# Funciones y subrutinas

## *Introducción*

En lecciones anteriores se ha descrito el concepto de diseño descendente; esta técnica permite desarrollar algoritmos que resuelvan un problema mediante un proceso de refinamiento progresivo, descomponiendo el problema original en subproblemas menores hasta obtener una granularidad suficientemente fina que permita resolver cada subproblema mediante un algoritmo sencillo.

Generalmente, por cada nivel del diseño descendente se desarrollo un pseudocódigo de alto nivel que hace uso de acciones no primitivas; si se detecta que alguna de estas acciones no primitivas aparece más de una vez es posible nombrarla y utilizarla de forma repetida. Tales acciones con nombre se denominan subprogramas y pueden ser, a su vez, funciones y subrutinas.

La utilización de subprogramas proporciona múltiples ventajas:

- Facilitan la modularidad y estructuración de los algoritmos.
- Facilitan la lectura e inteligibilidad de los algoritmos.
- Permiten una economización del esfuerzo del programador al poder escribir código reutilizable en muchas partes de un mismo algoritmo.
- Facilitan la depuración y mantenimiento de los programas.

## *Funciones*

Las funciones son subrutinas que pueden tener o no argumentos pero que siempre devuelven un valor de retorno. Así pues, las invocaciones a funciones son expresiones de un tipo determinado y deben emplearse igual que cualquier expresión de su tipos; es decir, una llamada a función puede formar parte de una expresión aritmética, lógica o de cadena en función de su tipo, puede constituir la parte derecha de una sentencia de asignación, aparecer en una sentencia de salida o constituir un argumento para otro subprograma. Por otro lado, las llamadas a funciones **nunca** pueden formar una sentencia aislada ni constituir la parte izquierda de una sentencia de asignación.

Las invocaciones a funciones siguen, tanto en la notación algorítmica como en FORTRAN, la siguiente sintaxis:

```
nombre_función ([argumento][,argumento]*)
```

Como se puede ver, es posible tener funciones con 0 o más argumentos, las funciones que se utilizarán en nuestros algoritmos pueden estar definidas por el propio usuario o, en muchas ocasiones, ser funciones estándar, esto es, definidas por el propio compilador.

### Funciones intrínsecas o estándar

FORTRAN proporciona un gran número de funciones intrínsecas, algunas de ellas se corresponden con funciones de nuestra notación algorítmica mientras que otras equivalen a operadores de dicha notación; a continuación se muestra una tabla con las funciones FORTRAN más empleadas y su homóloga en la notación algorítmica.

| Operación | FORTRAN | | | | Notación algorítmica | | | |
|---|---|---|---|---|---|---|---|---|
| | Función | Tipo | Tipo argumentos | Sintaxis | Homólogo | Tipo | Tipo argumentos | Sintaxis |
| Raíz cuadrada | SQRT | real | real | SQRT(arg1) | √ | real | entero real | √arg1 |
| $e^n$ | EXP | real | real | EXP(arg1) | $e^n$ | real | entero real | $e^{arg1}$ |
| Logaritmo neperiano | LOG | real | real | LOG(arg1) | ln | real | entero real | ln(arg1) |
| Logaritmo decimal | LOG10 | real | real | LOG10(arg1) | log | real | entero real | log(arg1) |
| Seno | SIN | real | real | SIN(arg1) | sin | real | entero real | sin(arg1) |
| Coseno | COS | real | real | COS(arg1) | sin | real | entero real | cos(arg1) |
| Tangente | TAN | real | real | TAN(arg1) | sin | real | entero real | tan(arg1) |
| Arco seno | ASIN | real | real | ASIN(arg1) | asin | real | entero real | asin(arg1) |
| Arco coseno | ACOS | real | real | ACOS(arg1) | acos | real | entero real | acos(arg1) |
| Arco tangente | ATAN | real | real | ATAN(arg1) | atan | real | entero real | atan(arg1) |
| Valor absoluto | ABS | entero real | entero real | ABS(arg1) | \|\| | entero real | entero real | \|arg1\| |
| Módulo/resto | MOD | entero | entero | MOD(arg1,arg2) | % | entero | entero | arg1%arg2 |

## Funciones definidas por el usuario

Obviamente, resultan mucho más interesantes para nosotros las funciones definidas por el usuario, aquellas cuya naturaleza es especificada por el desarrollador del algoritmo. La invocación o llamada de estas funciones se realiza de forma idéntica a la de las funciones intrínsecas, sin embargo, en este caso existe además una sintaxis específica para poder definir el funcionamiento de la función.

Para definir una función se dispone de la palabra reservada `funcion` en la notación algorítmica y `function` en FORTRAN. Es preciso señalar además que dado que las funciones van a ser empleadas dentro del algoritmo principal deberán ser definidas **dentro** del mismo; a continuación se muestra la zona de definición de funciones para un algoritmo escrito en nuestra notación y en FORTRAN:

```
constantes
  definición de constante1
  definición de constante2
  ...
  definición de constanteN

variables
  declaración de variable1
  declaración de variable2
  ...
  declaración de variableN

inicio
  sentencia1
  sentencia2
  ...
  sentenciaN

  DEFINICIONES DE FUNCIONES
fin
```

```
program nombre_programa
  variables y constantes

  sentencia1
  sentencia2
  ...
  sentenciaN

  contains
  DEFINICIONES DE FUNCIONES
end
```

Para definir una función es preciso, tanto en la notación algorítmica como en FORTRAN, indicar:

- El nombre de la función.
- El tipo de la función.
- Los argumentos de la función y el tipo de los mismos.

Como se verá a continuación, las diferencias entre la notación y FORTRAN son fundamentalmente idénticas.

### *Definición de funciones en la notación algorítmica*

La sintaxis empleada para definir una función en la notación algorítmica es la siguiente:

```
tipo funcion nombre_funcion (arg1 ∈ tipo1, arg2 ∈ tipo2, ..., argN ∈ tipoN)
inicio
  sentencia1
  sentencia2
  ...
  sentenciaN
fin
```

A continuación se muestran una serie de ejemplos:

- Función sin argumentos que siempre retorna el valor lógico `verdadero`:

```
logico funcion siempreVerdad ()
inicio
  siempreVerdad ← verdadero
fin
```

- Función que recibe dos enteros y retorna el mayor de los dos:

```
entero funcion maximo (a ∈ entero, b ∈ entero)
inicio
  si a>b entonces
    maximo ← a
  si no
    maximo ← b
  fin si
fin
```

Así, un ejemplo de algoritmo completo que utilice las funciones anteriores sería el que se muestra a continuación.

```
inicio
  escribir siempreVerdad()
  escribir maximo(4,5)

  logico funcion siempreVerdad ()
  inicio
    siempreVerdad ← verdadero
  fin

  entero funcion maximo (a ∈ entero, b ∈ entero)
  inicio
    si a>b entonces
      maximo ← a
    si no
      maximo ← b
    fin si
  fin
fin
```

### *Definición de funciones en FORTRAN*

Para definir funciones en FORTRAN la sintaxis es la siguiente:

```
tipo function nombre_funcion (arg1, arg2, ..., argN)
  declaración arg1
  declaración arg2
  ...
  declaración argN

  sentencia1
  sentencia2
  ...
  sentenciaN
end
```

A continuación se muestran los ejemplos anteriores codificados en FORTRAN:

- Función sin argumentos que siempre retorna el valor lógico `verdadero`:

```
logical function siempreVerdad ()
  siempreVerdad=.true.
end function
```

- Función que recibe dos enteros y retorna el mayor de los dos:

```
integer function maximo (a, b)
  integer a,b

  if (a>b) then
    maximo=a
  else
    maximo=b
  end if
end function
```

El algoritmo del apartado anterior traducido a FORTRAN quedaría como sigue:

```
program nombre_programa
  print *, siempreVerdad()
  print *, maximo(4,5)

  contains

  logical function siempreVerdad ()
    siempreVerdad=.true.
  end function

  integer function maximo (a, b)
    integer a,b

    if (a>b) then
      maximo=a
    else
      maximo=b
    end if
  end function
end
```

## *Ámbito de las variables*

Como se puede apreciar por la sintaxis y los ejemplos presentados las funciones disponen de una serie de variables a las que tienen acceso denominadas argumentos; además, también es posible declarar variables (o definir constantes) dentro del cuerpo de una función (en general, dentro del cuerpo de cualquier subprograma). Tales variables y constantes pertenecen a un ámbito local mientras que las variables y constantes declaradas en el programa principal pertenecen a un ámbito global.

Es necesario señalar un par de aspectos importantes sobre las variables de ámbito local:

1. Estas variables sólo "existen" mientras se está ejecutando la subrutina.
2. Si una variable local es nombrada de la misma forma que una variable global es la local la que es utilizada en la subrutina.

**¡Atención!** También es necesario hacer notar un peculiaridad de FORTRAN respecto al ámbito de las variables que lo hace diferente de la notación algorítmica (y de la mayor parte de lenguajes de programación): los argumentos son pasados a las subrutinas por dirección de tal manera que si una subrutina modifica el valor de algún argumento dicha modificación afectará a la variable empleada en la llamada desde el programa principal; est fenómeno se conoce con el nombre de **efectos laterales**.

En el ejemplo siguiente se muestran las consecuencias que puede tener la mala utilización de esta característica de FORTRAN:

```
01. program programa
02.   implicit none
03.
04.   integer x,y
05.
06.   x=1;y=1
07.
08.   print *,suma(x,y)
09.   print *,suma(x,y)
10.
11.   contains
12.     integer function suma (a,b)
13.       implicit none
14.       integer a,b
15.
16.       a=a+b
17.       suma=a
18.     end function
19. end
```

A simple vista podría parecer que el programa debería producir dos salidas idénticas puesto que las líneas 08 y 09 invocan a la misma función con los mismos argumentos.

Sin embargo, si se observa la línea 16 se puede apreciar como la función `suma` modifica el valor del argumento `a` (que se corresponde con la variable `x` de la invocación) de tal manera que al volver a invocar la función el argumento que recibe ha cambiado su valor y la salida que se obtiene no es la esperada.

En resumen, **una función NUNCA debe modificar el valor de los argumentos que recibe**.

## *Subrutinas o procedimientos*

En muchas ocasiones puede interesarnos desarrollar un subprograma que no se vea afectado por las limitaciones de las funciones; es decir, puede interesarnos un subprograma que sea capaz de "retornar" varios valores o ninguno. Para esos casos existen las denominadas subrutinas o procedimientos; las subrutinas son subprogramas que no devuelven ningún resultado, por tanto no tienen tipo, y en los que es "lícito" emplear los efectos laterales antes mencionados para permitir al programa principal obtener varios valores "resultantes" de la ejecución del subprograma.

Las subrutinas se diferencian de las funciones fundamentalmente en la sintaxis de la definición y en la forma de invocarlos; dado que no tienen tipo alguno las subrutinas no pueden formar parte de expresiones ni aparecer en la parte derecha de una sentencia de asignación, deben aparecer única y exclusivamente en una sentencia de llamada a procedimiento.

### Definición de subrutinas en la notación algorítmica

La sintaxis empleada para definir una subrutina en la notación algorítmica es la siguiente:

```
accion nombre_subrutina ([ent|sal|ent sal] arg1∈tipo1, ..., [ent|sal|ent sal] argN∈tipoN)
inicio
  sentencia1
  sentencia2
  ...
  sentenciaN
fin
```

Como se puede ver aparecen dos nuevas palabras reservadas, `ent` y `sal`; si un argumento va precedido de la primera de ellas se trata de un argumento de entrada, si va precedido de `sal` de salida y si va precedido de ambos de entrada/salida. En caso de que no se especifique ninguna el argumento será únicamente de entrada.

A continuación se muestran una serie de ejemplos:

- Subrutina que dibuja n asteriscos:

```
accion asteriscos (n ∈ entero)
inicio
  i ∈ entero
  desde i←1 hasta n hacer
    escribir '*'
  fin desde
fin
```

- Subrutina que intercambia los dos argumentos que recibe:

```
accion intercambio (a,b ∈ entero)
inicio
  temporal ∈ entero
  temporal ← a
  a ← b
  b ← temporal
fin
```

Un ejemplo de algoritmo completo que utilice las subrutinas anteriores es el siguiente:

```
inicio
  x,y ∈ entero

  llamar asteriscos(5)

  x ← 0
  y ← 1
  escribir x,y
  llamar intercambio(x,y)
  escribir x,y

  accion asteriscos (n ∈ entero)
  inicio
    i ∈ entero
    desde i←1 hasta n hacer
      escribir '*'
    fin desde
  fin

  accion intercambio (a,b ∈ entero)
  inicio
    temporal ∈ entero
    temporal ← a
    a ← b
    b ← temporal
  fin
fin
```

## Definición de subrutinas en FORTRAN

En FORTRAN las subrutinas se definen con la siguiente sintaxis:

```
subroutine nombre_subrutina (arg1, ..., argN)
inicio
  declaración arg1
  ...
  declaración argN

  sentencia1
  sentencia2
  ...
  sentenciaN
fin
```

Obsérvese que no es posible indicar si los argumentos son de entrada, de salida o de entrada/salida; en FORTRAN **todos** los argumentos son de entrada/salida así que hay que extremar las precauciones para no provocar efectos laterales indeseables.

A continuación se muestran los ejemplos del apartado anterior traducidos a FORTRAN.

- Subrutina que dibuja n asteriscos:

```
subroutine asteriscos (n)
  implicit none

  integer n
  integer i

  do i=1, n
    print *, '*'
  end do
end subroutine
```

- Subrutina que intercambia los dos argumentos que recibe:

```
subroutine intercambio (a,b)
  implicit none

  integer a,b
  integer temporal

  temporal=a
  a=b
  b=temporal
end subroutine
```

En FORTRAN existe una palabra reservada relacionada con las subrutinas, denominada `RETURN`; esta sentencia permite detener la ejecución de una subrutina en cualquier momento y retornar al programa principal, su utilización es altamente desaconsejable pues puede llevar a una desestructuración del algoritmo, es preferible diseñar las subrutinas de tal forma que el retorno siempre se produzca al llegar al final de las mismas.

El siguiente programa muestra la forma de invocar las subrutinas anteriores:

```
program programa
  implicit none

  integer x,y

  call asteriscos(5)

  x=0;y=1;
  print *,x,y
  call intercambio(x,y)
  print *,x,y


  contains

  subroutine asteriscos (n)
    implicit none

    integer n
    integer i

    do i=1, n
      print *, '*'
    end do
  end subroutine

  subroutine intercambio (a,b)
    implicit none

    integer a,b
    integer temporal

    temporal=a
    a=b
    b=temporal
  end subroutine
end
```

## *Resumen*

1. Los subprogramas facilitan la utilización de técnicas de diseño descendente para la construcción de programas.
2. Los subprogramas:
   - Facilitan la modularidad y estructuración de los algoritmos.
   - Facilitan la lectura e inteligibilidad de los algoritmos.
   - Permiten una economización del esfuerzo del programador al poder escribir código reutilizable en muchas partes de un mismo algoritmo.
   - Facilitan la depuración y mantenimiento de los programas.
3. Los subprogramas pueden ser funciones y subrutinas.
4. Las funciones son subrutinas con 0 ó más argumentos y que devuelven un único valor de retorno.
5. Las funciones pueden formar parte de expresiones o aparecer en la parte derecha de una sentencia de asignación pero nunca pueden constituir una sentencia aislada o aparecer en la parte izquierda de una asignación.
6. Las funciones son invocadas mediante su nombre seguido de los argumentos entre paréntesis.
7. Existen dos tipos de funciones: intrínsecas y definidas por el usuario.
8. Las funciones intrínsecas son funciones de uso muy común: raíz cuadrada, logaritmos, funciones trigonométricas, etc.
9. Las funciones definidas por el usuario deben describirse dentro del algoritmo principal; la sintaxis de la definición de funciones en la notación algorítmica y en FORTRAN es la siguiente:

```
tipo funcion nombre_funcion (arg1 ∈ tipo1, ..., argN ∈ tipoN)
inicio
  sentencia1
  sentencia2
  ...
  sentenciaN
fin


tipo function nombre_funcion (arg1, ..., argN)
  declaración arg1
  ...
  declaración argN

  sentencia1
  sentencia2
  ...
  sentenciaN
end
```

10. Los argumentos y variables declaradas dentro del cuerpo de una función (o subrutina) se denominan variables locales, las variables declaradas dentro del programa principal son variables globales. Los subprogramas tienen acceso a las variables globales aunque en el caso de que una variable local se denomine igual que una variable global tiene preferencia la primera.
11. Las subrutinas son subprogramas que no devuelven ningún resultado; sin embargo, gracias a la utilización de los efectos laterales es posible su utilización para permitir el “retorno” de varios resultados.
12. La sintaxis de la definición de subrutinas o procedimientos en la notación algorítmica y en FORTRAN es la que sigue:

```
accion nombre_subrutina ([ent|sal|ent sal] arg1∈tipo1, ..., [ent|sal|ent sal] argN∈tipoN)
inicio
  sentencia1
  sentencia2
  ...
  sentenciaN
fin

subroutine nombre_subrutina (arg1, ..., argN)
inicio
  declaración arg1
  ...
  declaración argN

  sentencia1
  ...
  sentenciaN
fin
```