Joan Clarke (1917-1996)

"A veces, la persona a la que nadie imagina capaz de nada, es la que hace cosas que nadie imagina"

Joan Clarke (1917)
Descrifó los mensajes alemanes de Enigma.

- Informática do exército británico especializada en criptografía
- Conseguiu descifrar o código Enigma dos alemáns durante a 2ª guerra mundial
- Nomeada cabaleira da Orde do imperio británico en 1947

Constantes e variábeis

- Os datos poden ser:
  - constantes: non se poden modificar durante a execución; dánselle valores no momento en que se comezan a usar: poden ter nome ou non
  - variábeis: pódense modificar, sempre teñen nome
- O nome dun dato pode ter letras, números, “_”:
  - Non pode comezar por números
  - Non pode ter símbolos especiais: +?-=)/*$#

Nomes e tipos de datos

- Fortran NON distingue entre maiúsculas e minúsculas. Sempre usaremos minúsculas
- Os nomes deben ter significado: radio, temperatura, altura, ... Tamén deben ser curtos
- Os datos almacénanse en memoria RAM
- Datos de distintos tipos: enteiros, reais, reais de dobre precisión, complexos, lóxicos e carácter. Ocupan un nº de bytes distinto, e teñen un rango de valores distinto
- Fortran proporciona funcións intrínsecas para realizar operacións estándar (p. ex: funcións matemáticas estándar)

Declaración de variábeis e constantes

- En xeral, os datos deben ser declarados. Na declaración indícase o seu nome e tipo. Ex:

integer x

integer :: x

- Toda declaración debe estar antes de calquer outra sentenza que non sexa unha declaración. Se non, erro de compilación.
- Pódense inicializar na declaración: *integer :: x = -5*
- As constantes con nome decláranse co atributo *parameter* e sempre hai que inicializalas. Ex:

integer, parameter :: x = -10

- Tamén se poden usar constantes sen nome dos distintos tipos en expresións aritméticas: *print *, x + 3.5*

Funcións relacionadas cos tipos dos datos

- *huge(...)*: indica o valor máximo que pode acadar un dato do tipo indicado. Ex:

  integer x

  *print *, huge(x)* => 2147483647

- *precision(...)*: indica o nº de decimais

  real y

  real(kind = 8) z

  *print *, precision(y), precision(z)* => 6 15

- *kind(...)*: indica o nº de bytes que ocupa

  *print *, kind(x), kind(z)* => 4 8

Datos enteiros e reais

- Datos **enteiros**: *integer x*
  - ocupan 4 bytes, signo +/-, sen punto decimal
  - rango valores: $-2^{31}...2^{31}-1$
- Datos **reais**: *real x*
  - ocupan 4 bytes: signo +/-, partes enteira e decimal, expoñente (máx. 2 díxitos)
  - sen expoñente: *x=-1.2*
  - con expoñente: *x=-0.45e-18*
  - rango: $-3.4 \cdot 10^{38}...-3.4 \cdot 10^{-38}$, $3.4 \cdot 10^{-38}...3.4 \cdot 10^{38}$
  - precisión: 6 cifras decimais

Overflow e underflow; reais dobres

- Overflow / underflow: erro que se produce cando superamos o rango de valores dunha variábel real
  - Overflow: supérase o límite máximo: $\pm 3.4 \cdot 10^{38}$
  - Underflow: supérase o límite mínimo: $\pm 3.4 \cdot 10^{-38}$

```
real :: y = -1.2e+80
           1
Error: Real constant overflows its kind at (1)
```

```
real :: y = -1.2e-80
           1
Warning: Real constant underflows its kind at (1)
```

- Reais de **dobre precisión**:
  - 8 bytes
  - Precisión: 15 cifras decimais

Exemplo: cálculo da media de 100 millóns de datos:
- Se sumamos e dividimos por N => overflow
- **Solución**: calcular 100 medias de 1 millón e a media destas 100

Reais dobres e complexos

- Reais de dobre precisión (continuación):
  - Declaración:

    real(8) x

    real(kind=8) x

    double precision :: x
  - Rango: $-1.79 \cdot 10^{308}$, ..., $-1.79 \cdot 10^{-308}$, $1.79 \cdot 10^{-308}$, ..., $1.79 \cdot 10^{308}$
- **Complexos**: parte real e imaxinaria

complex c

c=(-1.3, 1.5e-18)

complex :: c=(-1,2)

Mostrar por pantalla: *print *, c* => (-1.3, 1.5e-18)

Lóxicos e carácter

- **Lóxicos**: só poden toma-los valores *.true.* e *.false.* (constantes)
  - Declaración: *logical x*
  - Inicialización: *x=.true.*
  - Mostrar por pantalla: *print *, x* => T
- **Cadeas de caracteres**:
  - Hai que indica-lo seu tamano máximo: non se pode superar
  - Se non se indica o tamano máximo, vale 1: *character* :: *s='a'*
  - Declaración: *character(100) s*
  - Inicialización: s='ola que tal'
  - Lonxitude da cadea (nº de caracteres que realmente ten): *len_trim*(*s*)
  - Mostrar por pantalla: *print *, s* => hola que tal
  - Teste de igualdade: *s==t*
  - Relación de orde alfabética: *s<t*

Conversión entre tipos

- De enteiro a real: *x=i*
- De real a enteiro: *i=x*, pero podes perder información (p.ex. de 3.2 a 3).
- Podes redondear ao enteiro mais cercano con *i=nint(x)*, por defecto con *i=floor(x)*, por exceso con *i=ceiling(x)*.
- De carácter (p.ex. ‘32’) a enteiro: *s=’32’;read (s,*) i*
- De carácter (p.ex. ‘3.2’) a real: *s=’3.2’;read (s,*) x*
- De enteiro a carácter: *i=32; write (s,’(i0)’) i*
- De real a carácter: *x=3.2; write (s,’(f3.1)’) x*
- De carácter a real/enteiro, só funciona se o carácter ten un número real/enteiro. Se *s=’ola’*, o *read* da erro de entrada/saída.

Declaración implícita (I)

- Na práctica, os datos básicos (reais e enteiros) non é necesario declaralos:
- Os datos enteiros non é necesario declaralos cando os seus nomes comezan polas letras *i j k l m n*
- Os datos reais non é necesario declaralos cando os seus nomes comezan polo resto de letras
- En resumo: se non declaramos un dato:
  - Se o seu nome comeza por {i j k l m n}, é enteiro
  - En caso contrario, é real

Declaración implícita (II)

- Se queremos datos doutro tipo (real dobre, complexo, lóxico, carácter, vector ou matriz), hai que declaralos
- A sentenza *implicit none* anula a declaración implícita no subprograma actual (*f95* da erro de compilación se hai variábeis sen declarar)
- A sentenza:

implicit tipo(rango), ..., tipo(rango)

permite asignar rangos de letras a tipos. Ex:

implicit integer(a-b), real(c-d), complex(e-z)

Vectores e matrices (I)

- Vector: **colección de datos** do mesmo tipo almacenados xuntos na memoria RAM. Cada elemento ten un índice.
- Matriz: cada elemento ten dous índices (fila e columna).
- Acceso a elementos e grupos de elementos: *x(i), x(i:j), x(i:), a(i,:), a(:,i), a(i:j,k:l)*. Vectores e matrices son **arrais**. Deben declararse.
- Inicialización de vector: *x=[1,2,3], x=[(i,i=1,10,2)], x=(/1,2,3/)*
- Inicialización de matriz *x*: a=*reshape(x,shape(a))*: o vector vai por columnas
- **Vector/matriz estático:** mesma lonxitude en tódalas execucións do programa.

```
real v(3)
integer :: a(3,3)
v(1)=2
v=[1,2,3]
a(2,3)=5
```

```
real v(10),a(2,2)
v=[(2*i+1,i=1,10)]
a=reshape(([((i*j,j=1,2),i=1,2)],shape(a)))
```

Vectores e matrices (II)

- **Vector/matriz dinámico**: distintas lonxitudes en distintas execucións.
- Resérvase a memoria cando se coñece o nº de elementos
- Se se accede a un elemento do matriz/vector **dinámico** antes do *allocate* ou despois do *deallocate* da erro de segmentación

```
real,allocatable :: v(:)
integer,allocatable :: a(:,:)
read *,n,nf,nc
allocate(v(n),a(nf,nc))
v(3)=2
a(2,3)=5
deallocate(v,a)
```

No *deallocate* non se indica o nº de elementos

```
real,allocatable :: v(:)
v=[1]
do i=1,3
      v=[v,i]
end do
```

- Pódese engadir elementos a un vector dinámico:

Vectores e matrices (III)

Dinámico

- **Declaración**: *real x(10)*, ou *real :: x(10)*, ou *real,allocatable :: x(:)*

  integer a(3,3), ou *integer :: a(3,3)*, ou *integer,allocatable :: a(:,:)*

- Alternativa: *tipo :: nome(N_1:N_2)*

Se é dinámico:
allocate(x(-15:15))

  Exemplo: *integer :: x(-5:5)*. Ten elementos x(-5)...x(5): $N_2 - N_1 + 1$ elementos: as funcións *lbound(x)* e *ubound(x)* dan N_1 e N_2

- Declaración vella (en exames resoltos):

  int,dimension(5) :: x
  real,dimension(-5:5) :: y

Le a matriz por columnas: hai que traspoñer

- **Ler** vector por teclado: *read *,x*
- **Ler** matriz:

```
do i = 1,n
   read *, (a(i, j),j=1,m)
end
```

```
read *,a
a=transpose(a)
```

Vectores e matrices (IV)

- **Inicialización** con bucle *do* explícito:

```
do i=1,n
    v(i)=i**2
end do
```

```
do i=1,n
    do j=1,m
        a(i,j)=i*j+i
    end do
end do
```

- **Inicialización** con bucle *forall*:

```
forall(i=1:10) v(i)=i*i+i-1
```

```
forall(i=1:5,j=1:6) a(i,j)=i*j-i+j
```

- **Escribir** vector: *print *,x* (tódolos elementos),

```
print *,x(1:n)
print *,(x(i),i=1,n)
```

só *n* primeiros elementos

- **Escribir** matriz:

```
do i = 1,2
   print *, (a(i, j),j=1,2)
end
```

Vectorización de expresións (I)

- Escribir unha expresión con vectores/matrices como se fose con números
- Sexan *x,y* dous vectores (de igual lonxitude) e *a,b* dúas matrices (de iguais dimensións).
- Dimensións: *size(x), size(a),size(a,1), size(a,2), shape(x), v=shape(a)*
- Asignación de valores a un arrai ou asignación dun arrai a outro: *x(:)=5; a(:,:)=7; x=y; b=a*
- Operacións aritméticas compoñente a compoñente: *x+2*y, x*y, x/y, 1/x, x**y, 1/a, a*b, 2**a, a**2, a**b*
- Suma e producto de elementos: *sum(x), sum(x(2:)), sum(a), sum(a,1), sum(a,2), product(x), product(a)*. Media dun vector: *sum(x)/size(x)*
- Valores e índices dos elementos máximo e mínimo dun vector: *maxval(x), minval(x), maxloc(x), minloc(x)*. O mesmo para matrices.
- Producto escalar: *dot_product(x,y)*
- Transposta dunha matriz: *b=transpose(a)*
- Producto de dúas matrices: *p=matmul(a,b)*


```
integer :: a(2,2)=reshape((/1,2,
    3,4/), shape(a)),x(2)
x=sum(a,2)
print *,sum(a,1)
```