EJERCICIOS EXÁMENES ANTERIORES

1. Implementar una función cuyos argumentos de entrada sean los valores de tres ángulos (medidos en grados) y que devuelva:

   0 si los tres ángulos no pueden pertenecer a un triángulo

   1 si los tres ángulos pueden pertenecer a un triángulo rectángulo

   2 si los tres ángulos pueden pertenecer a un triángulo obtusángulo

   3 si los tres ángulos pueden pertenecer a un triángulo acutángulo

   Téngase en cuenta:

   - la suma de los ángulos de un triángulo es 180º
   - un triángulo rectángulo tiene un ángulo de 90º
   - un triángulo obtusángulo tiene un ángulo mayor de 90º
   - un triángulo acutángulo tiene los tres ángulos menores de 90º

Solución1:

```
Integer Function Triangulo (angulo1,angulo2,angulo3)
! Esta función clasifica un triángulo por los valores de sus ángulos
! dados como enteros positivos
Implicit none
! Argumentos
    Integer angulo1,angulo2,angulo3
    Intent (in) angulo1,angulo2,angulo3
! Cuerpo de la función
      If (angulo1+angulo2+angulo3 /= 180) Then
            Triangulo = 0
      Else If ((angulo1==90).or.(angulo2==90).or.(angulo3==90)) Then
            Triangulo = 1
      Else If ((angulo1>90).or.(angulo2>90).or.(angulo3>90)) Then
            Triangulo = 2
      Else
            Triangulo = 3
      End If
End Function Triangulo
```

Solución2:

```
Integer Function Triangulo (angulo1,angulo2,angulo3)
! Los angulos son reales y no necesariamente positivos
Implicit none
! Argumentos
    Real angulo1,angulo2,angulo3
    Intent (in) angulo1,angulo2,angulo3
! Cuerpo de la función
      If (angulo1+angulo2+angulo3 /= 180 .or. angulo1>0 .or. angulo2>0
          .or. angulo3>0) Then
            Triangulo = 0
      Else If ((angulo1==90).or.(angulo2==90).or.(angulo3==90)) Then
            Triangulo = 1
      Else If ((angulo1>90).or.(angulo2>90).or.(angulo3>90)) Then
            Triangulo = 2
      Else
            Triangulo = 3
```

```
      End If
End Function Triangulo
```

---

2. Implementar una subrutina que calcule la derivada de un polinomio de a lo sumo grado 3.

Ambos polinomios (el de entrada y su derivada) se representarán mediante vectores de enteros de cuatro componentes, por ejemplo:

La representación del polinomio 7 -2x -8x^2 +6x^3 sería:

7	-2	-8	6

y la de su derivada -2 -16x+18x^2 sería:

-2	-16	18	0

Por tanto, la subrutina debe de tener dos argumentos de tipo vector, uno de entrada con la información del polinomio a derivar y otro de salida donde se calculará la derivada del polinomio.

Solución:

```
Subroutine Derivar (polinomio,derivada)
! Esta subrutina calcula la derivada de un polinomio
Implicit none
! Constantes
    Parameter  max = 4
! Argumentos
    Integer  polinomio(max), derivada(max)
    Intent (in) polinomio
    Intent (out) derivada
! Variables
    Integer  i
! Cuerpo de la subrutina
      Do i = 1, max-1
            derivada(i) = polinomio(i+1) * i
      End Do
      derivada(max) = 0

   End Subroutine Derivar
```

---

3. Implementar una función cuyos argumentos de entrada sean los coeficientes de una ecuación de segundo grado y que devuelva:

0 si la ecuación no tiene soluciones reales

1 si la ecuación tiene una solución real doble

2 si la ecuación tiene dos soluciones reales diferentes

3 si la ecuación no es de segundo grado

Téngase en cuenta:

- La ecuación $ax^2 + bx + c = 0$ es de segundo grado si $a \neq 0$
- Si $ax^2 + bx + c = 0$ es una ecuación de segundo grado su discriminante es: $\Delta = b^2 - 4ac$
- Una ecuación de segundo grado no tiene soluciones reales si su discriminante es negativo.
- Una ecuación de segundo grado tiene una solución real doble si su discriminante es nulo.
- Una ecuación de segundo grado tiene dos soluciones reales diferentes si su discriminante es positivo.

Solución:

```
Integer Function Soluciones (a,b,c)
! Esta función determina el nº de soluciones reales de una ecuación de
! segundo grado
Implicit none
! Argumentos
      Real a,b,c
      Intent (in) a,b,c
! Variables
      Real discriminante
! Cuerpo de la función
      If (a == 0) Then
            Soluciones = 3
      Else
            discriminante = b * b - 4 * a * c
            If (discriminante < 0) Then
                  Soluciones = 0
            Else If (discriminante == 0) Then
                  Soluciones = 1
            Else
                  Soluciones = 2
            End If
      End If
End Function Soluciones
```

4. Implementar una subrutina que calcule la integral de un polinomio de a lo sumo grado 2.

Ambos polinomios (el de entrada y su integral) se representarán mediante vectores de reales de cuatro componentes, por ejemplo:

La representación del polinomio $-2 -16x+18x^2$ sería

-2	-16	18	0

y la de su integral $-2x -8x^2 +6x^3$ sería:

0	-2	-8	6

Por tanto, la subrutina debe de tener dos argumentos de tipo vector, uno de entrada con la información del polinomio a integrar y otro de salida donde se calculará la integral del polinomio.

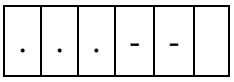
Solución:

```
Subroutine Integrar (polinomio,integral)
! Esta subrutina calcula la integral de un polinomio
Implicit none
! Constantes
      Parameter  max = 4
! Argumentos
      Real  polinomio(max), integral(max)
      Intent (in) polinomio
      Intent (out) integral
! Variables
    Integer i
! Cuerpo de la subrutina
      integral(1) = 0
      Do i = 2, max
            integral(i) = polinomio(i-1) / (i-1)
      End Do
End Subroutine Integrar
```

5. La versión internacional de la codificación de los dígitos en Morse es la siguiente:

1	.----	6	-....
2	..---	7	--...
3	...--	8	---..
4	....-	9	----.
5	.....	0	-----

Implementar una función en Fortran que reciba como argumento de entrada la representación de un digito en código Morse y devuelva como resultado el dígito descodificado. El digito codificado viene representado mediante un vector de caracteres que contiene una sucesión de puntos y rayas terminada con un espacio en blanco. Por ejemplo:



➔ 3

Solución 1

```
Integer Function DigMorse (digito)
Implicit none
! Constantes
      Parameter MaxLong = 6
! Argumentos
      Character(1) digito (MaxLong)
      Intent (in) digito
! Variables
      Integer i
! Cuerpo de la función
      DigMorse = 1
      i = 2
      Do While (digito (i) == digito (i-1))
            DigMorse = DigMorse + 1
            i = i + 1
      End Do
      If (digito (1) == '-') Then
            DigMorse = mod (DigMorse + 5, 10)
      End If
End Function DigMorse
```

Solución 2

```
Integer Function DigMorse (digito)
Implicit none
! Constantes
      Parameter MaxLong = 6
! Argumentos
      Character(1) digito (MaxLong)
      Intent (in) digito
! Variables
      Integer i
! Cuerpo de la función
      i = 1
      DigMorse = 0
      Do While (digito (i) == '.' .and. i < MaxLong )
            DigMorse = DigMorse + 1
            i = i + 1
      End Do
      If (DigMorse == 0) Then
            i = 1
            Do While (digito (i) == '-' )
                  DigMorse = DigMorse +1
                  i = i + 1
            End Do
            DigMorse = mod (DigMorse+5, 10)
      End If
End Function DigMorse
```

Solución 3

```
Integer Function DigMorse (digito)
Implicit none
! Constantes
      Parameter maxLong = 6
! Argumentos
      Character(1) digito (MaxLong)
      Intent (in) digito
! Variables
      Integer i, contPuntos, contRayas
! Cuerpo de la función
      contPuntos = 0
      contRayas = 0
      i = 1
      If (digito (i) == '.')  Then
            Do While (digito (i) == '.' )
                  i = i + 1
                  contPuntos = contPuntos + 1
            End Do
      Else
            Do While (digito (i) == '-' )
                  i = i + 1
                  contRayas = contRayas + 1
            End Do
      End If
      If (contPuntos /= 0) Then
            DigMorse = contPuntos
      Else If (contRayas == 5) Then
            DigMorse = 0
      Else
            DigMorse = 5 + contRayas
      End If
End Function DigMorse
```

6. Los resultados de una liga deportiva a doble vuelta de N equipos se pueden representar mediante una matriz de caracteres. Cada resultado se corresponde con un elemento de la matriz cuya fila identifica al equipo local y la columna al visitante. Los caracteres utilizados y su significado son:

'1' → vence el equipo local

'X' → empate entre ambos equipos

'2' → vence el equipo visitante

Implementar una subrutina en lenguaje Fortran que reciba una matriz (tal como se ha descrito anteriormente) con los resultados de una liga de 20 equipos. La subrutina tendrá como argumento de salida un vector con las puntuaciones conseguidas por cada equipo teniendo en cuenta que las victorias aportan 3 puntos, los empates 1 y las derrotas 0.

El siguiente ejemplo muestra una matriz con los resultados de una liga de 4 equipos y el vector con las puntuaciones obtenidas por cada uno de ellos.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} & 2 & X & 1 \\ X & & 1 & 2 \\ 2 & 2 & & 1 \\ X & 1 & 1 & \end{pmatrix} \end{array} \Rightarrow \begin{pmatrix} 9 \\ 10 \\ 4 \\ 10 \end{pmatrix}$$

Solución 1

```
Subroutine Puntuacion (resultados, puntos)
Implicit none
! Constantes
      Parameter NumEquipos = 20
! Argumentos
      Character(1) resultados(NumEquipos, NumEquipos)
      Integer puntos(NumEquipos)
      Intent (in) resultados
      Intent (out) puntos
! Variables
      Integer i, j
! Cuerpo de la subrutina
      Do i = 1, NumEquipos
            puntos (i) = 0
      End Do
      Do i = 1, NumEquipos
            Do j = 1, NumEquipos
                  If (resultados (i,j) == '1') Then
                        puntos (i) = puntos (i) + 3
                  Else If (resultados (i,j) == '2') Then
                        puntos (j) = puntos (j) + 3
                  Else If (resultados (i,j) == 'X') Then
                        puntos (i) = puntos (i) + 1
                        puntos (j) = puntos (j) + 1
                  End If
            End Do
      End Do
End Subroutine Puntuacion
```

Solución2

```
Subroutine Puntuacion (resultados, puntos)
Implicit none
! Constantes
      Parameter NumEquipos = 20
! Argumentos
      Character(1) resultados(NumEquipos, NumEquipos)
      Integer puntos(NumEquipos)
      Intent (in) resultados
      Intent (out) puntos
! Variables
      Integer i, j
! Cuerpo de la subrutina
      Do i = 1, NumEquipos
            puntos (i) = 0
            Do j = 1, NumEquipos
                  If (resultados (i,j) == '1') Then
                        puntos (i) = puntos (i) + 3
                  Else If (resultados (i,j) == 'X') Then
                        puntos (i) = puntos (i) + 1
                  End If
                  If (resultados (j, i) == '2') Then
                        puntos (i) = puntos (i) + 3
                  Else If (resultados (j,i) == 'X') Then
                        puntos (i) = puntos (i) + 1
                  End If
            End Do
      End Do
End Subroutine Puntuacion
```

7. Implementar una función en fortran que dada una matriz de 3x3 que contiene $\boldsymbol{X}$, $\boldsymbol{O}$ y $-$, determine si se ha conseguido tres en raya y quien lo ha conseguido. Por ejemplo:

$$\begin{matrix} O & X & - \\ X & O & O \\ X & - & O \end{matrix}$$ En este caso el ganador será el jugador de la $\boldsymbol{O}$.

La función debe recibir como parámetro la matriz de caracteres y devolver un carácter:

- **E** si ninguno ha conseguido tres en raya,
- **X** si el jugador de la **X** ha conseguido tres en raya,
- **O** si el jugador de la **O** ha conseguido tres en raya.

Tener en cuenta que:

- se puede conseguir tres en raya en horizontal (1ª , 2ª o 3ª línea), en vertical (1ª , 2ª o 3ª columna) o en las dos diagonales,
- solo uno de los jugadores puede conseguir tres en raya.

Se valorará positivamente la utilización de constantes para generalizar la función para matrices de *n x n*, es decir, la posibilidad de jugar al *n* en raya.

Solución

```
Character(1) Function TresEnRaya (tablero)
Implicit none
! Constantes
      Parameter  max = 3
! Argumentos
      Character(1) tablero(max,max)
      Intent (in) tablero
! Variables
      Integer i,j,contO, contX
! Cuerpo de la subrutina
      tresEnRaya = 'E'
      i = 1
      !Comprobamos las filas
      Do while (TresEnRaya == 'E' .AND. i <= max)
          contO = 0
          contX = 0
          Do j = 1, max
                If (tablero(i,j)== 'O') Then
                    contO = contO + 1
                Else If (tablero (i,j) == 'X') Then
                    contX = contX + 1
                End If
          End Do
          If (contO == max) Then
                TresEnRaya = 'O'
          Else If (contX == max) Then
                TresEnRaya = 'X'
          Else
                i = i + 1
          End If
      End Do
      j=1
```

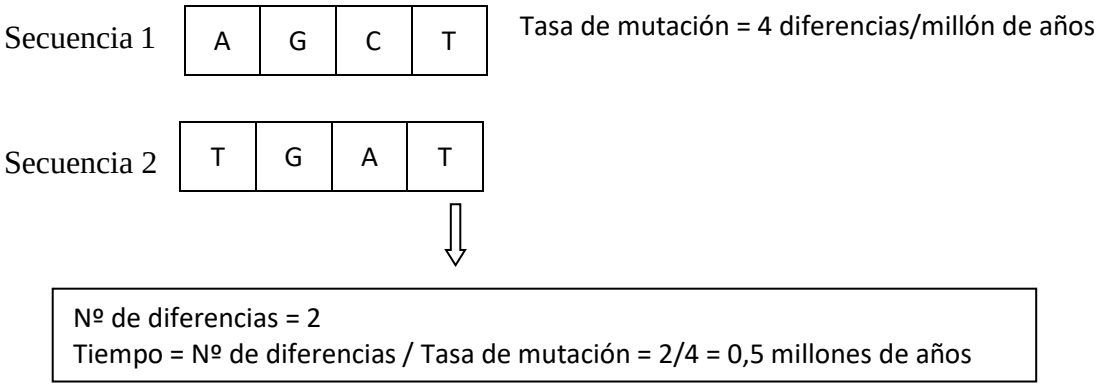
```
!Comprobamos las columnas
      Do while (TresEnRaya == 'E' .AND. j <= max)
          contO = 0
          contX = 0
          Do i = 1, max
                If (tablero(i,j)== 'O') Then
                    contO = contO + 1
                Else If (tablero (i,j) == 'X') Then
                    contX = contX + 1
                End If
          End Do
          If (contO == max) Then
                TresEnRaya = 'O'
          Else If (contX == max) Then
                TresEnRaya = 'X'
          Else
                j = j + 1
          End If
      End Do
      !Comprobamos la diagonal principal
      If (TresEnRaya == 'E' .AND. tablero (1,1)/= '-') Then
          TresEnRaya = tablero (1,1)
          Do i = 1, max - 1
              If (tablero(i,i) /= tablero (i+1,i+1)) Then
                  TresEnRaya = 'E'
              End if
          End Do
      End If
      !Comprobamos la otra diagonal
      If (TresEnRaya == 'E' .AND. tablero (1,max)/= '-') Then
          TresEnRaya = tablero (1,max)
          Do i = 0, max - 2
              If (tablero(i+1,max-i) /= tablero (i+2,max-i-1)) Then
                  TresEnRaya = 'E'
              End if
          End Do
      End If
End Function TresEnRAya
```

8. Con el fin de determinar el tiempo en que divergieron dos especies se comparan sus secuencias de ADN.

El número de diferencias encontradas, comparando posición a posición, dividido entre la tasa de mutación (nº diferencias /millones de años) proporciona la cantidad de tiempo en que ambas especies se separaron.

En una versión simplificada de este problema se pide implementar una subrutina en lenguaje Fortran que reciba dos secuencias de ADN (representadas por dos vectores de caracteres de tamaño 100) y la tasa de mutación. Como resultado proporcionará en dos argumentos de salida el número de diferencias entre ambas secuencias y el tiempo transcurrido desde que las dos especies divergieron.

Ejemplo con dos secuencias de tamaño 4:



El tres en raya es un juego entre dos jugadores (O y X) que marcan los espacios de un tablero de 3×3 alternadamente. Un jugador gana si consigue tener una línea de tres de sus símbolos. La línea puede ser horizontal, vertical o diagonal.

9. Codificar una función en Fortran que a partir de un tablero de 3 en raya, el jugador que tiene el turno y una línea concreta (columna, fila o diagonal), devuelva un valor negativo (por ejemplo -1) si no es posible alcanzar situación de victoria por ese jugador en esa línea. En caso contrario, la función devolverá el nº de fichas que el jugador tiene colocadas en dicha línea.

Consideraciones:

En una línea no se puede alcanzar 3 en raya por un jugador si en la misma aparecen fichas de su adversario.

Los jugadores quedan identificados por el tipo de ficha que utilizan ('O' , 'X').

La línea se identifica con un valor entero del 1 al 8. Del 1 al 3 para las filas 1ª, 2ª y 3ª respectivamente, del 4 al 6 para las columnas 1ª, 2ª y 3ª respectivamente, el 7 para la diagonal principal y el 8 para la secundaria.

Las fichas en el tablero se representan con los caracteres 'O' y 'X' y las posiciones libres por el espacio en blanco.

El código debe ser genérico en el sentido de poder desarrollarse en un tablero de n x n ("n en raya"). Con tan sólo modificar el valor de una constante (en el caso del "3 en raya" el valor de la constante es 3).

Ejemplo 1:

O	X	
O		X
		X

, 'O' , 1 → -1

Interpretación: En la línea 1 (1ªfila) no se puede conseguir 3 en raya con fichas 'O'

Tablero, jugador, línea
(argumentos de entrada)

Ejemplo 2:

O	X	
O		X
		X

, 'X' , 6 → 2

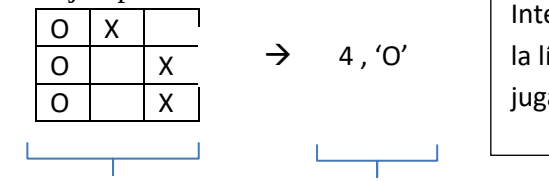
Interpretación: En la línea 6 (3ª columna) se puede conseguir 3 en raya con fichas 'X' y de momento hay colocadas 2

Tablero, jugador, línea
(argumentos de entrada)

De forma opcional los alumnos que deseen mejorar la calificación obtenida en la prueba de la traza de la evaluación continua pueden realizar la traza de su implementación con los dos ejemplos.

10. Implementar una subrutina en Fortran que, a partir de un tablero de tres en raya recibido como argumento y apoyándose en la función del ejercicio anterior, devuelva, en caso de haberse alcanzado situación de victoria, la línea donde se ha conseguido y el jugador que ha vencido. En caso de que no exista situación de 3 en raya se devolverá un valor negativo en el argumento que indica la línea y el espacio en blanco en el argumento que indica el jugador.

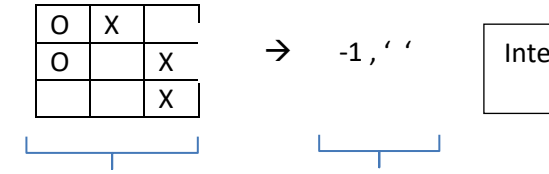
Ejemplo 1:



Tablero (argumento de entrada) — línea, jugador (argumentos de salida)

Interpretación: Hay situación de 3 en raya en la línea 4 (1ª columna) y la ha conseguido el jugador que juega con el tipo de ficha 'O'

Ejemplo 2:



Tablero (argumento de entrada) — línea, jugador (argumentos de salida)

Interpretación: No hay situación de 3 en raya

Además de ser válidas las consideraciones del ejercicio anterior ha de tenerse en cuenta que se puede llamar a la función del ejercicio 1., sin necesidad de haberla codificado, una vez se haya definido su cabecera.

11. Implementar una función cuyos argumentos de entrada sean dos polinomios y que devuelva:

- -1 si el primer polinomio tiene mayor grado que el segundo
- -2 si el segundo polinomio tiene mayor grado que el primero
- El grado de los polinomios en caso de que sea el mismo en ambos.

Ejemplos:

- Dados los polinomios $-2x -8x^2 +6x^3$ y $-2 -16x+18x^2$ la función devolverá -1
- Dados los polinomios $-2 -16x+18x^2$ y $-2x -8x^2 +6x^3$ la función devolverá -2
- Dados los polinomios $-2x -8x^2 +6x^3$ y $-2 -16x+18x^2-9x^3$ la función devolverá 3

La forma de representar los polinomios se corresponde con el siguiente ejemplo:

La representación del polinomio $7 - 2x - 8x^2 + 6x^3$ sería:

	1	2	3	4	5	6	....	51	índice vector
coeficientes	7.0	-2.0	-8.0	6.0	0.0	0.0	....	0.0	
	0	1	2	3	4	5	....	50	grado

La componente 1 del vector contendrá el coeficiente correspondiente al monomio de grado 0 del polinomio, la componente 2 el coeficiente de grado 1,..., hasta completar el vector. Si no existe monomio correspondiente a un grado su coeficiente será 0.

El grado máximo representable debe ser 50 y se reflejará en una constante.

Solución

```
Integer Function CompararGrados (polinomio1, polinomio2)
Implicit none
! Constantes
      Parameter max=51
! Argumentos
      Real polinomio1(max), polinomio2(max)
      Intent (in) polinomio1, polinomio2
! Variables
      Integer gradoPol1, gradoPol2, i
! Cuerpo de la función
      gradoPol1=0
      gradoPol2=0
      Do i=2, max
            If (polinomio1(i)/=0) Then
                  gradoPol1 = i-1
            End If
            If (polinomio2(i)/=0) Then
                  gradoPol2 = i-1
            End If
      End do
      If (gradoPol1>gradoPol2) Then
            CompararGrados = -1
      Else If (gradoPol1<gradoPol2) Then
            CompararGrados = -2
      Else
            CompararGrados = gradoPol1
      End If
End Function CompararGrados
```

12. Implementar una subrutina que calcule el límite de un cociente de polinomios cuando x → ∞.

$$\lim_{x\to\infty} \frac{P(x)}{Q(x)}$$

Recibirá dos polinomios como argumentos de entrada y los argumentos de salida serán tres: el signo del límite, el valor del límite para los casos en que este acotado (es decir, no sea ∞) y otro que tomará el valor '∞' en los casos en que el límite no esté acotado, y ' ' en el caso contrario.

Ejemplos:

- Dados los polinomios $-2x -8x^2 +6x^3$ y $-2 -16x+18x^2$ la subrutina devolverá: +, , '∞'
- Dados los polinomios $-2 -16x+18x^2$ y $-2x -8x^2 +6x^3$ la subrutina devolverá: +, 0, ' '
- Dados los polinomios $-2x -8x^2 +6x^3$ y $-2 -16x+18x^2-9x^3$ la subrutina devolverá: -, 0,66.., ' '

Téngase en cuenta:

1. Se debe llamar a la función del primer ejercicio para la resolución de éste
2. El signo del límite de un cociente de polinomios cuando x -> ∞ será positivo si coinciden los signos de los respectivos monomios de mayor grado de ambos polinomios y negativo en caso contrario.
3. El valor será ∞ si el grado del polinomio numerador es mayor que el del polinomio denominador.
4. El valor será 0 si el grado del polinomio numerador es menor que el del polinomio denominador.
5. El valor será el cociente de los respectivos coeficientes de mayor grado de los polinomios, prescindiendo del signo, si coinciden los grados de los mismos.

Solución

```
Subroutine Limite (poli1, poli2, signo, valorAcotado, infinito)
Implicit none
! Constantes
    Parameter max=51
! Argumentos
    Real poli1(max), poli2(max), valorAcotado
    Character signo, infinito
    Intent (in) poli1, poli2
    Intent (out) signo, valorAcotado, infinito
! Variables
    Integer i, grados
! Funciones
    Integer CompararGrados ! Función del ejercicio11
    Integer GradoPolinomio ! Función auxiliar que calcula el grado de
                           ! un polinomio dado
! Cuerpo de la subrutina
    grados = CompararGrados (poli1, poli2) ! Llamada a la función del ejercicio11
    If (grados == -1) Then
        valorAcotado = -9999.9999   ! Cómo es un entero no podemos asignarle
                                    ! blanco, le asignamos un valor que debería
                                    ! tratarse en el programa principal
        infinito = '∞' ! Para programarlo realmente habra que buscar el símbolo
        If (poli1(GradoPolinomio(poli1)+1)<0) Then ! Llamada a la función auxiliar
            signo = '-'
```

```
        Else
            signo = '+'
        End If
    Else If (grados == -2) Then
        valorAcotado = 0
        infinito = ' '
        signo = '+'
    Else
        infinito = ' '
        valorAcotado = abs(poli1(grados+1)/poli2(grados+1))
        If ((poli1(grados+1)>0 .AND. poli2(grados+1)>0) .OR.
(poli1(grados+1)<0 .AND. poli2(grados+1)<0)) Then
            signo = '+'
        Else
            signo = '-'
        End If
    End If

End Subroutine Limite


! Esta función se utiliza en el Ejercicio12 para facilitar la legibilidad del
! código y no ser repetitivos. Tambien podria utilizarse en el Ejercio11
! sustituyendo el Do por dos llamadas a la función:
! gradoPol1 = GradoPolinomio (polinomio1)
! gradoPol2 = GradoPolinomio (polinomio2)

Integer Function GradoPolinomio (polinomio)
Implicit none
! Constantes
    Parameter max=51
! Argumentos
    Real polinomio(max)
    Intent (in) polinomio
! Variables
    Integer i
! Cuerpo de la función
    GradoPolinomio=0
    Do i=2, max
        If (polinomio(i)/=0) Then
            GradoPolinomio = i-1
        End If
    End do

End Function GradoPolinomio
```

Se dice que un **número de N dígitos** es *"narcisista"* si resulta ser igual a la **suma de las potencias de orden N de sus dígitos**. Por ejemplo, los números 153, 1634 y 54748 son *"narcisistas"*:

$$153 = 1^3+5^3+3^3$$
$$1634 = 1^4+6^4+3^4+4^4$$
$$54748 = 5^5+4^5+7^5+4^5+8^5$$

13. Codificar una función en Fortran que indique si un número es "narcisista". El número vendrá representado por dos argumentos: un vector cuyas componentes se corresponderán con los dígitos del número y otro argumento que contendrá la cantidad de dígitos significativos del número.

Ejemplos:

Los argumentos correspondientes al número 153 son:

10	9	8	7	6	5	4	3	2	1	
0	0	0	0	0	0	0	1	5	3	3

Los argumentos correspondientes al número 1634 son:

10	9	8	7	6	5	4	3	2	1	
0	0	0	0	0	0	1	6	3	4	4

Los argumentos correspondientes al número 54748 son:

10	9	8	7	6	5	4	3	2	1	
0	0	0	0	0	5	4	7	4	8	5

El código debe ser genérico en el sentido de poder desarrollarse en un vector de N componentes con tan sólo modificar el valor de una constante (en el ejemplo el valor de la constante es 10).

De forma opcional los alumnos que deseen mejorar la calificación obtenida en la prueba de la traza de la evaluación continua pueden realizar la traza de su implementación con los números 370 y 371.

Solución:

```
Logical Function Narcisista (n, numDig)
!.....................................................................
Implicit none
! Constantes
      Parameter Max = 10
! Argumentos
      Integer n(Max), numDig
      Intent (in) n, numDig
! Variables
      Integer i, suma, valor, pot
! Comienzo parte ejecutiva
      suma = 0
      valor = 0
      pot = 1
      Do i = 1, numDig
            suma = suma + n(i)**numDig
```

```
            valor = valor + n(i) * pot  ! se podría utilizar el operador
                                          **, es menos eficiente
            pot = pot * 10
      End Do
      Narcisista = (valor == suma)
End Function Narcisista
```

---

14. Implementar una subrutina en Fortran que reciba como argumentos dos números y que apoyándose en la función del ejercicio 1., imprima por pantalla los números "narcisistas" que hay entre ellos.

Además de ser válidas las consideraciones del ejercicio anterior ha de tenerse en cuenta que se puede **llamar a la función** del ejercicio 1., sin necesidad de haberla codificado, **una vez se haya definido su cabecera**.

Nota: En ambos ejercicios se pueden implementar los subprogramas que se consideren necesarios.

Solución:

```
Subroutine NumNarcisista (num1, num2)
!.....................................................................
Implicit none
! Constantes
      Parameter Max = 10
! Argumentos
      Integer num1, num2
      Intent (in) num1, num2
! Variables
      Integer v(Max), numDigV, I, aux
! Funciones
      Logical Narcisista
! Comienzo parte ejecutiva
      Do i = num1 + 1, num2 - 1
            numDigV = 0 ! hasta el end do podría ser: CALL convertir
                        (i, v, numDigV)
            aux = i
            Do While (aux /= 0)  ! convierte el entero en un vector
                  numDigV = numDigV + 1
                  v(numDigV) = MOD (aux, 10)
                  aux = aux / 10
            End Do
! como la función narcisista no tiene en cuenta que el vector está
! relleno de ceros no hace falta rellenarlo
            If (Narcisista (v, numDigV)) Then
                  Print*, i
            End If
      End Do
End Subroutine NumNarcisista
```

---

15. Dado un número N, su crápulo es un número que se obtiene de la siguiente forma: se suman los dígitos de N, si el valor resultante tiene un único dígito ese es el crápulo y si tiene más entonces se vuelven a sumar los dígitos del resultado hasta que la suma sea un único dígito. Por ejemplo:

- 5 → crápulo 5
- 52 → crápulo 7
  5+2=7
- 97 → crápulo 7
  9+7=16 → 1+6=7
- 9859 → crápulo 1
  9+8+5+6=28 → 2+8=10 → 1+0=1

Implementar una función en frotran que reciba como argumento un número entero y que devuelva su crápulo (téngase en cuenta que se recibe el número completo y no sus dígitos)

Solución 1

```
Integer Function Crapulo (numero)
! Función que dado un número entero devuelve su crapulo, es decir la
! suma de sus cifras hasta que el resultado sea un único dígito
Implicit none
! Argumentos
  Integer numero
  Intent (in) numero
! Variables
  Integer cociente
! Cuerpo de la función
  cociente = abs(numero)      ! Cogemos el valor absoluto del número
  Crapulo = mod(cociente,10) ! Inicializamos el crapulo con el primer dígito
  Do While (cociente>9)    ! Si el nº sólo tiene un digito no entra
     cociente = cociente/10  ! Le quitamos el dígito que ya hemos sumado al
                             ! crapulo
     Crapulo = Crapulo + mod(cociente,10)  ! Sumamos otro dígito
     If (cociente<10 .and. Crapulo>9) Then ! Si ya hemos sumado todos los
                                           ! dígitos y el crapulo tiene más
           cociente = Crapulo              ! de un dígito cambiamos cociente
           Crapulo = mod (cociente,10)     ! por crapulo y volvemos a calcular
     End If
  End Do
End Function Crapulo
```

Solución 2

```
Integer Function Crapulo (numero)
! Función que dado un número entero devuelve su crapulo, teniendo en
! cuenta que el crapulo del número coincide con el resto de dividir el
! número entre 9. Si el resto es cero puede ser 0 o 9
Implicit none
! Argumentos
      Integer numero
      Intent (in) numero
! Cuerpo de la función
      Crapulo = mod(numero,9)
      If (Crapulo == 0 .and. numero /= 0) then
            Crapulo = 9
      End If
End Function Crapulo
```

16. Las cadenas de ADN están formadas por una disposición secuencial de las bases:

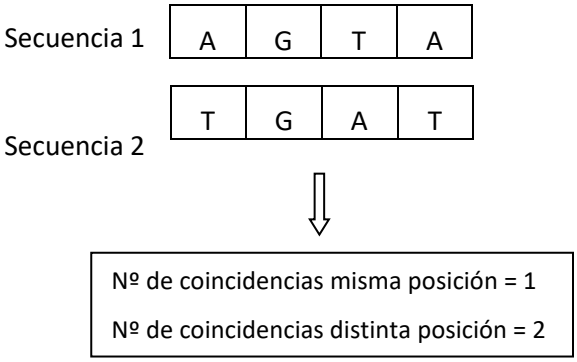
A → adenina, T → timina, C → citosina o G → guanina

Por ejemplo, una secuencia de ADN puede ser ATGCTAGATCGC...

Dos cadenas de ADN serán más similares cuantas más bases coincidan en ambas posición a posición. También se tiene en cuenta las bases coincidentes en ambas cadenas que no están en las mismas posiciones.

En una versión simplificada de este problema se pide implementar una subrutina en lenguaje Fortran que reciba dos secuencias de ADN (de tamaño 100). Como resultado proporcionará en dos argumentos de salida el número de coincidencias entre ambas secuencias en las mismas posiciones y el número de coincidencias en diferentes posiciones.

Ejemplo con dos secuencias de tamaño 4:



El código debe ser genérico en el sentido de poder desarrollarse con diferentes longitudes de las secuencias de ADN con tan sólo modificar el valor de una constante.

Nota: En ambos ejercicios se pueden implementar los subprogramas que se consideren necesarios

Solución

```
Subroutine ADN_coincidencias (secuencia1, secuencia2, coinPosi, coinDistPosi)
! Subroutina que dadas dos secuencias de ADN devuelve el numero de bases que
! tienen en la misma posición y el numero de bases coincidentes en distinta posición
Implicit none
! Constantes
  Parameter max = 100
! Argumentos
  Character(max) secuencia1, secuencia2
  Integer coinPosi, coinDistPosi
  Intent (in) secuencia1, secuencia2
  Intent (out) coinPosi, coinDistPosi
! Variables
  Character(4) bases
  Integer i, j, aux1, aux2
! Cuerpo de la subrutina
  bases = 'ATCG'
  coinPosi = 0
  coinDistPosi = 0
  Do j=1, max
     If (secuencia1 (j:j)== secuencia2(j:j)) Then
        coinPosi = coinPosi + 1
```

```
      End if
   End do
   Do i=1, 4
      aux1 = 0
      aux2 = 0
      Do j=1, max
         If (bases (i:i)== secuencia1(j:j))  Then
            aux1 = aux1 + 1
         End if
         If (bases (i:i)== secuencia2(j:j))  Then
            aux2 = aux2 + 1
         End if
      End do
      If (aux1>aux2)Then
         coinDistPosi = coinDistPosi + aux2
      Else
         coinDistPosi = coinDistPosi + aux1
      End if
   End do
   coinDistPosi = coinDistPosi - coinPosi
End Subroutine ADN_coincidencias
```

Dos números ***amigos*** son dos enteros positivos (n1, n2) tales que la suma de los divisores propios de uno de ellos es igual al otro y viceversa (la unidad se considera divisor propio, pero no lo es el mismo número).

Por ejemplo, 220 y 284 son amigos, ya que:

- Suma de divisores de 284: 1 + 2 + 4 + 71 + 142 = 220
- Suma de divisores de 220: 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284

También son números amigos 1.184 y 1.210

17. Implementar una función en lenguaje Fortran que reciba como **argumentos de entrada dos números enteros** y que **devuelva si son o no números amigos**.

Solución

```
Logical Function Amigos(numero1, numero2)
Implicit none
! Argumentos
  Integer numero1, numero2
  Intent (in) numero1, numero2
! Variables
  Integer sumaDivisores1, sumaDivisores2, i
! Cuerpo de la función
  sumaDivisores1 = 0
  sumaDivisores2 = 0
  ! Sumamos los divisores propios del primer número
  Do i = 1, numero1/2
     If (mod(numero1,i) == 0) Then  ! Si el resto de la división es 0
                                    ! entonces i es divisor de numero1
        sumaDivisores1 = sumaDivisores1 + i
     End If
  End Do
  ! Sumamos los divisores propios del segundo número
  Do i = 1, numero2/2
     If (mod(numero2,i) == 0) Then  ! Si el resto de la división es 0
                                    ! entonces i es divisor de numero2
        sumaDivisores2 = sumaDivisores2 + i
     End If
  End Do
  ! Una vez sumados los divisores propios de los dos números
  ! comprobamos si son amigos
  If (numero1 == sumaDivisores2 .AND. numero2 == sumaDivisores1) Then
     Amigos = .TRUE.
  Else
     Amigos = .FALSE.
  End If
  ! Éste If es equivalente a la asignación
  ! Amigos = (numero1==sumaDivisores2 .AND. numero2==sumaDivisores1)
End Function Amigos
```

18. Implementar una subrutina en lenguaje Fortran que, **utilizando la función del ejercicio anterior**, calcule los **números amigos** que hay en un **intervalo dado**. La subrutina debe recibir como **argumentos de entrada dos números enteros** que representan el intervalo y devolverá los números amigos en una **matriz de 2 columnas y n filas (argumento de salida)**, de forma que cada fila guardará dos números amigos contenidos en el intervalo. Ejemplo:

Si consideramos el intervalo (100,2.000) la subrutina deberá devolver la siguiente matriz

	1	2
1	220	284
2	1.184	1.210
3	0	0
.	.	.
.	.	.

Ha de tenerse en cuenta que se puede **llamar a la función** del ejercicio 1, sin necesidad de haberla codificado, **una vez se haya definido su cabecera**.

Nota: En ambos ejercicios se pueden implementar los subprogramas que se consideren necesarios.

Solución

```
Subroutine EncontrarAmigos (nIntervalo1, nIntervalo2, matrizAmigos)
Implicit none
! Constantes
  Parameter maxFilas = 100
! Argumentos
  Integer nIntervalo1, nIntervalo2, matrizAmigos(maxFilas,2)
  Intent (in) nIntervalo1, nIntervalo2
  Intent (out) matrizAmigos
! Variables
  Integer inferior, superior, contAmigos, i, j
! Funciones
  Logical Amigos
! Cuerpo de la subrutina
  Do i = 1, maxFilas    ! Inicializamos la matriz a ceros
     Do j = 1, 2
        matrizAmigos(i,j) = 0
     End Do
  End Do
  If (nIntervalo1 < nIntervalo2) Then     ! Colocamos los extremos del
     inferior = nIntervalo1               ! intervalo
     superior = nIntervalo2
  Else
     inferior = nIntervalo2
     superior = nIntervalo1
  End If
  contAmigos = 0  ! Cuenta el número de amigos en el intervalo y
                  ! marca la fila de la matriz donde serán guardados
  Do i = inferior, superior
     Do j = i, superior  ! Para excluir los números perfectos empezar en i+1
        If (Amigos(i, j)) Then
           contAmigos = contAmigos + 1
           matrizAmigos(contAmigos,1) = i
           matrizAmigos(contAmigos,2) = j
        End If
     End Do
  End Do
End Subroutine EncontrarAmigos
```

Un *centro numérico* es un número que separa una secuencia de números enteros consecutivos (comenzando en 1) en dos sub-secuencias de números, cuyas sumas son iguales. El primer *centro numérico* es el 6, el cual separa la secuencia (1 a 8) en las secuencias: (1, 2, 3, 4, 5) y (7, 8) cuyas sumas son ambas iguales a 15. El segundo *centro numérico* es el 35, el cual separa la secuencia (1 a 49) en las secuencias: (1 a 34) y (36 a 49) cuyas sumas son ambas iguales a 595.

19. Escribir una **subrutina** en fortran que reciba como **argumento de entrada un número entero** y devuelva **como argumento de salida un vector** que contenga los *centros numéricos* entre 1 y ése número dado.

Ejemplo:

num=1000 → **centrosNum**

6	35	204	0	0	...	0

Solución

```
Subroutine HallarCentrosNumericos (num, centrosNum)
Implicit none
! Constantes
    Parameter DimVector = 15
! Argumentos
    Integer num, centrosNum(DimVector)
    Intent (in) num
    Intent (out) centrosNum
! Variables
    Integer i, j, k, sumaIzq, sumaDer
      ! Declarando los argumentos y varables como 'Integer*8' podemos hallar
      ! centros numéricos para 'num' más grandes hasta ≈4.294.967.295
! Cuerpo de la subrutina
    Do i = 1, DimVector
        centrosNum(i) = 0  ! Inicializamos el vector a '0'
    End Do
    k = 0  ! Variable que nos permitirá rellenar el vector de centros numéricos
    Do i = 1, num ! Recorremos los números menores o iguales al número dado
        sumaIzq = 0 ! Inicializamos la suma de los números menores que 'i'
        Do j = 1, i - 1
            sumaIzq = sumaIzq + j
        End Do
        j = i+1     ! Sirve para calcular la suma de los números mayores que 'i'
        sumaDer = j ! Inicializamos la suma de los números mayores que 'i'
        Do While (sumaIzq > sumaDer) ! Sumamos hasta que 'sumaDer' sea mayor o
             j = j + 1                    ! igual que 'sumaIzq'
            sumaDer = sumaDer + j
        End Do
        If (sumaIzq == sumaDer) Then   ! Si 'i' es centro numérico lo metemos en
            k = k + 1                       ! el vector
            centrosNum(k) = i
        End If
    End Do
End Subroutine HallarCentrosNumericos
```

Solución optimizada

Cada vez que comprobamos si un número **'i'** es centro numérico almacenamos en 'sumaIzq' y 'sumaDer' sumas que podemos aprovechar para comprobar si el siguiente número **'i+1'** es centro numérico, ya que a 'sumaIzq' sólo habrá que sumarle 'i' para completarla y a 'sumaDer' habrá que restarle 'i+1' y continuar sumando términos a partir del último sumado.

```
Subroutine HallarCentrosNumericos (num, centrosNum)
! En ésta solución aprovechamos las sumas calculadas para los números anteriores
Implicit none
! Constantes
    Parameter DimVector = 15
! Argumentos
    Integer num, centrosNum(DimVector)
    Intent (in) num
    Intent (out) centrosNum
! Variables
    Integer i, j, k, sumaIzq, sumaDer
      ! Declarando los argumentos y varables como 'Integer*8' podemos hallar
      ! centros numéricos para 'num' más grandes hasta ≈4.294.967.295
! Cuerpo de la subrutina
    Do i = 1, DimVector
        centrosNum(i) = 0  ! Inicializamos el vector a '0'
    End Do
    k = 0 ! Variable que nos permitirá rellenar el vector de centros numéricos
    sumaIzq = 0
    sumaDer = 2
    j = 2
    Do i = 2, num
        sumaIzq = sumaIzq + i-1 ! Añadimos el siguiente término de la secuencia
        sumaDer = sumaDer - i ! Restamos el término que queremos ver si es
                              ! centro numérico y que habíamos sumado en la
                              ! iteración anterior
        Do While (sumaIzq > sumaDer)
            j = j + 1           ! 'j' guarda el último elemento sumado
            sumaDer = sumaDer + j
        End Do
        If (sumaIzq == sumaDer) Then
            k = k + 1
            centrosNum(k) = i
        End If
    End Do

End Subroutine HallarCentrosNumericos
```

20. Escribir una **función** en fortran que reciba como **argumento de entrada un número entero** y que, una vez encontrado el mayor *centro numérico* menor o igual que éste argumento de entrada, **devuelva el extremo superior** de la secuencia de la que es centro numérico. Para encontrar este centro numérico **deberá utilizarse la subrutina del ejercicio 19**, sin necesidad de haberla codificado, una vez se haya definido su cabecera. Ejemplos:

- Para **n=10** el **centro numérico mayor** es el **6** por lo que tendrá que **devolver 8** extremo superior de la secuencia **(1 a 8)**. Ver ejemplos en definición de centro numérico.
- Para **n=100** el **centro numérico mayor** es el **35** por lo que tendrá que **devolver 49** extremo superior de la secuencia **(1 a 49)**. Ver ejemplos en definición de centro numérico.

Nota: En ambos ejercicios se pueden implementar los subprogramas que se consideren necesarios

Solución

```
Integer Function ExtremoMaximo(n)
Implicit none
! Constantes
    Parameter DimVector = 15
! Argumentos
    Integer n
    Intent (in) n
! Variables
    Integer centNumericos(DimVector), i, j, maxCent, sumaAnt, sumaPos
       ! Declarando la función, los argumentos y varables como 'Integer*8' podemos
       ! hallar centros numéricos para 'n' más grandes hasta ≈4.294.967.295
! Cuerpo de la función
    Call HallarCentrosNumericos(n,centNumericos)    ! Llamada a la subrutina del
                                                    ! ejercicio1
    i= DimVector
    Do While (centNumericos(i) == 0 .And. i >1)
        i = i - 1
    End Do
    maxCent = centNumericos(i)       ! Variable donde vamos a guardar el centro
                                     ! numérico máximo. Si no existen centros
                                     ! numéricos menores que 'n' tendrá valor '0'
    sumaAnt = 0       ! Inicializamos la suma de los números menores que 'maxCent'
    Do i = 1, maxCent-1
        sumaAnt = sumaAnt + i
    End Do
    j = maxCent + 1   ! Sirve para calcular la suma de los números mayores que
                      ! 'maxCent' y el extremo
    sumaPos = j       ! Inicializamos la suma de los números mayores que 'maxCent'
    Do while (sumaAnt > sumaPos)  ! Sumamos los números mayores hasta que se
        j = j + 1                 ! igualen las sumas
        sumaPos = sumaPos + j
    End Do
    ExtremoMaximo = j ! Como sabemos que 'maxCent' es centro numérico sale del
                      ! bucle cuando las sumas son iguales y el ultimo sumando
                      ! será el extremo buscado. Se devolverá 0 en el caso en
                      ! el que 'n' sea menor que 6
End Function ExtremoMaximo
```

Solución alternativa EJERCICIOS 19 y 20

El problema puede plantearse como la igualdad de las sumas de las progresiones aritmétcas (1.. CentNum-1) y (CentNum+1..Estremo), siendo 'CentNum' el centro numérico dado y 'Extremo' el valor buscado.
Planteando la ecuación tenemos:

$$\frac{(1+(\text{CentNum}-1))\times(\text{CentNum}-1)}{2}=\frac{((\text{CentNum}+1)+\text{Extremo})\times(\text{Extremo}-\text{CentNum})}{2}$$

De donde:

$$\text{Extremo}^2+\text{Extremo}-2\times\text{CentNum}^2=0$$

También se puede deducir a partir de la primera ecuación que el centro numérico al cuadrado es igual a la suma de la progresión completa (1..Extremo):

$$\text{CentNum}^2=\frac{(1+\text{Extremo})\times\text{Extremo}}{2}$$

Desde donde llagaríamos a la misma ecuación de segundo grado.

Ejercio1

```
Subroutine HallarCentrosNumericos (num, centrosNum)
```

```
Implicit none
! Constantes
    Parameter DimVector = 15
! Argumentos
    Integer num, centrosNum(DimVector)
    Intent (in) num
    Intent (out) centrosNum
! Variables
    Integer i, k, extremo
! Cuerpo de la subrutina
    Do i = 1, DimVector
        centrosNum(i) = 0  ! Inicializamos el vector a '0'
    End Do
    k = 0  ! Variable que nos permitirá rellenar el vector de centros numéricos
    Do i = 2, num
        extremo = (-1 + sqrt(1.0+8.0*(i**2)))/2 ! Extremo de la serie si 'i'
                                                ! fuese centro numérico
        If (extremo**2+extremo==2*(i**2)) Then ! Comprobamos si 'i' es centro
           k = k + 1                           ! numérico y lo metemos en el
           centrosNum(k) = i                   ! vector
        End If
    End Do

End Subroutine HallarCentrosNumericos

! Se puede sustituir la condición del If por i == sqrt((extremo**2+extremo)/2.0)
```

Ejercio2

```
Integer Function ExtremoMaximo(n)
Implicit none
! Constantes
    Parameter DimVector = 15
! Argumentos
    Integer n
    Intent (in) n
! Variables
    Integer centNumericos(DimVector), i, maxCent
! Cuerpo de la función
    Call HallarCentrosNumericos(n, centNumericos)
    maxCent = 0
    i=1
    Do While (centNumericos(i) /= 0 .And. i <= DimVector)
        maxCent = centNumericos(i)
        i = i + 1
    End Do
    ExtremoMaximo = (-1 + sqrt(1.0+8.0*(maxCent**2)))/2 ! Resolvemos la ecuación
                                                        ! de segundo grado
End Function ExtremoMaximo
```

En diversas variantes del **juego de las damas** se da situación de captura cuando, en turno de juego, un peón encuentra en casilla diagonal contigua, en el sentido de avance, a otra ficha del contrario estando la posterior vacía. En ese caso puede saltar por encima del peón oponente hasta la casilla vacía, retirándolo del tablero.

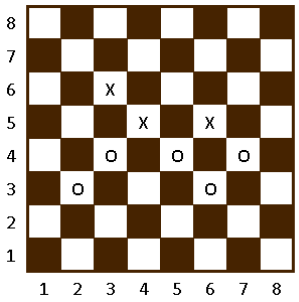
21. Implementar una **función** en lenguaje Fortran que reciba **como argumentos de entrada:**

- la posición de un peón (fila, columna), que se supone en turno de juego,
- el tipo del peón ('O' si es blanco, 'X' si es negro)
- y el tablero con la situación de la partida,

y que **devuelva** si dicho peón está en situación de capturar alguna ficha oponente.

Se entenderá que el sentido de avance se corresponde con el de la numeración creciente de las filas. Por ejemplo, si un peón está en la fila 4 sólo podría capturar a fichas oponentes de la fila 5 de cumplirse los requisitos indicados.

Ejemplo:



Para turno de juego blancas, 'O':

- Peón (3,2) ***no está*** en situación de capturar

- Peón (4,3) ***está*** en situación de capturar

Solución

```
! Función que dada una coordenada, el turno de juego y el tablero
! devuelve si en esa pasoción hay un peon en posición de captura

Logical Function Captura(fil, col, jugador, tablero)
Implicit none
! Constantes
  Parameter :: dimTablero = 8
! Argumentos
  Integer fil, col
  Character jugador, tablero(dimTablero,dimTablero)
  Intent(IN) fil, col, jugador, tablero
! Variables
  Character oponente
! Cuerpo de la función
  oponente = 'O'
  If (jugador == 'O') Then
     oponente = 'X'
  End If
  Captura = .False.

  ! Comprobamos si en la posición indicada hay una ficha del jugador en turno
  ! y si está en una fila en la que puede comer (fil<7)
  If (tablero(fil,col) == jugador.and. fil<dimTablero-1) Then

     ! Comprobamos si puede comer por la izquierda
     If (col>2 .and. tablero(fil+1,col-1) == oponente .and. tablero(fil+2,col-2) == ' ') Then
        Captura = .True.

     ! Comprobamos si puede comer por la derecha
     Else If(col<dimTablero-1 .and. tablero(fil+1,col+1)==oponente .and. tablero(fil+2,col+2) == ' ')Then
        Captura = .True.
     End If
  End If
End Function
```

22. Implementar una subrutina en lenguaje Fortran que reciba como argumentos de entrada:

- un tablero con la situación de una partida
- y el tipo de ficha que tiene el turno de juego ('O' o 'X')

y que **devuelva**, **utilizando la función del ejercicio anterior**:

- el número de fichas en situación de capturar
- y las situaciones de capturas, es decir, que indique las posiciones de los peones que pueden comer a fichas adversarias (las coordenadas deberán devolverse en **dos vectores**, uno para las posiciones de las filas y otro para el de columnas).

Ha de tenerse en cuenta que se puede **llamar a la función** del ejercicio 1, sin necesidad de haberla codificado, **una vez se haya definido su cabecera**.

Ejemplo:

Siguiendo el ejemplo del ejercicio 1, las blancas, 'O', ***tienen 3 situaciones de captura*** en las posiciones:

VectorFila

1	2	3	.	.	.	12
4	4	4	.	.	.	.

VectorColumna

1	2	3	.	.	.	12
3	5	7	.	.	.	.

Notas:

- El código debe ser genérico en el sentido de poder desarrollarse en un tablero de n x n con tan sólo modificar el valor de una constante.
- En ambos ejercicios se pueden implementar los subprogramas que se consideren necesarios.

Solución

```
! Subrutina que dados un tablero y el turno de juego, devuelve el
! número de fichas en situación de capturar para el turno de juego y
! las coordenadas de las fichas en esa situación

Subroutine SituacionesCaptura(tableroTurno, turno, numCapturas,
vFilas, vColumnas)
Implicit none
! Constantes
  Parameter dimTablero = 8, numFichas = 12 ! numFichas depende de la
                                           ! dimensión del tablero
! Argumentos
  Character tableroTurno(dimTablero, dimTablero), turno
  Integer  numCapturas, vFilas(numFichas), vColumnas(numFichas)
  Intent(IN) tableroTurno, turno
  Intent (OUT) numCapturas, vFilas, vColumnas
! Funciones
  Logical Captura
! Variables
  Integer i,j
! Cuerpo de la subrutina
  numCapturas = 0       ! Inicializamos el número de capturas
                        ! numCaptura sirve tambien como indice de los
                        ! vectores vFilas y vColumnas
  Do i = 1, dimTablero
     Do j = 1, dimTablero
        If (Captura(i,j, turno, tableroTurno)) Then  ! Llamada a la funcion
                                                     ! del ejercicio 1
```

```
            numCapturas = numCapturas + 1  ! Como hay situación de captura
                                           ! actualizamos el número de capturas
            vFilas(numCapturas) = i        ! La i marca la fila de la ficha
                                           ! con situación de captura
            vColumnas(numCapturas) = j     ! La j marca la columna de la ficha
                                           ! con situación de captura
         End If
      End Do
   End Do
End Subroutine
```
