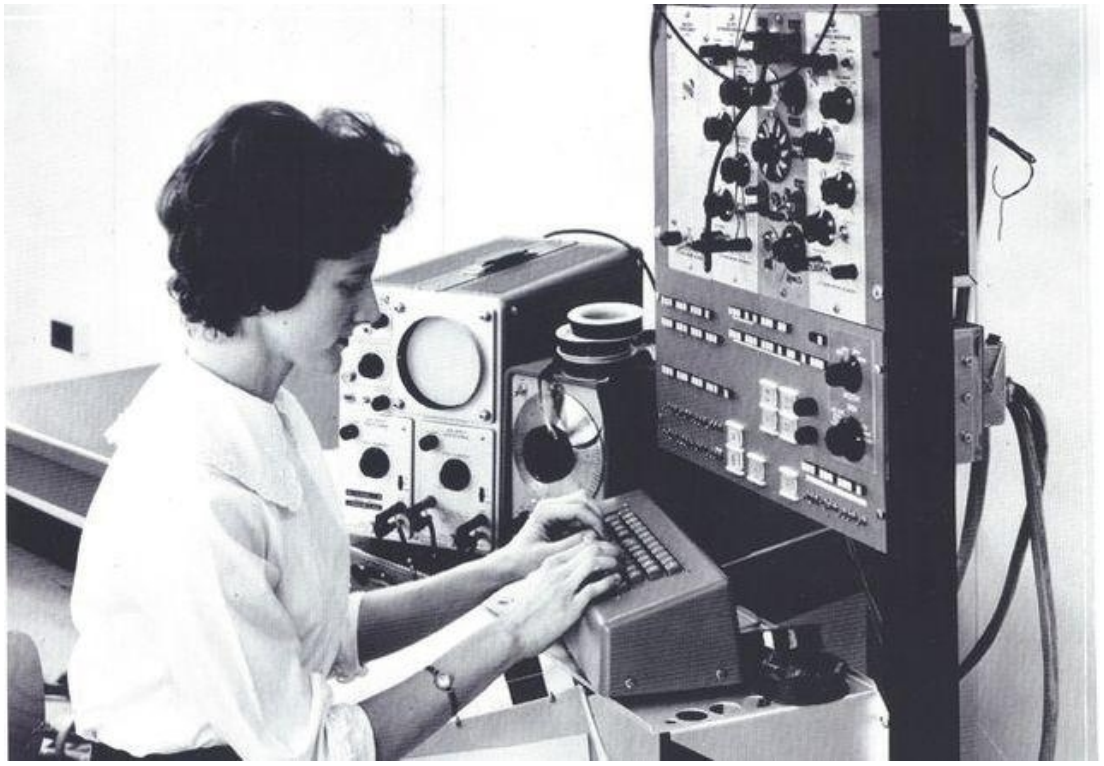
Mary Allen Wilkes (1937)



- Creadora (1965) del primer ordenador personal para trabajo en casa
- Desarrolladora de sistemas operativos e linguaxe ensamblador (LAP6)
- Traballou no MIT e na Univ. Washington

Módulo

- Bloque de código que contén datos e subprogramas: ***use*** *modulo*

```
module nome
    tipo :: var
    interface
        tipo subprograma(...)
        ...
        end fipo
    end interface
contains:
    tipo subprograma(...)
    ...
    end tipo
end module
```

```
module proba
      integer :: x
contains
      subroutine sub(y)
            integer,intent(in) :: y
            print *,y
      end subroutine sub
end module
```

```
program modulo
use proba
call sub(5)
end program modulo
```

- Pode estar en arquivo distinto do programa principal
- Se está en arquivo distinto: *f95 modulo.f90 principal.f90*

antes do arquivo que o usa!

Interface

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

interface
bloque subprograma1
...
bloque subprograma N
end ***interface***

```
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
```

- Empréganse para cousas especiais:

  Retornar arrais dinámicos
  Pasar argumentos arrai sen a súa dimensión
  Pasar argumentos nomeados ou opcionais

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

```
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
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
```

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

- Cun subprograma interno
- Cun módulo
- Cunha interface

```
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
```

```
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
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
```

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
```

Tipos de datos definidos por usuario

- Podemos definir tipos de datos agregados.

  type nome
  tipo1 :: var1
  tipoN :: varN
  end type nome

- Defínense en módulos para poder usarse en varios subprogramas.

```
module modulo_persoa
type persoa
      character(10) :: nome
      integer :: idade
end type persoa
end module modulo_persoa
```

```
program exemplo_tipos
use modulo_persoa
type(persoa) :: p=persoa('carlos',14)
call mostra(p)
end program exemplo_tipos

subroutine mostra(p)
use modulo_persoa
type(persoa),intent(in) :: p
print *,'nome=',p%nome,'idade=',p%idade
end subroutine mostra
```

- Declaración: *type(nome) :: x=nome(y,z)*
- Acceso a campos: *x%y,x%z*

Sobrecarga de operadores

- Define un operador a medida para un dato agregado.
- Bloque *interface operator*.
- Función que define o operador aritmético ou relacional.

```
program principal
use exemplo
type(persoa) :: x=persoa('alba',65.2,1.84)
type(persoa) :: y=persoa('clara',70.1,1.98)
if(x>y) then
      print *,x%nome,'>',y%nome
else
      print *,x%nome,'<=',y%nome
end if
stop
end program principal
```

```
module exemplo
      type persoa
            character(10) :: nome
            real :: peso,altura
      end type persoa
      interface operator (>)
            module procedure maior
      end interface
contains
      logical function maior(x,y)
      type(persoa),intent(in) :: x,y
      if(x%peso/x%altura > y%peso/y%altura) then
            maior=.true.
      else
            maior=.false.
      end if
      end function maior
end module
```

Creación dunha libraría en Fortran

- Libraría: código máquina de moitos subprogramas (en Fortran) empaquetado nun arquivo
- Non contén o código fonte (non se pode depurar ou ver como opera).
- Proporciónanos subprogramas que permiten facer operacións (p.ex., determinantes, resolución de sistemas de ecuacións, ...)
- Para usar unha libraría, hai que enlazar (*link*) o noso programa coa libraría
- O compilador *f95* xa usa algunhas librarías incluídas co compilador (p.ex. funcións intrínsecas)

Librarías estáticas

- Ficheiro con extensión *.a*: *libproba.a*. Usado só na compilación:

  f95 -L. programa.f90 -lproba

- O código máquina da libraría empótrase no programa executábel.
- Non se necesita nada para a execución: *a.out*
- Programa executábel de tamano grande (carga en memoria RAM máis lenta).
- Listado de arquivos *.o* contidos en libraría *.a*: *ar tv libproba.a (ou nm X.a*, ou *readelf -s X.a)*

Librarías dinámicas

- Ficheiro con extensión *.so*: *libproba.so*
- O código máquina da libraría úsase na compilación e na execución:

  f95 -L. programa.f90 -lproba

  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.

  a.out

- Executábel máis pequeno (carga máis rápida en memoria).
- Listado de arquivos *.o*: *nm X.so* ou *readelf -s X.so*.
- Non se pode executar sen que a variábel de entorno indique a ruta onde se atopan o(s) arquivo(s) **.so**
- Ver a variábel: *echo $LD_LIBRARY_PATH*

Creación dunha libraría estática en Fortran (I)

- Edición/compilación/depuración de código
- Compilación sen enlazado dos arquivos que conteñen subprogramas (non incluir o programa principal). Crea só arquivos obxecto (.o), non crea executábel:

  f95 -c arquivo.f90 (crea *arquivo.o*)

- Empaquetado de arquivos *.o* e creación de libraría (***libXX.a***, debe comezar por *lib*)

  *ar qv libXX.a *.o*

- Para ver arquivos .o empaquetados: *ar tv libXX.a*

Creación dunha libraría estática en Fortran (II)

- É recomendábel meter unha función ou subrutina en cada *arquivo .o*
  - Alomenos os subprogramas que van ser usados dende fóra da libraría).
- Así coinciden os nomes dos arquivos *.o* empaquetados en *libXX.a* cos nomes dos subprogramas que proporciona a libraría.
- Así podemos saber, co comando *ar tv libXX.a*, os subprogramas que contén esa libraría.

Exemplo de creación e uso dunha libraría estática

- Libraría *libestatistica.a* que proporcione subprogramas para calcula-la media, desviación, mediana e ordear un vector
- Arquivos media.f90, desviacion.f90, mediana.f90, ordea.f90, principal.f90 (descárgaos)
- Comando de compilación (sen enlazado):

  f95 -c media.f90 desviacion.f90 mediana.f90 ordea.f90

- Empaquetado da libraría: *ar qv libestatistica.a *.o*
- Listado de arquivos *.o*: *ar tv libestatistica.a* (ou *readelf -s X.a*)
- Para enlazar o programa *principal.f90* coa libraría:

  f95 -L. principal.f90 -lestatistica

- Execución: *a.out*

Exemplo de creación e uso dunha libraría dinámica

- Libraría *libestatistica.so*. Mesmos arquivos *.f90* de antes.
- Comando de compilación (sen enlazado):

  f95 -fpic -c media.f90 desviacion.f90 mediana.f90 ordea.f90
- Empaquetado: *f95 -shared -o libestatistica.so *.o*
- Listado de arquivos *.o*: *nm libestatistica.so* ou *readelf -s X.so*
- Uso da libraría dende programa *principal.f90*:

  f95 -L. principal.f90 -lestatistica
- Engade a ruta de libraría ao *LD_LIBRARY_PATH*:

  export LD_LIBRARY_PATH = $LD_LIBRARY_PATH:.
- Execución: *a.out*

media.f90 e desviacion.*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
```

```
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
```

mediana.f90

```
real function mediana(x, n)
   real, intent(in) :: x(n)
   integer, intent(in) :: n
   call ordear(x, n)
   if(mod(n, 2) == 0) then
      mediana = (x(n/2)+x(n/2+1))/2
   else
      mediana = x(n/2+1)
   endif
   return
   end function mediana
```

ordea.f90

```
subroutine ordea(x, n)
real, intent(inout) :: x(n)
integer, intent(in) :: n
do i = 1, n
    vmin = x(i); imin = i
    do j = i + 1, n
        if(x(j) < vmin) then
            vmin = x(j); imin = j
        end if
    end do
    x(imin) = x(i); x(i) = vmin
end do
return
end subroutine ordear
```

principal.f90

```
program principal
 real,allocatable :: x(:)
 real :: media, mediana, m
 print *, "introduce n: "
 read *, n
 allocate(x(n))
 print *, "introduce x: "
 read *, x
 y = media(x, n)
 d = desviacion(x, n, y)
 m = mediana(x, n)
 call ordear(x, n)
 print *, "media= ", y, "desviacion= ", d, "mediana= ", m
 print *, "vector ordeado= ", x
 end program principal
```

Uso dunha libraría feita por outros (I)

- **Instalación** da libraría como administrador ou usuario. Dúas alternativas:
  - Paquetes precompilados (binarios), co synaptic,apt-get,rpm,...
  - Compilación do código fonte: *configure*, *make*, *make install* (como administrador)
- Coñecer o directorio no que se atopa o(s) arquivo(s) **libXX.a** ou **libXX.so**
- Acceder á **documentación** da libraría, para saber que subprogramas ten, os seus argumentos, valores retornados, etc.

Uso dunha libraría feita por outros (II)

- Dende o noso programa, chamamos aos subprogramas da libraría (ver documentación), pasándolle os argumentos axeitados (en número e tipo) e usando os valores retornados
- Coñecido o directorio do arquivo **libXX.a** ou **libXX.so**:

  f95 -Ldir programa.f90 -lXX

  dir: directorio no que *f95* debe buscar *libXX.a/libXX.so*

  -lXX: ***libXX.a*** é a librería que usará *programa.f90*

- Con librarías dinámicas:
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:dir
- Librarías libres en Fortran: http://www.fortran.com/tools.html
- Destacamos: Lapack, Linpack (http://www.netlib.org), Cernlib, Plplot, Libg2