Ana María Prieto López (1942-2018)

- Primeira programadora en España, de Santiago
- Dende os 21 anos traballou en IBM e foi a primeira programadora en España da empresa informática Bull
- Programou en COBOL e código máquina
- Traballou dende 1969 na Caixa de Aforros de Santiago

Sentenzas de iteración

- Repiten a execución dun bloque de sentenzas ...
  - Un certo nº de veces predeterminado (*iteración definida*)
  - Mentres se cumpra unha condición (ou ata que deixa de cumprirse): non se sabe de antemán o nº de repeticións (*iteración indefinida*)
- Grande importancia en programación
- Sentenza *do*: ambos tipos de iteración

Sentenza *do* definida

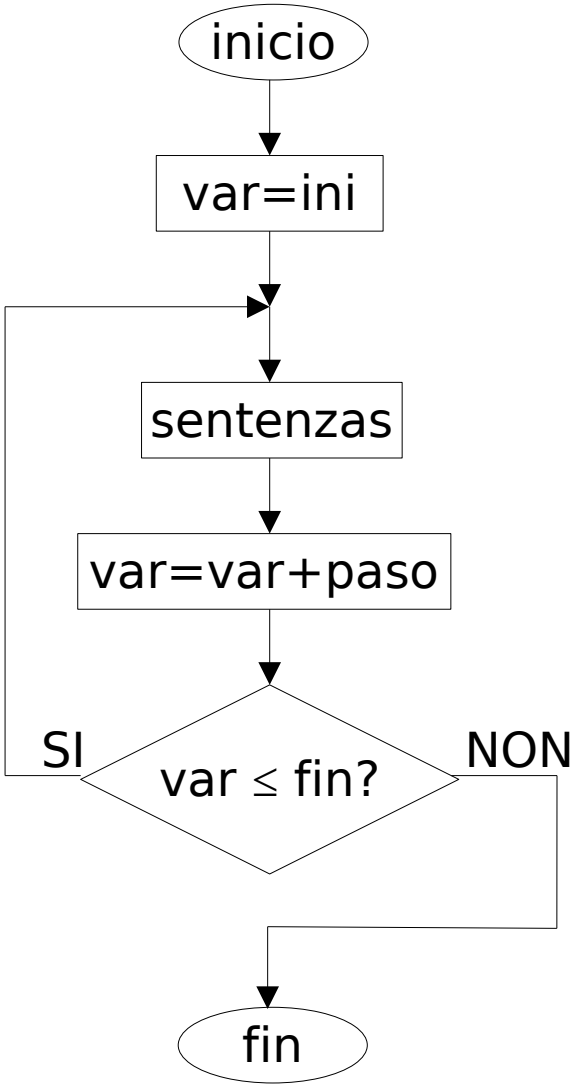
```
do var = ini, fin, paso
    sentenzas
end do
```

```
do i=1,10
    print *, v(i)
end do
```

imprime os elementos dun vector

- Antes de executar nada, asigna *ini* a *var*
- Repite as sentenzas mentres *var* <= *fin*
- Ao rematar unha iteración (repetición completa das sentenzas), executa *var* = *var* + *paso*
- O valor *paso*: opcional, por defecto vale 1
- Se modificamos *var* dentro das sentenzas: erro de compilación
- O parámetro *ini* debe ser enteiro, non real

Diagrama de fluxo *do* definido



Sentenza *do* definida

- Nº de iteracións do bucle *do*:

$$N = max\left\{0, \left\lfloor \frac{fin - ini + paso}{paso} \right\rfloor\right\}$$

- Non pode ser *paso = 0*: erro de compilación
- Se *paso < 0*, entón debe ser *fin < ini*, e a condición de continuidade do bucle é que *var >= fin*. Ex:

```
do i = 10, 1, -1
      print *, v(i)
end do
```

- Un bucles *do* pode estar dentro doutros bloques *do* ou *if/else*, ou conter bloques *if/else*.

Exemplo 1: cálculo de desviación típica dun vector

```
real,allocatable :: v(:)
real :: m
read *, n
allocate(v(n))
read *, v
m=sum(v)/n;d=0
do i=1,n
    t=v(i)-m;d=d+t*t
end do
print *,'desv=',sqrt(d/n)
deallocate(v)
```

Declara un vector dinámico

Le por teclado a lonxitude do vector

Reserva memoria para o vector

Le o vector por teclado

Calcula a media dos elementos do vector

Calcula a desviación típica dos elementos do vector de modo iterativo

Mostra a desviación por pantalla

Exemplo 2: búsqueda de elementos comúns a dús vectores

- So con bucles *do*:

```
integer,parameter :: n=10
integer :: x(n),y(n),z(n)
k=0
do i=1,n
    u=x(i)
    do j=1,n
        if(u==y(j)) then
            k=k+1;z(k)=u;exit
        end if
    end do
end do
print *,(z(i),i=1,k)
```

- Vectorizado:

```
integer,parameter :: n=10
integer :: x(n),y(n),z(n)
k=0
do i=1,n
    u=x(i)
    if(any(u==y)) then
        k=k+1;z(k)=u
    end if
end do
print *,(z(i),i=1,k)
```

Exemplo 3: acumulador, vectorización e incremental

- Queremos calcular:

$$y_i = \sum_{j=1}^{i} x_j, i = 1, \ldots, n$$

- **Versión 1**: acumulador

```
do i=1,n
    s=0
    do j=1,i
        s=s+x(j)
    end do
    y(i)=s
end do
```

Acumulador e dous bucles, fácil de entender, moitas iteracións e operacións

- **Versión 2**: vectorizada

```
s=0
do i=1,n
    y(i)=sum(x(1:i))
end do
```

Máis curto, fai sumas innecesarias

- **Versión 3**: incremental

```
s=0
do i=1,n
    s=s+x(i)
    y(i)=s
end do
```

Aforra moitas operacións (sumas)

Erro de segmentación

- Usualmente a xestión de vectores e matrices realízase mediante bucles *do* definidos ou indefinidos. Exemplo:

```
integer,dimension(3) :: v
do i = 1, 3
    v(i) = 2*i + 4
end do
```

- É posíbel que nos saiamos dos límites do vector, p.ex. se *i* chega a valer *10*
- En tal caso: *erro de segmentación*: é un exemplo de erro de execución (ver metodoloxía da programación), provoca o remate anticipado do programa.

Detección de erros de segmentación

- Compila coa opción -*fcheck=all*
- Cando se produce o erro de segmentación durante a execución, indica en que liña se produce o erro
- Exemplo:
- Compilación:
  f95 -fcheck=all proba.f90 -o proba

```
program erro_segmentacion
real :: x(10)
do i=0,20
    x(i)=i
    print *,x(i)
end do
stop
end program erro_segmentacion
```

Erro de segmentación por chegar a ser i=0 ou i>10

```
At line 5 of file proba.f90
Fortran runtime error: Index '0' of dimension 1 of array
'x' below lower bound of 1
```

Uso de depurador gdb para correxir erros de segmentación

- Compila coa opción -g: f*95 -g programa.f90 -o programa* (isto engade información de depuración no executábel)
- Cargamo-lo executábel no gdb: *gdb programa*
- Executámolo no gdb: *run*
- Para ver ónde remata: *where*
- Seleccionar o nivel situado en *programa.f90*: *frame 3* (se o nivel #3 é o de *programa.f90*)
- Inspeccionar variábeis: *print i*. Ver en que punto toman valores incorrectos, adiviñar por que, e correxir o fallo.
- Sair do gdb: *quit*

Sentenza *exit*

- Provoca o remate inmediato das iteracións
- Sempre vai dentro dunha sentenza de selección (asociada a unha condición)
- Se ésta se cumpre, execútase o *exit* e remata o bucle (se hai varios anidados, o bucle máis interno)
- A condición debe ter variábeis, e algunha delas debe cambiar o seu valor dentro do bucle para que éste remate

```
do i = 1, 10
    print *, “introduce un número (-1 para rematar):”
    read *, n
    if(-1 == n) exit  !remata a iteración neste intre
    suma = suma + n
end do
```

Bucle indefinido *do/exit*

```
do
   sentenzas
   if(condición) exit
end do
```

Executa as sentenzas antes de avaliar a condición (execútanse sempre polo menos unha vez)

```
do
   if(condición) exit
   sentenzas
end do
```

Executa as sentenzas logo de avaliar a condición (poden non executarse nunca)

```
do
   sentenzas1
   if(condición) exit
   sentenzas2
end do
```

Remata cando se cumpre a condición: hai sentenzas antes e despois do *exit*: é máis xeral

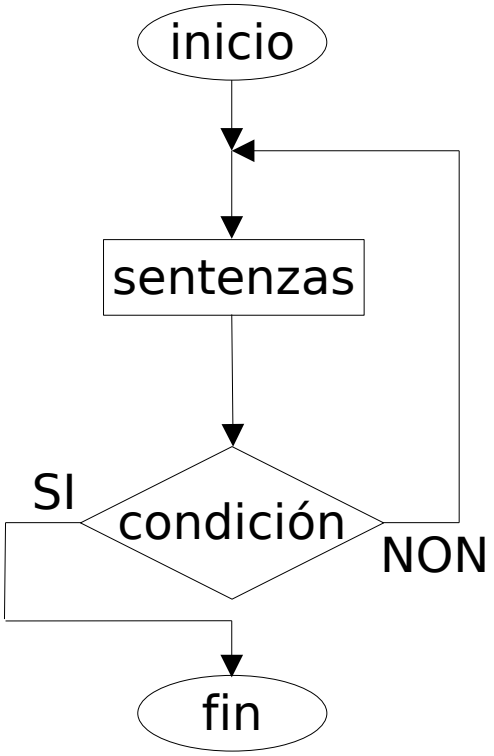
```
x2=0.9
do
   x1=x2;x2=x1**2
   if(abs(x1-x2)<0.01) exit
   print *,x1,x2
end do
```

```
x=1
do
   if(x>10) exit
   x=x+0.1
   print *,x,x+1
end do
```

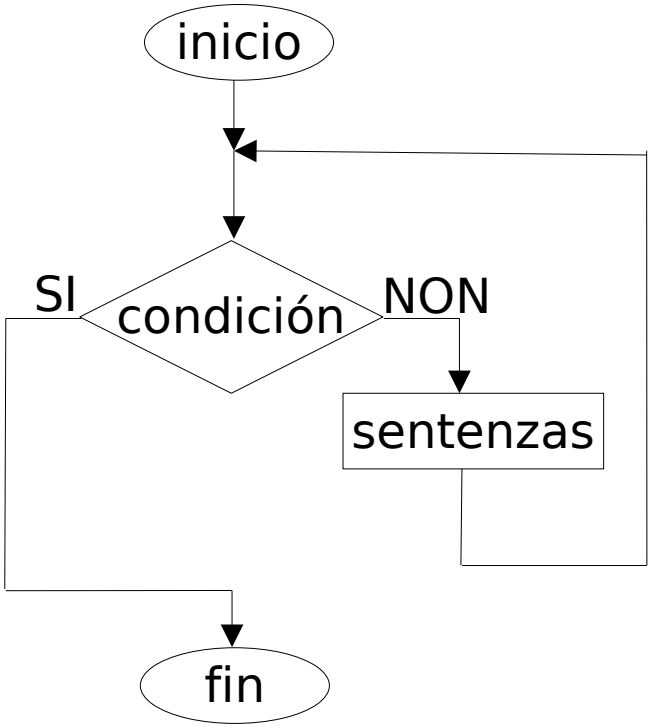
O bucle *do/exit* executa as sentenzas ata que se cumpre a condición, que polo tanto é **condición de remate**.

Diagramas de fluxo *do/exit*

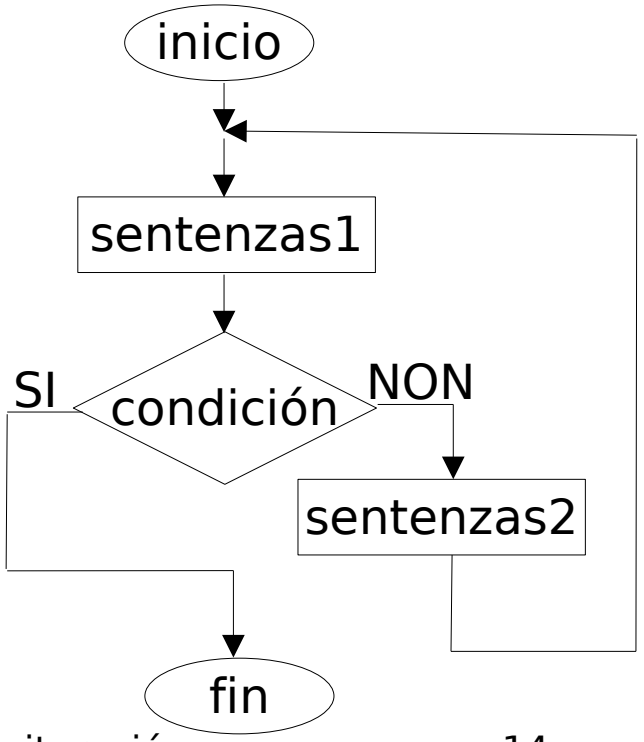
```
do
    sentenzas
    if(condición) exit
end do
```




```
do
    if(condición) exit
    sentenzas
end do
```




```
do
    sentenzas1
    if(condición) exit
    sentenzas2
end do
```



Bucle indefinido *do/while*

- Executa as sentenzas mentres se cumpre a condición.
- A **condición é de continuidade**, non de remate.
- A condición avalíase antes de cada iteración.
- É menos flexíbel que o *do/exit*, porque a condición non se pode avaliar en calquera parte do bucle.
- Outro inconvinte: debes inicializar as variábeis da condición antes do *while*
- Tamén pode levar un *if/exit* dentro do bucle.

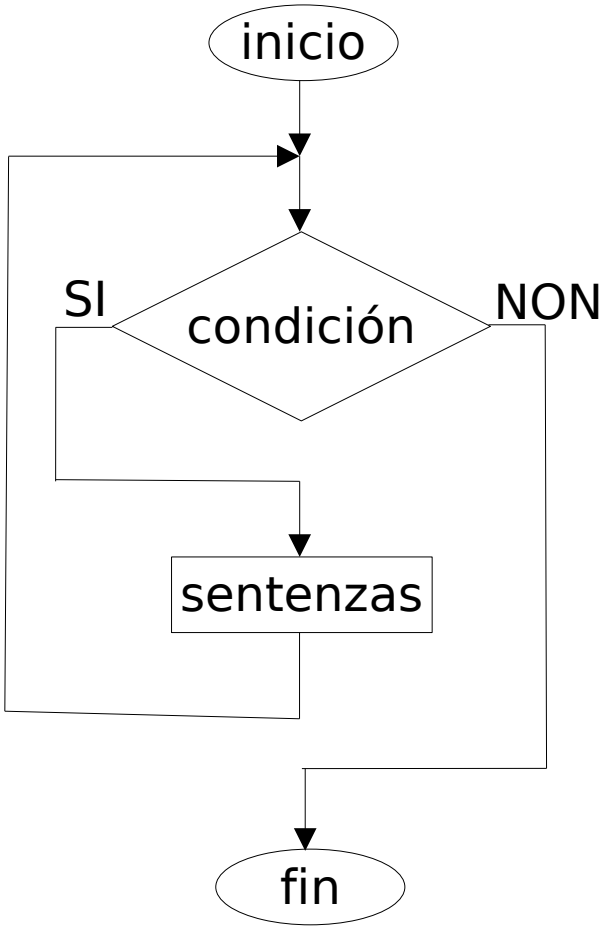
```
do while(condición)
   sentenzas
end do
```

```
s=0
do while(s<10)
   read *,x
   s=s+x
end do
```



```
s=0
do while(s<10)
   read *,x
   if(x==-1) exit
   s=s+x
end do
```

Diagrama de fluxo *do/while*



Bucle *do* indefinido

- O nº de iteracións depende do nº de veces que se non se cumpre a condición de remate

```
do
      print *, “introduce un nº (-1 para rematar)”
      read *, n
      if(n == -1) exit
end do
```

- Non se deben usar == nas condicións os erros en punto flotante poden evitar a igualdade: execución infinita. P. ex: *if(x==y) exit; if(abs(x-y)<1E-5) exit*
- Bucle *do* infinito:

```
do
      sentenzas
end do
```

```
do while(.true.)
      sentenzas
end do
```

O programa debería rematarse dende fóra, p.ex. con CTRL+C

Bucle *do* indefinido

- **Exemplo**: aproximación dunha suma (serie) de infinitos sumandos: sumar só os sumandos > 1e-5

$$\sum_{n=1}^{\infty} \frac{1}{n^2} \simeq \sum_{n \in A} \frac{1}{n^2}, A = \left\{ n \in \mathbb{N} : \frac{1}{n^2} > 10^{-5} \right\}$$

```
suma=0;n=1
do
   sumando=1./n**2
   if(sumando<1e-5) exit
   suma=suma+sumando
   n=n+1
end do
```

Debe ser real ou dará resultado 0

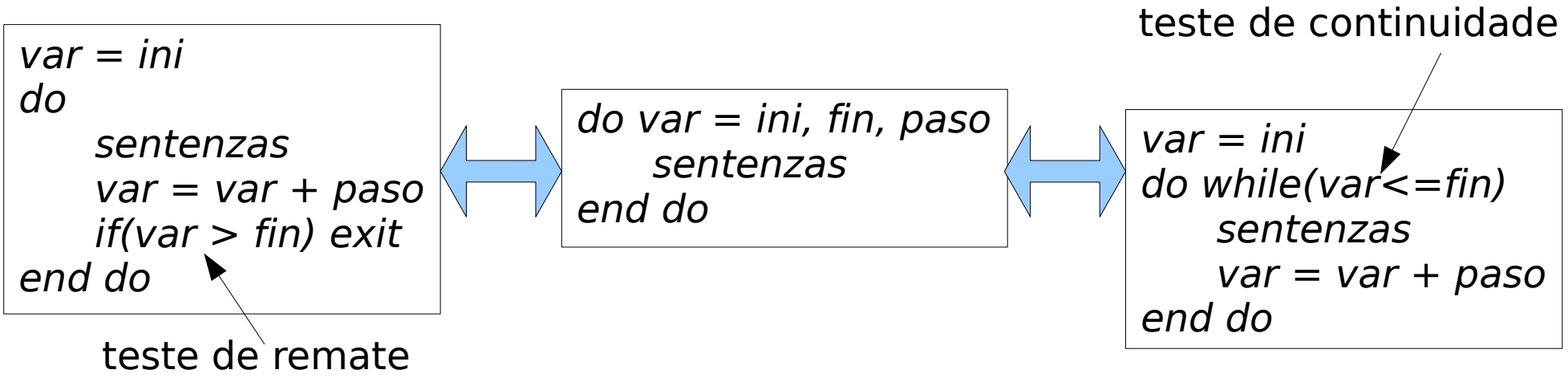
Versión con *do/exit*

Versión con *do-while*

```
suma=0;sumando=1;n=1
do while(sumando>1e-5)
   sumando=1./n**2
   suma=suma+sumando
   n=n+1
end do
```

Relación entre bucles definidos e indefinidos

- Un **bucle definido** pódese construir cun **bucle indefinido**, no cal a inicialización, actualización e teste de continuidade ou remate sobre a variábel fanse manualmente:



- As versións indefinidas son útiles se queremos que a variábel a actualizar sexa real, xa que o bucle *do* definido so permite variábeis enteiras.

Exemplo de bucle indefinido: percorrido cíclico dun vector

- Cíclico: cando chegas ao final do vector, volves ao principio
- O *do* definido non vale: usar *do-exit* ou *do-while*
- Cando chegas ao final do vector, voltas ao principio
- Tamén pode ser decrecendo o índice

```
integer :: x(5)=[1,2,3,4,5],s=0
i=1
do while(s<10)
    print *,'i=',i,' x=',x(i),' s=',s
    s=s+x(i)
    i=i+1
    if(i>10) i=1
end do
print *,'fin: s=',s
```

Bucle *do* híbrido

- É unha mestura de bucle definido e indefinido:

```
do var = ini, fin, paso
   sentenzas
   if(condición) exit
   sentenzas
end do
```

```
suma = 0
do n=1,100
   sumando = 1./n**2
   if(sumando < 1e-5) exit
   suma = suma + sumando
end do
```

```
sumando=1;suma = 0;n=1
do while(sumando>1e-5.and.n<100)
   sumando = 1./n**2
   suma = suma + sumando
   n=n+1
end do
```

- Executa as sentenzas un nº máximo de veces, pero pode rematar antes no *exit* se se cumpre a condición
- Evita unha execución infinita se a condición non está ben deseñada.
- Evita saírse dun vector ou matriz:

```
integer :: x(10)
s=0;i=1
do while(s<1.and.i<=10)
   s=s+x(i);i=i+1
end do
```

Sentenza *cycle*

- Provoca que se salte as sentenzas que restan por executar na iteración actual. Sempre vai asociado a unha condición mediante unha sentenza de selección
- A execución salta ao comezo da seguinte iteración (con bucles *do*'s aniñados, salta á seguinte iteración do bucle *do* máis interno)

```
do i = 1, 10
      print *, “introduce un número natural:”
      read *, n
      if(n <= 0) then
         print *, “dixen que fora natural!!”
         cycle    !salta ao print na iteración i+1
      end if
      suma = suma + n
   end do
```

Nomeamento de bucles *do*

- Como pode haber bucles aniñados, pode resultar útil nomear os bucles:

```
nome1: do i = 1, 10,2
   nome2: do j = 1, 50, 4
      sentenzas
      if(condición) exit nome1
   end do nome2
end do nome1
```

- Permite, p. ex., rematar con *exit* o bucle *do* máis externo (se se desexa)
- Con *cycle*, permite saltar á seguinte iteración dun bucle *do* que non é o máis interno: *cycle nome1*

Sentenza *where/elsewhere*

- Executa sentenzas de asignación para tódolos elementos dun vector/matriz nos que se cumpre (ou non) unha condición.

```
where condición-arrai
    asignacións_arrai_1
elsewhere
    asignacións_arrai_2
end where
```

```
real,dimension(10) :: v,w
where(v>w)
    v=v+w
elsewhere
    v=v-w
end where
```

```
real :: a(3,3),b(3,3)
where(a==b)
        a=b+3
end where
```

Permite vectorizar expresións, evitando bucles *do* e executando operacións para moitos elementos á vez.

Sentenza *forall*

- **Sentenza *forall***: executa iterativamente sentenzas de asignación sobre vector ou matriz, similar a sentenza *where*

  forall (var=ini:paso:fin,condición) asignacións

  forall (var1=ini1:paso1:fin1,var2=ini2:fin2,condición)

  asignacións

  end forall

```
forall (i=1:n,j=1:m,y(i,j)/=0.and.i/=j)
   x(i,j)=1./y(i,j)
end forall
```

- **Bucle *do* implícito**: *print *, (v(i), w(i), i = 1, 10)*
  - Imprime un vector nunha única liña
  - Imprime unha matriz, cada fila nunha liña da terminal

```
do i = 1, 3
   print *, (m(i, j), j = 1, 3)
end do
```

Sentenza *pack* con vectores

- Retorna un vector *k* cos índices, ou os valores, dos elementos do vector que cumpren unha condición. Se ningún a cumpre, vector baleiro (*size(k)*=0): similar á función *find* de Octave
- *pack(x,condición):* valores que cumpren a condición
- *pack([(i,i=1,n)],condición)*: índices dos elementos

  n=nº elementos do vector ou matriz que se testea

  condición: expresión lóxica que ten un vector *x*: p.ex. *x>=2*

Retorna valores que cumpren a condición

```
integer :: x(4)=[2,3,5,2]
print *,'x>2: ',pack(x,x>2)
```

```
integer :: x(4)=[2,3,5,2]
integer,allocatable :: z(:)
z=pack(x,mod(x,2)==0)
print *,'x par: ',z
```

```
integer :: x(4)=[2,3,5,2]
integer,allocatable :: k(:)
k=pack([(j,j=1,4)],x==2)
print *,k  ! k(1)=1, k(2)=4
```

Retorna os índices dos elementos que cumpren a condición

Se queres almacenar os valores ou índices nun vector, hai que declaralo como *allocatable* sen reservar memoria

Sentenza *pack* con matrices

Retorna os valores da matriz que cumpren a condición

```
integer :: a(2,3)=reshape((/7,2,1,8,10,3/),shape(a))
integer,allocatable :: z(:)
print *,'a>2: ',pack(a,a>2)
z=pack(a,mod(x,2)==0)   ! valores pares
```

Retorna os índices dos elementos que cumpren a condición

```
integer :: a(2,3)=reshape((/7,2,1,8,10,3/),shape(a))
integer :: i(2,3)=reshape((/1,2,3,4,5,6/),shape(i))
integer,allocatable :: k(:)
k=pack(i,mod(a,2)==0)  ! k=(2,4,5)
k=pack(i,a>2)  ! k=(1,4,5,6)
k=pack(i,a==7)  ! k=1
```

Uso de *pack* para vectorizar expresións

- Eliminar elementos dun vector:

```
integer,allocatable :: x(:),y(:),j(:),k(:)
x=[(i**2,i=1,10)]
y=pack(x,x>=3.and.x<=5)  ! baseándose no valor de x(i)
j=[(i,i=1,10)]
k=pack(j,j/=2)
x=x(k) ! baseándose no índice: borra x(2)
```

- Eliminar filas e columnas dunha matriz

```
integer :: a(3,3)=reshape((/1,2,3,4,5,6,7,8,9/),shape(a))
integer,allocatable :: b(:,:),k(:),m(:)
k=[(i,i=1,3)]
m=pack(k,k/=2)  ! borra o elemento 2 de vector k
b=a(m,m)    ! colle filas 1,3 e columnas 1,3 de a
```

Conversión entre vectores e matrices

Función *reshape(x,forma)*: o argumento *forma* pode ser *(/n/)* (para convertir a vector de lonxitude *n*), ou (/*nf*,*nc*/), para convertir a unha matriz *nf*x*nc*.

- Conversión de vector a matriz:
  a=reshape(v,(/nf,nc/))
- Conversión de matriz a vector:
  v=reshape(a,(/n/))

```
real :: v(4)=(/1,2,3,4/)
real,dimension(2,2) :: a
a=reshape(v,(/2,2/))
```

Convirte de vector a matriz por columnas

```
real :: v(4)
real,dimension(2,2) :: a=reshape((/1,2,3,4/),shape(a))
v=reshape(a,(/4/))
print *,'v=',v
```

Función *shape(v)*, *shape(a)*: retorna un vector coas dimensións do vector/matriz.

Programa de conversión de vector a matriz

```
program vector_matriz
real,allocatable :: x(:)
real,allocatable :: a(:,:)
real,parameter :: rand_max=2147483647
n=5;m=floor(sqrt(real(n)));k=1
allocate(x(n),a(m,m))
do i=1,n
      x(i)=floor(10.*irand()/rand_max)      !   nº enteiro aleatorio entre 0 e 10
end do
do i=1,m  ! conversión por filas
      do j=1,m
            a(i,j)=x(k);k=k+1          ! para convertir por columnas: a(j,i)=x(k)
      end do
end do
print *,'x=',x                ! imprime vector
print *,'a='                  ! imprime matriz
do i=1,m
      print *,(a(i,j),j=1,m)
end do
deallocate(x,a)
stop
end program vector_matriz
```

Programa de conversión de matriz a vector

```
program matriz_vector
real,allocatable :: x(:)
real,allocatable :: a(:,:)
n=3;m=n**2;k=1
allocate(x(m),a(n,n))
do i=1,n
      do j=1,n
            a(i,j)=i**2*j+i     !   inicializa a(i,j)
      end do
end do
do i=1,n  ! conversión por filas
      do j=1,n
            x(k)=a(i,j);k=k+1          ! para convertir por columnas: x(k)=a(j,i)
      end do
end do
print *,'a='                  ! imprime matriz
do i=1,n
      print *,(a(i,j),j=1,n)
end do
print *,'x=',x                ! imprime vector
deallocate(x,a)
stop
end program matriz_vector
```