# PRACTICA 1a

# Elementos de programación Fortran: Estructura general y tipos de datos simples.

*Dios es real.*
*A menos que sea declarado entero.*

**Estructura general.** Un programa en Fortran consiste de un programa principal (*main*, en inglés) y posiblemente varios subprogramas. Por el momento asumiremos que el código consta sólo de un programa principal. La estructura del programa principal es

```
PROGRAM nombre
declaraciones de tipo
sentencias ejecutables
END PROGRAM nombre
```

Cada línea del programa forma parte de una *sentencia* que describe una instrucción a ser llevada a cabo y tales sentencias se siguen en el orden que están escritas. En general, las sentencias se clasifican en *ejecutables* (las cuales realizan acciones concretas como ser cálculos aritméticos o entrada y salida de datos) y *no ejecutables* (las cuales proporcionan información sin requerir en sí ningún cómputo, como ser las declaraciones de tipos de datos involucrados en el programa). La sentencia (no ejecutable) `PROGRAM` especifica el comienzo del programa principal. La misma está seguida de un nombre identificatorio para el programa el cual no necesita ser el mismo que el de su código fuente ni el del programa ejecutable que genera el compilador. La sentencia (no ejecutable) `END PROGRAM` indica el final lógico del programa principal.

☞ Una sentencia Fortran sobre una línea puede extenderse hasta 132 caracteres. Si la sentencia es más larga ésta puede continuarse en la siguiente linea insertando al final de la línea actual el carácter &. De esta manera una sentencia puede ser continuada sobre 40 líneas, en Fortran 95, o 256 líneas en Fortran 2003.

☞ Originalmente la codificación de un programa Fortran sólo podía utilizar letras mayúsculas. En la actualidad todos los compiladores aceptan que en el programa haya letras minúsculas. Debe tenerse presente, sin embargo, que el compilador Fortran *no* distinguirá entre las letras mayúsculas y minúsculas (excepto en las constantes literales de carácter).

☞ Cualquier oración precedida de un signo de exclamación `!` es un *comentario*. Un comentario es una indicación del programador que permite aclarar o resaltar lo que realiza cierta parte del código. Por lo tanto los comentarios son de gran ayuda para documentar la operación de un programa o una parte de ella y deben usarse profusamente. Nótese que los comentarios son ignorados por el compilador.

☞ Es recomendable ser consistente con cierto estilo al escribir un programa Fortran. Un estilo posible es que las instrucciones de Fortran, tales como `PROGRAM`, `READ`, `WRITE`, son escritas con mayúsculas, mientras que las variables definidas por el programador son escritas con minúsculas. Asimismo los nombres que sean compuestos a partir de varias palabras son unidos con el guión bajo, como ser, `mi_primer_programa`. Por otra parte, constantes con nombres, utilizadas para valores que no cambian durante la ejecución del programa, como ser `PI` para almacenar el valor del número $\pi$, son escritos con mayúsculas.

---

**Los dialectos de Fortran.**

El primer compilador Fortran data de 1957. Al igual que los lenguajes humanos, con el paso del tiempo el lenguaje va evolucionando adaptándose a las necesidades (y filosofía) de programación de la época. Cada cierto período de tiempo un comité internacional fija la sintaxis y gramática del lenguaje originando un *estándar*. Con un estándar del lenguaje a seguir, todos los programadores nos podemos asegurar que los programas funcionaran en cualquier computadora que tenga un compilador Fortran. Tradicionalmente, los estándares de Fortran se denominan con la fecha de su constitución. El primer estándar fue Fortran 66, al cual le siguió el Fortran 77, el cual introdujo mejoras significativas al lenguaje y aún hoy continúa siendo ampliamente utilizado en la comunidad científica. Una mayor revisión del lenguaje fue el Fortran90. El Fortran 90 contiene como caso particular al Fortran 77 pero va más allá incorporando muchas nuevas características. Una revisión menor del Fortran 90 condujo al estándar Fortran 95. El estándar más reciente es el Fortran 2003, el cual constituye una nueva revisión completa del lenguaje. En este curso trabajaremos con el estandar Fortran 95 y algunas extensiones del Fortran 2003 implementadas en el compilador `gfortran`.

---

**Ejercicio 1.** Identificar y clasificar las sentencias del programa de la PRÁCTICA 0 correspondiente al algoritmo que calcula el área de un círculo.

**Ejercicio 2.** Indicar los errores en el siguiente programa Fortran.

```
PROGRAMent
* Un simple programa
integer :: ent
ent = 12
WRITE(*,*) 'El valor del entero es ',
ent
END PROGRAM ent
ent = 14
```

**Tipos de datos simples.** Los diferentes objetos de información con los que un programa puede operar se conocen colectivamente como *datos*. Todos los

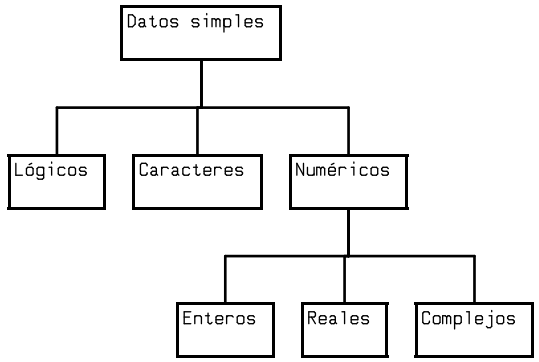


**Figura 1.** Tipos de datos simples.

datos tienen un *tipo* asociados a ellos. La asignación de tipos a los datos permite indicar la clase de valores que pueden tomar y las operaciones que se pueden realizar con ellos. Fortran dispone de cinco tipos de datos simples: *enteros*, *reales*, *complejos*, de *carácter* y *lógicos* (siendo los tres primeros datos de tipo numérico). Esta clasificación se ilustra en la figura 1. Por otra parte, los datos aparecen en el programa como *constantes literales*, como *variables* o como *constantes con nombres*. Una constante literal es un valor de cualquiera de los tipos de datos que se utiliza como tal y por lo tanto permanece invariable. Una variable, en cambio, es un dato, referido con un nombre, susceptible de ser modificado por las acciones del programa. Una constante con nombre es un valor referenciado con un nombre y su valor permanece fijo, es decir, no puede ser alterado por las acciones del programa.

Existen razones tanto matemáticas como computacionales para considerar distintos tipos de datos numéricos. Un dato entero permite la representación *exacta* de un número entero en la computadora (dentro de cierto rango). Un dato real permite la representación de un número real, aunque tal representación *no* es exacta, sino aproximada.

**Precisión de los datos reales.**

En las computadoras personales (PC) actuales, un dato de tipo real tiene una precisión de unos 6–7 dígitos significativos.

Una *constante literal entera* es un número entero, como ser `-1`, `0`, `2`, y por lo tanto no puede tener parte decimal. Una *constante literal real* es un número real con punto decimal, como ser `3.14159`, `-18.8`, `0.0056`. Números muy grandes o muy pequeños se representan en notación científica en la forma

$$\pm \texttt{m.n E} \pm \texttt{p},$$

donde `m.n` es un número decimal y la `E` (de exponente) significa multiplicar por 10 a la potencia entera $\pm$`p`. Así, por ejemplo, $3.49 \times 10^{-2}$ se codifica como `3.49E-2`.

**Constantes literales numéricas.**

Fortran establece la diferencia entre constantes literalesnenteras y reales por la presencia o ausencia del punto decimal, respectivamente. Estas dos formas *no* son intercambiables puesto que se almacenan y procesan de forma diferente en la computadora.

Por razones de claridad y estilo es conveniente utilizar el cero explícitamente al escribir constante reales sin parte fraccionaria con parte entera nula. Así escribimos `11.0` en vez de `11.` y `0.2` en vez de `.2`. También resulta conveniente codificar el cero real como `0.0`.

Un dato de tipo complejo permite representar un número complejo mediante un par ordenado de datos reales: uno que representa la parte real, y otro, la parte imaginaria del número complejo. Una *constante literal compleja* consiste un par ordenado de constantes literales reales encerradas entre paréntesis y separadas por una coma. Por ejemplo `(3.5,4.0)` corresponde al número complejo con parte real 3.5 y parte imaginaria 4.0.

Un dato tipo carácter permite representar caracteres alfanuméricos, esto es letras (a, b, A, B, etc.) y dígitos (tal como 6), y caracteres especiales (tales como \$, &, $\star$, el espacio en blanco, etc.). Una *constante literal de carácter* consiste en una secuencia de caracteres encerradas entre apóstrofes (comillas sencillas), por ejemplo: `'AbC'`, o comillas dobles, `"AbC"` por ejemplo.

Finalmente, un dato de tipo lógico es un dato que solo puede tomar un valor entre dos valores posibles: verdadero o falso. Sus valores literales se codifican en Fortran como `.TRUE.` y `.FALSE.` respectivamente.

**Ejercicio 3.** La siguiente lista presenta una serie de constantes válidas o inválidas en Fortran. Indicar si cada una de ellas es válida o inválida. En el caso que sea válida, especificar su tipo. Si es inválida indicar porque razón.

10.0, $-100,000$, 123E-5, 'Es correcto', 3.14159, '3.14159', 'Distancia =', 17.8E+6, 13.0^2.

**Ejercicio 4.** Escribir los siguientes números como constantes literales reales.

256, 2.56, $-43\,000$, $10^{12}$, 0.000000492, $-10$, $-10^{-16}$.

**Ejercicio 5.** Escribir los siguientes números complejos como constantes literales complejas.

$$i, \quad 3+i, \quad 1.$$

Las *variables* son los datos de un programa cuyo valores pueden cambiar durante la ejecución del mismo.

Existen tantos tipos de variables como tipos de datos diferentes. Una variable, en realidad, es una región dada en la memoria de la computadora a la cual se le asigna un nombre simbólico. El nombre simbólico o identificador se llama *nombre de la variable* mientras que el valor almacenado en la región de memoria se llama *valor de la variable*. La extensión de la región que ocupa la variable en la memoria viene dada por el tipo de dato asociado con ella. Una imagen mental útil de la situación es considerar a las variables como cajas o buzones, cada una de las cuales tiene un nombre y contiene un valor.

En Fortran los nombres de las variables sólo pueden formarse con letras (incluyendo al guión bajo _ como un carácter válido) o números pero siempre deben comenzar con una letra. El número máximo de caracteres que permite Fortran 95 para formar el nombre de una variables es de 31 caracteres, aunque en Fortran 2003 el límite es de 63.

> ✏ **Sobre los nombres de las variables**
>
> Constituye una buena práctica de programación utilizar nombres de variables que sugieran lo que ellas representan en el contexto del problema considerado. Esto hace al programa más legible y de más fácil comprensión.

Antes de ser utilizadas, *las variables deben ser declaradas* en la sección de declaración de tipo de variables. Para ello se utilizan las siguientes sentencias no ejecutables de acuerdo al tipo de dato que almacenarán:

```
INTEGER :: lista
REAL :: lista
COMPLEX :: lista
CHARACTER(tamaño) :: lista
LOGICAL :: lista
```

Si la lista de variables cuenta con más de un elemento, las mismas deben estar separadas por comas. Puede también usarse una sentencia de declaración de tipo por cada variable del programa. Esto último permite introducir un comentario para el uso que se hará en el programa, construyendo así un útil *diccionario de datos* que permite comprender fácilmente el uso dado a las variables del programa.

> ✏ **Declaración de variables implícita y explícita.**
>
> Fortran dispone de una (desafortunada) característica cual es la *declaración implícita de tipos*: nombres de variables que comienzan con las letras en el rango `a-h`, `o-z` se consideran variables reales, mientras que el resto, es decir las que comienza con las letras en el rango `i-n`, se consideran variables enteras. Este estilo de programación es altamente desaconsejado ya que es una fuente continua de errores. Por el contrario, nosotros propiciamos la declaración explícita de todas las variables utilizadas en el programa. Mas aún, con el fin de evitar completamente la posibilidad de declaraciones implícitas utilizamos la sentencia `IMPLICIT NONE` al comienzo de la declaración de tipo. Con esta sentencia el uso de variables no declaradas generará un error de compilación.

**Ejercicio 6.** Indicar cuales de los siguientes son identificadores aceptables como variables en Fortran.

```
gamma, j79-12, epsilon, a5, 5a,
is real, is_real, r(2)19, stop, _ok.
```

**Ejercicio 7.** Escriba las declaraciones de tipo apropiadas para declarar la variables enteras `i`, `contador`, las variables reales `x`, `y`, `vx`, `vy`, la variable lógica `clave` y la variable carácter `mensaje` (la cual tendrá un máximo de 80 caracteres).

> ✏ **Palabras reservadas.**
>
> En la mayoría de los lenguajes de programación las palabras que constituyen instrucciones del lenguaje no pueden ser utilizadas como nombres de variables (se dice que son *palabras reservadas*). En Fortran, sin embargo, *no* existen palabras reservadas y, por lo tanto, por ejemplo, es posible dar el nombre `stop` a una variable. Esta forma de proceder, sin embargo, no es recomendable porque puede introducir efectos no deseados.

Finalmente, las *constantes con nombres* permiten asignar un nombre simbólico a una constante literal y, como tal, no podrán ser alteradas. Esto resulta útil para nombrar números irracionales tales como $\pi$ o constantes físicas como la velocidad de la luz $c$ que aparecen repetidamente, y también para reconfigurar valores que pueden intervenir en un algoritmo. La declaración de una constante con nombre se realiza en su declaración de tipo como sigue:

```
tipo, PARAMETER :: nombre = constante
```

**Ejercicio 8.** Escriba sentencias de declaración apropiadas para definir como parámetros la velocidad de la luz en el vacío $c = 2.9979 \times 10^8$ m s$^{-1}$ y la constante de Plank $h = 6.6256 \times 10^{-34}$ J s.

**Sentencias de asignación.** La *sentencia de asignación* permite asignar (almacenar) un valor en una variable. La operación de asignación se indica tanto en el pseudocódigo de nuestros algoritmos como en un programa Fortran en la forma general

```
variable = expresión
```

Aquí el signo = no debe ser interpretado como el signo de igualdad matemático, sino que representa la operación en la cual el valor de la expresión situada a la derecha se almacena en la variable situada a la izquierda. Por *expresión* entendemos aquí un conjunto de variables o constantes conectadas entre sí mediante los operadores que permiten sus tipos.

☞ El tipo de dato correspondiente a la expresión debe ser el mismo tipo de dato correspondiente a la variable. Para tipos numéricos si éste no es el caso ciertas conversiones de tipo implícitas se realizan (las cuales serán discutidas enseguida).

☞ La operación de asignación es una operación *destructiva* para la variable del miembro de la izquierda, debido a que cualquier valor almacenado previamente en dicha variable se pierde y se sustituye por el nuevo valor. Por el contrario, los valores de cualesquiera variables que se encuentren en la expresión del miembro de la derecha de la asignación no cambian sus valores.

☞ En una sentencia de asignación cualquier cosa distinta de un nombre de variable en el miembro de la izquierda conduce a una sentencia incorrecta.

Entre los tipos de expresiones que podemos considerar nos interesan especialmente las *expresiones aritméticas*, las cuales son un conjunto de datos *numéricos* (variables y constantes) unidos por *operadores aritméticos* y *funciones* sobre los mismos. Cuando una expresión aritmética se evalúa el resultado es un dato numerico y por lo tanto puede ser asignada una variable de tipo numérico a través de una sentencia de asignación aritmética. Los operadores aritméticos son los usuales, cuya codificación en Fortran es como se indica en la siguiente tabla.

| Símbolo | Significado | Ejemplo |
|---|---|---|
| + | adición | `a+b` |
| - | sustracción | `a-b` |
| | opuesto | `-b` |
| / | división | `a/b` |
| * | multiplicación | `a*b` |
| ** | potenciación | `a**b` |

Pero además Fortran dispone de un conjunto de *funciones intrínsecas* que implementan una gran variedad de funciones matemáticas, algunas de las cuales se presentan en la tabla 1. Para utilizar una función se emplea el nombre de la misma seguido por la expresión sobre la que se va a operar (*argumento*) dentro de un juego de paréntesis. Por ejemplo, la sentencia `y = ABS(x)` calcula el valor absoluto de `x` y lo asigna a `y`.

✏ **Funciones intrínsecas del compilador gfortran.**

Una lista de todas las funciones implícitas que proporciona el compilador `gfortan` puede verse en la página info del mismo, ejecutando en una terminal el comando:

```
$ info gfortran
```

y dirigiéndonos a la sección titulada `Intrinsic Procedures`. (Para salir de la página info simplemente presionamos la tecla `q`).

**Orden de precedencia de las operaciones aritméticas.** Cuando en una expresión aparecen dos o más operadores se requiere de un *orden de precedencia* de las operaciones que permita determinar el orden en que se realizaran las operaciones. En Fortran estas reglas son las siguientes:

1) Todas las subexpresiones entre paréntesis se evalúan primero. Las expresiones con paréntesis anidados se evalúan de adentro hacia fuera: el paréntesis más interno se evalúa primero.
2) Dentro de una misma expresión o subexpresión, las funciones se evalúan primero y luego los operadores se evalúan en el siguiente orden de prioridad: potenciación; multiplicación y división; adición, substracción y negación.
3) Los operadores en una misma expresión o subexpresión con igual nivel de prioridad se evalúan de izquierda a derecha, con excepción de la potenciación, que se evalúa de derecha a izquierda.

Así, por ejemplo, `a + b*c/d` es equivalente a `a + ((b*c)/d)`, mientras que `x**y**z` es equivalente a `x**(y**z)`.

**Conversión implícita de tipos.** Cuando en un operador aritmético los dos operandos tienen el mismo tipo de dato numérico, el resultado tiene el mismo tipo que éstos. En particular, *la división de dos tipos enteros es un entero*, mientras que la división de dos tipos reales es un real. Si, en cambio, los operandos tienen tipos numéricos distintos, una *conversión de tipo implícita* se aplica, llevando el tipo de uno de ellos al otro, siguiendo la siguiente dirección: enteros se convierten a reales, reales se convierten a complejos. La excepción a esta regla es la potenciación: cuando la potencia es un entero el cálculo es equivalente a la multiplicación repetida (por ejemplo, con `x` real, `x**2 = x*x`) Sin embargo, si la potencia es de otro tipo numérico el cómputo se realiza implícitamente a través de las funciones logarítmica y exponencial (por ejemplo, `x**2.0` es calculado como $e^{2.0 \log x}$). De la misma manera una conversión de tipo implícita se realiza cuando en una sentencia aritmética la expresión y la variable difieren en tipo numérico, en tal caso, la expresión después de ser evaluada es convertida al tipo de la variable a la que se asigna el valor. Ciertamente, estas conversiones implícitas, producto de la "mezcla" de tipos en una misma expresión o sentencia debe ser consideradas con mucho cuidado. En expresiones complejas conviene hacerlas explícitas a través de las apropiadas *funciones intrínsecas de conversión de tipo* que se detallan en la tabla 1.

| | |
|---|---|
| `r = SIN(r)` | seno del ángulo en radianes |
| `r = COS(r)` | coseno del ángulo en radianes |
| `r = TAN(r)` | tangente del ángulo en radianes |
| `r = ASIN(r)` | arco seno (en el rango $-\pi/2$ a $+\pi/2$) |
| `r = ACOS(r)` | arco coseno (en el rango 0 a $+\pi$) |
| `r = ATAN(r)` | arco tangente (en el rango $-\pi/2$ a $+\pi/2$) |
| `r = ATAN2(r,r)` | arco tangente de arg1/arg2 (en el rango $-\pi$ a $\pi$) |
| `r = SQRT(r)` | raíz cuadrada |
| `r = EXP(r)` | función exponencial |
| `r = LOG(r)` | logaritmo natural |
| `r = LOG10(r)` | logaritmo decimal |
| `* = ABS(ir)` | valor absoluto |
| `* = MOD(ir,ir)` | resto de la división de arg1 por arg2 |
| `* = MAX(ir,ir)` | devuelve el máximo entre arg1 y arg2 |
| `* = MIN(ir,ir)` | devuelve el mínimo entre arg1 y arg2 |
| `i = INT(r)` | convierte a un tipo entero truncando la parte decimal |
| `r = REAL(i)` | convierte a tipo real |

**Tabla 1.** Funciones intrínsecas importantes proporcionadas por Fortran. El tipo del dato del argumento y del resultado que admiten es indicado por una letra: `i` = entero, `r` = real. Un asterisco en el miembro de la derecha indica que el resultado es del mismo tipo que el argumento.

**Ejercicio 9.** Escribir sentencias de asignación que efectúen lo siguiente:

a) Incrementar en 1 el valor actual de una variable entera `n` y remplazar el valor de `n` por dicho incremento. Una variable entera que se incrementa en una unidad o una cantidad constante se conoce como *contador*.
b) Incrementar en `x` (siendo `x` una variable numérica) el valor de la variable numérica `suma` y reemplazar el valor de `suma` por tal incremento. Una variable que actúa de esta forma se conoce como un *acumulador*.
c) Asignar a una variable lógica el valor verdadero.
d) Intercambiar el valor de dos variables `a` y `b` del mismo tipo (*Ayuda*: el intercambio requiere de una tercera variable).

**Ejercicio 10.** Considere el siguiente programa Fortran.

```
PROGRAM problema
 IMPLICIT NONE
 REAL :: x,y
 y = x + 1.0
 WRITE(*,*) y
END PROGRAM problema
```

¿Qué sucede al ejecutarlo varias veces? ¿A qué se debe tales resultados?

> ✏ **Variables no inicializadas.**
>
> Una variable en el lado derecho de una sentencia de asignación debe tener un valor antes de que la sentencia de asignación se ejecute. Hasta que una sentencia no le de un valor a una variable, esa variable no tendrá un valor definido. Una variable a la que no se le ha dado un valor se dice que no se ha *inicializado*. Entonces si, por ejemplo, `x` no tiene un valor antes de ejecutarse la sentencia `y = x + 1.0`, se produce un error lógico. Muchos lenguajes de programación inicializan automáticamente sus variables numéricas en cero. Sin embargo, este no es el caso de Fortran. Así una variable sin inicializar contendrá esencialmente un valor espurio proveniente de lo que exista en dicho momento en la posición de memoria correspondiente a la variable. Su uso ciertamente conducirá a una situación de error en los datos de salida. En Fortran 95 la inicialización de una variable puede ser realizada *en tiempo de compilación* asignando su valor en la sentencia de declaración correspondiente, como ser, por ejemplo, `REAL :: x = 0.0`.

**Ejercicio 11.** Determinar el valor de la variable real `a` o de la variable entera `i` obtenido como resultado de cada una de las siguientes sentencias de asignación aritmética. Indicar el orden en que son realizadas las operaciones aritméticas y las conversiones de tipo implícitas (si existen).

a) `a = 2*6 + 1`
b) `a = 2/3`
c) `a = 2.0*6.0/4.0`
d) `i = 2*10/4`
e) `i = 2*(10/4)`
f) `a = 2*(10/4)`
g) `a = 2.0*(10.0/4.0)`
h) `a = 2.0*(1.0e1/4.0)`
i) `a = 6.0*1.0/6.0`
j) `a = 6.0*(1.0/6.0)`
k) `a = 1.0/3.0 + 1.0/3.0 + 1.0/3.0`
l) `a = 1/3 + 1/3 + 1/3`
m) `a = 4.0**(3/2)`
n) `a = 4.0**3.0/2.0`
ñ) `a = 4.0**(3.0/2.0)`

o) `i = 19/4 + 5/4`
p) `a = 19/4 + 5/4`
q) `i = 100*(99/100)`
r) `i = 10**(2/3)`
s) `i = 10**(2.0/3.0)`

**Ejercicio 12.** Supongase que las variables reales `a,b,c,d,e,f` y la variable enterag han sido inicializadas con los siguientes valores:

$$\texttt{a} = 3.0,\ \texttt{b} = 2.0,\ \texttt{c} = 5.0,$$
$$\texttt{d} = 4.0,\ \texttt{e} = 10.0,\ \texttt{f} = 2.0,\ \texttt{g} = 3.$$

Determine el resultado de las siguientes sentencias de asignación, indique el orden en que se realizan las operaciones.

a) `resultado = a*b+c*d+e/f**g`
b) `resultado = a*(b+c)*d+(e/f)**g`
c) `resultado = a*(b+c)*(d+e)/f**g`

**Ejercicio 13.** Escriba sentencias de asignación aritméticas para los siguientes expresiones matemáticas.

a) $t = 3 \times 10^3\, x^4$,
b) $y = (-x)^n$,
c) $x = a^{(1/n)}$,
d) $z = \dfrac{xy}{\sqrt{x^2+y^2}}$,
e) $y = \left(\dfrac{2}{\pi x}\right)^{1/2} \cos x$,
f) $z = \cos^{-1}(|\log x|)$.
g) $y = \dfrac{1}{2} \log \dfrac{1+\sin x}{1-\sin x}$,

**Ejercicio 14.** Considere el siguiente programa

```
PROGRAM test
 IMPLICIT NONE
 REAL :: a,b,c

 READ(*,*) a,b
 c = ( (a+b)**2 - 2.0*a*b - b**2 )/a**2
 WRITE(*,*) c

END PROGRAM test
```

¿Cuál sería el valor esperado de `c`, cualquiera sean los valores de `a` y `b` ingresados? Ejectue el programa ingresando los pares de valores `a` $= 0.5$, `b` $= 888.0$; `a` $= 0.0001$, `b` $= 8888.0$ y `a` $= 0.00001$, `b` $= 88888.0$. ¿A qué se debe los resultados obtenidos?

**Entrada y salida por lista.** Si se desea utilizar un conjunto de datos de entrada diferentes cada vez que se ejecuta un programa debe proporcionarse un método para leer dichos datos. De manera similar, para visualizar los resultados del programa debe proporcionarse un mecanismo para darles salida. El conjunto de instrucciones para realizar estas operaciones se conocen como *sentencias de entrada/salida*. En los algoritmos tales instrucciones las describimos en pseudocódigo como

```
Leer lista de variables de entrada.
Imprimir lista de variables de salida.
```

Existen dos modos básicos para ingresar datos en un programa: *interactivo* y por *archivos*. Aquí solo discutiremos el primero, dejando el segundo para otra práctica. En el modo interactivo el usuario ingresa los datos por teclado mientras ejecuta el programa. La sentencia Fortran apropiada para ésto es

```
READ(*,*) lista de variables
```

donde la lista de variables, si contiene más de un elemento, está separada por comas. Con el fin de guiar al usuario en la entrada de datos interactiva es conveniente imprimir un mensaje indicativo previo a la lectura de los datos. Por ejemplo,

```
WRITE(*,*) 'Ingrese radio del círculo'
READ(*,*) radio
```

Para dar salida a los datos por pantalla utilizamos la sentencia

```
WRITE(*,*) lista de variables
```

Nuevamente podemos utilizar constantes literales de carácter para indicar de que trata el resultado obtenido. Por ejemplo,

```
WRITE(*,*) 'Area del círculo = ', area
```

**Ejercicio 15.** Escribir la sentencia de entrada (para leer por teclado) y la sentencia de salida (por pantalla) para los siguientes datos:

a) `0.1E13`
b) `(0.0,1.0)`
c) `'Hola mundo!'`
d) `3 1.5 -0.6`
e) `.TRUE.`

**Implementando lo aprendido.** Los siguientes ejercicios plantean diversos problemas. Para cada uno de ellos se debe diseñar un algoritmo apropiado el cual debe ser descrito en pseudocódigo y graficado por su diagrama de flujo. Luego implementar dicho algoritmo como un programa Fortran. Testear el programa utilizando datos de entrada que conducen a resultados conocidos de antemano.

**Ejercicio 16.** Dado los tres lados $a$, $b$ y $c$ de un triángulo, calcular su área $A$ por la fórmula de Herón,

$$A = \sqrt{s(s-a)(s-b)(s-c)},$$

donde $s = (a+b+c)/2$.

**Ejercicio 17.** Dadas las coordenadas polares $(r, \theta)$ de un punto en el plano $\mathbb{R}^2$, se desea calcular sus coordenadas rectangulares $(x, y)$ definidas por

$$\begin{cases} x = r\cos\theta, \\ y = r\sin\theta. \end{cases}$$

**Ejercicio 18.** Calcular el área $A$ y el volumen $V$ de una esfera de radio $r$,

$$A = 4\pi r^2, \quad V = \frac{4}{3}\pi r^3.$$

Implementar el algoritmo de manera de minimizar el número de multiplicaciones utilizadas.

> **Eficiencia de un algoritmo.**
> Una manera de medir la eficiencia de un algoritmo (y un programa) es contabilizar el número de operaciones utilizadas para resolver el problema. Cuanto menor sea este número más eficiente será el algoritmo (y el programa).

**Ejercicio 19.** La temperatura medida en grados centígrados $t_C$ puede ser convertida a la escala Farenheit $t_F$ según la fórmula:

$$t_F = \frac{9}{5}\, t_C + 32$$

Dado un valor decimal de la temperatura en la escala centígrada se quiere obtener su valor en la escala Fahrenheit.

**Ejercicio 20.** La magnitud de la fuerza de atracción gravitatoria entre dos masas puntuales $m_1$ y $m_2$, separadas por una distancia $r$ está dada por la fórmula

$$F = G\,\frac{m_1\, m_2}{r^2},$$

donde $G = 6.673 \times 10^{-8}$ cm$^3$ s$^2$ g$^{-1}$ es la constante de gravitación universal. Se desea evaluar, dada la masas de dos cuerpos y la distancia entre ellos, la fuerza de gravitación. El resultado debe estar expresados en dinas; una dina es dimensionalmente igual a un g cm s$^{-2}$.Nótese que las unidades de una fórmula deben ser *consistentes*, por lo cual, en este problema, las masas deben expresarse en gramos y la distancia en centímetros.

**Ejercicio 21.** Se denomina *longitud de penetración* de una onda electromagnética en un medio de cierta conductividad eléctrica a la distancia en la cual la amplitud de la onda decae a $1/e$ (alrededor del 37 %) de su valor original. La misma se expresa como

$$\delta = \sqrt{\frac{2}{\sigma\mu\omega}},$$

donde $\sigma$ es la conductividad eléctrica del medio, medida en Siemens/m, $\mu$ es su permeabilidad magnética (medida en Newton/Ampere$^2$) y $\omega = 2\pi f$ donde $f$ es la frecuencia de la onda medida en Hertz (1/segundo). Si consideramos un medio no magnético, podemos considerar $\mu = 4\pi \times 10^{-7}$ N/A$^2$, mientras que la conductividad de los primeros cientos de metros de la Tierra puede estimarse en 0.01-1.0 S/m, y la frecuencia de la onda depende del método de prospección, variando de $10^{-4}$-10 Hz en magnetotelúrica, a 25-1500 MHz ($10^6$ Hz) en *ground penetrating radar*. Implementar un programa para evaluar $\delta$ en distintos valores de frecuencia y conductividad.