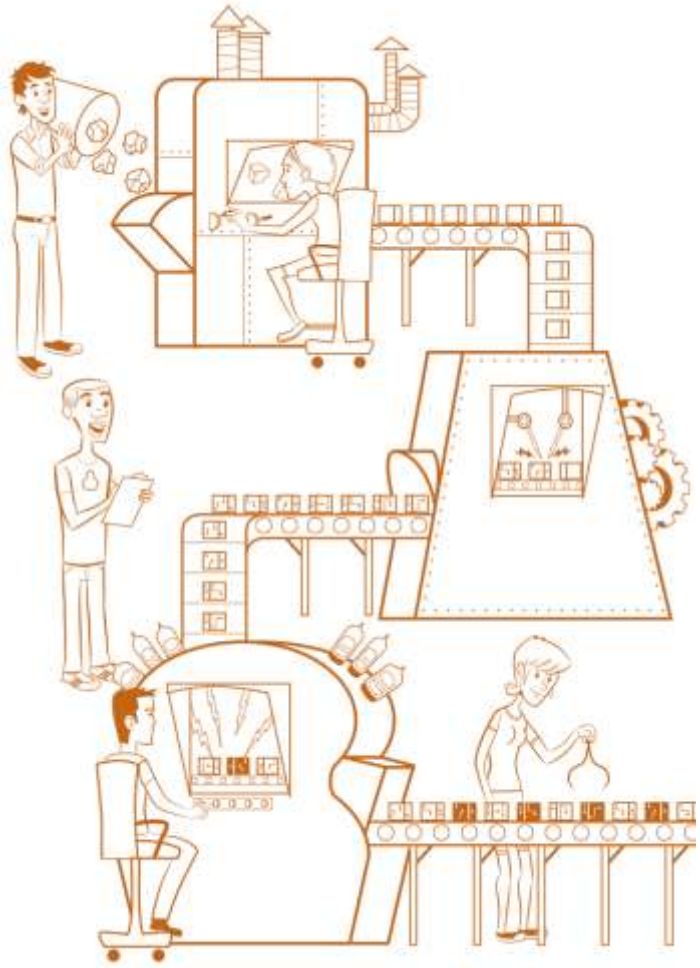
UNIDAD IV: PROGRAMACIÓN MODULAR



Estructura General de Programa

Los programas, en general, presentan una estructura común en la que pueden distinguirse las siguientes partes:

- Cabecera del programa: permite indicar el nombre de programa, a través de una palabra reservada. Por ejemplo, PROGRAMA *Calcula_Promedio*
- Declaración de Constantes: permite especificar los valores constantes que se utilizarán en el programa. Por ejemplo, MAXIMO=40
- Declaración de Tipos: permite especificar los tipos de datos definidos por el usuario que serán utilizados en el programa.
- Declaración de Variables: permite especificar las variables que serán utilizadas en el programa, indicando para cada una el tipo requerido. Por ejemplo, contador: entero (la variable contador se declara de tipo entero)
- Declaración de Procedimientos y Funciones: permite especificar las subrutinas que contendrá el programa y que facilitarán la escritura del programa. Por ejemplo, puede escribirse un subprograma que realice el cálculo del factorial de un número ingresado por el usuario.
- Programa Principal: permite especificar las secuencias de pasos generales del algoritmo, e invocar a los subprogramas necesarios cuando se requiera.

El siguiente ejemplo ilustra la estructura general de un programa.

```
PROGRAMA Calculo_Area_Circular
CONSTANTES
        PI=3.141592654
VARIABLES
        radio: real
        area:real
INICIO
    ESCRIBIR 'Ingrese el radio del círculo:'
    LEER radio
    area←PI * radio^2
    ESCRIBIR 'El área del círculo es:', area
FIN
```

En la cabecera del programa se especificó el nombre de programa, en este caso, *Calculo_Area_Circular*. En la sección de constantes, se definió la constante *PI* igual a 3.141592654. En la sección de variables, se definieron las variables de tipo real *radio* y *area*. En el cuerpo del programa principal, delimitado por las palabras reservadas *INICIO* y *FIN*, se especificaron las sentencias que implementan el algoritmo. Nótese que no se utilizaron procedimientos o funciones porque es preciso definir formalmente estos conceptos antes de ejemplificar su aplicación.

Descomposición de Problemas

En general los problemas que se presentan en el mundo real son complejos y extensos, por lo que su resolución no es directa. Para encarar tales problemas la programación cuenta con herramientas de abstracción y descomposición de problemas. La abstracción permite representar los objetos relevantes de un problema, y la descomposición, basada en el paradigma "Divide y vencerás", permite dividir problemas complejos en subproblemas y éstos a su vez en subproblemas más pequeños, y así sucesivamente. Cada uno de los subproblemas finales será entonces más simple de resolver.

Los programas que dan solución a problemas complejos están diseñados de tal forma que sus componentes (partes independientes, llamadas módulos) realizan actividades o tareas específicas (que resuelven subproblemas). Cada uno de estos componentes se programa utilizando las estructuras de control (secuenciales, selectivas y repetitivas) de la programación estructurada. La integración de los módulos, a través de métodos ascendentes o descendentes, permite obtener un programa modular que resuelve el problema planteado.

Programación Modular

La programación modular es un método de diseño flexible y potente que mejora la productividad de un programa. En programación modular, un programa se divide en subprogramas (módulos) que realizan tareas específicas y que se codifican independientemente (figuras 4 y 5). Cada módulo se analiza, codifica y pone a punto por separado.

En cada programa existe un módulo especial, llamado principal, que controla la ejecución de las tareas del programa. El principal transfiere temporalmente el control a los módulos (y éstos a su vez a submódulos, si es necesario) para que realicen una determinada tarea y luego retornen el control al módulo que los invocó.

En general los módulos o subrutinas se clasifican en procedimientos y funciones.

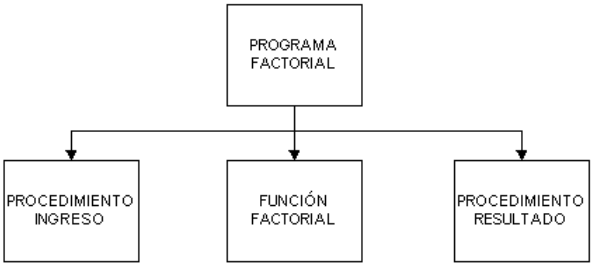


Figura 4. Estructura modular de un programa para el cálculo del factorial de un número.

```
function factorial (num:integer):integer;
var
      fact,i:integer;
begin
      fact:=1;
      for i:=1 to num do
            fact:=fact*i;
      factorial:=fact;
end;
```

Figura 5. Módulo (función) para el cálculo del factorial de un número.

Módulos: Procedimientos y Funciones.

La resolución de problemas complejos se facilita considerablemente si se dividen en problemas más pequeños (subproblemas). La resolución de estos subproblemas se realiza mediante subalgoritmos. Los subalgoritmos pueden ser de 2 tipos: procedimientos y funciones. Los subalgoritmos o subprogramas se diseñan para realizar una tarea específica, y pueden ser llamados desde distintas partes del programa. Cuando un subprograma es invocado, quien realiza el llamado (el programa principal u otro subprograma) transfiere el control de las acciones temporalmente a esta subrutina para que ejecute las acciones que contiene. Finalizado el procedimiento o la función, devuelve un resultado y el control a quién realizó la invocación.

Funciones

Matemáticamente una función es una operación que toma uno o más valores llamados argumentos y produce un valor denominado resultado (valor de la función para los argumentos dados).

En programación, una función es un módulo o subprograma que toma una lista de valores llamados argumentos o parámetros y devuelve un único valor. Como las funciones retornan un valor, éstas deben definirse del tipo de dato apropiado (entero, real, carácter, lógico), por lo que deben incluirse en expresiones (el nombre de la función está asociado a un valor) o visualizarse.

Todos los lenguajes de programación tienen funciones incorporadas o intrínsecas (por ejemplo, seno, coseno) y funciones definidas por el usuario. Las funciones incorporadas al sistema se denominan funciones internas o intrínsecas y las funciones definidas o creadas por el usuario, funciones externas.

Las funciones son diseñadas para realizar tareas específicas: tomar una lista de valores llamados argumentos o parámetros, procesarlos y devolver un único valor.

Cuando se invoca una función debe utilizarse su nombre en una expresión, indicando los argumentos actuales o reales encerrados entre paréntesis. Por ejemplo, la función factorial se invocará

Nombre_variable←factorial(numero**)**

ESCRIBIR 'El factorial es: **', factorial(**numero**)**

En el llamado a una función los argumentos deben coincidir exactamente en cantidad, tipo y orden con los especificados en su definición. Además el valor que devuelve la función debe corresponder con el tipo definido para ésta.

¿Cómo se declara una función?

Una función constará de una cabecera que comenzará la palabra reservada *función*, seguida del nombre de la función, argumentos y tipo de la función.

La declaración de la función será

```
FUNCIÓN Nombre_función (Lista de parámetros formales): Tipo_de_Función
VARIABLES
        Lista_de_Variables_de_la_Función
INICIO
    ACCIONES
    Nombre_función←resultado_de_la_función
FIN
```

Nombre_función: especifica el nombre de la función.

Lista de Parámetros Formales: especifica los valores que recibe la función, con los que realizará alguna operación.

Tipo_de_Función: tipo de la función, puede ser entera, real, carácter, lógica.

Lista_de_Variables_de_la_Función: especifica las variables de la función, éstas sólo están definidas para la función y desaparecen cuando finaliza su ejecución.

ACCIONES: sentencias secuenciales, selectivas o repetitivas que realizan la operación definida para la función.

Nombre_función←resultado_de_la_función: cuando se obtiene el resultado de la operación de la función, éste se asigna al nombre de la función como última acción para retornarlo al programa que la invocó.

¿Cómo se invoca una función?

Una función puede ser llamada de la siguiente forma:

Nombre_variable←Nombre_Función(Lista_parámetros_actuales**)**

ESCRIBIR 'El resultado de la función es: **', Nombre_Función(**Lista_parámetros_actuales**)**

Cada vez que se llama a una función desde el algoritmo principal se establece automáticamente una correspondencia entre los parámetros formales (cabecera de la declaración de función) y los parámetros actuales (valores que se envían a la función cuando se realiza la invocación). Debe haber exactamente el mismo número de parámetros actuales que de parámetros formales en la declaración de la función y se supone una correspondencia uno a uno de izquierda a derecha entre los parámetros formales y los actuales.

Una llamada a una función implica los siguientes pasos:

1. A cada parámetro formal se le asigna el valor real de su correspondiente parámetro actual.
2. Se ejecuta el cuerpo de acciones de la función.
3. Se asigna el resultado al nombre de la función y se retorna al punto de llamada.

Procedimientos

En programación, con frecuencia, se requieren subprogramas que calculen varios resultados (una función sólo retorna un valor), o que realicen la ordenación de una serie de números, etc. En estas situaciones una función no resulta adecuada y es necesario disponer de otro tipo de subprograma: "Procedimientos".

Un procedimiento o subrutina es un programa que ejecuta un proceso específico. Ningún valor está asociado con el nombre del procedimiento, como ocurre con la función, y es por ello que no puede utilizarse en una expresión. Un procedimiento se invoca escribiendo su nombre (y los parámetros necesarios) en el cuerpo de un programa o subprograma. Cuando se realiza el llamado a un procedimiento, se ejecutan los pasos que lo definen y se retorna el control al programa/subprograma que lo llamó.

¿Cómo se declara un procedimiento?

Un procedimiento constará de una cabecera que comenzará con la palabra reservada *procedimiento*, seguida del nombre y los argumentos del procedimiento.

La declaración del procedimiento será

```
PROCEDIMIENTO Nombre_procedimiento (Lista de parámetros formales)
VARIABLES
        Lista_de_Variables_del_Procedimiento
INICIO
    ACCIONES
FIN
```

Nombre_procedimiento: especifica el nombre del procedimiento.

Lista de Parámetros Formales: especifica los valores que recibe el procedimiento, con los que realizará algún procesamiento.

Lista_de_Variables_del_Procedimiento: especifica las variables del procedimiento, éstas sólo están definidas para el procedimiento y desaparecen cuando finaliza su ejecución.

ACCIONES: sentencias secuenciales, selectivas o repetitivas que realizan la operación definida para el procedimiento.

¿Cómo se invoca un procedimiento?

Un procedimiento puede ser llamado de la siguiente forma:

Nombre_Procedimiento (Lista_parámetros_actuales**)**

Cada vez que se llama a un procedimiento desde el algoritmo principal se establece automáticamente una correspondencia entre los parámetros formales (cabecera de la declaración de procedimiento) y los parámetros actuales (valores que se envían al procedimiento cuando se realiza la invocación). Debe haber exactamente el mismo número de parámetros actuales que de parámetros formales en la declaración del procedimiento y se supone una correspondencia uno a uno de izquierda a derecha entre los parámetros formales y los actuales.

Un llamado a un procedimiento implica los siguientes pasos:

1. Los parámetros actuales sustituyen a los parámetros formales.
2. El cuerpo de la declaración del procedimiento sustituye el llamado del procedimiento.
3. Se ejecutan las acciones escritas por el código resultante.

Pasaje de Parámetros

Para que las funciones calculen resultados o los procedimientos ejecuten determinados procesos, frecuentemente, requieren de información que les permita llevar a cabo tales tareas. Esta información se especifica en la declaración del subprograma y los valores apropiados se transmiten al procedimiento o función al ser invocados.

Los métodos más utilizados para realizar el paso de parámetros son:

- Paso por Valor
- Paso por Referencia

Las variables que se transmiten a una función o procedimiento, utilizando el *Paso de Parámetros por Valor*, son copias de las variables del programa que realiza el llamado a un subprograma. Por tanto, cualquier operación realizada sobre dichas variables no modifica el valor de las variables originales. En este caso se dice que los parámetros son de Entrada.

Las variables que se transmiten a una función o procedimiento, utilizando el *Paso de Parámetros por Referencia*, son las variables del programa que realiza el llamado y cualquier modificación a su valor implica un cambio de las variables para el programa que las transmitió. En este caso se dice que los parámetros son de Entrada/Salida (las variables pueden accederse y modificarse).

Ejemplo: Codifique un programa que calcule del área de un triángulo utilizando procedimiento y funciones.

```
PROGRAMA calculo_triangulo
VARIABLES
{Declaración de las variables base, altura y area utilizadas en el programa}
        base, altura, area: real
{Procedimiento que carga los valores de base y altura de un triángulo}
PROCEDIMIENTO Leer_datos(E/S b: real, E/S h: real)
INICIO
    ESCRIBIR 'Ingrese la base del triángulo:'
    LEER b
    ESCRIBIR 'Ingrese la altura del triángulo:'
    LEER h
FIN
{Función que calcula el área de un triángulo con los datos de base y altura ingresados por el usuario}
FUNCIÓN Calculo_area(E b:real, E h:real): real
INICIO
    Calculo_area<-b * h / 2
FIN
{Programa Principal que llama al procedimiento Leer_datos y a la función Calculo_area}
INICIO
    LEER_datos(base,altura)
    area<-Calculo_area(base,altura)
    ESCRIBIR 'El área calculada es:', area
FIN
```

Ámbito de la Variables

En el desarrollo de un sistema complejo con gran número de módulos, realizados por diferentes programadores, debe tenerse en cuenta que cada uno de ellos escribirá el código correspondiente a un módulo. Estos implica que cada cual define sus propios datos, con identificadores apropiados y como consecuencia de ello se podrían observar algunos inconvenientes tales como:

- Demasiados identificadores.
- Conflictos entre los nombres de los identificadores de cada programador.
- Integridad de los datos lo que implica que se puedan utilizan datos que tengan igual identificador pero que realicen funciones diferentes.
- Side Effects no previstos.

Si se considera que los módulos también pueden anidarse (módulos dentro de otros módulos), deben aplicarse 2 reglas muy importantes que gobiernan el alcance de los identificadores:

- El alcance de un identificador es el bloque de programa donde se lo declara.
- Si un identificador declarado en un bloque es nuevamente declarado en un bloque interno al primero, el segundo bloque es excluido del alcance de la primera sección.

Variables Globales y Locales

Las variables utilizadas en los programas principales y subprogramas se clasifican en 2 tipos:

- Variables Locales
- Variables Globales

Una variable local es aquella que está declarada y definida dentro de un subprograma, en el sentido de que está dentro de ese programa y es distinta de las variables con el mismo nombre declaradas en cualquier parte del programa principal. El significado de una variable se confina al subprograma en el que está declarada. Una variable local no tiene ningún significado en otros subprogramas, es decir, el valor asignado a una variable local en un subprograma no es accesible por los otros procedimientos o funciones del programa.

Un variable global es aquella que está declarada en el programa o algoritmo principal, del que dependen todos

los subprogramas. Las variables globales son conocidas por todos los procedimientos y funciones que conforman un programa, y pueden ser modificadas por éstos. Esta situación permite utilizar variables sin necesidad de usarlas como parámetros, sin embargo, las modificaciones no intencionales a una variable global pueden ocasionar errores en el programa.

Ocultamiento y Protección de Datos

El concepto *Data Hidding* se utiliza para significar que todo dato que es relevante para un módulo debe ocultarse a otros módulos. De esta manera se evita que en el programa principal se declaren datos que sólo son relevantes para algún módulo en particular y, además, se protege la integridad de los datos.

Recursividad

La recursividad consiste en realizar la definición de un concepto en términos del propio concepto que se está definiendo. Para que una definición recursiva sea válida, la referencia a sí misma debe ser relativamente más sencilla que el caso considerado.

Una definición recursiva indica cómo obtener nuevos conceptos a partir del concepto que se intenta definir. Esto es, la recursividad expresa un concepto complejo en función de las formas más simples del mismo concepto.

Un razonamiento recursivo tiene dos partes:

- caso base y
- regla recursiva de construcción

El caso base no es recursivo y es el punto tanto de partida como de terminación de la definición.

La regla recursiva de construcción intenta resolver una versión más simple (pequeña) del problema original.

Por lo tanto, para resolver problemas en forma recursiva debe considerarse:

1. la división sucesiva del problema original en uno o varios problemas más pequeños, del mismo tipo que el inicial;
2. la resolución de los problemas más sencillos, y
3. la construcción de las soluciones de los problemas complejos a partir de las soluciones de los problemas más sencillos.

Teniendo en cuenta todo esto, un algoritmo recursivo se caracteriza por:

1. el algoritmo debe contener una llamada a sí mismo,
2. el problema planteado puede resolverse atacando el mismo problema pero de tamaño menor,
3. la reducción del tamaño del problema eventualmente permite alcanzar el caso base, que puede resolverse directamente sin necesidad de otro llamado recursivo.

En resumen, un algoritmo recursivo consta de una parte recursiva, otra iterativa o no recursiva y una condición de terminación. La parte recursiva y la condición de terminación siempre existen. En cambio la parte no recursiva puede coincidir con la condición de terminación. Es importante destacar que siempre debe alcanzarse el caso base, si esto no se cumple el algoritmo nunca se finalizará.

Ventajas e inconvenientes

La principal ventaja es la simplicidad de comprensión y su gran potencia, favoreciendo la resolución de problemas de manera natural, sencilla y elegante; y facilidad para comprobar y convencerse de que la solución del problema es correcta.

Los algoritmos que por naturaleza son recursivos y donde la solución iterativa es complicada y debe manejarse

explícitamente una pila para emular las llamadas recursivas, deben resolverse por métodos recursivos

En general, las soluciones recursivas son menos eficientes que las iterativas (coste mayor en tiempo y memoria). Tenga en cuenta que por cada invocación se crea una copia en memoria del programa recursivo con los parámetros indicados en el llamado.

Cuando haya una solución obvia al problema por iteración, debe evitarse la recursividad.

Tipos de Recursividad

La recursividad puede ser de 2 tipos:

- Recursividad Directa (simple): un algoritmo se invoca a sí mismo una o más veces directamente.
- Recursividad Indirecta (mutua): un algoritmo A invoca a otro algoritmo B y éste a su vez invoca al algoritmo A.

Problemas resueltos mediante recursividad

Un ejemplo clásico de un problema que se resuelve utilizando recursividad es el cálculo del factorial de un número. Suponga que se calcula el factorial de 5, éste es:

5! = 120

Esto puede expresarse también como:

Problema	**Descomposición del Problema**
5! = 5 * 4!	Factorial de 5 (Original)
4! = 4 * 3!	Factorial de 4
3!= 3 * 2!	Factorial de 3
2! = 2 * 1!	Factorial de 2
1! = 1 * 0!	Factorial de 1
0! = 1	Solución Directa Factorial de 0 = 1

Como puede observarse este problema se expresa en términos cada vez más simples hasta que se alcanza el caso base, donde la solución más simple se obtiene directamente (por definición se sabe que el factorial de 0 es 1). A partir del caso base pueden obtenerse las soluciones para los problemas más complejos.

Procedimientos y Funciones Recursivos

Un procedimiento *P* que contiene una sentencia de llamada a sí mismo, se dice que es un procedimiento recursivo. Un procedimiento recursivo debe cumplir con lo siguiente:

1. incluye un cierto criterio, llamado caso base, por el que el procedimiento no se llama a sí mismo.
2. cada vez que el procedimiento se llame a sí mismo (directa o indirectamente), debe acercarse más al caso base.

Un procedimiento recursivo con estas dos propiedades se dice que está bien definido.

Asimismo una función es recursiva si su definición contiene una invocación a sí misma y cumple con lo siguiente:

1. debe haber ciertos argumentos, llamados valores base, para los que la función no se refiera a sí misma.
2. cada vez que la función se refiera a sí misma, el argumento de la función debe acercarse más al valor base.

Una función recursiva con estas dos propiedades se dice que está bien definida.

Los siguientes ejemplos ilustran procedimientos y funciones recursivos.

```
PROCEDIMIENTO Mostrar_Numeros(E cantidad: entero)
INICIO
    SI cantidad=1 ENTONCES
        ESCRIBIR cantidad
    SINO
        Mostrar_Numeros(cantidad-1)
        ESCRIBIR cantidad
    FINSI
FIN
```

El procedimiento *Mostrar_Numeros* visualiza, en orden creciente, valores enteros entre 1 y cantidad. Por ejemplo, si se invoca a *Mostrar_Numeros* con el parámetro cantidad igual a 5, el algoritmo se comporta como sigue:

1. Se invoca al procedimiento con el parámetro *cantidad* igual a 5, se evalúa la condición (*cantidad=1*) y como ésta no se cumple, se invoca a *Mostrar_Numeros* con parámetro *cantidad* igual a 4. Tenga en cuenta que la acción *ESCRIBIR cantidad* (con valor 5 en este caso) no se realizará hasta que finalice la ejecución del nuevo llamado.

Parámetro cantidad	Condición (cantidad=1)	Acción
5	FALSO	Invocar Mostrar_Numeros(4) **Escribir cantidad = 5**

2. *Mostrar_Numeros*, ahora con parámetro *cantidad* igual a 4, comprueba la condición (*cantidad=1*). Al evaluar dicha condición se determina que es FALSA, y por lo tanto se realiza el llamado al procedimiento *Mostrar_Numeros* con parámetro *cantidad* igual a 3. Esto se repite en tanto no se cumpla que *cantidad* sea igual a 1. Las acciones *ESCRIBIR cantidad* con valores 4, 3 y 2 no se llevan a cabo hasta que finalice la ejecución de las invocaciones realizadas.

Parámetro cantidad	Condición (cantidad=1)	Acción
4	FALSO	Invocar Mostrar_Numeros(3) **Escribir cantidad = 4**

Parámetro cantidad	Condición (cantidad=1)	Acción
3	FALSO	Invocar Mostrar_Numeros(2) **Escribir cantidad = 3**

Parámetro cantidad	Condición (cantidad=1)	Acción
2	FALSO	Invocar Mostrar_Numeros(1) **Escribir cantidad = 2**

3. Finalmente, cuando se invoca a *Mostrar_Numeros* con parámetro *cantidad* igual a 1, se verifica la condición (*cantidad=1*) que finaliza la recursión (no se realizan más llamados al procedimiento) y se ejecuta la sentencia *ESCRIBIR cantidad* (con valor 1). Al terminar la ejecución de este llamado, se realiza la instrucción *ESCRIBIR cantidad* del procedimiento *Mostrar_Numeros* que tiene parámetro *cantidad* igual a 2, al finalizar éste se ejecutan las instrucciones ESCRIBIR de todos los llamados anteriores.

Parámetro cantidad	Condición (cantidad=1)	Acción
1	VERDADERO	Escribir cantidad = 1

Nótese que el procedimiento ejemplificado contiene las 2 partes indicadas para los algoritmos recursivos, una que realiza el llamado al mismo procedimiento y otra que finaliza la recursión (condición *cantidad=1*).

La función *Potencia* calcula el resultado de elevar un número entero a al exponente entero b, mediante productos sucesivos. Por ej., si se invoca a *Potencia* con los parámetros *a* igual a 3 y *b* igual a 4, el algoritmo se comportaría así:

```
FUNCIÓN Potencia(E a:entero, E b:entero): entero
INICIO
    SI b=1 ENTONCES
        Potencia←a
    SINO
        Potencia←a * Potencia(a,b-1)
    FINSI
FIN
```

1. Al ejecutarse la función *Potencia* con los parámetros *a=3* y *b=4* se evalúa la condición (*b=1*) indicada en el algoritmo. Dicha condición resulta FALSA, por lo que se realiza *Potencia←a*Potencia(a,b-1)*, sentencia que invoca a *Potencia* con los parámetros *a=3* y *b=3*. Tenga en cuenta el valor final de la función no se obtiene hasta que todos los llamados recursivos finalicen su ejecución.

Parámetro *a*	Parámetro *b*	Condición (b=1)	Acción
3	4	FALSO	Potencia<-3 * Potencia(3,3) **Potencia = 81**

2. *Potencia*, con parámetros *a=3* y *b=3*, comprueba la condición (*b=1*) del algoritmo. Al evaluar dicha condición se determina que es FALSA, y por lo tanto se realiza el llamado a la función *Potencia* con parámetros *a=3* y *b=2*. Esto se repite en tanto no se cumpla que el parámetro *b* sea igual a 1. El valor de la función para cada llamado no se calcula hasta que finalice la ejecución de las invocaciones realizadas.

Parámetro *a*	Parámetro *b*	Condición (b=1)	Acción
3	3	FALSO	Potencia<-3 * Potencia(3,2) **Potencia = 27**

Parámetro *a*	Parámetro *b*	Condición (b=1)	Acción
3	2	FALSO	Potencia<-3 * Potencia(3,1) **Potencia = 9**

3. Finalmente, cuando se llama a *Potencia* con parámetros *a=3* y *b=1*, se verifica la condición (*b=1*) que finaliza la recursión (no se realizan más llamados a la función) y se ejecuta la sentencia *Potencia←a*, que asigna a la función el valor 3. Al terminar la ejecución de este llamado, se calcula el valor de la función *Potencia* con los parámetros *a=3* y *b=2* cuyo resultado es 9. Al finalizar ésta se calcula el valor de la función para todos los llamados anteriores.

Parámetro *a*	Parámetro *b*	Condición (b=1)	Acción
3	1	VERDADERO	Potencia=3

Nótese que el función ejemplificada contiene las 2 partes indicadas para los algoritmos recursivos, una que realiza el llamado a la misma función y otra que finaliza la recursión (condición *b=1*).