Informática

Clase 6: Funciones y Subrutinas I

Mario Merino Martínez
mario.merino@upm.es

Escuela de Ingeniería Aeronáutica y del Espacio
Universidad Politécnica de Madrid

22 de noviembre de 2011

Subprogramas I

- Programar bien permite **reutilizar nuestro código** en otros programas. *Por ejemplo, un código para calcular el producto matricial*. Para ello, ponemos ese código en una función o subrutina independiente.
- Desde el programa principal, los subprogramas son **cajas negras**, que recogen unos inputs y devuelven unos outputs. Lo que ocurre dentro de ellas es **ajeno al programa principal**.

Subprogramas II

- El objetivo es **encapsular unidades de programa**: células independientes y mínimas, dedicadas cada una a **una tarea bien diferenciada**, obedeciendo los axiomas de:
  - **Máxima independencia**: las partes interfieren lo mínimo posible entre sí.
  - **Mínima información**: las partes requieren la menor información posible del programa principal para funcionar.
- En un programa largo, es **fundamental estructurar correctamente el código en partes**. Orden y limpieza facilitan el desarrollo y mantenimiento.

Subrutinas I

- Una subrutina es un subprograma declarado con la instrucción **subroutine** en lugar de **program**, y con una lista de argumentos después del nombre:

```
subroutine nombre_subrutina(argumentos)
implicit none
! Declaracion de los argumentos y otras variables
! Órdenes ejecutables
end subroutine nombre_subrutina
```

- **Desde el programa principal**, la subrutina se ejecuta con la orden **call**:

```
call nombre_subrutina(argumentos)
```

Subrutinas II

- La subrutina sólo se comunica con el mundo exterior a través de los argumentos, que pueden ser de **entrada (in), de salida (out), o ambos (inout)**.
- El **atributo opcional** `intent` permite dejar claro qué queremos hacer con cada argumento (el uso equivocado dará un **error**).

```
subroutine nombre_subrutina(arg1, arg2)
implicit none
real*8, intent(in) :: arg1
integer, intent(inout) :: arg2
! ...
```

Funciones I

- Las funciones son subprogramas declarados con la instrucción **function**. También incluye una lista de argumentos. El **resultado es el propio nombre de la función**, que **también hay que declarar**:

```
function fun(arg1, arg2)
implicit none
real*8 :: fun, arg1 ! Declaracion de fun, argumentos y demás
integer :: arg2, temp
! Órdenes ejecutables
end function fun
```

- También puede especificarse el **tipo** de "fun" en la primera línea, en vez de declarandola aparte:

```
real*8 function fun(arg1, arg2)
implicit none
! ...
```

Funciones II

- Antes de usarlas en un programa principal, hay que **declararlas como una variable más**, con atributo `external`. El tipo ha de coincidir con el tipo de funcion:

```
program main
implicit none
real*8, external :: fun
! ...
```

- Las funciones **se ejecutan sin `call`**, y pueden formar parte de expresiones más complejas (las subrutinas, no):

```
z = 3.2 + fun(x,y)
```

Funciones III

- En una función, los argumentos de entrada (inputs) se listan **entre paréntesis después del nombre de la función**, y el resultado (output) es **la propia función**.
- **Nunca** se deberían **modificar los argumentos de entrada dentro de la función**: hacer esto también los cambiará en el programa principal cada vez que se llame a la función: *normalmente no queremos eso*. Declarar siempre los inputs con **intent**(in) para asegurarse de ello.

```
function f(x)
implicit none
real*8 :: f, x
f=x**2+2 ! Asignamos el resultado a f
x= 27d0 ! Esto NO suele ser deseado
end function f
```

Comparativa subrutinas/funciones

- Las **subrutinas...**
  - permiten tener varios argumentos de salida (out), o de entrada/salida (inout)
  - Requieren usar `call` para ser invocadas.
  - **No hay que declararlas en el programa principal** como si fuesen una variable, pues no lo son.
  - Las tareas más complejas se suelen hacer en subrutinas.
- Las **funciones...**
  - solo deben usarse para producir un único resultado (el nombre de la función es una variable en sí).
  - **Hay que declararlas con external en el programa principal**.
  - Pueden llamarse dentro de expresiones, por tanto son más versátiles.

Ejemplo

```
program prueba
implicit none
real*8, external :: f ! Declaramos f como external
real*8 :: x
  call numeropi(x) ! Llama a la subrutina numeropi para calcular x
  print*, "resultado: ", f(x) ! Llama a f dentro del propio print
end program prueba

real*8 function f(w) ! f declarada implicitamente como real*8
implicit none
real*8, intent(in) :: w ! Argumento interno, input
f=w**2+sin(w*5) ! Asigna el resultado a f
end function f

subroutine numeropi(pi)
implicit none
real*8, intent(out) :: pi ! Argumento interno, output
pi = acos(-1d0) ! Calcula el número pi
end subroutine numeropi
```

*Imprescindibles I

- **Dentro** del código de una función/subrutina, los argumentos tienen un nombre (ficticio).
- Al llamarla **desde el programa principal**, los nombres no tienen por qué coincidir, o **incluso** se pueden usar datos literales en vez de variables para los argumentos de entrada.
- **Lo único que importa, es que coincidan en orden, tipo y dimensiones**.

"Los subprogramas son "cajas negras" que esperan recibir datos de un tipo determinado, los utilizan en sus cálculos, y devuelven unos resultados"

*Imprescindibles II

- Las **funciones y subrutinas** pueden escribirse en el mismo fichero (después de end program), o en ficheros independientes *(en Plato hay que usar "**proyectos**")*.
- Los **subprogramas** también pueden llamar a otros subprogramas (con `call` o declarando las funciones como `external`)