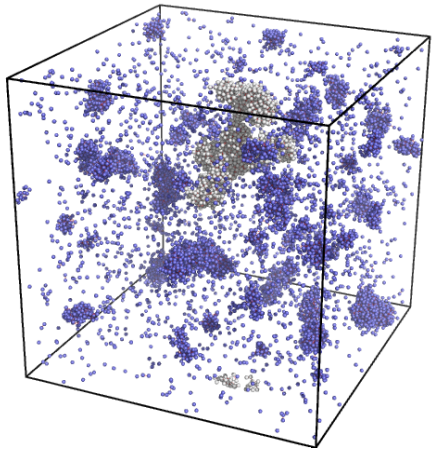
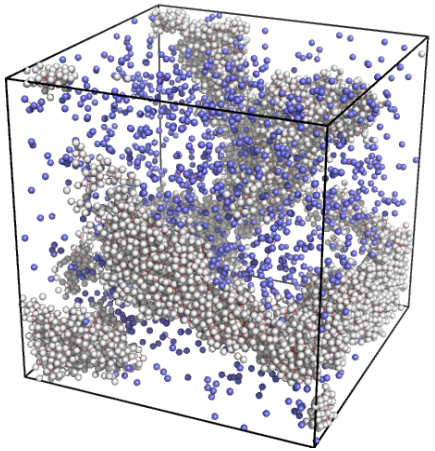
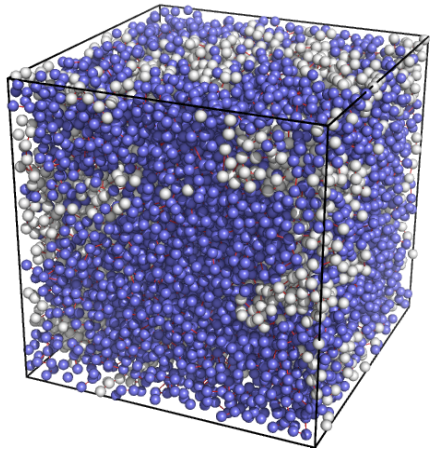
FORTRAN
Una breve introducción

Física Computacional
Universidad de Granada

Estructura del curso

- Introducción
- Programas y su estructura
- Variables
- Operaciones aritméticas
- Funciones intrínsecas
- *Input* y *output*
- Iteraciones y bucles
- Vectores
- Decisiones
- Funciones y subrutinas
- Formato y manipulación de archivos
- Problemas

Introducción

- El lenguaje de programación de **FORTRAN** fue concebido en 1957 por IBM.
- Su nombre proviene de ***FOR****mula* ***TRAN****slation*.
- FORTRAN está especialmente adaptado al **cálculo numérico** y a la **computación científica**
- En 1966 se hizo estándar y nació FORTRAN IV. Una primera revisión del lenguaje condujo al **FORTRAN 77** que ha sido y es muy popular por su eficiencia.
- Tras muchos años con el estándar FORTRAN 77, desde la década de los 90 se han introducido nuevas revisiones, FORTRAN 90, 95, 2003, para desembocar en el estándar actual, **FORTRAN 2008**.
- FORTRAN fue diseñado por científicos e ingenieros, y ha dominado esos campos los últimos 50 años.
- Es uno de los lenguajes más populares en el área de la computación de alto rendimiento y es el lenguaje usado para programas que evalúan el desempeño (*benchmark*) y el ranking de los **supercomputadores** más rápidos del mundo.

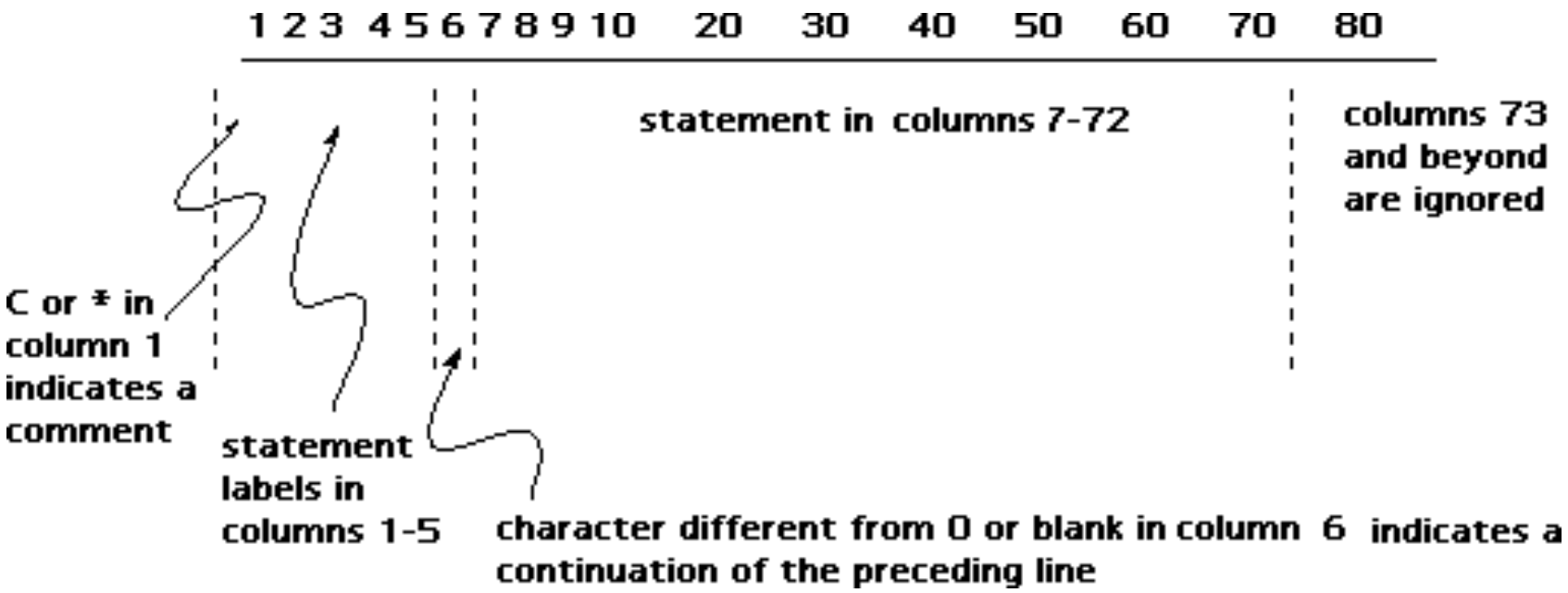
Programas

- Un programa es una **lista de instrucciones** que permiten a un ordenador resolver un problema.
- **El problema a resolver se rompe en partes** que definen una estructura: el problema complejo original al principio del programa y las partes sencillas y más pequeñas al final.
- ***Divide et impera***: divide y vencerás (Julio César)
- Los **pasos** para realizar un programa son:
  - Examinar el problema y dividirlo en varias partes.
  - Examinar cada parte y dividirla en partes más pequeñas.
  - Establecer un **plan lógico** que una todas las partes.
  - Escribir el programa y utilizar subprogramas y funciones.
  - Comprobar el programa: esto es un ***arte***

Formato del programa

- Todas las instrucciones en FORTRAN deben de escribirse entre las **columnas 7 hasta la 72**. Los caracteres escritos a partir de la columna 73 no se consideran. Esto se puede obviar usando **formato libre**.
- Si una instrucción necesita una **etiqueta**, ésta debe de aparecer en las columnas 1-5. Las etiquetas son números enteros que van del 1 hasta el 99999.
- Ocasionalmente, una instrucción ocupa más espacio que el correspondiente a una línea (7-72). En este caso, la instrucción puede **continuarse en líneas consecutivas** siempre que en la **columna 6** de las líneas continuación se coloque un carácter numérico o alfanumérico (excepto el 0 o el espacio). Un 0 o un espacio indica que se trata de la primera línea de la instrucción.
- Líneas que están vacías o que tienen una letra C o un asterisco * en la columna 1 representan **líneas de comentarios** que no son ejecutadas.

Formato del programa

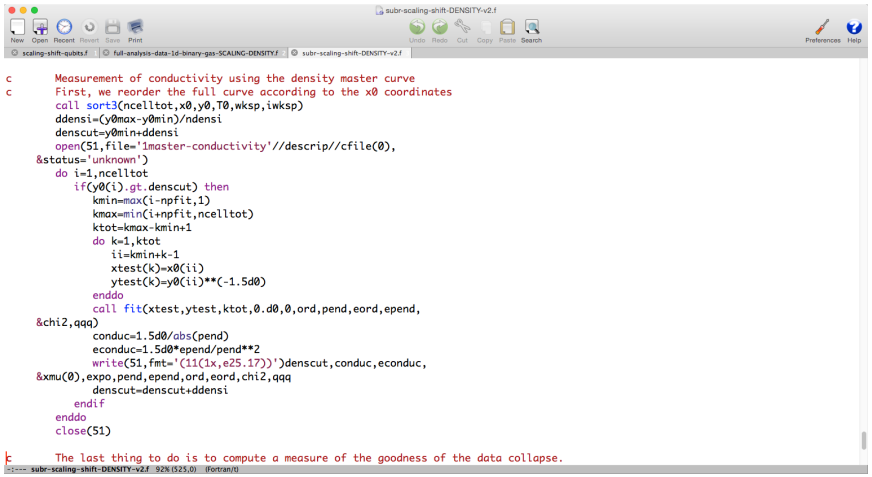


Estructura típica de una simulación

```
      program abc

c     Definición de variables
      implicit none
      integer i, j, k, ...
      real*8 x, y, z, ...
      ...

c     Definición de constantes
      parameter (cte1=0.7654, pi=3.1415927, ...)

c     Dimensionado de vectores
      dimension x(0:1000), y(0:1000), z(0:1000)

c     Lectura de datos de entrada
      open(10,file='input.dat',status='old')
      read(10,*) x0,y0,z0
      ...
      close(10)

c     Cuerpo principal del programa
      do while (time<tmax)
         call evolución
         call medir_observables
         if (mod(time,twrite)==0) call escribir_resultados
      enddo

c     Fin del programa
      stop

      end
```

Edición, compilación y ejecución

- Para escribir un programa FORTRAN, podemos usar **cualquier editor** de texto estándar: vi, pico, emacs, kwrite, gedit, etc.



```
c       Measurement of conductivity using the density master curve
c       First, we reorder the full curve according to the x0 coordinates
        call sort3(ncelltot,x0,y0,T0,wksp,iwksp)
        ddensi=(y0max-y0min)/ndensi
        denscut=y0min+ddensi
        open(51,file='1master-conductivity'//descrip//cfile(0),
     &status='unknown')
        do i=1,ncelltot
           if(y0(i).gt.denscut) then
              kmin=max(i-npfit,1)
              kmax=min(i+npfit,ncelltot)
              ktot=kmax-kmin+1
              do k=1,ktot
                 ii=kmin+k-1
                 xtest(k)=x0(ii)
                 ytest(k)=y0(ii)**(-1.5d0)
              enddo
              call fit(xtest,ytest,ktot,0.d0,0,ord,pend,eord,epend,
     &chi2,qqq)
              conduc=1.5d0/abs(pend)
              econduc=1.5d0*epend/pend**2
              write(51,fmt='(11(1x,e25.17))')denscut,conduc,econduc,
     &xmu(0),expo,pend,epend,ord,eord,chi2,qqq
              denscut=denscut+ddensi
           endif
        enddo
        close(51)

c       The last thing to do is to compute a measure of the goodness of the data collapse.
```

- Salvamos el archivo con **extensión .f**, por ejemplo prog.f
- A continuación **compilamos el programa** con el comando **gfortran** para generar un archivo ejecutable:

gfortran prog.f -o prog.exe -O3

- La **opción -o** nos permite elegir el nombre del ejecutable (prog.exe en este caso), y la **opción -O3** optimiza el código.
- Para **ejecutar** el programa, escribimos en la terminal **./prog.exe&**

Variables

- Una variable es una magnitud que puede cambiar de valor.
- El nombre de una variable debe de comenzar con una letra.
- Hay varios **tipos de variables** en FORTRAN. El tipo de variable se especifica al principio del programa, antes de cualquier linea de ejecutables.
- Si no se declaran las variables, el compilador hará ciertas suposiciones (**variables intrínsecas**).
- Por ejemplo, cualquier variable que comience con las letras I,J,K,L,M,N es ENTERA. El resto de variables se consideran REALES (de 4 bytes) de manera intrínseca.
- Se recomienda que las variables sean declaradas para evitar errores. Para eliminar los tipos implícitos escribimos ***implicit none*** al principio del programa y procedemos a definir cada variable según su tipo.

Variables enteras y reales

- **Enteros**
  - Enteros son todos los números sin punto decimal.
  - La forma de declarar enteros es:

```
INTEGER namel, name2
```

  - **Ejemplo**: **INTEGER WHOLE, AVERAGE, SUM**

- **Reales**
  - Reales en FORTRAN son en realidad todos los números racionales.
  - Un número real no puede ser almacenado con precisión infinita.
  - La forma general de la declaración de una variable real es:

```
REAL*8 namel, name2
```

  - **Ejemplo**: **REAL*8 FRACT, MEAN, STDDEV**

Variables de doble precisión

- Las variables reales tienen por defecto precisión simple pues se almacenan en una única posición de memoria. Para números no muy grandes estos números llegan a tener 7 dígitos de precisión.
- Para incrementar el número de dígitos de precisión se deber de declarar la variable como de **doble precisión**. Cada una de estas variables usa dos posiciones de memoria para almacenarse y provee hasta 13-14 dígitos de precisión.
- La forma de declarar las variables en doble precisión es:

```
DOUBLE PRECISION namel, name2
```

- Ejemplo: **DOUBLE PRECISION ROOT, VELO**

Números enteros

- La representación de números (enteros, reales, complejos, etc.) en el ordenador se hace mediante el **sistema binario**.
- El número binario $[b_{31}, b_{30},\ldots, b_0]_2$, de **4 bytes** (o **32 *bits***, **1 *byte*=8 *bites***), con b_k=0,1, representa el entero

$$[b_{31}, b_{30}, \ldots, b_0]_2 = (-1)^{b_{31}} \sum_{k=0}^{30} b_k 2^k$$

- El entero más grande de 32 bits es **2147483647~ 2.147x10^9**.

Números en coma flotante

- La **representación de coma flotante** permite representar números racionales extremadamente grandes y pequeños de una manera muy eficiente y compacta.
- Representación de números reales en coma flotante

$$num = (-1)^{s} \times mantisa \times 2^{exp-bias}$$

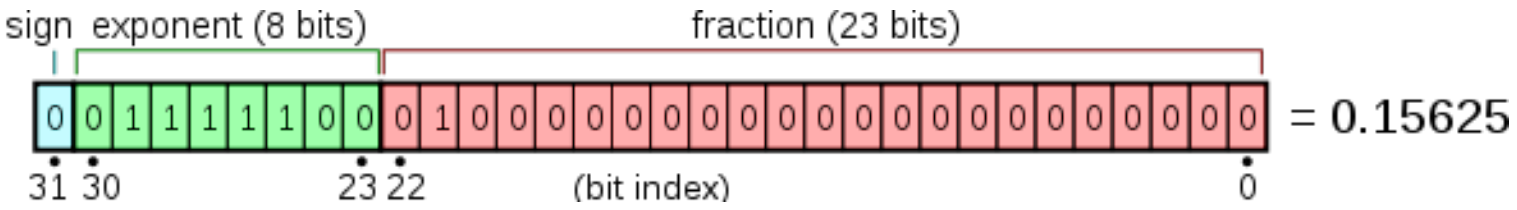
- ***s=0,1*** es el bit de signo, la ***mantisa*** contiene los dígitos del número a representar, y el exponente (***exp***) indica donde colocar el punto decimal (una vez sustraido el ***bias***)
- Para un número **REAL*4** (de 4 *bytes*, o **32 *bites***), se reservan **23 bits para la mantisa** y **8 para el exponente**: eso nos da **7-8 cifras significativas**

$$x = (-1)^{b_{31}} \left(1 + \sum_{k=1}^{23} b_{23-k} 2^{-k}\right) \times 2^{\text{exp-bias}}$$

- Para un número **REAL*8** (de 8 *bytes*, o **64 *bites***), se reservan **52 bits para la mantisa** y **11 para el exponente**: eso nos da **15-16 cifras significativas**
- El **rango total de los REAL*4**, positivos o negativos, va **de 1.47×10^{-39} a 1.70×10^{38}**. Para los números de **doble precisión** abarca de **2.23×10^{-308} a 1.80×10^{308}**.

Números en coma flotante

Ejercicio: Compruebe la siguiente representación binaria de un número REAL*4.



Caracteres y constantes

- **Caracteres** son variables que contienen uno o más caracteres (por ejemplo G o GRANADA)
- La forma general de la **declaración** es:

CHARACTER namel,name2	donde name1 y name2 tienen 1 carácter cada uno
CHARACTER*n namel,name2	donde name1 y name2 tiene n caracteres cada uno
CHARACTER namel*nl,name2*n2	donde name1 tiene n1 caracteres y name2 n2

- **Constantes**: son cantidades cuyos valores no varían durante la ejecución del programa.
- Para definir **constantes**, usamos la orden ***parameter***:

parameter (pi=3.1415927, …)

Operaciones aritméticas

- Las operaciones básicas son:

+	suma
-	resta
*	multiplicación
/	división
**	exponenciación

- **Aritmética real**: Si todas las variables y constantes en una expresión son reales las operaciones se realizarán sin truncar ningún decimal.
- **Aritmética entera**: Si todas las variables y constantes en una expresión son enteros, las sumas, restas, multiplicaciones y exponenciaciones se realizarán sin problema. Sin embargo las divisiones entre enteros ignoran en su resultado la parte decimal que es automáticamente truncada.
- **Ejemplos de aritmética entera**:
  - 5/2 da el resultado de 2 en lugar de 2.5
  - 3/4 da el resultado de 0 en lugar de 0.75

Aritmética mezcla

- **Aritmética en modo mezcla**: Esto ocurre cuando en una expresión hay **mezclados enteros y reales**.
  - Si cualquiera de los operandos es real el resultado es real.
  - Hay que utilizar con **cuidado** la mezcla de tipos de operandos.
- Ejemplo:
  - 5/2*3.0 = 6.0 porque el orden de la operación es **de izquierda a derecha**
  - En cambio, 3.0*5/2 = 7.5
- **Asignación a variables en modo mezcla**: Si la variable a la que se asigna una expresión ha sido declarada como real, todos los decimales resultantes de la evaluación de la expresión se conservarán.
- **Ejemplo**: la variable real x=5*2.1 tendrá el valor de 10.5
- Si la variable a la que se asigna una expresión ha sido declarada como entero, todos los decimales resultantes de la evaluación de la expresión se perderán.
- **Ejemplo**: la variable entera i=5*2.1 tendrá el valor de 10

Reglas de prioridad

- Las expresiones aritméticas son realizadas de acuerdo con las siguientes reglas de prioridad:
- Todas las exponenciaciones se realizan primero. Exponenciaciones consecutivas se realizan de derecha a izquierda.
- Todas las multiplicaciones y divisiones se realizan después, en el orden en el que aparecen de izquierda a derecha.
- Las sumas y las restas se realizan las últimas, en el orden en el que aparecen de izquierda a derecha.
- Es muy recomendable usar profusamente el parentesis, (…), para evitar errores y conflictos.

Funciones

- FORTRAN tiene funciones definidas de manera intrínseca para llevar a cabo cálculos sobre una serie de argumentos devolviendo un resultado.
- Para usar una función simplemente se da el nombre de la función, seguido de sus argumentos entre paréntesis:

```
funtion_name(name1, name2,...)
```

- La lista de funciones intrínsecas de FORTRAN es muy amplio (consultar documentación para un listado exhaustivo). A continuación proporcionamos una tabla con las más usuales

Ejercicio: Calcular a mano MOD(10,3), MOD(4,3) y MOD(10.3,6)

Funciones

ABS (x)	Valor absoluto de x
SIN(x)	Seno de x en radians
COS (x)	Coseno de x en radianes
EXP(x)	Exponencial de x
LOG(x)	Logartimo natural de x
SQRT(x)	Raiz cuadrada de x
MAX(xl, . . . , xn)	Maximo de xl, . . .,xn
MIN(xl, . . . , xn)	Minimo de xl, . . ., xn
MOD(x,y)	x (mod y); x - INT(x/y)*y
DBLE(x)	Conversion de x a doble precision
INT(x)	Parte entera de x
NINT(x)	x redondeado al entero más próximo
REAL*8(x)	Conversion of x al tipo real

Input y output

- Supongamos el siguiente **problema**: Un cohete es lanzado desde una altura inicial H00 con velocidad inicial V0 y una aceleración vertical A. Entonces la altura y velocidad en el instante T después del lanzamiento vienen dados por las ecuaciones:

$$H = 0.5*A*T**2 + V0*T + H0$$

$$V = A*T + V0$$

- El siguiente programa asigna el valor -9.807 (m/s**2) a A, 150.0 (m) a H0, 100.0 (m/s) a V0 y 5.0 (sec) a T, calculando los valores de H y V

Input y output

```
      PROGRAM PROJECT
************************************************************************
* This program calculates the velocity and height of a projectile      *
* given its initial height, initial velocity, and constant             *
* acceleration.  Variables used are:                                   *
*    H0   :  initial height                                            *
*    H    :  height at any time                                        *
*    V0   :  initial vertical velocity                                 *
*    V    :  vertical velocity at any time                             *
*    A    :  vertical acceleration                                     *
*    T    :  time elapsed since projectile was launched                *
*                                                                      *
* Input:   none                                                        *
* Output:  none                                                        *
************************************************************************

      REAL*8 H0, H, V0, V, A, T

      A = -9.807
      H0 = 150.0
      V0 = 100.0
      T = 5.0
      H = 0.5*A*T**2 + V0*T + H0
      V = A*T + V0
      END
```

Input/output

- El programa calcula los valores de H y V como se le ha ordenado pero los almacena internamente sin que pueda verlos el usuario. Más aun, si queremos realizar el cálculo con otra altura o velocidad iniciales, hemos de reescribir el programa, volverlo a compilar y a ejecutar.
- Vamos a explicar cómo ver la información generada por el programa (***output***) y cómo variar la información inicial (***input***) sin tener que modificar el programa cada vez.
- Las instrucción para leer datos en el programa es ***READ***, mientras que la orden para escribir resultados del programa es ***WRITE***. El formato estándar es:

READ (origen, formato) lista_variables

WRITE (origen, formato) lista_variables

- **Ejemplo** (el * es la entrada/salida estándar, normalmente la terminal)

READ (*,*) x0, y0

WRITE (*,*) H, V

READ

- El formato estándar es:

READ (origen, formato) lista_variables

- ***lista_variables*** es una única variable o variables separadas por comas.
- ***READ*** transfiere valores desde alguna fuente externa (usualmente teclado o archivo) y los asigna a las variables de la lista de variables.
- Las siguientes **reglas** se aplican:
  - Una **nueva linea** de datos es procesada cada vez que una instrucción READ se ejecuta.
  - Cuando el programa llega a un READ(*,*), la ejecución de éste **se suspende** hasta que el usuario entra todos los valores de las variables de la lista.
  - Si en una linea de entrada hay menos valores que el número de variables en *lista_variables*, se procesan subsiguientes lineas de input hasta que todos los valores de las variables se han obtenido.
  - Si hay más entradas en una linea de entrada que el número de variables en *lista_variables*, se usan los primeros valores y el resto se ignora.
  - Las entradas deben de ser **constantes y del mismo tipo** que la de las variables a las que son asignadas.
  - Entradas consecutivas en una linea de entrada deben de ser separadas por una coma o por uno o más espacios.

WRITE

- El formato estándar es:

WRITE (origen, formato) lista_variables

- ***lista_variables*** es una única variable o variables separadas por comas.
- Cada una de estas expresiones es una constante, una variable, o una fórmula.
- Ejemplos:

WRITE (*,*) H, V

WRITE(10,75) x, sin(x), cos(x), sin(x)**2+cos(x)**2

- En el último ejemplo, el comando WRITE escribe cuatro valores en el **archivo etiquetado con el índice 10** y con **formato especificado en la línea con etiqueta 75**
- Para escribir una **cadena de caracteres,** usamos comillas:

WRITE (*,*) 'Así escribo una cadena de caracteres'

- Para escribir una **línea en blanco**:

WRITE (*,*)

Ejemplo de input/output

```
      PROGRAM PROJECT
************************************************************************
* This program calculates the velocity and height of a projectile      *
* given its initial height, initial velocity, and constant             *
* acceleration.  Variables used are:                                   *
*    H0   :  initial height                                            *
*    H    :  height at any time                                        *
*    V0   :  initial vertical velocity                                 *
*    V    :  vertical velocity at any time                             *
*    A    :  vertical acceleration                                     *
*    T    :  time elapsed since projectile was launched                *
*                                                                      *
* Input:   H0, V0, T                                                   *
* Output:  V, H                                                        *
************************************************************************

      REAL*8 H0, H, V0, V, A, T

      A = -9.807
      WRITE(*,*)'ENTER THE INITIAL HEIGHT AND VELOCITY:'
      READ(*,*) H0, V0
      WRITE(*,*)'ENTER TIME AT WHICH TO CALCULATE HEIGHT AND VELOCITY:'
      READ(*,*) T
      H = 0.5*A*T**2  +  V0*T  +  H0
      V = A*T + V0
      WRITE(*,*)'AT TIME ', T, ' THE VERTICAL VELOCITY IS ', V
      WRITE(*,*)'AND THE HEIGHT IS ', H
      END
```

Ejercicios: Input/output y aritmética

Ejercicio: Escriba el programa anterior, compílelo y ejecútelo.

Ejercicio: Escriba un programa que convierta los grados Celsius a Fahrenheit y muestre en pantalla el resultado. Recuerde que la relación entre las dos es:

$$[^\circ F] = \frac{9}{5}[^\circ C] + 32$$

Ejercicio: Escriba un programa que muestre en la pantalla la raíz cuadrada de cada número entero del 1 al 20.

Ejercicio: Escriba un programa que tome tres números desde el teclado, a, b y c, y muestre en pantalla las raíces de la ecuación

$$ax^2 + bx + c = 0$$

Note que el programa fallará cuando intente calcular raíces de números negativos. Este problema lo solventaremos más adelante.

Iteraciones y bucles

- Un bucle ***DO*** permite la repetición de una sección del programa un número prefijado de veces. Las instrucciones repetidas son las que se encuentran entre DO y ENDDO

```
DO I = START, STOP, STEP
   cuerpo del bucle
ENDDO
```

- En el bucle anterior: ***I*** es típicamente un entero., ***START*** es el valor de I en la primera iteración, ***STOP*** es el valor de I para el que se detiene la iteración, ***STEP*** es el incremento de I en cada iteración. Cuando STEP se omite de la expresión se supone igual a 1.
- Ejemplos:

```
do i=1,10
   write(*,*) i, log(i**2)
enddo
```

```
do i=10,0,-1
   write(*,*) i
enddo
```

- Otra estructura iterativa útil es el ***DO WHILE***:

```
DO WHILE (cond_logica)
   cuerpo del bucle
ENDDO
```

Ejercicios: Bucles

Conversión de Temperaturas: Escriba un programa que imprima en pantalla una tabla de conversión de grados Celsius a Fahrenheit de 0ºC a 100ºC de uno en uno.

Rango de las variables reales en FORTRAN: Escriba un programa que encuentre el valor máximo que puede almacenarse en una variable real. Por ejemplo, puede multiplicar repetidamente una variable por 2 dentro de un bucle, mostrando en pantalla el resultado hasta que el ordenador dé un error.

Rango de las variables enteras en FORTRAN: Reescriba el programa anterior para números enteros.

Vectores

- Un vector es un conjunto de variables ordenadas que comparten el mismo nombre y son del mismo tipo (REAL*8, ENTERAS o CARACTERES).
- El acceso a cada una de las variables ser realiza variando su índice. Por ejemplo, un grupo de 10 estudiantes han recibido notas por un trabajo entregado. Si la nota de cada estudiante se almacena en una variable: ***MARK1, MARK2, ...,MARK10***, la nota media será:

SUM = (MARKI+MARK2+MARK3+...+MARK9+MARK10)/10.0

- Es más conveniente utilizar un **vector *MARK(I)*** donde I puede tomar valores entre 1 y 10, y utilizar un ***DO***

```
sum = 0.d0
do i=1,10
   sum=sum+mark(i)
enddo
sum=sum/10.d0
write(*,*)'Media=',sum
```

- Piense en las ventajas de esta forma de asignación si quisiéramos sumar 100 ó 1000 datos.

Declaración de un vector

- Los vectores deben declarase **al principio del programa**. Para ello, hay que establecer el tipo de datos que contendrán, seguido por el nombre del vector y el número total de elementos.

INTEGER MARK(10)

REAL*8*8 TABLE(25), TIME(50)

- A veces queremos que los índices del vector comiencen en un número diferente de 1, que es el **índice inicial** por defecto. En ese caso se ha de especificar los valores del índice inicial y final, por ejemplo:

INTEGER ERROR(-10:10)

DOUBLE PRECISION SECNDS(0:50)

- Podemos especificar el tipo de variable (entera, real, etc.) y su carácter vectorial o tensorial de manera separada, usando ***DIMENSION***

INTEGER ERROR, TIME

DIMENSION ERROR(-10:10), TIME(-3:47)

- Podemos definir **tensores de cualquier rango**:

REAL*8*4 CURVATURA(0:4, -3:3, 7)

Inicialización de vectores

- Se puede asignar valores a los elementos de un vector: ***MARK(1) = 23.5***
- También se pueden obtener uno a uno desde el teclado:

```
do i=1,n
   read(*,*) mark(i)
enddo
```

- Sin embargo, para vectores de gran tamaño es más eficaz leer sus valores iniciales de un archivo:

```
do k=1,count
   read(10,*) name(k), number(k)
enddo
```

- Para inicializar todas las componentes de un vector (o tensor en general) a un valor dado, hacemos:

MARK = 0.d0

- De esta manera, MARK(j)=0.d0 para todo j en el rango del vector.

Dimensionado dinámico de vectores

- Un vector (o tensor) A puede ser dimensionado dinámicamente, i.e. se le puede asignar espacio en la memoria durante la ejecución del programa. Para ello escribimos:

REAL*8, DIMENSION(:), ALLOCATABLE :: A

- A la hora de ejecutar el programa, los límites del vector dinámico A se especifican con la orden

ALLOCATE (A(N))

donde N es una variable entera que ha sido asignada previamente.

- Podemos eliminar un vector dinámico de la memoria usando la orden DEALLOCATE

DEALLOCATE (A)

- También podemos dimensionar dinámicamente tensores con multiples índices

REAL*8, DIMENSION(:, :, :), ALLOCATABLE :: CURV

- Durante la ejecución, asignamos memoria mediante

ALLOCATE (CURV(N1, N2, N3))

con N1, N2 y N3 tres variables enteras previamente asignadas.

Ejercicios: Vectores

Ejercicio: Escriba un programa que lea algunos números desde el teclado y los almacene en un vector. Imprima el vector en la pantalla y compruebe que lo hace correctamente.

Ejercicio: Escriba el programa anterior para que los números almacenados se impriman en pantalla en orden inverso.

Decisiones

- Las decisiones en FORTRAN se realizan con la estructura ***IF…THEN…ENDIF***
- La decisión se toma comprobando si una **expresión lógica** es verdadera o falsa. Si es verdadera, se ejecutan las instrucciones que aparecen entre IF...THEN y ENDIF. Si es falsa el programa se salta el bloque anterior y continua a partir de ENDIF.

```
if (logical_expression) then
   instrucciones
endif
```

- La forma más común de una expresión lógica es: ***(exp_aritm Operador exp_aritm)***. Por ejemplo: ***(A .GT. 0)***.

Decisiones

- Los **operadores** posibles son:

.GT. (>)	Más grande que
.LT. (<)	Más pequeño que
.EQ. (==)	Igual a
.GE. (>=)	Más grande o igual a
.LE. (<=)	Más pequeño o igual a
.NE. (/=)	No es igual a

- Las expresiones lógicas puede ser encadenadas usando los operadores lógicos ***.AND.*** y ***.OR.***
- Ejemplo:

```
if ((A.GT.B).AND.(B.LE.C)) then
   instrucciones
endif
```

Tabla ***True/False***

T.AND.T=T	T.OR.T=T
T.AND.F=F	T.OR.F=T
F.AND.T=F	F.OR.T=T
F.AND.F=F	F.OR.F=F

Decisiones

- Para la toma de decisiones alternativas, usamos la estructura ***IF…THEN…ELSE…ENDIF***
- Esta estructura se utiliza si hay instrucciones que deben de ser ejecutadas únicamente cuando la expresión lógica es verdad y otras únicamente cuando es falsa. Por ejemplo:

```
if (logical_exp) then
   instrucciones si logical_exp=.TRUE.
else
   instrucciones en caso contrario
endif
```

- Si hay **varias posibilidades**, usamos la estructure ***ELSEIF***

```
if (logical_exp1) then
   instrucciones caso 1
elseif (logical_exp2) then
   instrucciones caso 2
else
   instrucciones caso 3
endif
```

- En **casos simples**, se usa **sólo *IF***

```
if (A.GT.B) write(*,*)'A es mayor que B'
```

Ejercicios: Decisiones

Ejercicio: Escriba un programa que toma un número del teclado y comprueba su tamaño. Si el número es menor que 100 el programa debe de escribir en pantalla ***Es pequeño***, y si el mayor o igual a 100 debe de escribir ***Es grande***. Coloque todo en un bucle de forma que se puedan entrar varios números uno tras otro.

Ejercicio: Reescriba el programa que resolvía la ecuación de segundo grado. Ahora, antes de calcular la raíz cuadrada, compruebe si el discriminante es negativo y, si lo es, escribir en pantalla ***No hay raíces reales***.

Ejercicio: Escriba un programa que lea desde el teclado 10 números y los memorice en un vector. **Ordénelos** de forma que el más pequeño sea el primero de la lista. Un **algoritmo** sencillo (aunque no eficiente) de ordenación consiste en comparar cada número con todos los demás y ver cuántos son más pequeños: si hay n de éstos, el orden del número es n+1.

Funciones y subrutinas

- Las funciones y subrutinas son subprogramas de FORTRAN. La mayor parte de los problemas son tan complejos que resulta conveniente trocearlos en pequeños problemas que se resuelven en cada función o subrutina.
- Una ***función*** toma un conjunto de valores como argumentos, realiza algún cálculo y **devuelve un único resultado**. Hay algunas funciones ya escritas en FORTRAN y que pueden ser usadas por el programador directamente, son las llamadas **funciones intrínsecas**. La mayor parte son funciones matemáticas que se utilizan de la siguiente forma: ***answer = functionname(argl, arg2,…)***

WRITE(*,*) ABS(T)	El compilador evalúa el valor absoluto de T y lo escribe.
Y = SIN(X) + 45	El compilador calcula el valor del seno de x, añade 45 y luego pone el resultado en la variable y.
M=MAX(a,b,c,d)	El compilador averigua el valor máximo de entre a,b,c y d y lo asigna a la variable M.
C=SQRT(a**2 + b**2)	El compilador evalúa, a**2+b**2, envía el resultado a la función raiz cuadrada y pone el resultado en la variable C.

Funciones externas

- Las funciones internas de FORTRAN son pocas y el programador puede **definir sus propias funciones**. Una vez definidas se utilizan de igual forma que las funciones internas.
- Hay unas pocas **reglas** para escribir una función externa:
  - Las funciones y cualquier otro subprograma se coloca **después de la instrucción END** del programa principal (o en **archivo aparte**)
  - Comienzan con una linea que incluye el **tipo** de valor de la función que devolverá, el **nombre** de la función y la lista de **argumentos** que usará como inputs.
  - Todas las variables que usa la función, incluidos los argumentos, deben de ser **declarados** inmediatamente después de la línea 2. El nombre de la función NO se declara dentro de la función.
  - El nombre de la función se utiliza como la variable de asignación del resultado de las operaciones del subprograma. Ese valor será el que el compilador devolverá al programa principal.
  - Una función debe de finalizar con un ***RETURN*** y un ***END***.

Ejemplo: cálculo de la media

Ejercicio: Escriba un programa que calcule la media de un número arbitrario de valores. Usar para ello vectores con dimensión dinámica y una función.

```
      PROGRAM FUNDEM
C     Declarations for main program
      REAL*8*8 A,B,C
      REAL*8*8 AV, AVSQ1, AVSQ2
      REAL*8*8 AVRAGE
C     Enter the data
      DATA A,B,C/5.d0,2.d0,3.d0/

C     Calculate the average of the numbers
      AV = AVRAGE(A,B,C)
      AVSQ1 = AVRAGE(A,B,C)**2
      AVSQ2 = AVRAGE(A**2,B**2,C**2)

      WRITE(*,*)'Statistical Analysis'
      WRITE(*,*)'The average of the numbers is: ',AV
      WRITE(*,*)'The average squared of the numbers: ',AVSQl
      WRITE(*,*)'The average of the squares is: ', AVSQ2
      STOP
      END

     REAL*8 FUNCTION AVRAGE(X,Y,Z)
     REAL*8*8 X,Y,Z,SUM
     SUM = X + Y + Z
     AVRAGE = SUM /3.0
     RETURN
     END
```

Subrutinas

- Subrutinas actúan de igual forma que las funciones pero **pueden devolver varios resultados a la vez**. Una subrutina no puede asignarse a una variable pues en si misma no tiene un valor asociado.
- En el programa principal una subrutina se activa con la instrucción ***CALL*** que incluye el nombre de la subrutina seguida por una **lista de inputs y de outputs**.
- No es necesario declarar el nombre de las subrutinas en el programa principal.
- Las subrutinas comienzan por una línea que incluye la palabra ***SUBROUTINE***, el nombre de la subrutina y los argumentos.
- Todos los argumentos de la subrutina deben de ser declarados en la misma.
- Una subrutina acaba con un ***RETURN*** y un ***END***.

Ejemplo: cálculo de la media

```
PROGRAM SUBDEM
REAL*8 A,B,C,SUM,SUMSQ
CALL INPUT(A,B,C)
CALL CALC(A,B,C,SUM,SUMSQ)
CALL OUTPUT(SUM,SUMSQ)
END

SUBROUTINE INPUT(X, Y, Z)
REAL*8 X,Y,Z
WRITE(*,*)'ENTER THREE NUMBERS => '
READ *,X,Y,Z
RETURN
END

SUBROUTINE CALC(A,B,C, SUM,SUMSQ)
REAL*8 A,B,C,SUM,SUMSQ
SUM = A + B + C
SUMSQ = SUM **2
RETURN
END

SUBROUTINE OUTPUT(SUM,SUMSQ)
REAL*8 SUM, SUMSQ
WRITE(*,*)'The sum of the numbers you entered are: ',SUM
WRITE(*,*)'And the square of the sum is:',SUMSQ
RETURN
END
```

Ejercicios: funciones y subrutinas

Ejercicio: Calcule la función de distribución radial (FDR) para los orbitales 1s, 2s y 2p y normalice el resultado usando el método de Simpson. La FDR es el valor del modulo al cuadrado de la función de ondas multiplicado por $4\pi r^2$. Las funciones de ondas son:

$$
\begin{aligned}
1s &\rightarrow \exp(-r/2) \\
2s &\rightarrow \frac{(2-r)}{\sqrt{32}} \exp(-r/2) \\
2p &\rightarrow \frac{(6-6r+r^2)}{\sqrt{972}} \exp(-r/2)
\end{aligned}
$$

Formato y manipulación de archivos

- Introducir datos en el programa usando el teclado e imprimirlos en la pantalla se puede realizar usando las instrucciones ***READ*** y ***WRITE***. Sin embargo se tiene poco control sobre el formato de los mismos.
- La información sobre el formato de entrada y salida se realiza por la instrucción ***FORMAT***.
- ***FORMAT*** contiene los siguientes descriptores de edición:

Ejemplos de formato

Valor	Descriptor
2099	i4
-72.81	f6.2
0.186E+06	e10.3
Cup of Tea	a10

Tipo de Variable	Descriptor de edición	Especificaciones
Entero	Iw	**w** es el número de dígitos del entero
Real	Fw.d	**w** es el número total de dígitos incluyendo el signo, punto decimal y número de cifras decimales. **d** el número de cifras decimales mostradas
Caracter	Aw	**w** es el número de caracteres
Espacios	nX	**n** es el número de espacios
Exponencial	Ew.d	**d** digitos de la mantisa **w** es el tamaño total

La instrucción FORMAT

- La instrucción FORMAT obedece la siguiente sintaxis

  etiqueta FORMAT(c,ed1,ed2,'some text',ed3...)

- donde los diferentes descriptores son

etiqueta	Es un número de identificación que asocia el formato a una instrucción WRITE o READ.
edl, ed2, ed3	Son descriptores de edición separados por comas
text	Mensages que son rodeados por comillas simples
c	Es el control de carro para la salida y sólo puede ser: lx: mover la salida hasta la siguiente linea O: doble espaciado +: mover al comienzo de la linea (sobreescribiendo) l: comenzar una nueva página

Manipulación de archivos

- Para escribir con o sin formato en un archivo o leer desde un archivo debemos utilizar las instrucciones ***READ*** y ***WRITE***.

READ/WRITE (control_list) variable_list

donde ***control_list*** pueden ser algunos de las siguientes instrucciones:

UNIT = identificador de la unidad

FMT = identificador del formato

END = etiqueta ERR = etiqueta

- **Ejemplo**:

```
READ(UNIT=1, FMT=10) A,B,C
READ(1,10) A,B,C

WRITE(UNIT=2, FMT=20) X,Y,Z
WRITE(2,20) X,Y,Z
```

Manipulación de archivos

- Cada lista de control debe de contener un **identificador de unidad**. Éste identifica el archivo que debe de accederse. Si el identificador se coloca en primer lugar se puede omitir ***UNIT=***. Ej: ***WRITE(2,FMT=20)X,Y,Z***
- Se puede elegir **cualquier número** para un identificador **excepto** dos que están reservados y tienen significados predefinidos: 5 representa que el input será por el teclado y 6 representa que la escritura se realizará en la pantalla. Así ***READ(5,*)a,b,c*** es equivalente a ***READ(*,*)a,b,c***. De igual forma ***WRITE(6,*)a,b,c*** es equivalente a ***WRITE(*,*)a,b,c***.
- El **identificador del formato** identifica cómo se organizaran los datos por referencia a una instrucción ***FORMAT*** o por una asterisco. Ejemplos:

READ(UNIT=l, FMT = 10)A,B,C (La instrucción FORMAT está en la etiqueta 10)

READ(UNIT=1, FMT = *)A,B,C (No hay formato)

- Si el identificador del formato es el segundo en la lista de control entonces se puede omitir ***FMT=***. Ejemplo: READ(1,10)A,B,C

Condición de fin y/o error

- **Condición FIN del archivo**: Sólo una condición *end-of-file* puede aparecer en la lista de control. Cuando la instrucción READ llega al final del archivo de datos sin que ocurra ningún error, no se leen más datos y el programa continua en el lugar donde está la etiqueta que acompaña a END=.
- **Condición de error**: Si algún error ocurre durante el input/output y hay una condición de error, el input/output se detiene y el programa salta a la etiqueta que es especificada por ERR=.

```
80  READ(l,*,END=99)A,B,C
c   do something with A, B, and C
    GOTO 80
99  WRITE(*,*)'ALL DATA READ'
```

```
    READ(l,*,ERR=100,END=200)A,B,C
100 WRITE(*,*)'ERROR ON READING, STOPPING PROGRAM RUN'
    STOP
c   or else continue with the rest of the program
200 CONTINUE
```

Abrir y cerrar archivos

- La instrucción ***OPEN*** conecta el archivo y define sus especificaciones.

  OPEN(UNIT = number, ERR = label, FILE = 'nombre', STATUS = 'character')

- ***ERR***: Si una condición de error ocurre cuando el archivo va a abrirse el programa salta a la etiqueta.
- ***FILE***: Especifica el nombre del archivo que se va a abrir.
- ***STATUS***: Puede ser ***OLD*** (el archivo ya existe), ***NEW*** (el archivo no existe y se crea), o ***UNKNOWN*** (el archivo puede o no existir)
- La instrucción ***CLOSE*** desconecta el archivo: ***CLOSE([UNIT=] number)***

```
      open(10,file='datos.dat',err=100,status='unknown')
c     operamos con el archivo
      close(10)
```

Problemas

Ejercicio: Ecuación de van der Waals: La ecuación de estado de van der Waals es una extensión de la ecuación para los gases ideales que tiene en cuenta algunos aspectos de la desviación de los gases reales con respecto a su comportamiento ideal. La ecuación es

$$(p + \frac{an^2}{V^2})(V - bn) = nRT$$

en la que n es el número de moles del gas, p es su presión, T es la temperatura, V el volumen, R la constante de los gases (8.414) y a y b son parámetros fenomenológicos que dependen de cada gas particular. Escriba un programa que imprima una tabla con varias p para al menos 10 volúmenes entre V1 y V2. Las variables a, b, T, V1 y V2 deben de ser entradas por el teclado y la salida debe de ir a un archivo. Use para el hidrógeno a=0.0247, b=26.6x10^{-6}.

Ejercicio: Estime el número π con métodos Monte Carlo (ver notas pdf)

Ejercicio: Calcule el mayor autovalor de una matriz simétrica usando el método de las potencias (ver notas pdf)