PRACTICA 1b

Elementos de programación Fortran: Estructuras de control.

Se encuentra un programador muerto en la bañadera. En una de sus manos hay una botella de shampoo que dice: Modo de uso: Aplicar, enjuagar, repetir.

Estructuras de control. Las *estructuras de control* permiten especificar el orden en que se ejecutaran las instrucciones de un algoritmo. Todo algoritmo puede diseñarse combinando tres tipos básicos de estructuras de control:

- *secuencial*: las instrucciones se ejecutan sucesivamente una después de otra,
- *de selección*: permite elegir entre dos conjuntos de instrucciones dependiendo del cumplimiento (o no) de una condición,
- *de iteración*: un conjunto de instrucciones se repite una y otra vez hasta que se cumple cierta condición.

Combinando estas tres estructuras básicas es posible producir un flujo de instrucciones más complejo pero que aún conserve la simplicidad inherente de las mismas. La implementación de un algoritmo en base a estos tres tipos de estructuras se conoce como *programación estructurada* y este estilo de programación conduce a programas más fáciles de escribir, leer y modificar.

Estructura secuencial. La estructura de control más simple está representada por una sucesión de operaciones donde el orden de ejecución coincide con la aparición de las instrucciones en el algoritmo (o código del programa). La figura 1 ilustra el diagrama de flujo correspondiente a esta estructura. En Fortran una estructura secuencial es simplemente un conjunto de sentencias simples unas después de otra.

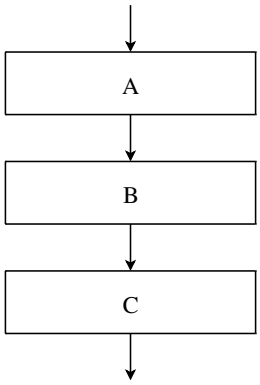

Figura 1. Estructura secuencial.

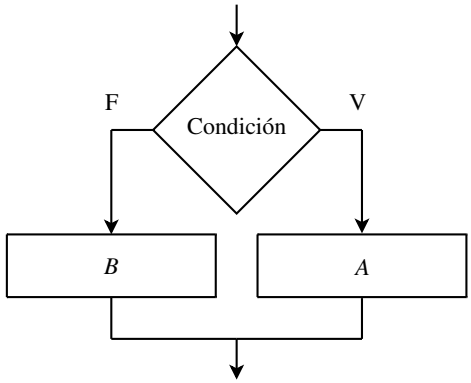

Figura 2. Estructura de selección.

Estructura de selección. La *estructura de selección* permite que dos conjuntos de instrucciones alternativas puedan ejecutarse según se cumpla (o no) una determinada condición. El pseudocódigo de esta estructura es descrito en la forma *si ... entonces ... sino ...*, ya que si p es una condición y A y B respectivos conjuntos de instrucciones, la selección se describe como *si* p es verdadero *entonces* ejecutar las instrucciones A, *sino* ejecutar las instrucciones B. Codificamos entonces esta estructura en pseudocódigo como sigue.

```
Si condición entonces
 instrucciones para condición verdadera
sino
 instrucciones para condición falsa
fin_si
```

El diagrama de flujo correspondiente se ilustra en la figura 2. En Fortran su implementación tiene la siguiente sintaxis:

```
IF (condición) THEN
 sentencias para condición verdadera
ELSE
 sentencias para condición falsa
ENDIF
```

Sangrado ("indentación")

Mientras que la separación de líneas en la codificación de una estructura de control es sintácticamente necesaria, el sangrado en los conjuntos de sentencias es opcional. Sin embargo, la sangría favorece la legibilidad del programa y, por lo tanto, constituye una buena práctica de programación.

La condición en la estructura de selección es especificada en Fortran por una *expresión lógica*, esto es, una expresión que devuelve un dato de tipo lógico: verdadero (`.TRUE.`) o falso (`.FALSE.`). Una expresión lógica puede formarse comparando los valores de expresiones aritméticas utilizando *operadores*

Operador	Significado
<	menor que
>	mayor que
==	igual a
<=	menor o igual que (*less than o equal to*)
>=	mayor o igual que (*grater than o equal to*)
/=	distinto a

Tabla 1. Operadores relacionales.

Operador	Significado
`.NOT.`	negación
`.AND.`	conjunción
`.OR.`	disyunción (inclusiva)
`.EQV.`	equivalencia

Tabla 2. Operadores lógicos.

relacionales y pueden combinarse usando *operadores lógicos*. El conjunto de operadores relacionales involucra a las relaciones de igualdad, desigualdad y de orden, las cuales son codificadas en Fortran como se indica en la tabla 1. Por otro lado, los operadores lógicos básicos son la *negación*, la *conjunción*, la *disyunción* (inclusiva) y *equivalencia*, cuya codificación en Fortran se indica en la tabla 2. El operador `.NOT.` indica la negación u opuesto de la expresión lógica. Una expresión que involucra dos operandos unidos por el operador `.AND.` es verdadera si ambas expresiones son verdaderas. Una expresión con el operador `.OR.` es verdadera si uno cualquiera o ambos operandos son verdaderos. Finalmente en el caso de equivalencia lógica la expresión es verdadera si ambos operandos conectados por el operador `.EQV.` son ambos verdaderos.[1].

Cuando en una expresión aparecen operaciones aritméticas, relacionales y lógicas, el *orden de precedencia* en la evaluación de las operaciones es como sigue:

1. Operadores aritméticos.
2. Operadores relacionales.
3. Operadores lógicos, con prioridad: `.NOT.`, `.AND.` y `.OR.`, `.EQV.`.

Operaciones que tienen la misma prioridad se ejecutan de izquierda a derecha. Por supuesto, las prioridades pueden ser modificadas mediante el uso de paréntesis.

Ejercicio 1. Dadas las variables con los valores que se indican:

```
a =  2.0   d =  2.5   i =  2   f = .FALSE.
b =  5.0   e = -4.0   j =  3   t = .TRUE.
c = 10.0              k = -2
```

deducir el valor lógico de cada una de las expresiones lógicas siguientes. Indicar el orden en que se evalúan.

a) `t .AND. f .OR. .FALSE.`

b) `a**i+b <= b/c+d`

c) `i/j == 2+k .AND. b/c+d >= e+c/d-a**j`

d) `(b*j+3.0) == (d-e) .AND. (.NOT. f)`

Ejercicio 2. Implemente una estructura de selección para verificar si un número es negativo o no negativo (positivo o cero).

Ejercicio 3. Las raíces de una ecuación cuadrática $a\,x^2+b\,x+c=0$ serán reales o complejas dependiendo del signo del discriminante $\Delta = b^2-4\,a\,c$. Implemente una estructura de decisión para determinar la naturaleza de las raíces conocidos los coeficientes de la ecuación.

Ejercicio 4. Implemente una estructura de decisión para determinar si un número entero es par o impar (*Ayuda:* utilice la función `MOD(x,y)`, la cual devuelve el resto de la división de `x` por `y`).

Ejercicio 5. Dada una esfera de radio R, considerando su centro como origen de coordenadas se quiere determinar si un punto de coordenadas (x, y, z) está dentro o fuera de la esfera. Implemente un algoritmo para éste problema.

Ejercicio 6. Considérese en el plano un rectángulo dado por las coordenadas (x_S, y_S) de su vértice superior izquierdo y las coordenadas (x_I, y_I) de su vértice inferior derecho. Se quiere determinar si un punto (x, y) del plano está dentro o fuera del rectángulo. Implemente la solución en un algoritmo.

Ejercicio 7. Dado tres números reales *distintos* se desea determinar cual es el mayor. Implemente un algoritmo apropiado (*Ayuda:* considere ya sea un conjunto de estructuras de selección *anidadas* o bien dos estructuras de selección en secuencia).

> ✏ **Estructuras de selección anidadas**
>
> La sentencia que comienza en el bloque de instrucciones verdadero o falso de la estructura de selección puede ser cualquiera, incluso otra sentencia `IF-THEN-ELSE`. Cuando ésto ocurre en una o ambas bifurcaciones de la estructura, se dice que las sentencias `IF` están *anidadas*.

En muchas circunstancias se desea ejecutar un conjunto de instrucciones sólo si la condición es verdadera y no ejecutar ninguna instrucción si la condición es falsa. En tal caso, la estructura de control se simplifica, codificándose en pseudocódigo como sigue (y con un diagrama de flujo como se indica en la figura 3)

```
Si condición entonces
  instrucciones para condición verdadera
fin_si
```

La sintaxis correspondiente en Fortran es

[1] Estos enunciados no son más que las conocidas *tablas de verdad* de la lógica matemática.

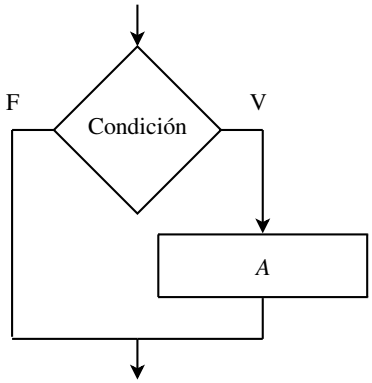

Figura 3. Estructura de selección sin sección *sino*.

```
IF (condición) THEN
 sentencias para condición verdadera
ENDIF
```

Si, además, sólo debe realizarse *una* sentencia ejecutable cuando la condición es verdadera, Fortran permite codificar esta situación en un *if lógico*:

```
IF (condición) sentencia ejecutable
```

Ejercicio 8. Implemente un algoritmo que intercambie los valores de dos números reales si están en orden creciente pero que no realice ninguna acción en caso contrario.

Ejercicio 9. Implementar el cálculo del valor absoluto de un número real con un if lógico.

Otra circunstancia que se suele presentar es la necesidad de elegir entre más de una alternativa de ejecución. En este caso podemos utilizar la *estructura multicondicional* cuya lógica se puede expresar en la forma *si ... entonces sino si ... sino ...*. El pseudocódigo correspondiente es descrito como sigue (y su diagrama de flujo se ilustra en la figura 4)

```
Si condición 1 entonces
 instrucciones para condición 1 verdadera
sino si condición 2 entonces
 instrucciones para condición 2 verdadera
...
sino si condición N entonces
 instrucciones para condición N verdadera
sino
 instrucciones para todas
las condiciones falsas
fin_si
```

Aquí, cada condición se prueba por turno. Si la condición no se se satisface, se prueba la siguiente, pero si la condición es verdadera, se ejecutan las instrucciones correspondientes para tal condición y luego se va al final de la estructura. Si ninguna de las condiciones son satisfechas se ejecutan las instrucciones especificadas en el bloque correspondientes al *sino* final. Debería quedar claro entonces que para que un estructura condicional sea eficiente sus condiciones deben ser *mutuamente excluyentes*. La codificación de esta estructura en Fortran se indica a continuación.

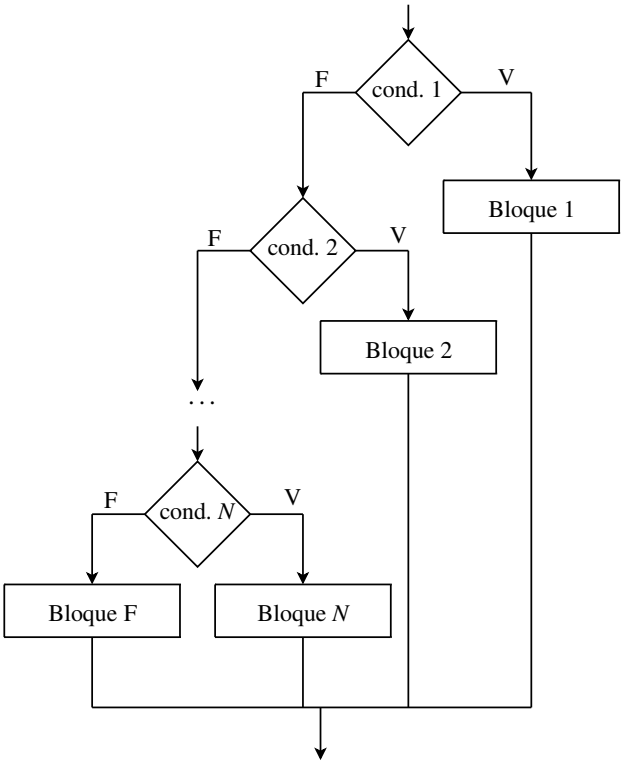

Figura 4. Estructura multicondicional.

```
IF (condición 1) THEN
 sentencias para condición 1 verdadera
ELSEIF (condición 2) THEN
 sentencias para condición 2 verdadera
...
ELSEIF (condición N) THEN
 sentencias para condición N verdadera
ELSE
 sentencias para todas condiciones falsas
ENDIF
```

Ejercicio 10. Implementar una estructura multicondicional para la evaluación de la función

$$f(x) = \begin{cases} e^{-x} & \text{si } x < -1, \\ e & \text{si } -1 \leq x \leq 1, \\ e^{x} & \text{si } x > 1. \end{cases}$$

Ejercicio 11. Un examen se considera desaprobado si la nota obtenida es menor que 4 y aprobado en caso contrario. Si la nota está entre 7 y 9 (inclusive) el examen se considera destacado, y entre 9 y 10 (inclusive) sobresaliente. Implementar un algoritmo para indicar el *estatus* de un examen en base a su nota.

Estructura de iteración. La *estructura de control iterativa* permite la repetición de una serie determinada de instrucciones. Este conjunto de instrucciones a repetir se denomina *bucle* (*loop*, en inglés) y cada repetición del mismo se denomina *iteración*. Podemos diferenciar dos tipos de bucles:

- Bucles donde el número de iteraciones es fijo y conocido de antemano.
- Bucles donde el numero de iteraciones es desconocido de antemano. En este caso el bucle se repite mientras se cumple una determinada condición (*bucles condicionales*).

Para estos dos tipos de bucles disponemos de sendas formas básicas de la estructura de control iterativa.

Comencemos con un bucle cuyo número de iteraciones es conocido *a priori*. La estructura iterativa, en tal caso, puede expresarse en pseudocódigo como sigue.

```
Desde índice=valor inicial hasta
                    valor final hacer
   instrucciones del bucle
fin_desde
```

El diagrama de flujo correspondiente se ilustra en la figura 5.

☞ *índice* es una variable entera que, actuando como un *contador*, permite controlar el número de ejecuciones del ciclo.

☞ *valor inicial* y *valor final* son valores enteros que indican los límites entre los que varía *índice* al comienzo y final del bucle.

☞ Está implícito que en cada iteración la variable *índice* toma el siguiente valor, incrementándose en una unidad. En seguida veremos que en Fortran se puede considerar incrementos mayores que la unidad e incluso negativos.

☞ El número de iteraciones del bucle es $N =$ *valor final* - *valor inicial* $+1$.

☞ Dentro de las instrucciones del bucle *no es legal* modificar la variable *índice*. Asimismo, al terminar todas las iteraciones el valor de la variable no tiene porque tener un valor definido, por lo tanto, la utilidad de la variable *índice* se limita a la estructura de iteración.

En Fortran la estructura repetitiva se codifica como sigue.

```
DO índice=valor inicial,valor final,incr.
  sentencias del bucle
ENDDO
```

Aquí *índice* es una variable entera, mientras que *valor inicial*, *valor final* e *incremento* pueden ser variables, constantes o expresiones enteras y pueden ser negativos. Si *incremento* no se especifica se asume igual a la unidad. El número de iteraciones del bucle está dado por

$$N = \max\{[(\textit{valor final} - \textit{valor inicial} + \textit{incr.})/\textit{incr.}\ ], 0\},$$

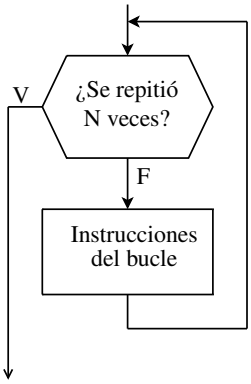

Figura 5. Estructura de iteración.

donde [ ] denota tomar la parte entera, descartando cualquier fracción decimal. Sólo si N es mayor que 0 se ejecutará el bucle.

Ejercicio 12. Imprimir una tabla de los cuadrados y cubos de los primeros N números enteros positivos. Ordenar la tabla primero en orden ascendente y luego en orden descendente.

Ejercicio 13. Calcular la suma y multiplicación de los N primeros números enteros positivos,

$$\sum_{i=1}^{N} i, \qquad \prod_{i=1}^{N} i.$$

✏ **Inicialización de los acumuladores.**
Siempre recordar inicializar a cero la variable que será utilizada para acumular una suma repetida y a uno la variable que acumulará un producto repetido.

Ejercicio 14. Considérese el siguiente conjunto de bucles repetitivos *anidados*.

```
DO i=1,5
 WRITE(*,*) 'Iteración externa = ',i
 DO j=1,4
   WRITE(*,*) 'Iteración interna = ',j
 ENDDO
ENDDO
```

¿Cuantas iteraciones del bucle interno se realizan por cada iteración del bucle externo? ¿Cuántas iteraciones se realizan en total?

Ejercicio 15. Tabular la función $f(x) = \dfrac{1}{\sqrt{2\pi}}\, e^{-x^2}$ para x en el intervalo $[0, 1]$ con un paso $h = 0.1$ (*Ayuda:* Notar que si $[a, b]$ es el intervalo bajo consideración, los puntos donde hay que evaluar f están dados por $x_i = a + ih$ con $i = 0, \ldots, N$, siendo $N = (b - a)/h$).

Consideremos, finalmente, los bucles condicionales. Aquí, el número de iteraciones no es conocido *a priori*,

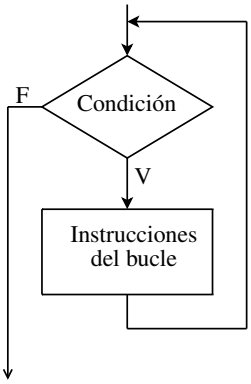

Figura 6. Estructura de iteración condicional.

sino que el bucle se repite *mientras* se cumple una determinada condición. En este caso, la estructura iterativa se describe en pseudocódigo de la siguiente forma (y su diagrama de flujo se ilustra en la figura 6)

```
Mientras condición hacer
  instrucciones del bucle
fin_mientras
```

☞ La condición se evalúa antes y después de cada iteración del bucle. Si la condición es verdadera las instrucciones del bucle se ejecutarán y, si es falsa, el control pasa a la instrucción siguiente al bucle.

☞ Si la condición es falsa cuando se ejecuta el bucle por primera vez, las instrucciones del bucle no se ejecutarán.

☞ Mientas que la condición sea verdadera el bucle continuará ejecutándose indefinidamente. Por lo tanto, para terminar el bucle, en el interior del mismo debe tomarse alguna acción que modifique la condición de manera que su valor pase a falso. Si la condición nunca cambia su valor se tendrá un *bucle infinito*, la cual no es una situación deseable.

En Fortan un bucle condicional se codifica como sigue.

```
DO WHILE (condición)
   sentencias del bloque
ENDDO
```

donde *condición* es una expresión lógica.

Ejercicio 16. Determinar cual es el primer valor de N para el cual la suma de los N primeros números enteros positivos, $\sum_{i=1}^{N} i$, excede a 10 000.

Fortran, además del `DO WHILE`, dispone de una estructura más general para bucles condicionales, cuya codificación es la siguiente:

```
DO
   sentencias del bloque pre-condición
   IF (condición) EXIT
   sentencias del bloque post-condición
ENDDO
```

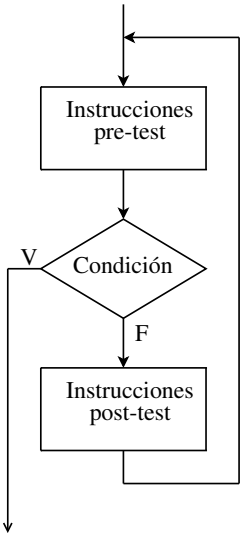

Figura 7. Estructura de iteración condicional general.

El pseudocódigo correspondiente puede ser descrito como sigue (y su diagrama de flujo se ilustra en la figura 7):

```
Repetir
  instrucciones pre-condición
  Si condición terminar repetir
  instrucciones post-condición
fin_repetir
```

☞ Las sentencias correspondientes tanto a los bloques previo y posterior al test de la condición son ejecutadas indefinidamente mientras la condición sea falsa. Cuando la condición resulta verdadera la repetición es terminada y la ejecución continúa con la sentencia que sigue a la estructura de control. Nótese que si la condición es verdadera cuando se inicia el bucle por primera vez, las sentencias del bloque pre-condición serán ejecutadas una vez y luego el control es transferido a la sentencia siguiente a la estructura, sin ejecutarse ninguna de las instrucciones correspondientes al bloque posterior a la condición.

☞ Un bucle condicional `DO WHILE`

```
DO WHILE (condición)
   sentencias del bloque
ENDDO
```

es equivalente a un bucle condicional general de la forma

```
DO
   IF (.NOT. condición) EXIT
   sentencias del bloque
ENDDO
```

Esto es, la condición lógica que controla la repetición del bucle es evaluada al comienzo del bucle y es la negación lógica a la condición que controla al bucle `DO WHILE`.

☞ Un bucle condicional general de la forma

```
DO
  sentencias del bloque
  IF (condición) EXIT
ENDDO
```

repetirá las sentencias del bloque hasta que la condición se haga verdadera. Pero debido a que la condición se verifica después de que el cuerpo del bucle se ha ejecutado, las instrucciones correspondientes se ejecutarán al menos una vez sin importar si la condición es verdadera o falsa. Este tipo de bucle condicional es conocido como *repetir hasta que* (*repeat-until*).

Ejercicio 17. Reimplementar el ejercicio anterior con una estructura *repetir hasta que*.

Implementando lo aprendido. Los siguientes ejercicios plantean diversos problemas. Diseñar un algoritmo apropiado implementando su pseudocódigo y su diagrama de flujo correspondiente. Luego codificarlo en un programa Fortran.

Ejercicio 18. Determinar si un año dado es bisiesto o no. Recordar que un año es bisiesto si es divisible por 4, aunque si es divisible por 100 no es bisiesto, salvo si es divisible por 400. Así, 1988 fue bisiesto, como también lo fue el año 2000, pero no 1800.

Ejercicio 19. Dado un conjunto de N números determinar cuantos de ellos son negativos, positivos o cero.

Ejercicio 20. Calcular las soluciones de una ecuación cuadrática $a\,x^2 + b\,x + c = 0$. Contemplar todas las alternativas posibles (raíces reales distintas, iguales y complejas). Testear el programa para el siguiente conjunto de coeficientes:

a) $a = 2, b = 2, c = -12$ (raíces reales 2 y -3),

b) $a = 2, b = 4, c = 2$ (raíz doble -1),

c) $a = 1, b = 0, c = 1$ (raíces conjugadas, i y $-i$).

Ejercicio 21. Dado un conjunto de N números reales determinar cual es el máximo, el mínimo y la media aritmética del conjunto.

Ejercicio 22. El *máximo común divisor*, mcd, (a, b) de dos número enteros positivos a y b, con $a \geq b > 0$ puede ser calculado por el *algoritmo de Euclides*, según el cual el mcd es igual al último resto no nulo que se obtiene por aplicación sucesiva de la división entera entre el divisor y el resto del paso anterior. Esto es,

$$\begin{aligned}(a, b) &= (b, r_1)\\ &= (r_1, r_2)\\ &= \dots\\ &= (r_{n-1}, r_n)\\ &= (r_n, 0)\\ &= r_n\end{aligned}$$

Implementar este algoritmo para calcular el máximo común divisor de dos números enteros *cualesquiera* dados, contemplando la posibilidad de que alguno de ellos, o ambos, sea negativo o cero (*Ayuda*: $(a, b) = (|a|, |b|)$, $(a, 0) = |a|$ y $(0, 0)$ no está definido). Verificar el programa calculando (25950,1095) = 15, (252,-1324) = 4.

Ejercicio 23. La fecha del domingo de Pascua corresponde al primer domingo después de la primera luna llena que sigue al equinocio de primavera en el hemisferio norte. Los siguientes cálculos permiten conocer esta fecha para un año comprendido entre 1900 y 2099:

```
a = MOD(año,19)
b = MOD(año,4)
c = MOD(año,7)
d = MOD(19*a+24,30)
e = MOD(2*b+4*c+6*d+5,7)
n = 22+d+e
```

donde `n` indica el número de días del mes de marzo (o abril si `n` es superior a 31) correspondiente al domingo de Pascua. Realizar un programa que determine esta fecha para un año dado y comprobar que para el 2010, el domingo de Pascua corresponde al 4 de abril.

Ejercicio 24. Todo número complejo $z = (x, y) = x + iy$ no nulo admite exactamente n raíces n-ésimas distintas dadas por

$$w_k = \sqrt[n]{\rho}\Big[\cos\Big(\frac{\theta + 2k\pi}{n}\Big) + i\ \sin\Big(\frac{\theta + 2k\pi}{n}\Big)\Big],$$

donde $k = 0, 1, \dots, n-1$, y

$$\rho = \sqrt{x^2 + y^2}, \qquad \tan\theta = y/x.$$

Dado un número complejo z no nulo determinar todas sus raíces n-ésimas (*Ayuda:* para determinar el argumento θ utilizar la función `ATAN2(y,x)`).

Ejercicio 25. Calcular el seno de un número x a partir de su serie de Taylor:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

Considerar tantos términos como sean necesarios para que el error cometido en la aproximación finita sea menor que cierta tolerancia prescrita (digamos $\leq 10^{-8}$). *Observación:* Notar que dado un término de la serie, el siguiente se obtiene multiplicando por $-x^2$ y dividiendo por el producto de los dos enteros siguientes. Testear el programa tanto con valores pequeños como con valores muy grandes. Comentar los resultados obtenidos.