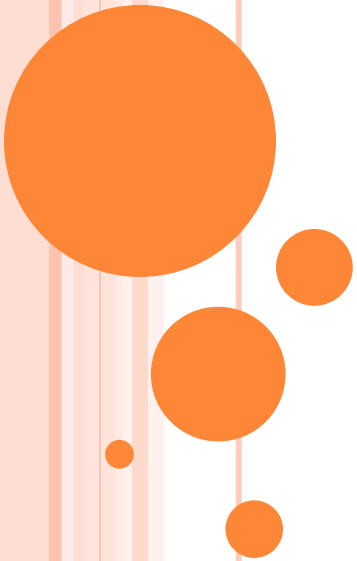


Lenguaje FORTRAN. Funciones y Subrutinas

Programación en Fortran

Valentín Moreno

ÍNDICE

1. Subprogramas
2. Funciones
3. Subrutinas

1. SUBPROGRAMAS

- Si necesitamos usar con frecuencia una funcionalidad determinada, podemos usar módulos que ejecutan ese código que se utiliza varias veces, y llamarlos cada vez que queremos usarlos.
- Si tenemos programas muy largos que no son manejables, podemos subdividir ese código en módulos más pequeños.
- La técnica de programación utilizando módulos independientes de código se denomina **programación modular.**
- Para referirnos a esos módulos reutilizables de código se usa el término ***subprogramas***.

- El programa principal es ejecutado de forma autónoma e independiente de cualquier otro.
- Los subprogramas únicamente pueden ser ejecutados cuando son llamados por el programa principal u otro subprograma.
- Los subprogramas pueden ser de dos clases:
  - Funciones **FUNCTION**
  - Subrutinas **SUBROUTINE**
    - también llamados procedimientos.

- ¿Dónde se ponen los subprogramas?
  - Normalmente a continuación del programa principal.
  - Aunque también se pueden poner antes del programa principal.
  - O se pueden poner en un archivo separado dentro del mismo proyecto.
  - Se compilan por separado del programa principal.
  - Se pueden programar independientemente del programa principal.

- ¿Cómo se pasan los datos del programa principal a los subprogramas?
  - La información se comunica a través de los ***argumentos***.
  - Cada vez que se llama a un subprograma se le pasan, a través de los argumentos, los valores con los que va a trabajar.
  - Argumentos formales y actuales:
    - Los argumentos que aparecen en el **encabezado** de la función o subrutina se denominan ***argumentos formales*** (ficticios o mudos).
    - Los argumentos que aparecen en la **llamada** a la función o subrutina se denominan ***argumentos actuales*** (reales, efectivos, verdaderos).

- Se establece automáticamente una correspondencia entre los ***argumentos de la llamada*** (***actuales***) y ***los del subprograma*** (***formales***).
- Esta correspondencia está definida por la posición:
  - El primer argumento actual se corresponde con el primer argumento formal; el segundo argumento actual, con el segundo formal y así sucesivamente.
- Deben existir el mismo número de argumentos formales que actuales.
- Los argumentos formales y los actuales deben coincidir en tipo.

- Además los argumentos formales de los subprogramas pueden tener tres finalidades:
  - entrada, salida y entrada/salida
- Si los argumentos se definen dentro del subprograma sin indicar su finalidad esta será de entrada/salida.
- Para especificarla es necesario utilizar la palabra reservada INTENT y a continuación su uso.
  - **argumentos de entrada**: su valor no deberá modificarse durante la ejecución del subprograma.

    ```
    tipo argumento
    INTENT (IN) argumento
    ```
  - **argumentos de salida**: su valor se modificará durante la ejecución del subprograma.

    ```
    tipo argumento
    INTENT (OUT) argumento
    ```
  - **argumentos de entrada y salida**

    ```
    tipo argumento
    INTENT (INOUT) argumento
    ```

- Las funciones y subrutinas también pueden tener variables de ámbito local, que existen sólo mientras se ejecuta la función y no pueden usarse en el programa principal.
- Se declaran dentro del subprograma.
- Las constantes también deben declararse en los subprogramas en los que se vayan a utilizar.

2. FUNCIONES

- Son un tipo de subprogramas cuyo **nombre** representa un valor.
- Devuelven al programa principal un valor (y sólo uno).
  - No se utilizan para lectura de variables ni escritura de valores.
- El tipo al que pertenece ese valor es el tipo de la función (real, entero,…).
- Sus argumentos formales son argumentos de entrada nunca de salida, aunque si realizamos cálculos sobre ellos pueden variar su valor si no se declaran correctamente (ver diapositiva 9) .

- Declaración de funciones

```
tipo FUNCTION nombre_funcion (lista de argumentos)
implicit none
! Argumentos
   tipo argumento1
   tipo argumento2
   …
   INTENT (IN) argumento1, argumento2, …

 ! Variables
   tipo variable1
   tipo variable2
   …
! Cuerpo de la función
    Sentencias ejecutables
END FUNCTION nombre_funcion
```

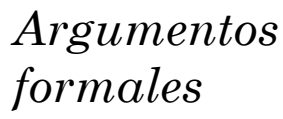
Sentencias de especificación y declaración

- Declaración de funciones (cont.)
  - **FUNCTION**
    - Todo función en FORTRAN debe comenzar con FUNCTION
    - `nombre_funcion`
      - Ha de empezar por una letra
      - Puede contener números y letras
      - NO puede contener espacios
      - Su longitud máxima es de 31 caracteres
      - NO puede coincidir con ningún nombre de variable que se use en la función
  - **END FUNCTION**
    - indica al compilador que ya no hay más instrucciones de la función

Declaración de funciones (cont.)

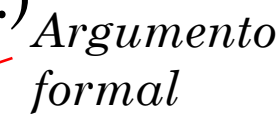
- **tipo**
  - Tipo de dato del valor que devuelve la función
- `lista de argumentos`
  - Valores separados por comas
  - Son los argumentos ficticios o formales.
  - Debe existir aunque este vacía.
- `Sentencias de especificación y declaración`
  - Declaración de constantes.
  - **Declaración de los argumentos ficticios**.
  - Declaración de las variables locales.
- `Sentencias ejecutables`
  - Son las que llevan a cabo el trabajo de la función.
  - ***Debe haber al menos una instrucción que asigne un valor al nombre de la función***.

- Declaración de funciones. Ejemplos




```
Integer FUNCTION suma ( n1, n2 )
implicit none
   ! Argumentos
      Integer n1, n2
      INTENT (IN) n1, n2
   ! Cuerpo de la función
      suma = n1+n2
END FUNCTION suma
```

- Declaración de funciones. Ejemplos (cont.)




```
Logical FUNCTION esAprobado ( calificacion )
implicit none
   ! Argumentos
      Real calificacion
      INTENT (IN) calificacion
   ! Cuerpo de la función
      If (calificacion <5) Then
            esAprobado = .False.
      Else
            esAprobado = .True.
      End
END FUNCTION esAprobado
```

- Para emplear una función creada por el programador hay que seguir los siguientes pasos:
  1. Definir (=declarar) la función utilizando la palabra reservada FUNCTION.
  2. Declarar en el programa principal o subprograma la función.

     ```
     tipo nombre_funcion
     ```
  3. Llamar a la función desde el programa principal u otro subprograma, pasándole los datos (argumentos) actuales con los que debe trabajar dicho módulo.

- El programa principal trata la función como una variable, por lo que podemos utilizarla como tal, con la única salvedad de que no le podemos asignar valores.
- Ejemplos
  - variable = nombre_funcion (lista argumentos)

```
num3 = suma (num1, num2)
```

Llamada a la función **suma**

Argumentos actuales

  - O usando nombre_funcion como una variable en una expresión

Llamada a la función **esAprobado**

```
If (esAprobado(nota)) then
    Print*, "enhorabuena"
End if
```

Argumento actual

3. SUBRUTINAS

- Son otro tipo de subprogramas. Una de las diferencias principales con las funciones es que no devuelven un valor "en el nombre", sino solamente a través de los argumentos formales.
- Pueden recibir argumentos de entrada y de salida.
- Se pueden utilizar para leer variables y escribir valores.

- Declaración de subrutinas

```
SUBROUTINE nombre_subrutina (lista de argumentos)
implicit none
 ! Argumentos
    tipo argumento1
    tipo argumento2
    …
    INTENT (in) lista_argumentos_entrada
    INTENT (out) lista_argumentos_salida
    INTENT (inout) lista_argumentos_entrada/salida
 ! Variables
     tipo variable1
     tipo variable2
    …
! Cuerpo de la función
     Sentencias ejecutables
END SUBROUTINE nombre_subrutina
```

Sentencias de especificación y declaración

- Declaración de subrutinas (cont.)
  - **SUBROUTINE**
    - Todo subrutina en FORTRAN debe comenzar con SUBROUTINE
    - `nombre_funcion`
      - Ha de empezar por una letra
      - Puede contener números y letras
      - NO puede contener espacios
      - Su longitud máxima es de 31 caracteres
      - NO puede coincidir con ningún nombre de variable que se use en la función
  - **END SUBROUTINE**
    - indica al compilador que ya no hay más instrucciones de la subrutina

Declaración de subrutinas(cont.)

- `lista de argumentos`: valores separados por comas
  - Son los argumentos ficticios o formales.
  - Debe existir aunque este vacía.
- `Sentencias de especificación y declaración`
  - Declaración de constantes.
  - **Declaración de los argumentos ficticios**.
  - Declaración de las variables locales.
- `Sentencias ejecutables`
  - Son las que llevan a cabo el trabajo de la función.
  - ***Debe haber al menos una instrucción que asigne un valor al nombre de la función***.

- Declaración de subrutinas. Ejemplos

```
SUBROUTINE  asteriscos ( n )
implicit none
! Argumentos
   integer n
   INTENT (IN) n
! Variables
   integer i
! Cuerpo de la subrutina
   do i=1, n
      print*, "*"
   end do
END SUBROUTINE asteriscos
```

Argumento formal

- Declaración de subrutinas. Ejemplos (cont.)

Argumentos formales

```
SUBROUTINE  cilindro ( radio, area, altura, volumen )
implicit none
! Constantes
   parameter Pi = 3.1416
! Argumentos
   real radio, area, altura, volumen
   INTENT (IN) radio
   INTENT (OUT) altura, volumen
   INTENT (INOUT) area  ! Contiene el área de la base y devuelve el área
                        ! del  cilindro
! Cuerpo de la subrutina
    print*, “Dame la altura del cilindro"
    read*, altura
    volumen = area * altura
    area  =  2 * Pi * radio * altura + 2 * area
END SUBROUTINE cilindro
```

- Llamada a una subrutina

  CALL nombre_subrutina (lista de argumentos)

  - Para llamar a una subrutina desde el programa principal se usa la sentencia CALL.
  - Cuando la subrutina devuelve el control al programa principal, devuelve los resultados en los argumentos de salida (si los tuviera).
  - Ejemplos

    CALL asteriscos (numAsteriscos)

    CALL cilindro (radioCi, areaCi, alturaCil, volumenCil)