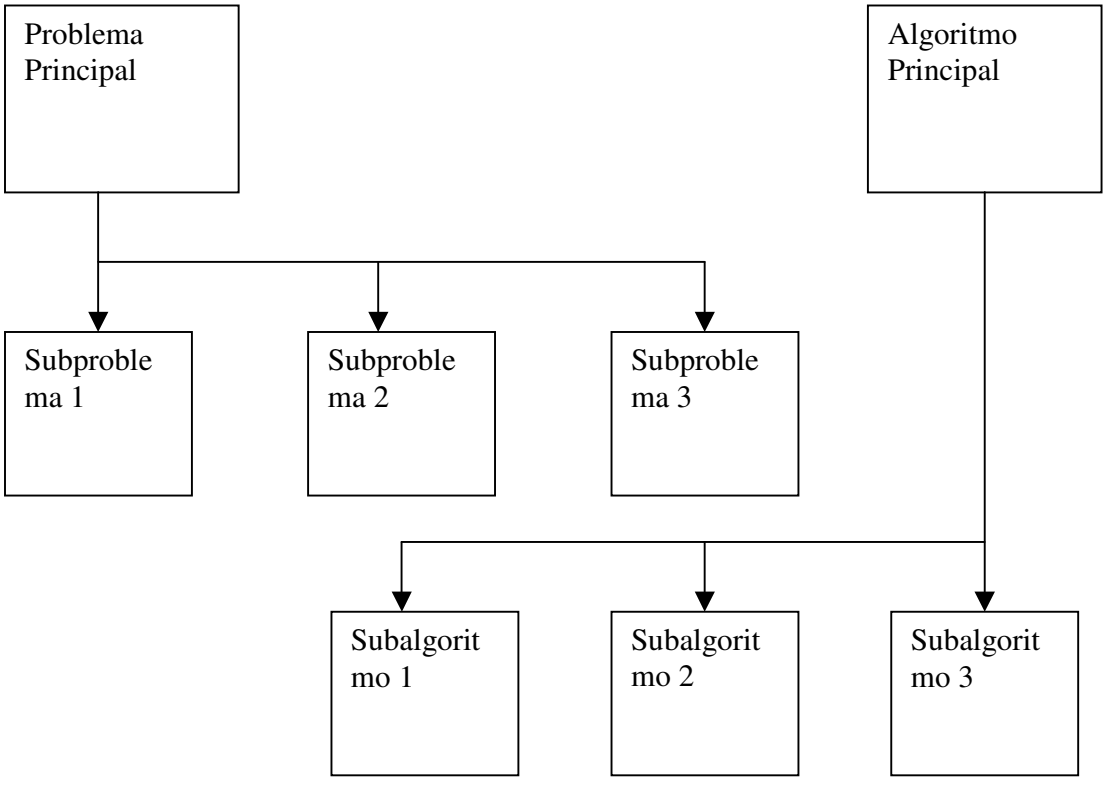
5. Subprogramas(subalgoritmos):procedimientos y funciones

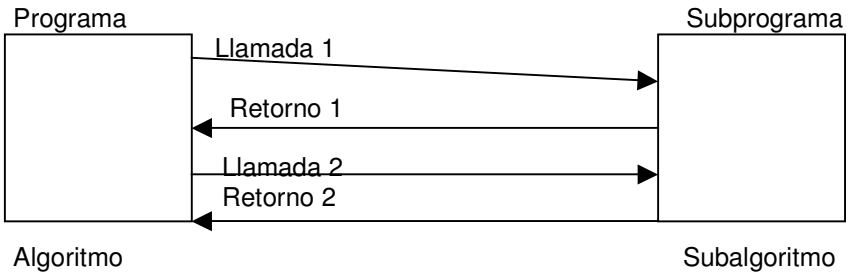
En este capitulo se describen las funciones y procedimientos, con los conceptos de variables locales y globales. Se introduce el concepto de recursividad como una nueva herramienta para resolver problemas.

5.1 Introducción a los subalgoritmos o subprogramas

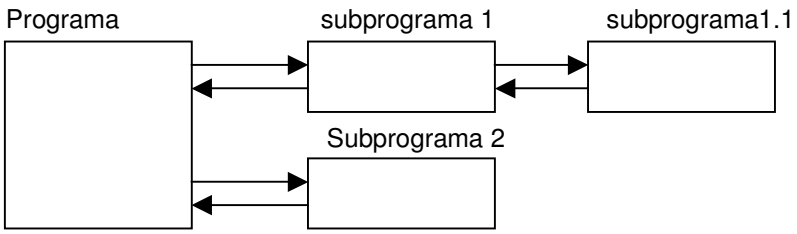
Soluciona problemas complejos al dividirlos en subprogramas y luego dividirlos estos en otros mas simples, hasta que estos sean mas fáciles de resolver. Esta técnica se llama “divide y vencerás” .
El problema principal denominado controlador o conductor (drive) y la solución de los subproblemas conocidos como procedimientos (subrutinas) o funciones.



Se dice que el programa principal invoca al subprograma, el subprograma ejecuta la tarea y luego devuelve el control al programa.



Un programa puede llamar a su ves a sus propios subprogramas .



5.2 Funciones:

Es una que toma una o mas valores llamados argumentos y produce un valor llamado resultado
Ejemplo:

F(x) =x/i+x*x

Para evaluar f debemos darle un valor a x.

5.2.1 Declaración de funciones

Requiere de una serie de pasos que la definen .
Consta de una cabecera , seguido de la palabra (`función`) y del nombre del argumento de la función después ira el cuerpo que es una serie de acciones cuya ejecución hará que se asigne un valor al nombre de la función , esto determina el resultado que ha de devolver al programa.
La declaración de la función será :

```
<tipo_de_resultado>función <nombre_fun>(lista de parámetro
[declaración locales]
inicio
      <acciones>      //cuerpo de la función
      devolver        (<exposición>)
fin_funcion
```

Ejemplo:

F(x,y)=x/1+x*x

Se define como :
Real `función f(x)`
```
Inicio
   Devolver (x/(1+x*x))
Fin_funcion
```

5.2.2 Invocación de las funciones.

Una función puede ser llamada de la siguiente forma:

```
Nombre_funcion(lista de parámetros actuales)
```

`Nombre_funcion:` función que llama
`Lista de parámetros actuales:` constantes variables , expresiones.

Cada vez que se llama a una función desde el algoritmo principal se establece una correspondencia entre los parámetros formales y los parámetros actuales .
Una llamada a la 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 cuero de acciones de la función .
3._Se devuelve el valor de la función y se retorna al punto de llamada.

Ejemplo:
Función potencia para el calculo de N elevada a A .El numero N deberá ser positivo aunque podrá tener parte fraccionaria , A es un real .

```
Algoritmo elevar_a_potencia
Var
   Real:a.n
Inicio
Escribir(N elevado a =potencia(n.a))
Fin
Real función potencia (E real :n,a)
Inicio
    Devolver(exp(a*1n(n))
Fin_funcion.
```

Ejemplo 2

Algoritmo que contiene y utiliza unas funciones(sen y cos) a las que les podemos pasar el Angulo en grados.

```
Algoritmo sen_cos_en_grados
Ver real :g
Inicio
   Escribir(deme ángulo en grados)
   Leer (g)
   Escribir(sen(g))
   Escribir(cos(g))
Fin
Real función coseno(E real:g)
Inicio
      Devolver(cos(g*2*3.141592/360))
Fin_funcion
Real_funcion seno(real:g)
Inicio
     Devolver(sen(g*2*3.141592/360))
Fin_funcion
```

La salida del algoritmo sería:

```
2 al cubo es 8
3 al cubo es 27
```

Las funciones pueden tener muchos argumentos, pero solamente un resultado: *el valor de la función*. Esto limita su uso, aunque se encuentran con frecuencia en cálculos científicos. Un concepto más potente es el proporcionado por el subprograma procedimiento que se examina en el siguiente apartado.

Ejercicio 5.5

Algoritmo que contiene y utiliza unas funciones (seno y coseno) a las que les podemos pasar el ángulo en grados.

```
algoritmo Sen_cos_en_grados
var real : g

inicio
  escribir(´Deme angulo en grados´)
  leer(g)
  escribir(seno(g))
  escribir(coseno(g))
fin

real funcion coseno (E real : g)
inicio
  devolver(cos(g * 2 * 3.141592/360))
fin_funcion

real funcion seno (E real g)
inicio
```

```
  devolver( sen(g * 2 * 3.141592/360))
fin_funcion
```

Ejercicio 5.6

Algoritmo que simplifique un quebrado, dividiendo numerador y denominador por su máximo común divisor.

```
algoritmo Simplificar_quebrado
var
  entero : n, d

inicio
  escribir(´Deme numerador´)
  leer(n)
  escribir(´Deme denominador´)
  leer(d)
  escribir(n, '/', d, '=', n div mod(n,d), '/', d div mod(n,d)))
  fin
  entero funcion mcd (E entero: n,d)
  var
    entero : n, d
  inicio
    r ← n mod d
    mientras r <> 0 hacer
      n ← d
      d ← r
      r ← n MOD d
    fin_mientras
  devolver(d)
fin_funcion
```

Ejercicio 5.7

Supuesto que nuestro compilador no tiene la función seno. Podríamos calcular el seno de x mediante la siguiente serie:

$$sen(x) = x - r$$

x (ángulo en radianes)

El programa nos tiene que permitir el cálculo del seno de ángulos en grados, mediante el diseño de una función seno (x), *que utilizará, a su vez, las funciones potencia (x, n) y factorial (n), que también deberán ser implementadas en el algoritmo.*

Se terminará cuando respondamos N (no) a la petición de otro ángulo.

```
algoritmo Calcular_seno
var real     : gr
    carácter : resp

inicio
  repetir
    escribir(´Deme angulo en grados´)
```

```
    leer(gr)
    escribir(seno(', gr, ')=', seno(gr))
    escribir('¿Otro angulo?')
    leer(resp)
  hasta_que resp = 'N'
fin

real funcion factorial (E entero:n)
var
  real   : f
  entero : i
inicio
    f ← 1
    desde i ← 1 hasta n hacer
      f ← f * i
    fin_desde
    devolver(f)
fin_funcion

real funcion potencia (E real: x; E entero:n)
var real : pot
  entero : i
inicio
    pot ← 1
    desde i ← 1 hasta n hacer
      pot ← pot * x
    fin_desde
    devolver(pot)
fin_funcion

real funcion seno (E real:gr)
var real : x, s
  entero : i, n
inicio
  x ← gr * 3.141592 / 180
  s ← x
  desde i ← 2 hasta 17 hacer
    n ← 2 * i - 1
    si i mod 2 <> 0 entonces
      s ← s - potencia(x, n) / factorial(n)
    si_no
      s ← s + potencia(x, n) / factorial(n)
    fin_si
  fin_desde
  devolver(s)
fin_funcion
```

5.3. PROCEDIMIENTOS (subrutinas)

Aunque las funciones son herramientas de programación muy útiles para la resolución de problemas, su alcance está muy limitado. Con frecuencia, se requieren subprogramas que calculen varios resultados en vez de uno solo, o que realicen la ordenación de una serie de números, etc. En estas situaciones la *función* no es apropiada y se necesita disponer del otro tipo de subprograma: el *procedimiento* o *subrutina.*

Un *procedimiento* o *subrutina* es un subprograma que ejecuta un proceso específico. Ningún valor está asociado con el nombre del procedimiento; por consiguiente, no puede ocurrir en una expresión. Un procedimiento se llama escribiendo su nombre, por ejemplo, SORT, para indicar que un procedimiento denominado SORT se va a usar. Cuando se invoca el procedimiento, los pasos que lo definen se ejecutan y a continuación se devuelve el control al programa que le llamó.

Procedimiento versus función

Los procedimientos y funciones son subprogramas cuyo diseño y misión son similares, sin embargo, existen unas diferencias esenciales entre ellos:

1. Un procedimiento es llamado desde el algoritmo o programa principal mediante su nombre y una lista de parámetros actuales, o bien con la instrucción **`llamar_a (call).`** Al llamar al procedimiento se detiene momentáneamente el programa que se estuviera realizando y el control pasa al procedimiento llamado. Después de que las acciones del procedimiento se ejecutan, se regresa a la acción inmediatamente siguiente a la que se llamó.
2. Las funciones devuelven un valor, los procedimientos pueden devolver 0,1 o *n* valores y en forma de lista de parámetros.
3. El procedimiento se declara igual que la función, pero su nombre no está asociado a ninguno de los resultados que obtiene.

La declaración de un procedimiento es similar a la de funciones.

```
procedimiento nombre [(lista de parametros formales)]
...
  <acciones>
fin_procedimiento
```

Los parámetros formales tienen el mismo significado que en las funciones; los parámetros variable – en aquellos lenguajes que los soporta, por ejemplo, Pascal – están precedidos cada uno de ellos por la palabra **`var`** para designar que ellos obtendrán resultados del procedimiento en lugar de los valores actuales asociados a ellos.

El procedimiento se llama mediante la instrucción

```
[llamar_a] nombre [(lista de parametros actuales)]
```

La palabra **`llamar_a(call)`** es opcional y su existencia depende del lenguaje de programación. El ejemplo siguiente ilustra la definición y uso de un procedimiento para realizar la división de dos números y obtener el cociente y el resto.

Variables enteras: DIVIDENDO
DIVISOR
COCIENTE

Procedimiento

```
procedimiento división (E entero:Dividendo,Divisor; S entero:Cociente,
Resto)
```

```
inicio
  Cociente ← Dividendo div Divisor
  Resto ← Dividendo - Cociente * Divisor
fin_procedimiento
```

Algoritmo princpal

```
algoritmo Aritmetica
var
  entero : M, N, P, Q, S, T
inicio
  leer(M, N)
  llamar_a división (m, N, P, Q)
  escribir(P, Q)
  llamar_a división (M * N, -4, N + 1, S, T)
  escribir(S, T)
fin
```

5.3.1. Sustitución de argumentos/parámetros

La lista de parámetros, bien *formales* en el procedimiento o *actuales* (reales) en la llamada, se conoce como lista de parámetros, pueden ser de Entrada (E), Salida (S) o Entrada/Salida(E/S).

```
procedimiento demo
.
.
.
fin_procedimiento
```

o bien:

```
procedimiento demo (lista de parámetros formales)
```

y la instrucción llamadora

```
llamar_a demo (lista de parámetros actuales)
```

Cuando se llama al procedimiento, cada parámetro formal toma como valor inicial el valor del correspondiente parámetro actual. En el ejemplo siguiente se indican la sustitución de parámetros y el orden correcto.

```
algoritmo Demo
  //definición del procedimiento
  entero: años
  real: numeros, tasa
inicio
  ...
  llamar_a calculo(numero, años, tasa)
  ...
fin

procedimiento calculo(S real: pl; E entero: p2; E real: p3)
inicio
```

```
  p3 ... p1 ... p2 ... p2
fin_procesimiento
```

Las acciones sucesivas a realizar son las siguientes:

1. Los parámetros reales sustituyen a los parámetros formales.
2. El cuerpo de la declaración del procedimiento se sustituye por la llamada del procedimiento.
3. Por último, se ejecutan las acciones escritas por el código resultante.

Ejemplo 5.8.

Algoritmo que transforma un número introducido por teclado en notación decimal a romana. El número será entero y positivo y no excederá de 3.000.

Sin utilizar programación modular:

```
algoritmo Romanos
var entero : n, digito, r, j

inicio
    repetir
      escribir(´Deme numero´)
      leer(n)
hasta_que (n >= 0) Y (n <= 3000)
r • n
digito • r div 1000
r • r mod 1000
desde j • 1 hasta digito hacer
  escribir(´M´)
fin_desde
digito • r div 100
r • r mod 100
si digito = 9 entonces
  escribir(´C´, ´M´)
si_no
si digito > 4 entonces
  escribir(´D´)
  desde j • 1 hasta digito - 5 hacer
    escribir(´C´)
  fin_desde
si_no
   si digito = 4 entonces
     escribir(´C´, ´D´)
   si_no
     desde j • 1 hasta digito hacer
       escribir(´C´)
     fin_desde
   fin_si
 fin_si
fin_si
digito • r div 10
r • r mod 10
si digito = 9 entonces
```

```
    escribir(´X´,´C´)
  si_no
    si digito > 4 entonces
      escribir(´L´)
      desde j ← 1 hasta digito - 5 hacer
        escribir(´X´)
        fin_desde
    si_no
      si digito = 4 entonces
        escribir(´X´, ´L´)
      si_no
        desde j ← 1 hasta digito hacer
          escribir(´X´)
        fin_desde
      fin_si
    fin_si
  fin_si
  digito ← r
  si digito = 9 entonces
    escribir(´Í´, ´X´)
  si_no
    si digito > 4 entonces
      escribir(´V´)
    desde j ← 1 hasta digito - 5 hacer
      escribir(´I´)
    fin_si
    si_no
      si digito = 4 entonces
        escribir(´Í´, ´V´)
      si_no
        desde j ← 1 hasta digito hacer
          escribir(´Í´)
        fin_desde
        fin_si
      fin_si
    fin_si
  fin
```

Mediante programción modular:

```
  algoritmo Romanos
  var n,r,digito: entero

  inicio
    repetir
    escribir(´Deme numero´)
    leer(n)
    hasta_que (n >= 0) Y (n <= 3000)
    r ← n
    digito ← r div 1000
    r ← r mod 1000
    calccifrarom(digito, ´M´,`D`, `L`)
    digito ← r div 100
    r ← r mod 100
    calccifrarom(digito, ´C´, ´D´, ´M´)
    digito ← r div 10
    r ← r mod 10
```

```
  calccifrarom(digito, ´X´, ´L´, ´C´)
  digito • r
  calccifrarom(digito, ´I´, ´V´, ´X´)
fin

procedimiento calccifrarom(E entero:digito; E carácter:v1, v2, v3)
var entero: j
inicio
  si digito = 9 entonces
    escribir( v1, v3)
  si_no
    si digito > 4 entonces
      escribir(v2)
      desde j • 1 hasta digito - 5 hacer
        escribir(v1)
      fin_desde
    si_no
    si digito = 4 entonces
      escribir(v1, v2)
    si_no
      desde j • 1 hasta digito hacer
        escribir(v1)
      fin_desde
    fin_si
    fin_si
  fin_si
fin_procesimiento
```

5.4. ÁMBITO: VARIABLES LOCALES Y GLOBALES

Las variables utilizadas en los principales y subprogramas se clasifican en dos 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 subprograma 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 procedimiento en el que está declarada.* Cuando otro subprograma utiliza el mismo nombre se refiere a una posición diferente en memoria. Se dice que tales variable son locales al subprograma en el que están declaradas.

Una *variable global* es aquellas que está declarada para el programa o algoritmo principal, del que dependen todos los subprogramas.

La parte del programa/algoritmo en que una variable se define se conoce como *ámbito* (*scope,* en inglés)

El uso de variables locales tiene muchas ventajas. En particular, hace a los subprogramas independientes, con la comunicación entre el programa principal y los subprogramas manipulados estructuralmente a través de la lista de parámetros. Para utilizar un procedimiento sólo necesitamos conocer lo que hace y no tenemos que estar preocupados por su diseño, es decir, cómo están programados.

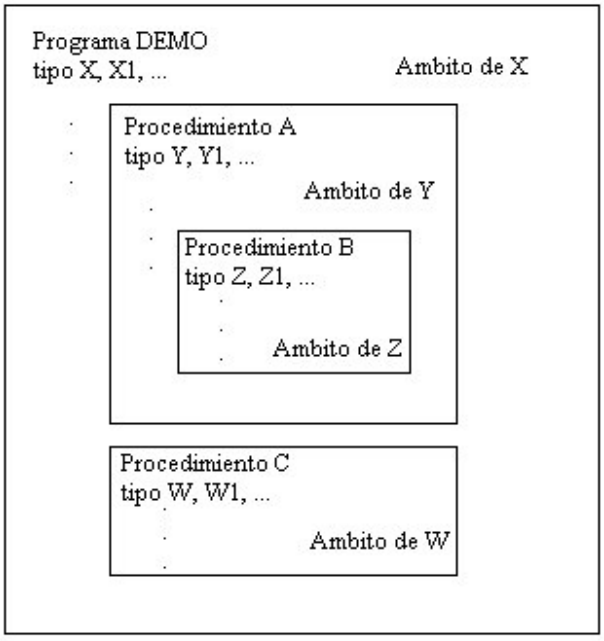


Figura 5.4. Ambito de identificadores.

Esta característica hace posible dividir grandes proyectos en piezas más pequeñas independientes. Cuando diferentes programadores están implicados, ellos pueden trabajar independientemente.

A pesar del hecho importante de los subprogramas independientes y las variables locales, la mayoría de los lenguajes proporcionan algún método para tratar ambos tipos de variables.

Una variable local a un subprograma no tiene ningún significado en otros subprogramas. Si un subprograma asigna un valor a sus variables locales, este valor no es accesible a otros programas es decir, no puede utilizar esta valor. A veces, también es necesario que una variable tenga el mismo nombre en diferentes subprogramas.

Por el contrario las variables globales tienen la ventaja de compartir información de diferentes subprogramas sin una correspondiente entrada en la lista de parámetros.

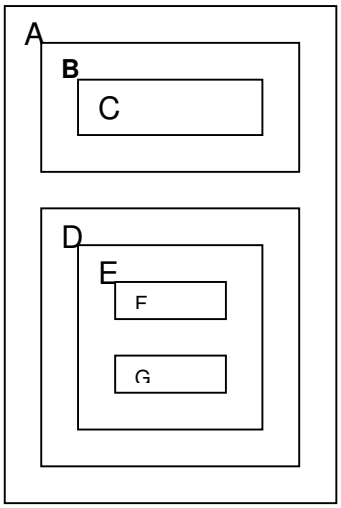
En un programa sencillo con un subprograma, cada variable u otro identificador es o bien local al procedimiento o global al programa completo. Sin embargo, si el programa incluye procedimientos que engloban a otros procedimientos –*procedimientos anidados* -, entonces la noción de global/local es algo más complicada de entender.

El *ámbito* de un identificador (variables, constantes, procedimientos) es la parte del programa donde se conoce el identificador. Si un procedimiento está definido localmente a otro procedimiento, tendrá significado solo dentro del ámbito de ese procedimiento. A las variables les sucede lo mismo; si están definidas local mente dentro de un procedimiento, su significa o uso se confina a cualquier función o procedimiento que pertenezca a esa definición.

La figura 5.5 muestra un esquema de un programa con diferentes procedimientos, algunas variables son locales y otras globales. En esta citada figura se muestra el ámbito de cada definición.

Los lenguajes que admiten variables locales y globales suelen tener la posibilidad explícita de definir dichas variables como tales en el cuerpo del programa o, lo que es lo mismo, definir su ámbito de actuación, para ello se utilizan las cabeceras de programas y subprogramas, con lo que se definen los ámbitos.

Las variables definidas en un ámbito son accesibles en el mismo, es decir, en todos los procedimientos interiores.



Variables definidas en	Accesibles desde
A	A, B, C, D, E, F, G.
B	B, C.
C	C
D	D, E, F, G.
E	E, F, G.
F	F
G	G

Figura 5.5 Ámbito de definión de variables

Ejemplo 5.9

La función (signo) realiza la siguiente tarea: dado un numero real x. Si x es 0*, entonces se devuelve un* 0*; si x es positivo, se devuelve* 1*, y si x es negativo, se devuelve un valor* –1.

La declaración de la función es:

```
Entero función signo (E real: x)
Var entero: s
Inicio
 //valor de signo: +1, 0, -1
 si X = 0 entonces S ⇓ 0
 si X > 0 entonces S ⇓ 1
 si X < 0 entonces S ⇓ -1
 devolver (S)
fin_función
```

Antes de llamar a la función. La variable (S). como se declara dentro del subprograma, es local al subprograma y solo se conoce dentro del mismo. Veamos ahora un pequeño algoritmo donde se invoque la función.

```
Algoritmo SIGNOS
var
 entero: a, b, c
 real: x, y, z
inicio
 x ⇓ 5.4
 a ⇓ signo(x)
 y ⇓ 0
 b ⇓ signo(y)
 z ⇓ 7.8975
 c ⇓ signo (z-9)
```

```
 escribir ( Las respuestas son a,  , b,  , c)
fin
```

Si se ejecuta este algoritmo, se obtienen los siguientes valores:

x ⇓	5.4	x es el parametro actual de la primera lla mada a *signo* (x)
a ⇓	signo(x)	a toma el valor 1
y ⇓	0	
b ⇓	signo(y)	b toma el valor 0
z ⇓	7.8975	
c ⇓	signo (z-9)	c toma el valor –1

La linea escrita al final será:

```
Las respuestas son 1 0 -1
```

Ejemplo 5.10

```
Algoritmo DEMOX
var entero: A, x, y
inicio
 x ⇓ 5
 A ⇓ 10
 y ⇓ F(x)
 escribir (x, A, y)
fin

entero función F (E entero: N)
var
 entero: X
inicio
 A ⇓ 5
 X ⇓ 12
 Devolver (N-A)
fin_función
```

A la bariable global A se puede acceder desde el algoritmo y desde la función. Sin embargo, X identifica a dos variables distintas: una local al algoritmo y solo se pude acceder desde el y otra local a la función.

Al ejecutar el algoritmo se obtendrian los siguientes resultados:

X = 5	
A = 10	
Y = F(5)	invocación a la función F(N) se realiza un paso del parámetro actual X al parámetro formal N
A = 5	se modifica el valor de A en el algoritmo principal por ser A global
X = 12	no se modifica el valor de X en el algoritmo principal porque X es local
F = 5+5=10	se pasa el valor del argumento X (5) a travéz del parámetro X
Y = 10	

se escribirá la linea

5 5 10

ya que X es el valor de la variable local X en el algoritmo; A, el valor de A en la función, ya que se pasa este valor al algoritmo; Y es el valor de la función F (X).

5.5. COMUNICACIÓN CON SUBPROGRAMAS:

PASO DE PARÁMETROS

Cuando un programa llama a un subprograma,la información se comunica a travéz de la lista de parámetrosy se establece una correspondencia automatica entre los parámetros formales y actuales.*Los parámetros actuales son sustituidos o utilizados en lugar de los parámetros formales.*

La declaración del subprograma se hace con

```
procedimiento nombre (clase
tipo_de_dato: F1
                      clase
tipo_de_dato: F2
                      . . . . . .
. . . . .
                      clase
```

Y la llamada al subprograma con

```
Llamar_a nombre (A1, A2, ...,An)
```

Donde F1, F2, . . . Fn son los parámetros formales y A1, A2, . . . An los parámetros actuales o reales.

Las clases de parámetros podrían ser:

```
(E) Entrada
(S) Salida
(E/S) Entrada/Salida
```

Existen dos métodos para establecer la correspondencia de parámetros:

1. *Correspondencia posicional.*la correspondencia se establece aparejando los parámetros reales y formales según su posición en las listas: así, Fi se corresponde con Ai, donde i = 1, 2, . . . n. este método tiene algunas desventajas de legibilidad cuando el número de parámetros es grande.
2. *Correspondencia por el nombre explicito,*también llamado *método de paso de parámetros por nombre.* En este método, en las llamadas se indica explicitamente la correspondencia entre los parámetros reales y formales. Este método se utiliza en Ada. Un ejemplo sería:

   ```
   Sub (Y => B, X => 30);
   ```

   Que hace corresponder el parámetro actual Bcon el formal Y, y el parámetro actual 30 con el formal X durante la llamada de SUB.

Por lo general, la mayoría de los lenguajes usan exclusivamente la correspondencia posicional y ése será el método empleado en este libro.

Las cantidades de información que pueden pasarce como parámetros son *datos, tipos* –en los lenguajes que admiten su declaración- y *subprogramas*.

5.5.1. Paso de parámetros

Existen diferentes métodos para la *transmición* o *el paso de parámetros o subprogramas.* Es preciso conocer el método adoptado por cada lenguaje , ya que la eleccion puede afectar ala semántica del lenguaje. Dicho de otro modo, un mismo programa puede producir diferentes resultados bajo diferentes sistemas de paso de parámetros.

Los parámetros pueden ser clasificados como:

entradas: (**E**)	las entradas proporcionan valores desde el programa que llama y que se utilizan dentro de un procedimiento. En los programas función, las entradas son los argumentos en el sentido tradicional:
salidas: (**S**)	Las salidas producen los resultados del subprograma: de nuevo si se utiliza el caso una función, éste devuelve un valor calculado por dicha función, mientras que con procedimientos puede calcularce cero, una o varias salidas:
entrada/salida ***(E/S)***	Un solo parámetro se utiliza para mandar argumentos a un programa y para devolver resultados

Desgraciadamente, el conocimiento del tipo de paámetros no es suficiente para caracterizar su funcionamiento; por ello, examinaremos los diferentes métodos que se utiolizan para pasar o transmitir parámetros.

Los métodos mas empleados para realizar el paso de parámetros son:

- *paso por valor (*también conocido por *parámetro valor*)
- *paso por referencia o diferencia* (también conocidopor *parametro variable*)
- *paso por nombre*
- *paso por resultado*

5.5.2. Paso por valor

El paso por valor se utiliza en muchos lenguajes de programación: por ejemplo, C, Modulo-2, Pascal, Algol y Snobol. La razón de su popularidad es la analogía con los argumentos de una función, donde los valores se proporcionan en el orden de cálculo de resultados. Los parámetros se tratan como variables locales y los valores iniciales se proporcionan copiando los valores de los correspondientes argumentos.

Los parametros formales –locales a la función- resiben como valores iniciales los valores de parametros actuales y con ellos se ejecutan las acciones descritos en el subprograma,

No se hace diferencia entre un argumento que es variable, constante o expreción, ya que solo importa el valor del argumento. La figura 5.6 muestra el mecanismo de paso por valor de un procedimiento con tres parámetros.

El mecanismo de paso se resume así:

Valor primer parámetro: A = 5
Valor segundo parámetro: constante = 18
Valor tercer parametro: expresión B * 3 + 4 = 25

El valor 5.18 y 25 se transforma en los parámetros X, Y, Z respectivamente, cuando se ejecuta el procedimiento.

Aunque el *paso por valor* es sencillo, tiene una limitación acusada: *no existe ninguna otra conexión con los parametros actuales,* entonces los cambios que se produscan por efecto del subprograma no producen cambios en los argumentos originales y, por consiguiente, no se pueden pasar valores de retorno al punto de llamada: es decir, todos los *parámetros* son sólo de *entrada.*El parametro actual no puede modificarse por el subprograma. Cualquier cambio realizado

en los valores de los parametros formales durante la ejecución del subprograma se destruye cuando se termina el subprograma.

La llamada por valor no devuelve información al programa que llama.

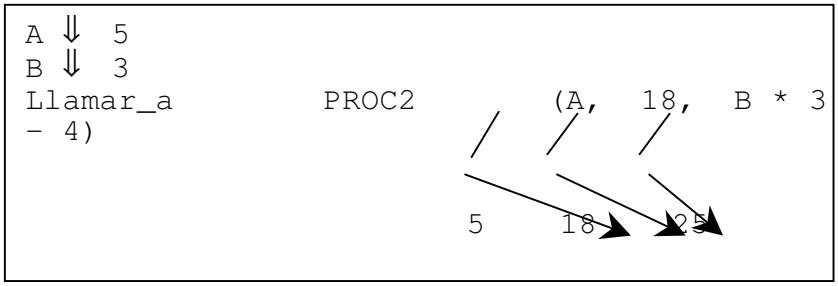


Figura 5.6. Paso por valor.

Existe una variante de la llamada por valor y es lamada por *valor resultado.*las variables indicadas por los parámetros formales se inicializan en la llamada por valor tras la ejecución del subprograma; los resultados (valores de parámetrosd formales) se transfieren alos actuales. Este método se utiliza en algunas versiones de FORTRAN.

5.5.4. Paso por referencia

En numerosas ocasiones se requieren que ciertos parámetros sirvan como parámetros de salida, es decir, se devuelvan los resultados a la mitad o programas que llama. Este método se denomina *paso por referencia* o también de *llamada por dirección o variable.* La unidad que llama pasa ala unidad llamada la dirección del parámetro actual (que está en el ambiente de la unidad llamante). Una referencia al correspondiente parámetro formal se trata como una referencia a la posición de memoria, cuya dirección se ha pasado. Entonces una variable pasada como parámetro real es compartida, es decir, se puede modificar directamente por el subprograma.

Este método existe en FORTRAN, COBOL, Modulo-2, Pascal, PL/1 y Algol 68. La característica de este método se debe a la simplicidad y su analogía directa con la idea de que las variables tienen una posición de memoria asignada desde la cual se pueden obtener o actualizar sus valores.

El área de almacenamiento (direcciones de memoria) se utiliza para pasar información de entrada y/o salida: en ambas direcciones.

En este método los parámetros son de entrada/salida y los parámetros se denominan *parámetros variables.*

Los parámetros valor y los parámetros variable se suelen definir en la cabeza del subprograma. En el caso de lenguajes como Pascal, los parámetros variables deben ir precedidos por la palabra clave var;

```
Program muestra:
//parámetros actuales a y b, c y d paso por referencia
  procedure prueba (var x, y: integer);
  begin //procedimiento
//procedimiento de los valores x e y
  end;

begin
    .
    .
    .
1. prueba(a, c);
```

```
    .
    .
    .
2. prueba(b, d);
    .
    .
    .
end.
```

La primera llamada en (1) produce que los parámetros a y c sean sustituidos por x e y si los valores de x e y se modifican dentro del a y c en el algoritmo principal. De igual modo, b y d son sustituidos por x e y, y cualquier modificación de x o y en el procedimiento afectará también al programa principal.

La llamada por referencia es muy útil para programas donde se necesita la comunicación del valor en ambas direcciones.

Notas:

Ambos métodos de paso de parámetros se aplican tanto a la llamada de funciones como a las de procedimientos:

1. Una función tiene la posibilidad de devolver los valores al programa principal de dos formas: a)como valor de la función, b)por medio de argumentos gobernados por la llamada de referencia en la correspondencia parámetro actual-parámetro formal.
2. Un procedimiento solo puede devolver valores por el método de devolución de resultados.

El lenguaje Pascal permite que el programador especifique el tipo de paso de parámetros y en un mismo subprograma, unos parámetros se pueden especificar por valor y otros por referencia.

```
Procedure Q (i:integer; var j:integer);
Begin
  i := i - 10;
    j := j - 10;
  write (i, j)
end;
```

Los parámetros formales son `i, j,` donde `i` se pasa por valor y `j` por referencia.

5.5.4. Comparaciones de los métodos de paso de parámetros

Para examinar de modo práctico los diferentes métodos, consideremos un ejemplo único y veamos los diferentes valores que toman los parámetros. El algoritmo corresponde con un procedimiento SUBR:

```
algoritmo DEMO
var
  entero: A,B,C.
inicio   //DEMO
  A ⇓ 3
  B ⇓ 5
  C ⇓ 17
  llamar_a SUBR (A,A,A-B,C)
  escribir(C)
fin      //DEMO
```

```
procedimiento SUBR <Modo> entero: x,y:E entero:z: <Modo> entero: y)


inicio
  x ⇓ x + 1
  y ⇓ y + 2
fin_procedimiento
```

Modo por valor:

a) *sólo por valor:*

no se admite ningún resultado, por consiguiente, C no varía C = 17

b) *valor_resultado:*

A = 3		X = A = 3
B = 5	pasa al procedimiento	Y = A = 3
C = 17		Z = A + B = 8
		V = C = 17

al ejecutar el procedimiento quedará:

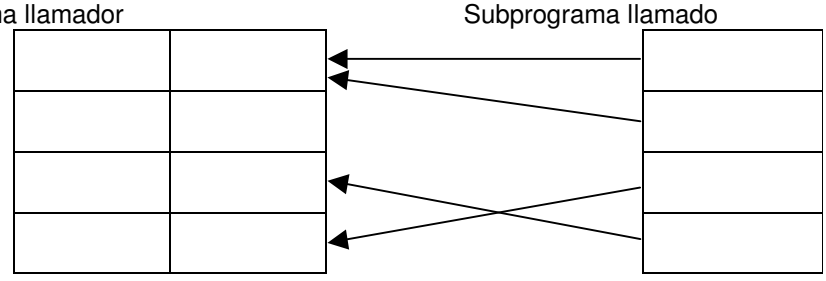
X = X + 1 = 3 + 1 = 4
V = Y + Z = 3 + 8 = 11

El parámetro llamado v pasa el valor del resultado v a su parámetro actual correspondiente, C. Por tanto, C = 11.

Modo por referencia:

Posiciones de la memoria del Programa llamador

Posiciones de la memoria del Subprograma llamado



C recibirá el valor 12.

Utilizando variables globales:

```
algoritmo DEMO
var entero: A,B,C
inicio
 A ⇓ 3
 B ⇓ 5
```

```
      C ⇓ 17
      llamar_a SUBR
      escribir (C)
     fin

procedimiento SUBR
inicio

a  ←        a + 1
c  ←        a + a – b
fin_procedimiento
```

Es decir, el valor de C será 13.
La llamada por referencia es el sistema estándar utilizado por FORTRAN para pasar parámetros.
La llamada por nombre es estándar en Algol 60. Simula 67 proporciona llamadas por valor ,referencia y nombre.
Pascal permite pasar bien por valor bien por referencia.

```
procedure demo (ytinteger; var z:real)
```

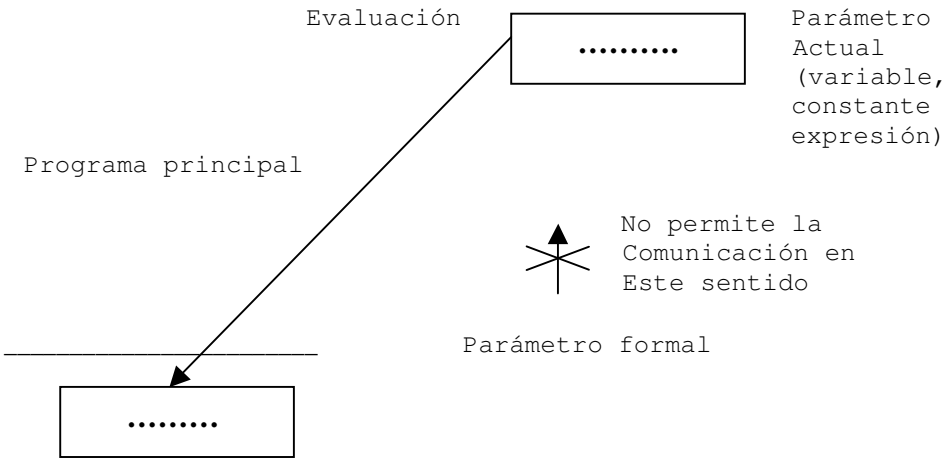
Especifica que y se pasa por valor mientras que z se pasa por referencia – indicado por la palabra reservada var - . La elección entre un sistema u otro puede venir determinado por diversas consideraciones, como evitar efectos laterales no deseados provocados por modificaciones inadvertidas de parámetros formales (véase 5.7)

5.5.5. Síntesis de la transmisión de parámetros.

Los métodos de transmisión de parámetros más utilizados son por valor y por referencia.

El paso de un parámetro por valor significa que el valor del argumento – parámetro actual o real – se asigna al parámetro formal. En otras palabras antes de que el subprograma comience a ejecutarse, el argumento se evalúa a un valor específico (por ejemplo, 8 ó 12). Este valor se copia entonces en el correspondiente parámetro formal dentro del subprograma.

Una vez que el procedimiento arranca, cualquier cambio del valor de tal parámetro formal no se refleja en un cambio en el correspondiente argumento. Esto es cuando el subprograma se termine el argumento actual tendrá exactamente el mismo valor que cuando el subprograma comenzó, con independencia de lo que haya sucedido al parámetro formal. Este método es el método por defecto en Pascal si no se indica explícitamente otro. Estos parámetros de entrada se denominan parámetros valor. En los algoritmos indicaremos como **<** modo **>** E (entrada). El paso de un parámetro por referencia o dirección se llama parámetro variable, en oposición al parámetro por valor. En este caso la



Subprograma

Figura 5.7. Paso de un parámetro por valor.

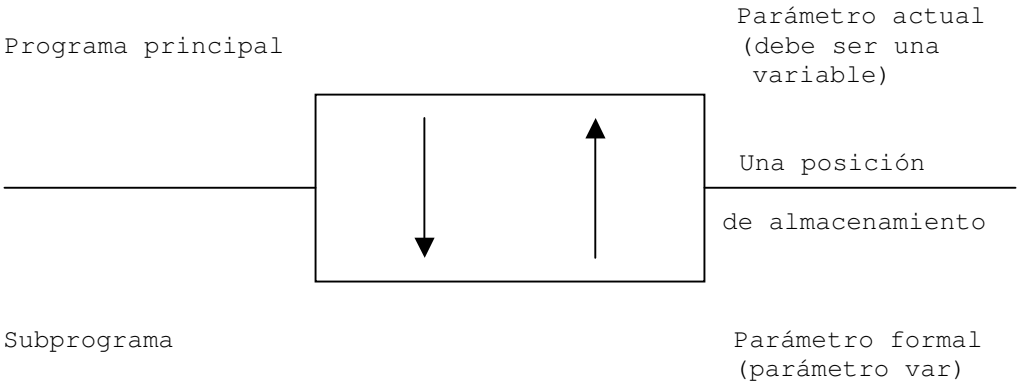


Figura 5.8. Paso de un parámetro por referencia

Posición o dirección (no el valor) del argumento o parámetro actual se envía al subprograma. Si a un parámetro formal se le da el atributo de parámetro variable – en Pascal con la palabra reservada **var** – y si el parámetro actual es una variable, entonces un cambio en el parámetro formal se refleja en un cambio en el correspondiente parámetro actual, ya que ambos tienen la misma posición de memoria.

Para indicar que deseamos transmitir un parámetro por dirección, lo indicaremos con la palabra variable – en pascal se indica con la palabra reservada **var – y especificaremos como** < modo > E/S (entrada/salida) o S (salida).

Ejemplo 5.11

Se trata de realizar el cálculo del área de un círculo y la longitud de la circunferencia en función del valor del radio leído desde el teclado.

Recordemos las fórmulas del área del círculo y de la longitud de la circunferencia:

```
A = pi x r = pi x rxr
C = 2pi x r = 2 x pi x r     donde pi = 3.141592
```

Los parámetros de entrada: `radio`
Los parámetros de salida: `área, longitud`.
El procedimiento `círculo` calcula los valores pedidos.

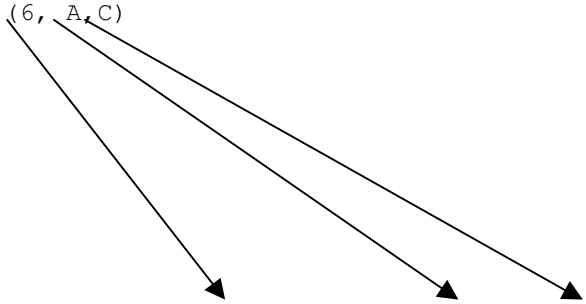
```
Procedimiento círculo (E real: radio, S real: área, longitud)
  // parámetros valor: radio
  //parámetros variable: área, longitud
var
  real: pi
inicio
  pi    ←  3.141592
  área  ←     pi * radio * radio
```

```
  longitud  ←  2 * pi * radio
fin _procedimiento
```

Los parámetros formales son: radio, área, longitud, de los cuales son valor (radio) y variable (área, longitud).

Invoquemos el procedimiento `círculo` utilizando la instrucción

```
Llamar_a círculo (6, A, C)
  // (programa principal)
inicio
//llamada al procedimiento
  llamar_a círculo (6, A, C)
```



```
fin

procedimiento círculo (E real: radio; S real: área, longitud)
  //parámetros valor: radio
  //parámetros variables: área, longitud.

Inicio
  pi ←  3.141592
  área ←  pi * radio * radio
  longitud  ←  2 * pi * radio
fin procedimiento
```

Ejemplo 5.12

Consideramos un subprograma N con dos parámetros formales: i, transmitido por valor, y j, por variable.

```
Algoritmo M
// variables A, B enteras
var
entero: A, B

inicio
A  ←  2
B  ←  3
Llamar_a N (A, B)
Escribir (A, B)
Fin    // algoritmo M

Procedimiento N (E entero: I; E/S entero: j)
// parámetros valor i
//parámetros variables j
inicio
i  ←  i + 10
```

```
j ← j + 10
escribir (i, j)
fin_procedimiento
```

Si se ejecuta el procedimiento N, veamos qué resultados se escribirán:

```
A y B son parámetros actuales.
I y j son parámetros formales.
```

Como i es por valor se transmite el valor de A a i, es decir, i = A = 2. Cuando i se modifica por defecto de i ← i + 10 a 12, A no cambia y, por consiguiente, a la terminación de N, A sigue valiendo 2.

El parámetro B se transmite por referencia, es decir, j es un parámetro variable. Al comenzar la ejecución de N, B se almacena como el valor j y cuando se suma 10 al valor de j, i en sí mismo no cambia. El valor del parámetro B se cambia a 13. Cuando los valores i, j se escriben en N, los resultados son:

12 y 13

pero cuando retornan a M y al imprimir los valores de A y B, sólo ha cambiado el valor B. El valor de i = 12 se pierde en N cuando éste ya termina. El valor de j también se pierde, pero éste es la dirección, no el valor 13.

Se escribirá como resultado final de la instrucción ***escribir*** (A, B) :

2 13

5.6. FUNCIONES Y PROCEDIMIENTOS COMO PARÁMETROS

Hasta ahora los subprogramas que hemos considerado implicaban dos tipos de parámetros formales: parámetros valor y parámetros variable. Sin embargo, en ocasiones se requiere que un procedimiento o función dado invoque a otro procedimiento o función que ha sido definido fuera del ámbito de ese procedimiento o función. Por ejemplo, se puede necesitar que un procedimiento P invoque la función F que puede estar o no definida en el procedimiento P; esto puede conseguirse transfiriendo como parámetro el procedimiento o función externa (P) o procedimiento o función dado (por ejemplo, el P). En resumen, algunos lenguajes de programación – entre ellos Pascal – admiten parámetros procedimiento y parámetros función.

Ejemplos:

```
Procedimiento P (E func:F1; E REAL; X,Y)
Real función F (E func: F1, F2, E entero: x,y)
```

Los parámetros formales del procedimiento P son la función F1 y las variables X e Y, y los parámetros formales de la función F son las funciones F1 y F2, y las variables x e y.

Procedimientos función

Para ilustrar el uso de los parámetros función, consideremos la función integral para calcular el área bajo una curva f (x) para un intervalo a < = x < = b.

La técnica conocida para el cálculo del área es subdividir la región en rectángulos, como se muestra en la Figura 5.9 y sumar las áreas de los rectángulos. Estos rectángulos se construyen subdividiendo el intervalo (a, b) en m subintervalos iguales y formando rectángulos con estos subintervalos como bases y alturas dadas por los valores de f en los puntos medios de los subintervalos.

La función integral debe tener los parámetros formales a, b y n, que son parámetros valor ordinarios actuales de tipo real; se asocian con los parámetros formales a y b; un parámetro actual de tipo entero – las subdivisiones – se asocia con al parámetro formal n y una función actual se asocia con el parámetro formal f.

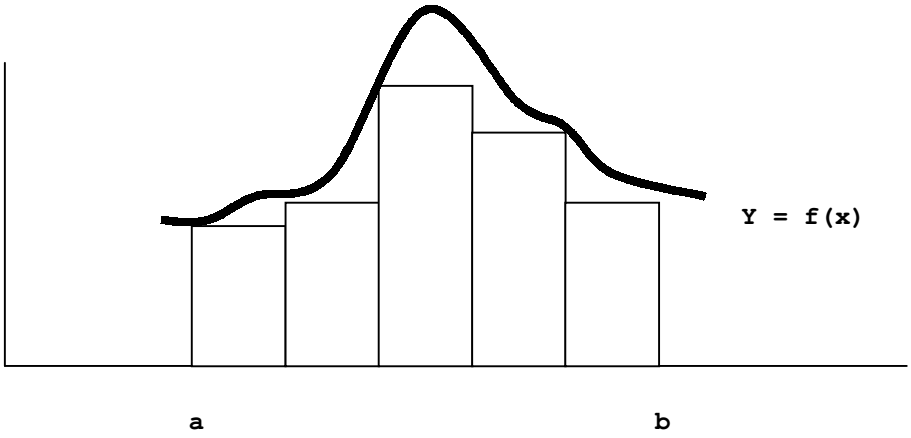


Figura 5.9. Cálculo del área bajo la curva f (x)

Los parámetros función se designan como tales con una cabecera de función dentro de la lista de parámetros formales. La función integral podrá definirse por

```
real: función integral (E real: a, b; E entero: n)
el tipo func-tip real función (E real: x) : func.
```

Aquí la función func: F especifica que f es una función parámetro que denota una función cuyo parámetro formal y valor son de tipo real. El correspondiente parámetro función actual debe ser una función que tiene un parámetro formal real y un valor real. Por ejemplo, si integrado es una función de valor real con un parámetro de tipo real función, es decir, de tipo: func.

```
Área ← Inegral (integrado, 0, 1, 5, 20)
```

Es una referencia válida a función.

Diseñar un algoritmo que utilice la función integral para calcular el área bajo el gráfico de las función es f1 (x) = $x^3 - 6x^2 + 10x$ e Integrado por Integrado (x) = $x^2 + 3x + 2$ para 0< = x < = 4.

```
Algoritmo Área_bajo_curvas

Tipo
Real función (E real : x) : func
Var
Real: a, b
Entero: n

Inicio
Escribir (¿Entre qué límites?)
Leer (n
Escribir (integral (f1, a,b,n))
Escribir (integral (integrado a, b, n))
Fin
Real función f1 (E real): x
Inicio
```

```
Devolver (x * x * 3 * x * 2)
Fin_funcón

Real función integral (E func: f; E real: a, a; E entero: n)
Var
Real: baserectángulo, altura,x,s
Entero: i

Inicio
baserectángulo ←——  (b – a) / n
x ←  a + baserectángulo / 2
s ←  0
desde i ←  1 hasta n hacer
altura ←  f (x)
fin_desde
devolver (s)
fin_función
```

5.7. LOS EFECTOS LATERALES

Las modificaciones que se produzcan mediante una función o procedimiento en los elementos situados fuera del subprograma (función o procedimiento) se denominan efectos laterales. Aunque en algunos casos los efectos laterales pueden ser beneficiosos en la programación, es conveniente no recurrir a ellos de modo general. Consideramos a continuación los efectos laterales en funciones y en procedimientos.

5.7.1. En procedimientos

La comunicación del procedimiento con el resto del programa se debe realizar normalmente a través de parámetros. Cualquier otra comunicación entre el procedimiento del programa se conoce como efectos laterales. Como ya se ha comentado, los efectos laterales son perjudiciales en la mayoría de los casos como se indica en la Figura 5.10.

Si un procedimiento modifica una variable global (distinta de un parámetro actual), este es un efecto lateral. Por ello, excepto en contadas ocasiones, no debe aparecer en la declaración del procedimiento. Si se necesita una variable temporal en un procedimiento, utilicé una variable local, no una variable global. Si se desea que el programa modifique el valor de una variable global, utilice un parámetro formal variable en la declaración del procedimiento y a continuación utilice la variable global como el parámetro actual en una llamada al procedimiento.

En general, se debe seguir la regla de “ninguna variable global en procedimientos”, aunque esta prohibición no significa que los procedimientos no puedan manipular5 variables globales. De hecho, el cambio de variables globales se debe pasar al procedimiento como parámetros actuales. Las variables globales no se deben utilizar directamente en las instrucciones en el cuerpo de un procedimiento; en su lugar, utilice un parámetro formal o variable local.

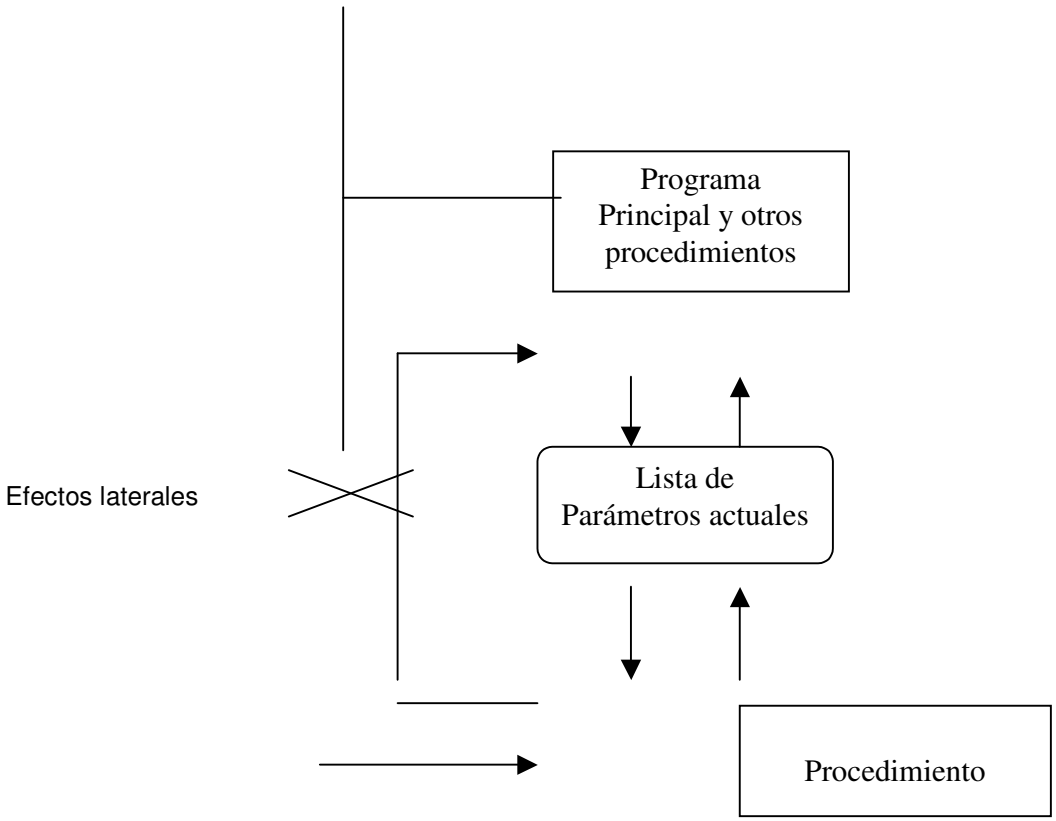
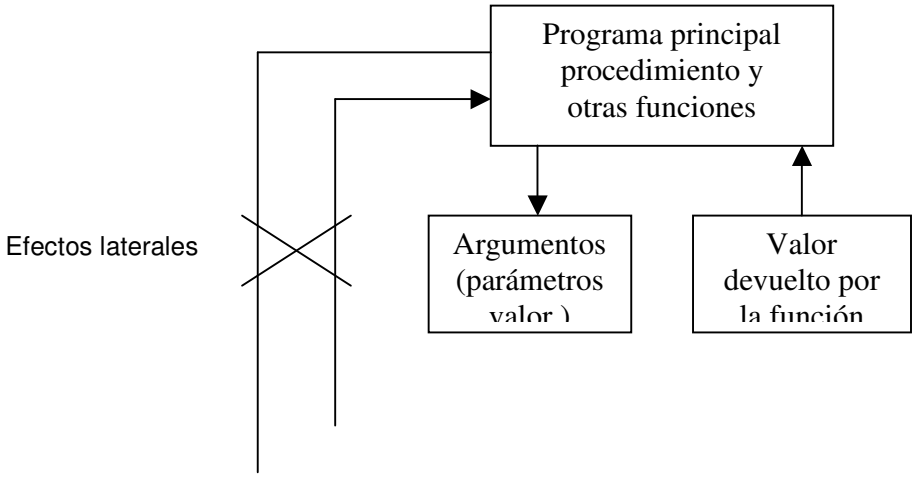


Figura 5.10. Efectos laterales en procedimientos

En aquellos lenguajes en que es posible declarar constantes - como Pascal - se pueden utilizar constantes globales en una declaración de procedimiento; la razón reside en el hecho de que las constantes no pueden ser modificadas por el procedimiento y por consiguiente, no existe peligro de que se pueda modificar inadvertidamente.

5.7.2. En funciones

Una función toma los valores de los argumentos y devuelve un único valor. Sin embargo, al igual que los procedimientos una función – en algunos lenguajes de programación – puede hacer cosas similares a un procedimiento o subrutina. Una función puede tener parámetros variables además de parámetros valor en la lista de parámetros formales. Una función puede cambiar el contenido de una variable global y ejecutar instrucciones de entrada / salida (escribir un mensaje en la pantalla, leer un valor del teclado, etc). Estas operaciones se conoce como parámetros laterales y se deben evitar.



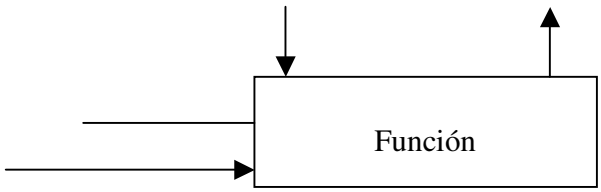


Figura 5.11. Efectos laterales en una función

Los efectos laterales están considerados – normalmente – como una mala técnica de programación, pues hacen más difícil de entender los programas.

Toda la información que se transfiere entre procedimientos y funciones debe realizarse a través de la lista de parámetros y no a través de variables globales. Esto convertirá el procedimiento o función en módulos independientes que pueden ser comprobados y depurados por sí solos, lo que evitará no preocuparnos por el resto de las partes del programa.

5.8. RECURSIÓN (RECURSIVIDAD)

Como ya se conoce, un subprograma puede llamar a cualquier otro subprograma y éste a otro y así sucesivamente; dicho de otro modo, los subprogramas se pueden anidar. Se puede tener

```
A llamar _a B, B llamar_a C,  C llamar_a D
```

Cuando se produce el retorno de los subprogramas a la terminación de cada uno de ellos, el proceso resultante será:

```
D retornar_a C,   C retornar_a B,  B retornar_a A
```

¿Qué sucedería si dos subprogramas de una secuencia son los mismos?

```
A llamar_a A
```

O bien:

```
A llamar_a B,  B llamar_a A
```

En primera instancia parece incorrecta. Sin embargo, existen lenguajes de programación – Pascal, C , entre otros – en que un subprograma puede llamarse así mismo.

Una función o procedimiento que se puede llamar a sí mismo se llama recursivo. La recursión (recursividad) es una herramienta muy potente el algunas aplicaciones, sobre todo de cálculo. La recursión puede ser utilizada como una alternativa a la repetición o estructura repetitiva.

El uso de la recursión es particularmente idóneo para la solución de aquellos problemas que pueden definirse de modo natural en términos recursivos.

La escritura de un procedimiento o función recursivo es similar a sus homónimos no recursivos; sin embargo, para evitar que la recursión continúe indefinidamente es preciso incluir una condición de terminación.

La razón de que existan lenguajes que admiten la recursividad se debe a la existencia de estructuras específicas tipo pilas (stack, en inglés) para este tipo de procesos y memorias dinámicas. Las direcciones de retorno y el estado de cada subprograma se guardan en estructuras

tipo pilas (véase Capítulo 11). En el Capítulo 11 se profundizará en el tema de las pilas; ahora nos centraremos sólo en el concepto de recursividad y en su comprensión con ejemplos básicos.

Ejemplo 5.13

Muchas funciones matemáticas se definen recursivamente. Un ejemplo de ello es el factorial de un número entero n

La función factorial se define como

1 si n = 0 0! = 1

n! =
(n) x (n-1) x (n-2) x ... x 3 x 2 x 1 si n>0. (n-1) . (n-2) ....3 . 2 . 1

Si se observa la formula anterior cuando n > 0, es fácil definir n! en función de (n – 1)!
Por ejemplo, 5!

5 ! = 5 x 4 x 3 x 2 x 1 = 120
4 ! = 4 x 3 x 2 x 1 = 24
3 ! = 3 x 2 x 1 = 6
2 ! = 2 x 1 = 2
1 ! = 1 x 1 = 1
0 ! = 1 = 1

Se pueden transformar las expresiones anteriores en

5 ! = 5 x 4 !
4 ! = 4 x 3 !
3 ! = 3 x 2 !
2 ! = 2 x 1 !
1 ! = 1 x 0 !

En términos generales sería:

1 si n = 0
n ! =
n (n – 1) ! si n > 0

La función FACTORIAL **de N expresada en términos recursivos sería:**

FACTORIAL ---N FACTORIAL (N – 1)

La definición de la función sería:

```
entero función  factorial ( E entero: n)
//calculo recursivo del factorial
inicio
   si n = 0 entonces
     devolver (1)
   si_no devolver (n * factorial (n - 1) )
   fin_si
fin_funcion
```

Para demostrar cómo esta versión recursiva de FATORIAL calcula el valor de n!, consideremos el caso de n = 3. Un gráfico se representa en la Figura 5.12.

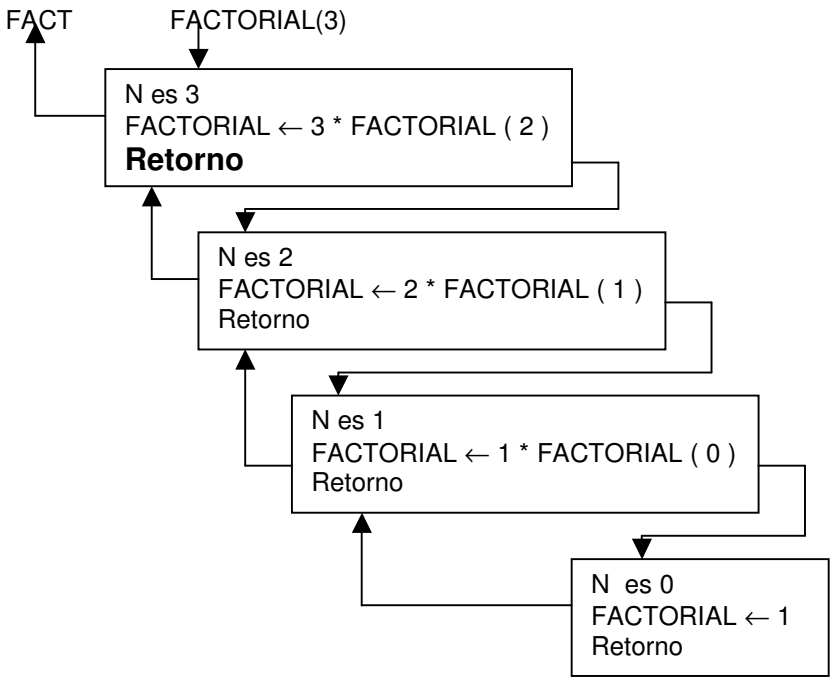
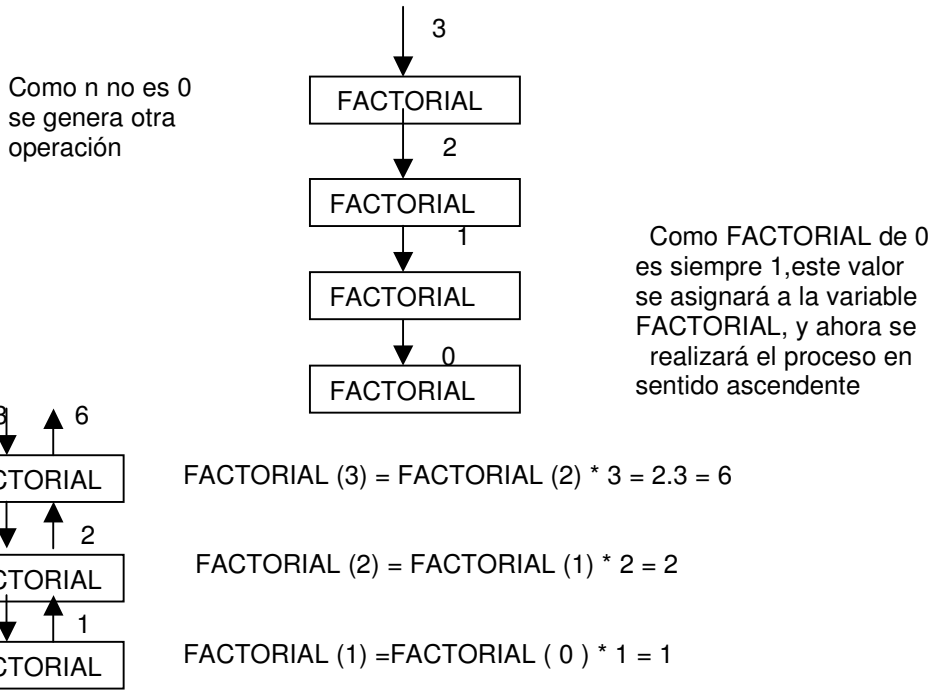
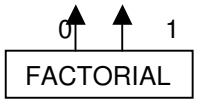


Figura 5.12. cálculo recursivo de FACTORIAL de 3.

Los pasos sucesivos extraídos de la figura 5.12 son:





Ejemplo 5.14

Otro ejemplo típico de una función recursiva es la serie Fibonacci. Esta serie fue concebida originalmente como modelo para el crecimiento de una granja de conejos (multiplicación de conejos) por el matemático italiano del siglo XVI, Fibonacci.

La serie es la siguiente:

1, 1, 2, 3, 5, 8, 13, 21, 34....
Esta serie crece muy rápidamente; como ejemplo, el término 15 es 610.
La serie de Fibonacci (*fib*) se expresa así:

fib(1) = 1
fib(2) =1
fib(n) =fib(n – 1) +fib(n- 2) para n >2

Una función recursiva que calcula el elemento enésimo de la serie de Fibonacci es:

```
entero funcion fibonacci(E entero: n)
//calculo del elemento n-esimo
inicio
  si (n = 1) o (n = 2) entonces
    devolver (1)
  si_no
    devolver (fibonacci(n - 2) + fibonacci(n - 1))
  fin_si
fin_funcion
```

Aunque es fácil de escribir la función de Fibonacci, no es muy eficaz definida de esta forma, ya que cada paso recursivo genera otras dos llamadas a la misma función.

ACTIVIDADES DE PROGRAMACIÓN RESUELTAS

5.1.Diseñar el algoritmo que calcule el factorial de un número entero.

```
entero funcion factorial(E entero: n)
var entero: f, i
inicio
  si = 0 entonces
    devolver (1)
  si_no
    desde i ← 1 hasta n hacer
      f ← f * i
    fin_desde
```

```
      devolver (f)
    fin_si
  fin_funcion
```

5.1.Diseñar un algoritmo que calcule el máximo común divisor de dos números mediante el algoritmo de Euclides.
Sean los dos números A y B. El método para hallar el máximo común divisor (mcd) de dos números A y B por el método de Euclides es:

1.Dividir el número mayor (A) por el menor (B). Si el resto de la división es cero, el número B es el máximo común divisor.
2.Si la división no es exacta, se divide el número menor (B) por el resto de la división anterior.
3.Se siguen los pasos anteriores hasta obtener un resto cero. El último divisor es el mcd buscado.

Algoritmo

```
entero función mcd (E entero: a , b)
inicio
  mientras a <> b hacer
      si  a > b entonces
        a ← a – b
     si_no
        b ← b – a
     fin_si
  fin_mientras
  devolver(a)
fin_funcion
```

5.3.Para calcular el máximo común divisor (mcd) de dos números se recurre a una función específica definida con un subprograma. Se desea calcular la salida del programa principal con dos números A y B, cuyos valores son 10 y 25, es decir, el mcd(A, B) y comprobar el método de paso de parámetros por valor.

```
   algoritmo Maxcomdiv
   var
     entero: N, X, Y
   inicio //programa principal
     x ← 10
     y ← 25
     n ← mcd(x, y, n)
     escribir (X, Y, Z)
   fin
   entero funcion mcd (E entero: a,b)
   inicio
     mientras a <> b hacer
       si a > b entonces
         a ← a – b
       si_no
         b ← b – a
       fin_si
     fin_mientras
     devolver (a)
```

```
fin
fin_funcion
```

Los parámetros formales son a y b y reciben los valores de X e Y.

a = 10
b = 25

Las variables locales a las función son X e Y del algoritmo principal.

Variables del Programa principal			*Variables de la función*		
X	Y	N	a	b	mcd (a, b)
10	25		10	25	

Las operaciones del algoritmo son:

a = 10 b = 25

1. b > a realizará la operación b ← b – a
   y, por consiguiente, b tomará el valor 25 – 10 = 15
   y a sigue valiendo 10.

2. a = 10 b = 15
   se realiza la misma operación anterior
   b ← b – a, es decir, b = 5
   a permanece inalterable

3. a = 10 b = 5
   como a > b entonces se realiza a ←a – b, es decir, a = 5.

Por consiguiente, los valores finales, serían:

a = 5 b =5 mcd (a, b) = 5

Como los valores A y B no se pasan al algoritmo principal, el resultado de su ejecución será:

10 25 5

5.4. Escribir un algoritmo que permita ordenar tres números mediante un procedimiento de intercambio en dos variables (paso de parámetro por referencia).

El algoritmo que permite realizar el intercambio de los valores de variables numéricas es el siguiente:

AUXI ← A
A ← B
B ← AUXI

y la definición del procedimiento será:

PROCEDIMIENTO intercambio (E/S real: a, b)
var real : auxi

```
inicio
      auxi ← a
      a ← b
      b  ←auxi
   fin_procedimiento
```

El algoritmo de ordenación se realizará mediante llamadas al procedimiento intercambio.

algoritmo Ordenar_3_numeros
var real : x, y, z

```
inicio
  escribir (´ Deme 3 numeros reales ´)
  leer ( x, y, z )
  si x > y entonces
     intercambio ( x, y )
  fin_si
  si y > z  entonces
    intercambio ( y, z )
  fin_si
  si x > y  entonces
     intercambio (x, y )
  fin_si
  escribir ( x, y, z )
fin
```

Paso de parámetro por referencia

Los tres números x, y, z que se van a ordenar son:
132 45 15

Los pasos sucesivos al ejecutarse el algoritmo o programa principal son:
1. Lectura x, y, z parámetros actuales

   X = 132
   Y = 45
   Z = 15

2. Primera llamada al procedimiento intercambio (a, b) X > Y.
   La correspondencia entre parámetros será la siguiente:

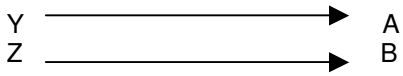
Parámetros actuales	Parámetros formales
X	A
Y	B

   Al ejecutarse el procedimiento se intercambiarán los valores de A y B que se devolverán a las variables X e Y; luego valdrán:

   X = 45
   Y = 132

3. Segunda llamada al procedimiento intercambio con Y > Z (ya que Y= 132 y Z = 35)

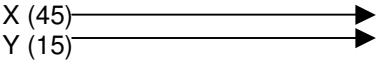
   Parámetro actuales Parámetros formales

   Antes llamada al procedimiento Y =132, Z = 15.
   Después terminación del procedimiento Z < 0 132, Y = 15, ya que A y B han intercambiado los valores recibidos, 132 y 15.

4. Los valores actuales de X, Y, Z son 45, 15, 132; por consiguiente, X < Y, y habrá que hacer otra nueva llamada al procedimiento intercambio.

   Parámetros actuales Parámetros formales

   Después de la ejecución del procedimiento A y B intercambiarán sus valores y valdrán A =15, B =45, por lo que se pasan al algoritmo principal X =15, Y= 45. Por consiguiente, el valor final de las tres variables será:

   X = 15 Y =45 Z = 132

   Ya ordenados de modo creciente.

5.5. Diseñar un algoritmo que llame a la función signo (x) y calcule: a) el signo de un número, b) el el signo de la función coseno.

Variables de entrada: P (real)
Variables de salida: Y –signo del valor P- (entero) Z –signo del coseno de P- (entero);

Pseudocódigo

```
Algoritmo Signos
var entero: Y, Z
      real: P

 inicio
    leer (P)
    Y ← signo (p)
    Z ← signo (cos (p))
    escribir (Y, Z)
 fin

 entero funcion signo (E real:x)
  inicio
     si  x > 0 entonces
         devolver ( 1)
     si_no
        si  x < 0 entonces
           devolver (-1)
        si_no
```

```
            devolver ( 0)
          fin_si
       fin_si
    fin_funcion
```

Notas de ejecución

Parámetro actual	Parámetro formal
p	x

El parámetro formal X se constituye por el parámetro actual. Así, por ejemplo, si el parámetro P vale –1.45. Los valores devueltos por la función signo que se asignarán a las variables Y, Z son:

Y ← signo (-1.45)
Z ← signo (cos (-1.45))

Resultado

Y = -1
Z = 1

EJERCICIOS

5.1. Diseñar una función que calcule la medida de tres números leídos del teclado y poner un ejemplo de su aplicación.

5.2. Diseñar la función FACTORIAL que calcule el factorial de un número entero en el rango 100 a 1.000.000.

5.3. Diseñar un algoritmo para calcular el máximo común divisor de cuatro números basado en un subalgoritmo función mcd (máximo común divisor de dos números).

5.4. Diseñar un procedimiento que realice el intercambio de valores de dos variables A y B.

5.5. Diseñar una función que encuentre el número el mayor de dos números enteros.

5.6. Diseñar una función que calcule Xn para X, variable real y n variable entera.

5.7. Diseñar un procedimiento que acepte un número de mes, un número de día y un número de año y los visualice en el formato.

dd /mm /aa

Por ejemplo, los valores 19, 09, 1987 se visualizarían como

19 /9 /87

y para los valores 3, 9. 1905

3 /9 /05

5.8. Realizar un procedimiento que realice la conversión de coordenadas polares (r, θ) a coordenadas cartesianas (X, Y)

X = r.cos (J)

Y = r.sen (J)

5.9. Escribir una función SALARIO que calcule los salarios de un trabajador para un número dado de horas trabajadas y un salario hora. Las horas que superen las 40 horas semanales se pagarán como extras con un salario hora 1.5 veces el salario ordinario.

5.10. Escribir una función booleana DIGITO que determine si un carácter es uno de los dígitos 0 al 9.