Fortran 90
Seminario de Computación 2009

Estructura de un programa Fortran y tipos de variables.
Funciones matemáticas intrínsecas.

Conjunto de caracteres (el “alfabeto” Fortran)

Caracteres alfanuméricos:

Letras: A, B, C, ..., Z, a, b, c, ..., z
Dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
A diferencia de otros lenguajes como C o Matlab, en el código de un programa Fortran A y a son la misma cosa (no hay distinción entre mayúsculas y minúsculas)

Caracteres especiales:

<blanco> = + - * / ( ) , . ‘ : ! “ % & ; < > ? $ _

Palabras reservadas

Combinaciones de caracteres con significado predefinido para el compilador: *CHARACTER, DO, END, GOTO, IF, INTEGER, MATMUL, MINVAL, OPEN, PRINT, PROGRAM, READ, REAL, STOP, WRITE,...* (es indistinto si estas palabras se escriben en mayúscula o minúscula).

Identificadores (nombres)

- Nombre utilizado por el programador para referirse a un objeto dentro del programa (nombres de variables, el nombre del programa, etc)
- F90: Máximo de 31 caracteres, el primero obligatoriamente alfabético
- Ejemplos de identificadores válidos
  - ✓ aAaA, num_manzanas, r2d2,
  - ✓ Rosa, ROSA, rosa (son equivalentes para el compilador porque este no distingue mayúsculas de minúsculas)
- Ejemplos de identificadores erróneos
  - ✓ _variable (porque comienza con guión bajo),
  - ✓ 2dedos (porque comienza con un número),
  - ✓ Carácter_ilegal (el acento no está permitido)

Estructura de una sentencia fortran:

Un programa Fortran está compuesto por “sentencias” que estan pensadas para cumplir con el objetivo del programa.

- **Sentencias ejecutables: Describen una acción que va a ser realizada por el programa, por ejemplo una suma, una división de dos números, etc.**
- **Sentencias no ejecutables: Son sentencias necesarias para que el programa sea interpretado y funcione correctamente. Por ejemplo proveen información sobre algún elemento del programa.**

Ejemplo de una sentencia de un programa fortran, el valor de la variable C será la suma del valor de A + el valor de B

C=A+B

Las sentencias de un código fuente se escriben en un archivo de texto convencional (generado por ejemplo con el bloc de notas).

Cada sentencia debe comenzar en un renglón aparte o estar separadas por el carácter **;**. Pueden comenzar en cualquier columna del renglón y pueden tener un largo máximo de **132 characteres**. Si alguna sentencia excede ese largo, se debe colocar un **&** al final de la misma para indicar que será continuada en el renglón de abajo.

Las sentencias pueden comenzar con un número que haga de etiqueta (**“label”**) que permita identificarla dentro del programa, pero no es necesario colocarlo en todas las sentencias.

20 C=A+B

Fortran 90 vs Fortran 77:

Fortran 77 tiene reglas más estrictas en cuanto a como escribir las sentencias. Las mismas deben empezar en la columna 7 y no se podía tener una extensión mayor a 74 caracteres, muchas de estas restricciones son heredadas de cuando cada sentencia se escribía en una tarjeta perforada y no tienen sentido actualmente. Nosotros en este curso vamos a utilizar el formato de Fortran 90 o formato libre ("Free Format"). El formato de Fortran 77 se conoce como formato fijo ("Fixed Format").

Comentarios:

En un programa Fortran además de aparecer sentencias también pueden aparecer comentarios. Los comentarios permiten incluir información que ayude a las personas a interpretar la finalidad de algunas sentencias o eventualmente de parte o de todo el programa. Los comentarios son ignorados a la hora de ejecutar el programa.

En Fortran 90 todo lo que se encuentre a la derecha del carácter ! es tratado como un comentario. El carácter ! puede ir al inicio de una línea o a continuación de una sentencia.

Ejemplos de comentarios:

! Este es un comentario

C=A+B ! Este es un comentario a continuación de una sentencia.

En Fortran 77 todas las lineas que tienen una C en la primera columna también son comentarios.

Estructura de un programa Fortran 90

```
PROGRAM mi_primer_programa
! Objetivo: Ilustrar la estructura básica de un programa Fortran.
!Declaracion de variables.
INTEGER :: a , b , c
!Asignamos valores a las variables.
a=1
b=2
!Multiplico a y b y asigno el resultado a la variable c.
c = a * b
!Escribo el resultado por pantalla.
WRITE(*,*) c
!Termino el programa.
STOP
END PROGRAM mi_primer_programa
```

Sección de declaración (no ejecutable)

Sección de ejecución, donde se hacen los cálculos.

Final del programa.

NOTA: Es saludable adoptar un estilo de programación, por ejemplo siempre escribir en mayúscula los nombres de las palabras claves como WRITE, PROGRAM, etc y también para las constantes. Una vez adoptado seguirlo para mantener la prolijidad en los programas.

Compilar un programa Fortran.

La máquina no puede interpretar directamente las sentencias escritas en el lenguaje Fortran (lo que vamos a llamar código fuente o “suorce code”). Antes de que puedan ser ejecutadas por la máquina esas sentencias deben ser “traducidas” al lenguaje de máquina. La tarea la lleva a cabo un programa que se denomina compilador. Además se vincula (“linkea”) al programa con todos aquellos recursos del sistema que va a necesitar para su correcta ejecución.

En nuestro caso vamos a utilizar el **Digital Fortran** para compilar nuestros programa. Veamos el siguiente ejemplo:

Escribir el codigo de **mi_primer_programa** en un archivo de texto al que nombraremos **miprimerprograma.f90** (el nombre del archivo no tiene por que coincidir con el del programa, aunque si es muy recomendable utilizar la extensión f90 en todos los códigos fuente escritos en **Fortran90**.

Compilar el código fuente, para eso vamos a abrir una consola de Windows (**inicio > todos los programas > accesorios > símbolo de sistema**), en dicha consola vamos a ubicarnos en el directorio donde generamos el archivo **miprimerprograma.f90**

En dicho directorio tipear el siguiente comando:

df miprimerprograma.f90

Al hacer esto se compilará el código fuente y se generará un archivo ejecutable llamado **miprimerprograma.exe**

Para ejecutar dicho programa solo tenemos que ingresar el nombre del ejecutable en la consola.

miprimerprograma.exe

Variables y constantes en un programa Fortran:

En Fortran 90 podemos tener 5 tipos diferentes de variables y constantes: **INTEGER , REAL , CHARACTER , COMPLEX , LOGICAL** . En esta primera clase nos vamos a ocupar de los primeros 3 tipos. A diferencia de otros lenguajes como el matlab, Fortran requiere que el tipo de cada variable sea declarado al comienzo del programa.

En los programa Fortran, es posible no declarar algunas variables y dejar que el compilador asuma su tipo a partir de la primera letra de su nombre. Esto es muy poco recomendable por lo que vamos a utilizar la modalidad en la que todas las variables son declaradas y le vamos a indicar al compilador que vamos a trabajar de esta manera utilizando la sentencia

IMPLICIT NONE

Esta sentencia se ubica luego del nombre del programa y antes de empezar con la declaración de variables.

En Fortran 90 se pueden definir más tipos de variables utilizando combinaciones de los 5 tipos fundamentales (tipos de datos derivados o “derived data types”). Esto se discutirá más adelante en la materia.

Variables y constantes INTEGER (enteros):

Las variables de tipo **integer** son enteros positivos o negativos. Ejemplos de variables o constantes integer podrían ser: **-10 , 1000**, etc. No pueden tener punto decimal, porque si lo tuvieran estarían indicando que no son enteros.

El máximo o mínimo entero que es posible utilizar en la ejecución de un programa Fortran depende de la máquina que estemos utilizando (actualmente lo más usual es encontrar máquinas de 32 o 64 bits).

El máximo y mínimo valor entero que podemos generar utilizando una máquina como función del número de bits con la que trabaja es:

$$\min\_integer = -2**n-1$$

$$\max\_integer = 2**(n-1)-1$$

Para una máquina de 32 bits los enteros pueden ir de **-2,147,483,648** hasta **2,147,483,647**

Si durante la ejecución del programa alguna operación produce un entero más grande que estos valores se produce un error de ejecución. **("overflow")**

Ejemplos de declaración de variables **integer**:

```
INTEGER        ::    A              !Declaramos una sola variable integer A

INTEGER        ::    A , B , C      !Podemos declarar varias variables del mismo tipo en
! la misma sentencia. O podemos declarar cada uno en una sentencia por separado.
```

Variables y constantes REAL (de punto flotante o “floating point”):

Permiten representar números con cifras decimales y de mayor tamaño que los números enteros. Las variables y constantes siempre deben incluir el punto decimal. Ejemplos: **1.0 , -2002.2134 , 1.23e20 , 2.3e-10** (la notación científica es aceptada).

Los valores de una variable real se almacenan en la máquina en dos partes, la mantisa y el exponente. La **mantisa** es un número entre 0.0 y 1.0, mientras que el **exponente** puede tomar valores enteros (negativos y positivos). Dada la cantidad de bits de la máquina, parte de esos bits serán destinados a almacenar la mantisa y parte a almacenar el exponente.

Por ejemplo para máquinas que siguen el **Estándar IEEE754**, la mantisa se almacena en **24 bits (7 a 8 cifras decimales)** y el exponente en **8 bits** para máquinas de **32 bits**. Esto nos permite tener números reales entre **10e-38 y 10e38**.

Para una máquina de **64 bits** en cambio tenemos **53 bits** asignados a la mantisa (15 a 16 cifras decimales) y **11** al exponente con lo cual podemos representar números entre **10e-308 y 10e308**.

En principio la precisión de las variables reales dentro de un programa **Fortran** va a estar dada por la máquina en la que estemos trabajando, pero de ser necesario para el correcto funcionamiento del programa podemos especificar la precisión deseada para las variables de nuestro programa independientemente de la máquina utilizando la función “**KIND**” que no será discutida en este curso.

La cantidad de bits asignados a la mantisa determinan la precisión del número real (cuan cerca van a estar los números reales que puede almacenar la computadora), mientras que la cantidad de bits asignados al exponente controlan el rango (cual es la distancia entre el mayor y el menor número real que puede ser definido).

Como la computadora no puede almacenar los números reales con “infinita” precisión, entonces solo podemos obtener el número real más cercano al resultado de una operación dada. En ese punto se produce lo que se llama el error de truncado o redondeo.

Ejemplos de declaraciones

```
REAL        ::   A , B , C        ! En este ejemplo A, B y C son variables reales.
```

Variables y constantes CHARACTER (alfanuméricas o texto):

Esta variable consiste en una cadena de caracteres **colocados entre comillas simples o dobles**. El tamaño mínimo es de un carácter y el máximo varía de compilador en compilador.

Todos los caracteres válidos para una computadora son válidos para formar parte de una variable carácter (esto incluye a los que no son válidos para escribir un programa Fortran).

Ejemplos: **“casa” , “esto es una prueba” , “[]{}” , “3.14159”** (notar que el último va a ser tomado como una cadena de caracteres y no como un número en la ejecución del programa).

Cuando declaramos una variable de tipo character podemos incluir el largo de la misma.

Ejemplos

```
CHARACTER       ::    A     !por defecto el largo va a ser de 1.

CHARACTER  (len=20)    ::  B    !B es una variable character de 20 caracteres.
```

En Fortran 77 la última declaración se escribía así (es posible que se encuentren con este tipo de sintaxis).

```
CHARACTER*20   B
```

Esta última forma fue declarada obsoleta en Fortran 95 por lo cual no se recomienda su utilización en nuevos programas ya que es un candidato a ser eliminado en futuras actualizaciones del lenguaje Fortran.

Constantes y variables:

Las constantes como su nombre lo indica no pueden variar a lo largo del programa (cualquier sentencia que intente contradecir esto producirá un error al momento de la compilación). Las variables pueden cambiar su valor a lo largo del programa.

Ejemplo de declaración de una constante de tipo **INTEGER**

```
INTEGER, PARAMETER   ::     J = 3    !Cantidad de pabellones en Cdad. Universitaria.
```

La inclusión de la palabra **PARAMETER** indica que el valor debe mantenerse constante en el programa.

En forma análoga lo podemos hacer para una variable **REAL**

```
REAL, PARAMETER   ::  PI=3.14159     !Valor de PI que voy a utilizar a lo largo del programa.
```

El uso de constantes que se definen al principio del programa es muy recomendable y permite darle mayor consistencia a los cálculos dentro del mismo (imaginemos que tenemos que usar el valor de PI varias veces y usemos diferentes valores en diferentes partes del programa… esto resultaría en cierta inconsistencia).

A las variables **REAL** e **INTEGER** también se les puede asignar un valor al momento de la declaración, pero este valor puede ser reemplazado por otro en la sección de sentencias ejecutables, mientras que en el caso de las constantes no.

Ejemplo

```
REAL       ::    TEMPERATURA=0.0     !Asigno un valor inicial a la variable temperatura.
INTEGER  ::    DIA=1                        !Asigno un valor inicial a la variable dia.
```

Asignación:

Para reemplazar el valor de una variable en un programa Fortran utilizamos la sentencia “asignacion” (=)

Ejemplo

INTEGER :: A

A= 20

En este caso la segunda sentencia asigna el valor 20 a la variable A. El igual no debe ser confundido como una igualdad en el sentido matemático estricto.

Ejemplo

INTEGER :: A

A = 20

A = A + 2

En este caso, luego de ejecutar la tercera sentencia el valor de A será 22. La tercera sentencia no tiene sentido desde el punto de vista matemático, pero es una asignación perfectamente válida en Fortran.

El ejemplo anterior también se puede escribir como

INTEGER :: A=20

A=A+2

En este caso asignamos un valor a A al momento de la declaración de la variable.

Aritmética:

Operadores aritméticos

+ - suma y resta

***** / multiplicación y división

****** potencia

Algunas reglas básicas para escribir operaciones aritméticas en Fortran

• **No se pueden colocar dos operadores juntos**

a * - b (esto produce un error de compilacíon)

a * (- b) (esto es la forma correcta)

• **La multiplicación implicita es ilegal.**

a(b) (esto produce un error de compilación)

a * b (esto es la forma correcta)

• **Los paréntesis pueden (y deben) ser usados para controlar el orden de calculo de las operaciones. Todas las operaciones en el interior de los paréntesis van a ser evaluadas primero.**

2 ** ((8 + 2) / 5) = 2 ** (10 / 5)

= 2 ** 2

= 4

Es importante controlar los paréntesis en forma cuidadosa, un paréntesis mal colocado puede producir que la expresión sea evaluada en el orden incorrecto y que el resultado obtenido no sea el deseado

Aritmética con números enteros:

Es el caso en el que las operaciones se realizan exclusivamente con números enteros. La aritmética de números enteros da como resultado un número entero.

Esto puede ser particularmente importante cuando se realizan divisiones, cuando el resultado de la división de 2 enteros no es entero, la computadora lo trunca automáticamente al entero inferior más próximo.

Ejemplos: En aritmética entera ocurre lo siguiente:

3 / 4 = 0

4 / 4 = 1

5 / 4 = 1

6 / 4 = 1

Por este motivo nunca se deben utilizar variables enteras para almacenar valores de variables que en el mundo real son continuas. Siempre tener mucho cuidado de cómo se definen las expresiones y como se utilizan los enteros en dichas expresiones.

Aritmética con números reales:

Es el caso en el que las operaciones se realizan exclusivamente con números reales. El resultado será además un número real.

3. / **4. = 0.75**

4. / **4. = 1.**

5. / **4. = 1.25**

6. / **4. = 1.5**

Es importante recordar que debido al error de truncado, el resultado de algunas operaciones no va a ser exacto. Por ejemplo en teoría, **1.** / **3. = 0.333333…….** sin embargo una computadora no puede representar infinitas cifras decimales, y solo representará algunas (que depende de la precisión).

Por este motivo hay que tener cuidado, ya que algunas operaciones que en teoría deberían arrojar cierto resultado en la práctica no lo hacen.

Ejemplos

En algunas computadoras **3. * (1.** / **3.)** no necesariamente da **1.** pero **2. * (1.** / **2.) = 1.**

Aritmética mixta:

En muchas oportunidades vamos a necesitar escribir expresiones en las que participen números reales y enteros.

Ejemplo

INTEGER :: A = 4

REAL :: B= 2.0 , C

C= B + 1 / A

En este caso, el resultado que esperamos es 2.25 , pero el que obtenemos es 2.0 porque 1 / A al ser una operación entre enteros nos da 0.

Ejemplo

C = A * B nos da un número REAL.

Las reglas de la aritmética mixta son complicadas y pueden producir resultados inesperados, por eso es mejor asegurarse que siempre estemos haciendo operaciones entre números reales. Para eso podemos utilizar la función de Fortran **REAL**

REAL convierte un valor entero en un valor real.

C = B + 1.0 / REAL(A) !Ejemplo en la que utilizamos el real que corresponde al valor entero de A al momento de hacer la operación. (Esto no modifica el valor de A, A sigue siendo un entero).

El resultado es 2.25 como esperábamos.

Otra posibilidad es **C = B + 1.0 / (1.0 * A)** ! El resultado de 1.0 * A es un número real tanto si A es real como si A es entero.

```
PROGRAM INT_REAL_ARITMETICS
IMPLICIT NONE

REAL :: A, B
INTEGER :: I, J

A=3.
B=4.

WRITE(*,*) A**(-2)
WRITE(*,*) A/B
WRITE(*,*) A*B

I=3
J=4

WRITE(*,*) I**(-2)
WRITE(*,*) I/J
WRITE(*,*) I*J

I=3.+0.25

WRITE(*,*)I

A=I/J
B=REAL(I)/REAL(J)

WRITE(*,*)A,B
STOP
END PROGRAM INT_REAL_ARITMETICS
```

Ejemplo de aritmética REAL, INTEGER y MIXTA.

Entrada y salida de datos básica (input / output o I/O):

Hasta ahora vimos como modificar el valor de las variables mediante una asignación. Pero como hacemos si queremos realizar una determinada operación pero variando el valor inicial de alguna variable, sin tener que modificar el código fuente y recompilar el programa cada vez.

Para eso existen las funciones de **I/O** que le permiten al programa acceder a información a través de distintos dispositivos (por ejemplo datos ingresados por el teclado, datos almacenados en el disco, etc). A su vez permiten mostrar los resultados obtenidos ya sea por pantalla o guardando la información en el disco.

Ejemplo, programa para calcular el cuadrado de un número que será ingresado por pantalla durante la ejecución del programa.

```
PROGRAM cuadrado

IMPLICIT NONE

REAL  :: numero , cuadrado_numero

WRITE(*,*)" Ingrese un número "  !Se muestra "ingrese un número" en la pantalla.

READ(*,*) numero                 !El número ingresado por el teclado es asignado a la variable A.

cuadrado_numero= numero ** 2

WRITE(*,*)"El cuadrado del número es: ",cuadrado_numero   !Muestra el mensaje "el cuadrado

! Del número es: " seguido del valor calculado.

STOP

END PROGRAM cuadrado
```

La sentencia **READ** permite ingresar datos por teclado al programa, esta sentencia nos permite asignar los valores ingresados a una o más variables.

READ(*,*) A, B, C

En este caso **A, B y C** pueden ser modificados por los valores ingresados. Para ingresar los datos correctamente se deben ingresar 3 valores del mismo tipo de las variables **A, B y C**, en el mismo orden como aparecen en la sentencia **READ**.

Para ingresar varios valores los mismos deberán estar separados por “,” o por un espacio en blanco.

La sentencia **WRITE** permite mostrar por pantalla el valor de una o de varias variables.

WRITE(*,*) A, B, C

WRITE(*,*) A*B !También podemos pedir que muestre el resultado de una expresión.

En este caso se mostrará el valor de las variables **A, B y C** separados por uno (o varios) espacios.

En el WRITE se pueden incluir además cadenas de caracteres ubicadas entre comillas y separadas por comas de las variables.

WRITE(*,*) “El valor de la variable A es : “, A

También se pueden colocar solo cadenas de caracteres.

WRITE(*,*)”Este es un programa Fortran”

Nota: A diferencia de lenguajes como el Matlab que muestran todas las operaciones por pantalla a no ser que se le indique lo contrario, el Fortran solo muestra por pantalla los valores de las variables cuando se le indica explícitamente.

Funciones intrínsecas: Una función es una expresión que acepta uno o más valores de entrada y calcula un resultado a partir de ellos.

Los valores de entrada son denominan **"argumentos"** (arguments) y aparecen entre paréntesis a continuación del nombre de la función.

Ejemplo: La función **LOG** calcula el logaritmo en base e de un número real.

```
PROGRAM ejemplo

REAL  :: numero , log_numero

numero= 20.

log_numero = LOG ( numero )

WRITE(*,*) "El logaritmo de ",numero," es ",log_numero

STOP

END PROGRAM ejemplo
```

Las funciones pueden aparecer en cualquier expresión (al igual que si fueran una variable o una constante) pero nunca pueden aparecer en el lado izquierdo de una asignación.

Ejemplo: (A y B variables reales)

```
A = ( 1. + 10. * LOG ( B ) ) / ( 1. – LOG ( B ) )
```

Funciones intrínsecas:

Una lista completa de funciones intrínsecas se puede encontrar en:
http://www.nsc.liu.se/~boein/f77to90/a5.html#section7

Algunos ejemplos:

Funciones que toman argumentos **REAL** y devuelven resultados **REAL**.

SIN(X) , COS(X), LOG(X) , LOG10(X) , EXP(X) , TAN(X) , ATAN(X) , ASIN(X) , ACOS(X)

Funciones que toman argumentos **REAL** o **INTEGER** y que devuelven un resultado del mismo tipo que el argumento.

ABS(X)

Funciones que utilizan más de un argumento.

MAX(X,Y) !Devuelve el máximo entre X e Y.

Funciones que toman un argumento **REAL** y devuelven un **INTEGER**

INT(X) , AINT(X)

Funciones que toman un argumento **INTEGER** y devuelven un **REAL**

REAL(X)

Funciones que toman un argumento **CHARACTER** y devuelven un **INTEGER**

LEN(X) ! Me da la longitud de la variable CHARACTER X.

Como estuvimos viendo la precisión de las variables REAL depende de la máquina que estemos utilizando.

Existen funciones que nos dan información sobre la precisión de una determinada variable:

```
PROGRAM que_precision
IMPLICIT NONE
REAL  :: X
!Ejemplo de funciones que me dan informacion sobre la precision de la maquina.
WRITE(*,*)DIGITS(X)              !el numero de bits en la mantisa.
WRITE(*,*)MAXEXPONENT(X)   !El máximo exponente
WRITE(*,*)MINEXPONENT(X)   !El mínimo exponente
WRITE(*,*)HUGE(X)                !El maximo numero positivo.
WRITE(*,*)TINY(X)                !El número positivo más pequeño.
WRITE(*,*)RADIX(X)               !La base de las operaciones de punto flotante.
WRITE(*,*)RANGE(X)               !El rango del exponente.
WRITE(*,*)EPSILON(X)             !El número más pequeño tal que si se lo sumamos a 1 el
resultado es distinto de 1.
WRITE(*,*)PRECISION(X)           !La precisión decimal.
STOP
END PROGRAM que_precision
```

Este es el mismo ejemplo que antes, solo que agregamos **(KIND=8)** en la declaración de la variable X, esto hace que la variable X tenga una precisión de 8 bytes **(64 bits).** Con lo cual se van a modificar los resultados de las funciones.

Importante el uso del número 8 para designar a variables de 64 bits puede depender de la máquina o el compilador que se esté utilizando. Es por eso que para hacer programas en donde la precisión sea independiente de la máquina es necesario utilizar funciones más complejas que no vamos a ver en el curso.

```
PROGRAM que_precision
IMPLICIT NONE
REAL  (KIND=8) :: X              !Ahora la variable X es de 64 bits.
```

También podemos probar con la siguiente declaración

```
REAL (KIND=16) :: X              !Ahora la variable X es de 128 bits (loco, no?)
```

Algunos consejos prácticos:

• Adopten un "**estilo**" de programación en cuanto al uso de mayúsculas y minúsculas.

• Utilicen abundantes **comentarios** en los programas. Piensen que ese mismo programa puede tener que ser utilizado o modificado por otra persona en el futuro que tiene que ser capaz de entender en forma lo más clara posible lo que el programa está haciendo.

• Utilicen nombres reconocibles para las variables y constantes que permitan identificar el significado de las mismas. Por ejemplo si tengo que utilizar la constante de la gravedad puedo llamarla **G**, **G_CONST** , **W984_334 o PI**. Los 4 nombres son válidos, pero los primeros 2 me ayudan a interpretar que es lo que se está calculando, el tercero no me dice nada y el cuarto es para confundir al enemigo.

• Tengan cuidado con los cálculos aritméticos que involucren variables **INTEGER** y **REAL**, en la medida de lo posible hagan siempre conversiones explícitas (usando por ejemplo la función **REAL**) de forma tal que todos los elementos involucrados en una expresión sean del tipo **REAL**.

• Tengan especial cuidado con el **uso correcto de los paréntesis**, es una importante fuente de errores sobre todo en expresiones con mayor grado de complejidad.

•Usen siempre la sentencia **IMPLICIT NONE**, declarar todas las variables que vamos a utilizar al comienzo de un programa puede implicar más trabajo al principio, pero esta forma hace más fácil que el compilador detecte errores de tipeo en el código fuente (los cuales son muy comunes)

•Si bien **Fortran 90** continúa siendo compatible con algunas formas y sentencias del **Fortran 77**, algunas de estas han sido declaradas obsoletas lo que implica que pueden ser eliminadas en futuras versiones del lenguaje. Es por eso que deben ser evitadas en la medida de los posible.


FIN