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 na memoria RAM (*Random Access Memory*) do ordenador
- 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:

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, neste caso hai que poñer o símbolo :: por exemplo *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

```
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*

Declaración implícita

- Enteiros: non hai que declaralos se o nome comeza por *i j k l m n*
- Reais: se comeza 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
- 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 (*gfortran* 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, un índice. Declaración: *integer :: x(10)*
- Matriz: dous índices (fila e columna). Declaración: *integer :: a(3,3)*
- 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 *a*: a=*reshape([1,2,3,4,5,6,7,8,9],[3,3])*: os valores van 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)],[2,2]))
```

Vector con valores equiespaciados entre *a* e *b*

- Vector con paso *p:* $x_i = a + p(i-1), i = 1..n$ $\quad n = 1 + \left\lfloor \frac{b-a}{p} \right\rfloor$
- Vector con *n* valores: $x_i = a + \frac{b-a}{n-1}(i-1), i = 1..n$ $\quad p = \frac{b-a}{n-1}$
- Hai *n* valores entre *a* e b
- Hai *n*-1 intervalos entre *a* e *b*

Se nos dan a lonxitude *n* (nº de puntos) do vector:

```
integer,parameter :: n=10
real,parameter :: a=0,b=1
real :: x(n),p=(b-a)/(n-1)
do i=1,n
    x(i)=a+p*(i-1)
end
```

Se nos dan o paso *p* (ancho do intervalo):

```
real,parameter :: p=0.1,a=0,b=1
real,allocatable :: x(:)
n=1+(b-a)/p
allocate(x(n))
do i=1,n
    x(i)=a+p*(i-1)
end
deallocate(x)
```

Vectores e matrices (III)

- Outra forma de declaración: *tipo :: nome(N_1:N_2)*. É para que o 1º elemento do vector non sexa o 1 e o último non sexa o *N*
- Exemplo: *integer :: x(-5:5)*. Ten elementos x(-5)...x(5)
- Con esta declaración, o vector ten $N_2 - N_1 + 1$ elementos: as funcións *lbound(x)* e *ubound(x)* dan N_1 e N_2. Cun vector dinámico sería: *real,allocatable :: x(:)*, e logo *allocate(x(-15:15))*
- **Ler** vector por teclado: *read *,x*
- **Ler** matriz:

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

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

Le a matriz por columnas: hai que traspoñer

Vectores e matrices (IV)

- **Escribir** vector: *print *,x* (tódolos elementos), ou *print *,x(:n)* (só *n* primeiros elementos)
- Con formato: *print '(f6.2," ",$)', x; print **
- **Escribir** matriz:

Con formato por defecto:

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

Con formato específico:

```
do i=1,n
     print '(i0," ",$)',a(i,:)
     print *
end
```

- **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
```

Vectorización de expresións (I)

- Vectorizar: 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)*, onde v é un vector
- 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 dun vector: *sum(x), product(x)*
- Suma dunha parte do vector: *sum(x(2:)), sum(x(:5))*
- Media dun vector: *sum(x)/size(x)*
- Suma e producto dunha matriz: *sum(a)*, *product(a)*
- Suma de matriz por columnas e filas*: sum(a,1), sum(a,2)*

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

Vectorización de expresións (II)

- Valores e índices dos elementos máximo e mínimo dun vector: *maxval(x), minval(x), maxloc(x), minloc(x)*. O mesmo para matrices.
- Producto escalar de dous vectores: *dot_product(x,y)*
- Transposta dunha matriz: *b=transpose(a)*
- Producto matricial de dúas matrices: *p=matmul(a,b)*
- Atopar o índice do 1º elemento dun vector cun certo valor:

*i=**findloc**(vector,valor,1)*

Retorna o índice *i* do primero elemento do vector con ese valor; 1 é a dimensión do vector

```
integer :: x(4)=[3,1,4,4]
i=findloc(x,1,1)
print *,'i=',i
```

mostra a posición 2 por pantalla