

Mary Allen Wilkes (1937)



- Creadora (1965) del primer ordenador personal para trabajo en casa
- Desarrolladora de sistemas operativos e lenguaje ensamblador (LAP6)
- Trabajó en MIT e en la Univ. Washington

Conversión entre tipos de datos

- De inteiro a real: $x=i$ ou $real(i)$
- De real a inteiro: $i=x$, pero podes perder información (p.ex. de 3.2 a 3).
- Podes redondear ao inteiro mais cercano con $i=nint(x)$, por defecto con $i=floor(x)$, por exceso con $i=ceiling(x)$.
- De carácter (p.ex. '32') a inteiro: $s='32'; read (s,*) i$. So funciona se o carácter ten un número inteiro. Se $s='ola'$, o $read$ da erro de entrada/saída.
- De carácter (p.ex. '3.2') a real: $s='3.2'; read (s,*) x$. So funciona igual que no anterior.
- De inteiro a carácter: $i=32; write (s, '(i0)') i$
- De real a carácter: $x=3.2; write (s, '(f3.1)') x$

Creación dunha librería en Fortran

- Librería: código máquina de moitos subprogramas (en Fortran) empaquetado nun arquivo
- Non contén o código fonte (non se pode depurar ou ver como opera).
- Proporcionan subprogramas que permiten facer operacións (p.ex., determinantes, resolución de sistemas de ecuacións, ...)
- Para usar unha librería, hai que enlazar (*link*) o noso programa coa librería
- O compilador *gfortran* xa usa algunhas librerías incluídas co compilador (p.ex. funcións intrínsecas)

Ficheiros para crear a librería

media.f90

```
real function media(x,n)
real,intent(in) :: x(n)
integer,intent(in) :: n
media=0
do i=1,n
    media=media+x(i)
end do
media=media/n
return
end function media
```

Podes descargalos
dende estes enlaces:

[media.f90](#)

[desviacion.f90](#)

[principal.f90](#)

principal.f90

```
program principal
real,allocatable :: x(:),media
print *, 'n? '
read *,n
allocate(x(n))
print *, 'x[]? '
read *,x
y=media(x,n)
d=desviacion(x,n,y)
print *, 'media=',m
print *, 'desviacion=',d
end program principal
```

desviacion.f90

```
function desviacion(x,n,media)
real,intent(in):: x(n)
integer, intent(in) :: n
real, intent(in) :: media
desviacion=0
do i=1,n
    y = x(i)-media
    desviacion=desviacion+y*y
end do
desviacion=sqrt(desviacion/n)
return
end function desviacion
```

Librería estática

- É un ficheiro con extensión *.a*: *libestatistica.a*
- Librería *libestatistica.a* que proporcione subprogramas para calcula-la media e desviación dun vector
- Comando de compilación (sen enlazado):

gfortran -c media.f90 desviacion.f90

- Creación da librería: *ar qv libestatistica.a media.o desviacion.o*
- Compilación de programa principal e enlazado con librería:

gfortran -L. principal.f90 -lestatistica

- Execución: *a.exe*

Librería dinámica

- Ficheiro con extensión *.so*: *libestatistica.so*
- Comando de compilación (sen enlazado):
gfortran -fpic -c media.f90 desviacion.f90
- Empaquetado: *gfortran -shared -o libestatistica.so *.o*
- Listado de arquivos *.o*: *nm libestatistica.so* ou *readelf -s X.so*
- Uso da librería dende programa *principal.f90*:
gfortran -L. principal.f90 -lestatistica
- Execución: *a.exe*

Interface

- Bloque no que se indican subprogramas (tipo, nome, argumentos, valores retornados).

interface

bloque subprograma1

...

bloque subprograma N

end interface

- Empréganse para cousas especiais:

Retornar arrais dinámicos

Pasar argumentos arrai

sen a súa dimensión

Pasar argumentos nomeados ou opcionais

Sobrecarga de operadores

interface

```
function fun(x,y,n) result(z)
    integer,intent(in) :: x(n),y(n)
    real :: z
end function fun
```

```
subroutine sub(x,y)
    real,intent(in) :: x
    real,intent(out) :: x
end subroutine sub
```

end interface

Exemplo de uso de módulos e interfaces: paso de vectores como argumentos

- Pasar o nome do arrai (*assumed-shape arrays*) sen a súa lonxitude como argumento a un subprograma:
- Cun subprograma interno
- Cun módulo
- Cunha interface

```
program proba
interface
    subroutine sub(x)
        integer,intent(out) :: x(:)
    end subroutine
end interface
integer,allocatable :: x(:)
call sub(x)
end program proba
subroutine sub(x)
integer,intent(out) :: x(:)
n=size(x)
...
end subroutine sub
```

```
module aux
contains
    subroutine sub(x)
        integer,intent(out) :: x(:)
        n=size(x)
        ...
    end subroutine sub
end module
program proba
use aux
integer,allocatable :: x(:)
...
call sub(x)
...
end program proba
```

```
program proba
integer,allocatable :: x(:)
...
call sub(x)
...
contains
    subroutine sub(x)
        integer,intent(out) :: x(:)
        n=size(x)
        ...
    end subroutine sub
end program proba
```

Exemplo de uso de módulos e interfaces: paso de matrices como argumentos

- Cun subprograma interno
- Cun módulo
- Cunha interface

```
program proba
interface
    function fun(a)
        integer,intent(...) :: a(:, :)
    end function
end interface
integer,allocatable :: a(:, :)
y=fun(a)
end program proba

function fun(a)
integer,intent(...) :: a(:, :)
nf=size(a,1);nc=size(a,2)
...
end function fun
```

```
module aux
contains
    function fun(a)
integer,intent(...) :: a(:, :)
nf=size(a,1);nc=size(a,2)
...
    end function fun
end module

program proba
use aux
integer,allocatable :: a(:, :)
...
y=fun(a)
...
end program proba
```

```
program proba
integer,allocatable :: a(:, :)
...
y=fun(a)
...
contains
    function fun(a)
integer,intent(...) :: a(:, :)
nf=size(a,1);nc=size(a,2)
...
    end function fun
end program proba
```

Exemplo de interface: subrutina que reserva un vector ou matriz *intent(out)* cunha interface

Con vector

```
program exemplo
interface
    subroutine sub(x)
        integer,allocatable,intent(out) :: x(:)
    end subroutine sub
end interface
integer,allocatable :: x(:)
call sub(x)
print *,'x=',(x(i),i=1,size(x))
deallocate(x)
end program exemplo
!-----
subroutine sub(x)
integer,allocatable,intent(out) :: x(:)
allocate(x(5))
do i=1,5
    x(i)=i*i
end do
return
end subroutine sub
```

Con matriz

```
program exemplo
interface
    subroutine sub(a)
        integer,allocatable,intent(out) :: a(:, :)
    end subroutine sub
end interface
integer,allocatable :: a(:, :)
call sub(a)
do i=1,size(a,1)
    print *,(a(i,j),j=1,size(a,2))
end do
deallocate(a)
end program exemplo
!-----
subroutine sub(a)
integer,allocatable,intent(out) :: a(:, :)
allocate(a(3,3))
do i=1,3
    do j=1,3
        a(i,j)=i*j+i-j
    end do
end do
return
end subroutine sub
```

Exemplo de interface: función externa que retorna un vector ou matriz reservado dinámicamente

Con vector

```
program exemplo
interface
  function func(x) result(y)
    integer,intent(in) :: x(:)
    integer,allocatable :: y(:)
  end function func
end interface
integer :: x(5)=[1,2,3,4,5]
integer,allocatable :: y(:)
print *, 'x=', x
y=func(x)
print *, 'y=', y
deallocate(y)
end program exemplo
!-----
function func(x) result(y)
integer,intent(in) :: x(:)
integer,allocatable :: y(:)
n=size(x);allocate(y(n))
do i=1,n
  y(i)=x(n-i+1)
end do
end function func
```

Con matriz

```
program exemplo
interface
  function func(a) result(b)
    integer,intent(in) :: a(:, :)
    integer,allocatable :: b(:, :)
  end function func
end interface
integer :: a(5,5)
integer,allocatable :: b(:, :)
b=func(a)
deallocate(b)
end program exemplo
!-----
function func(a) result(b)
integer,intent(in) :: a(:, :)
integer,allocatable :: b(:, :)
nf=size(a,1);nc=size(a,2)
allocate(b(nf,nc))
do i=1,nf
  do j=1,nc
    b(i,j)=a(nf-i+1,nc-j+1)
  end do
end do
end function func
```

Exemplo de interface: función que retorna un vector dinámico: *find*

- Atopa os índices dos elementos dun vector que cumpren unha condición (expresión lóxica)
- Moi útil para vectorizar expresións
- Retorna un vector reservado dinámicamente na función

```
function find(x) result(y)
logical,intent(in) :: x(:)
integer,allocatable :: y(:)
n=count(x);m=size(x)
allocate(y(n));j=1
do i=1,m
    if(x(i)) then
        y(j)=i;j=j+1
    end if
end do
end function find
```

```
program proba
interface
    function find(x) result(y)
        logical,intent(in) :: x(:)
        integer,allocatable :: y(:)
    end function find
end interface
integer :: x(5)=[1,2,3,4,5]
print *,count(x>2) !nº elementos >2
print *,find(x>2)  ! índices de elementos >2
print *,pack([(i=1,5)],x>2) !alternativa
print *,x(find(x>2)) ! valores de elementos >2
print *,pack(x,x>2) !alternativa
end program proba
```

Argumentos nomeados

- Na chamada ao subprograma pódese especificar os nomes dos argumentos.
- Se na chamada se nomea un argumento, hai que nomealos todos.
- Pode haber chamadas con argumentos nomeados e outras sen nomealos.
- O subprograma debe declararse nun bloque *interface* no programa principal.

```
program argumentos_nomeados
interface
    subroutine nomeados(x,y)
        real,intent(in) :: x,y
    end subroutine
end interface
call nomeados(y=3.2,x=2.1)
call nomeados(1.1,2.2)
end program
argumentos_nomeados
!-----
subroutine nomeados(x,y)
    real,intent(in) :: x,y
    print *, 'x=',x, 'y=',y
end subroutine nomeados
```

Argumentos opcionais

- Teñen o atributo *optional*, de modo que se pode chamar ao subprograma indicando este argumento ou non.
- Non pode haber ningún argumento obrigatorio logo dun argumento opcional.
- O subprograma debe declararse nun bloque *interface* no programa principal.
- Función intrínseca *present(x)*: retorna *.true.* se o argumento *x* está presente, e *.false.* en caso contrario.

```
program argumentos_opcionais
interface
    subroutine opcionais(x)
        real,intent(in),optional :: x
    end subroutine
end interface
call opcionais(2.1)
call opcionais()
end program argumentos_opcionais
!-----
subroutine opcionais(x)
    real,intent(in),optional :: x
    if(present(x)) then
        print *,'argumento x=',x
    else
        print *,'argumento x ausente'
    end if
end subroutine opcionais
```