.

Computación - Práctica 3

Programación en FORTRAN:
Formatos, ciclos DO, lectura y escritura de archivos

Formatos:

Ejercicio 1. Sean las siguientes variables con sus respectivos valores, por ejemplo: $POS = 1$; $PIL =$ "Juan Manuel Fangio"; $AUT =$ "Maserati"; $NAC =$ "Argentina"; $VUE = 22$; $H = 3$; $M = 30$; $S = 38.3$. Escribir un programa que declare correctamente el tipo de cada variable, y que escriba por pantalla $POS, PIL, AUT, NAC, VUE, H, M, S$ con un formato adecuado para reproducir de la siguiente manera el encabezado y la primera línea de la tabla:

```
Nro. Piloto              Auto      Pais        Vlts.  Tiempo
 1   Juan Manuel Fangio  Maserati  Argentina   22     3h 30m 38.3s
10   Mike Hawthorn       Ferrari   Inglaterra  22     3h 30m 42.2s
 7   Peter Collins       Ferrari   Inglaterra  22     3h 31m 13.9s
 6   Luigi Musso         Ferrari   Italia      22     3h 33m 15.8s
 8   Stirling Moss       Vanwall   Inglaterra  22     3h 35m 15.5s
```

¿Existe un único formato para hacerlo?

Ciclos DO:

Los ejercicios 2 y 3 deben realizarse a mano, de tal forma de predecir los valores solicitados sin tener que hacer todos los ciclos. Para verificar dichas predicciones, programar los ciclos DO y escribir los resultados.

Ejercicio 2. Al concluir los siguientes ciclos *DO*, ¿cuánto valen *K, L* y *M*?

```
      M = 1
      DO I = 2, 6
         K = I
         DO J = 1, 10
            L = J
            M = M + 1
         ENDDO
      ENDDO
```

```
      J=0
      DO I = 1, 10, 3
         K = J * I
         DO L = 5, 7
            M = K + L
            J = J - 1
         ENDDO
      ENDDO
```

A una variable entera k que incrementa su valor en una unidad se le llama *contador*. A una variable numérica que se incrementa en una cantidad x variable se le llama *acumulador*. La

forma de instrumentar esto, por ejemplo para el contador, es asignarle a la variable k el valor $k+1$. Para que esto funcione correctamente la variable k debe inicializarse, generalmente, como igual a cero (u otro valor si fuera necesario).

Ejercicio 3. Indicar cuántas veces se ejecutarán los siguientes ciclos *DO* y cuál es el valor de la variable al finalizar cada ciclo.

```
i)      DO I=25,25                  vi)     DO A=1.6,4.8
ii)     DO I=4,1                    vii)    DO A=1.6,4.8,1.6
iii)    DO I=1,10,-1                viii)   DO A=0.3,1.5,0.3
iv)     DO I=1,-10,-2               ix)     DO A=1.,6.
v)      DO I=1.6,4.8                x)      DO A=1.E0,1.E4,1.E1
```

Comprobar si sus afirmaciones son correctas programando los distintos ciclos *DO*, escribiendo por pantalla el valor final de la variable y el número de ciclos. Para esto último utilizar un *contador*.

Ejercicio 4. Escribir un programa que calcule la siguiente sumatoria, imprimiendo en pantalla los resultados parciales. Probar para distintos valores de $n \geq 2$.

$$\sum_{i=1}^{n} \sum_{j=i}^{n} (-1)^{i+j}(i+j)$$

Ejercicio 5. Escribir en pantalla una tabla de tres columnas mostrando todos los números naturales hasta M, sus cuadrados y sus cubos. La tabla debe tener encabezado, para hacer su lectura más fácil. El límite M debe ser dado por el usuario durante la ejecución del programa.

Lectura y escritura de archivos:

Para leer los datos necesarios para un programa desde un archivo o escribir los resultados de nuestro programa en un archivo, primero debemos *abrir* el mencionado archivo. Para ello usamos la sentencia *OPEN*. La sintaxis de la misma es:

```
      OPEN(UNIT=10, FILE="nombre-archivo")
```

O también,

```
      OPEN(10, FILE="nombre-archivo")
```

En el caso en que el nombre del archivo pueda variar, en vez de colocar el nombre del archivo entre comillas, usaremos una variable de tipo *caracter* cuyo valor será el nombre del archivo.

Si abrimos varios archivos, cada uno deberá tener un número de *unidad (UNIT)* distinto. Para leer los datos o escribir en el archivo usaremos respectivamente,

```
      READ(10,20) A, B
      WRITE(10,21) C, D
```

donde el primer número entre paréntesis corresponde a la unidad y el segundo al formato.
Una vez que finalizamos la utilización de un archivo deberemos *cerrarlo*, usando la sentencia *CLOSE*, por ejemplo,

```
      CLOSE(10)
```

Las sentencias *OPEN* y *CLOSE* tienen más opciones (consultar en la bibliografía).
Muchas veces puede ocurrir que no sepamos cuántas líneas tiene un archivo que debemos leer. En estos casos, si el programa ya leyó la última línea del archivo e intenta leer una nueva, abortará y nos mostrará un mensaje de error. Para evitar esto, cuando leemos un archivo podemos usar,

```
      READ(10,20, END=30) A, B
```

Esto hace que cuando el programa encuentra una línea en blanco en el archivo que está leyendo, en vez de tratar de leer, salte a la sentencia que en el programa está etiquetada como 30.

Ejercicio 6. Copiar la tabla del ejercicio 1 en un archivo. Realizar un programa que lea con un formato adecuado la tabla desde el archivo y para verificar lo leido lo vaya escribiendo en otro archivo, con el mismo formato definido en el ejercicio 1. Los nombres de los archivos deben ser ingresados por el usuario.

Ejercicio 7. Escribir un programa que lea con un formato adecuado los datos guardados en el archivo *P03-Estrellas-proximas.dat*. Sabemos que si la distancia d a una estrella está dada en pársecs, entonces su paralaje π en segundos de arco estará dada por $\pi = 1/d$. En un archivo cuyo nombre es ingresado por el usuario, el programa debe escribir el listado de las estrellas con sus respectivas paralajes y magnitudes. Recordar que 1 pc = 3.2616 años luz.

Ejercicio 8. En el archivo *P03-Puntos.dat* se encuentran listadas las coordenadas x e y de un gran número de puntos generados al azar. Realizar un programa que lea los puntos del archivo (no sabemos cuántos son) y nos entregue como resultado el número de puntos.

Ejercicio 9. Para x real y M entero, escribir un programa que calcule el valor aproximado de $\cos(x)$ según la expresión,

$$\cos(x) \approx \sum_{n=0}^{M} (-1)^n \frac{x^{2n}}{(2n)!}$$

Observar que salvo el primer término, y a menos del signo, el n-ésimo término se obtiene multiplicando al anterior por x^2 y dividiéndolo por $2n(2n-1)$. (Seguir esta observación ahorra operaciones aritméticas, los que tengan ganas pueden verificarlo).

Modificar el programa tal que realice el cálculo del $\cos(x)$ para valores de x entre 0.0 y 1.5 con paso 0.05, y para un dado M ingresado por el usuario. Guardar los resultados en un archivo cuyo

nombre también es provisto por el usuario. Ejecutar este programa para $M = 1, 2, 4$. Graficar los resultados obtenidos y compararlos con la función coseno provista por el *gnuplot*. ¿Qué observa?