Metodología de la Programación

Enunciados de Problemas (I)

Dpto. LSIIS. Unidad de Programación.

I. Problemas básicos.

Ejercicio 1 *(Longitud de una circunferencia)*

Calcular la longitud de una circunferencia a partir de su radio.

Ejercicio 2 *(Area de un círculo)*

Calcular el área de un círculo a partir de su radio.

Ejercicio 3 *(Area de un rectángulo)*

Calcular el área de un rectángulo a partir de su base y altura.

Ejercicio 4 *(Superficie de un cilindro)*

Calcular la superficie de un cilindro a partir del radio de la base y la altura.

Ejercicio 5 *(Volumen de un cilindro)*

Calcular el volumen de un cilindro a partir del radio de la base y la altura.

Ejercicio 6 *(Conversión de velocidad)*

Dado un valor de velocidad medido en kilometros/hora convertirlo a metros/seg.

Ejercicio 7 *(Temperatura: Celsius a Farenheit)*

Dado un valor de temperatura, medida en grados centigrados (o celsius), convertirlo a grados Farenheit.

Nota: La relacion entre grados Farenheit y Celsius es la siguiente: $^oC = 5/9(^oF - 32)$.

Ejercicio 8 *(Distancia entre 2 puntos en $\mathbb{R}^2$)*

Calcular la distancia entre dos puntos del plano $\mathbb{R}^2$.

Ejercicio 9 *(Añadir un digito a un número)*

Dado un numero $n \in \mathbb{N}^+$, añadirle un dígito d que represente las unidades en el número resultante.

Ejemplos:

- $AnyadirDigito\ (123, 5) = 1235$
- $AnyadirDigito\ (3687, 1) = 36871$
- $AnyadirDigito\ (1, 2) = 12$

Ejercicio 10 *(Valor absoluto)*

Calcular el valor absoluto de un número $n \in \mathbb{Z}$.

Ejemplos:

- $ValorAbsoluto\ (0) = 0$
- $ValorAbsoluto\ (11) = 11$
- $ValorAbsoluto\ (-34) = 34$

Ejercicio 11 *(Máximo de 2 números)*

Calcular el máximo de dos números n_1 y $n_2 \in \mathbb{Z}$.

Ejemplos:

- $Maximo2\ (-11, 23) = 23$
- $Maximo2\ (41, 128) = 128$
- $Maximo2\ (-67, -199) = -67$

Ejercicio 12 *(Máximo de 3 números)*

Calcular el máximo de tres números n_1, n_2 y $n_3 \in \mathbb{Z}$.

Ejemplos:

- $Maximo3\ (-11, 23, 89) = 89$
- $Maximo3\ (41, 12, 8) = 41$
- $Maximo3\ (-67, -19, 0) = 0$

Ejercicio 13 *(Ser un número par)*

Determinar si un número $n \in \mathbb{N}^+$ es *par* o no.

Ejercicio 14 *(Número de soluciones)*

Determinar el número de soluciones de un polinomio de segundo grado.

Ejercicio 15 *(Pertenencia a un cuadrante de $\mathbb{R}^2$)*

Determinar a qué cuadrante pertenece un punto del plano $\mathbb{R}^2$ sin considerar los ejes de coordenadas.

Ejercicio 16 *(Punto interior a un círculo)*

Determinar si un punto de $\mathbb{R}^2$ es interior a un círculo. El círculo se define en función de su radio y centro.

Ejercicio 17 *(Fecha correcta)*

Determinar si una fecha, dada como una tupla de tres números naturales, es correcta o no. Se considera que una fecha es *correcta* si:

- dia $\in \{1, \ldots, 31\}$
- mes $\in \{1, \ldots, 12\}$

Ejemplos:

- $EsFechaCorrecta\ (32, 9, 1998) = false$
- $EsFechaCorrecta\ (9, 5, 1966) = true$
- $EsFechaCorrecta\ (12, 90, 1198) = false$

Ejercicio 18 *(Ordenación de fechas)*

Dadas 2 fechas, expresadas como tres números naturales, determinar si la primera es cronológicamente menor que la segunda.

Ejercicio 19 *(Días del mes)*

Dada una fecha correcta, calcular el número de días que tiene el mes. Ampliar la solución anterior de forma que tenga en cuenta los años bisiestos.

Ejercicio 20 *(Polinomio de 2^o grado)*

Calcular las soluciones de un polinomio de segundo grado.

Ejercicio 21 *(Polinomio de 3^{er} grado)*

Calcular las soluciones de un polinomio de tercer grado.

Ejercicio 22 *(Polinomio de 4^o grado (bicuadrado))*

Definir una función que determine el número de soluciones de un polinomio de cuarto grado (bicuadrada).

Ejercicio 23 *(*Xor*: or-exclusivo)*

Calcular la función función booleana Xor, según la siguiente fórmula: $Xor(a, b) = (a \vee \neg b) \wedge (\neg a \vee b)$.

Ejercicio 24 *(Es vocal)*

Determinar si un carácter es una vocal o no.

Ejercicio 25 *(Es vocal fuerte)*

Determinar si un caracter es una vocal fuerte o no.

Ejercicio 26 *(Diptongos)*

Determinar si dos vocales forman diptongo o no.

Ejercicio 27 *(Múltiplo de un número)*

Determinar si un número es múltiplo de otro, o dicho de otra manera, si un número es divisor de otro.

Ejemplos:

- $EsMultiplo\ (42, 5) = false$
- $EsMultiplo\ (42, 1) = true$
- $EsMultiplo\ (42, 6) = true$

Ejercicio 28 *(Ordenar 2 números)*

Dados 2 números enteros devolver como resultado los números ordenados de menor a mayor.

Ejercicio 29 *(Ordenar 3 números)*

Dados 3 números enteros devolver como resultado los números ordenados de menor a mayor.

Ejercicio 30 *(Intervalos en $\mathbb{R}$)*

Determinar si un número real pertenece a un intervalo cerrado de $\mathbb{R}$. Igualmente, en el caso de que el intervalo sea abierto.

Ejercicio 31 *(Anillos de un dígito)*

Se define el número de anillos de un dígito como: *"el número de círculos que tiene el dígito"*.

Ejemplos:

- el 9 tiene 1 círculo,
- el 8 tiene 2 círculos,
- el 3 ningún círculo,
- etc..

Calcular el número de anillos de un dígito.

II. Problemas de recursividad sencillos.

Ejercicio 32 *(Número factorial)*

Calcular el número factorial de un número natural.

Ejercicio 33 *(Potencia de un número)*

Calcular la n-ésima potencia de un número, utilizando la operación producto.

Ejercicio 34 *(Números de* Fibonacci*)*

Calcular el n-ésimo número de *Fibonacci*.

Ejercicio 35 *(*Ackermann*)*

Se define la función de *Ackermann* como:

$$\begin{aligned} Ackermann(0,m) &= m+1 \\ Ackermann(n,0) &= Ackermann(n-1,1) \\ Ackermann(n,m) &= Ackermann(n-1, Ackermann(n,m-1)) \quad si, m>0 \end{aligned}$$

Calcular la función de *Ackermann*.

Ejercicio 36 *(Número de variaciones)*

Calcular el número de variaciones de n elementos tomados de m en m (el valor de V_m^n).

Ejercicio 37 *(Número de combinaciones)*

Calcular el número de combinaciones de n elementos tomados de m en m (C_m^n o $\binom{n}{m}$).

Ejercicio 38 *(Producto como sumas)*

Calcular el producto de dos números naturales en función de la suma.

Ejercicio 39 *(Potencias como productos)*

Calcular la potencia de dos números naturales (base y exponente) en función del producto.

Ejercicio 40 *(División entera)*

Calcular el resto de la división entera de dos números naturales.

Ejercicio 41 *(Cociente de una división entera)*

Calcular el cociente de la división entera de dos números naturales.

III. Problemas de manipulación de dígitos.

Ejercicio 42 *(Dígito menos significativo)*

Devolver el dígito menos significativo de un número natural.

Ejemplos:

- $DigitoMenosSignificativo\ (1234) = 4$
- $DigitoMenosSignificativo\ (673) = 3$

Ejercicio 43 *(Dígito más significativo)*

Devolver el dígito mas significativo de un número natural.

Ejemplos:

- $DigitoMasSignificativo\ (1234) = 1$
- $DigitoMasSignificativo\ (673) = 6$

Ejercicio 44 *(Añadir dígito menos significativo)*

Dado un número natural y un dígito, devolver el número resultante de añadir el dígito al número, como dígito menos significativo.

Ejemplos:

- $AnyadirDigitoMenosSignificativo\ (1234, 6) = 12346$
- $AnyadirDigitoMenosSignificativo\ (673, 9) = 6739$

Ejercicio 45 *(Añadir dígito más significativo)*

Dado un número natural y un dígito, devolver el número resultante de añadir el dígito al número, como dígito más significativo.

Ejemplos:

- $AnyadirDigitoMasSignificativo\ (1234, 6) = 61234$
- $AnyadirDigitoMasSignificativo\ (673, 9) = 9673$

Ejercicio 46 *(Número de dígitos de un número)*

Calcular el número de dígitos de un número natural.

Ejemplos:

- $NumeroDigitos\ (1234) = 4$
- $NumeroDigitos\ (673) = 3$

Ejercicio 47 *(Dígito i-ésimo de un número)*

Calcular el dígito i-ésimo de un número natural.

Ejemplos:

- $IesimoDigito\ (1234, 2) = 2$
- $IesimoDigito\ (673, 1) = 6$
- $IesimoDigito\ (87653, 4) = 5$

Ejercicio 48 *(Suma de dígitos de un número)*

Calcular la suma de los dígitos de un número natural.

Ejemplos:

- $SumaDigitos\ (1234) = 10$
- $SumaDigitos\ (4269) = 21$
- $SumaDigitos\ (3432) = 12$

Ejercicio 49 *(Suma de los dígitos pares de un número)*

Calcular la suma de los dígitos pares de un número natural.

Ejemplos:

- $SumaDigitosPares\ (1234) = 6$
- $SumaDigitosPares\ (4269) = 12$
- $SumaDigitosPares\ (3434) = 8$

Ejercicio 50 *(Intercambiar dígitos de un número)*

Dado un número natural y dos valores i y $j \in I\!N$, devolver el número resultante de intercambia los dígitos i-ésimo y j-ésimo del número inicial.

Ejemplos:

- $InterCambiar\ (1234, 1, 2) = 2134$
- $InterCambiar\ (4269, 4, 2) = 4962$

Ejercicio 51 *(Inverso de un número)*

Dado un número natural, devolver el número resultante de invertir el orden de sus dígitos.

Ejemplos:

- $Inverso\ (1234) = 4321$
- $Inverso\ (32377) = 77323$

Ejercicio 52 *(Número capicúa)*

Dadas dos secuencias, determinar si la primera secuencia es permutación de la segunda.

Ejemplos:

- $EsCapicua\ (1234) = false$
- $EsCapicua\ (43234) = true$

Ejercicio 53 *(Número binario)*

Determinar si un número natural representa un número binario, es decir, está compuesto sólo de 0's y 1's.

Ejemplos:

- $EsBinario\ (1234) = false$
- $EsBinario\ (1010) = true$
- $EsBinario\ (11) = true$

Ejercicio 54 *(Múltiplo de 3)*

Determinar si un número natural N es múltiplo de 3 o no.

Ejemplos:

- $EsMultiplo3\ (9) = false$
- $EsMultiplo3\ (1010)\ = false$
- $EsMultiplo3(89)\ = false$

Ejercicio 55 *(Producto de los dígitos de un número)*

Calcular el producto de los dígitos de un número natural.

Ejemplos:

- $ProductoDigitos(9) = 9$
- $ProductoDigitos(23) = 6$
- $ProductoDigitos(912) = 18$
- $ProductoDigitos(1234) = 24$

Ejercicio 56 *(Dígitos pares e impares de un número)*

Calcular la diferencia entre el número de dígitos pares e impares de número natural.

Ejemplos:

- $DiferenciaParesImpares\ (9) = -1$
- $DiferenciaParesImpares\ (29) = 0$
- $DiferenciaParesImpares\ (246) = 3$

Ejercicio 57 *(Frecuencia de un dígito en un número)*

Definir una función que calcule la frecuencia de aparición de un dígito d en un número natural N.

Ejemplos:

- $FrecuenciaDigito\ (1234323, 3) = 3$
- $FrecuenciaDigito\ (1234323, 2) = 2$

Ejercicio 58 *(Anillos de un número)*

Se define el número de anillos de un dígito como: *"el número de círculos que tiene el dígito"*.

Ejemplos:

- el 9 tiene 1 círculo,
- el 8 tiene 2 círculos,
- el 3 ningún círculo,
- etc..

Calcular el número de anillos de un dígito.

Ejercicio 59 *(Cambio de base)*

Escribir una función que tome un número $n \in I\!N$ –con representaciíon en base decimal– y un valor b –con $2 \leq b \leq 9$– y devuelva el número representado en base b.

Ejemplos:

- $CambioBase\ (6, 2) = 110$
- $CambioBase\ (8, 3) = 120$
- $CambioBase\ (18, 5) = 42$

IV. Problemas de recursividad con números.

Ejercicio 60 *(Suma de números)*

Calcular la suma de todos los números naturales entre N y M.

Ejercicio 61 *(Media aritmética de N números)*

Calcular la media de todos los números naturales del 1 al N, ambos inclusive.

Ejemplos:

- $Media\_1\_N(3) = 2$
- $Media\_1\_N(5) = 3$

Ejercicio 62 *(Números triangulares)*

Se dice que un número es *triangular* si se puede expresar como la suma de un numero determinado de numeros naturales empezando en el 1.

Determinar si un número natural es triangular o no.

Ejemplos:

- $EsTriangular(10) = true$, ya que $10 = 1 + 2 + 3 + 4$
- $EsTriangular(7) = false$

Ejercicio 63 *(Números perfectos)*

Se dice que un número es *perfecto*, si es amigo de si mismo, es decir, si es igual a la suma de sus divisores (sin incluirle a él mismo). Determinar si un número es perfecto o no, proponiendo dos soluciones: una recursiva y otra no recursiva.

Ejemplos:

- $EsPerfecto(6) = true$, ya que $SumaDivisores(6) = 1 + 2 + 3 = 6$
- $EsPerfecto(8) = false$, ya que $SumaDivisores(8) = 1 + 2 + 4 = 7$

Ejercicio 64 *(Números redondos)*

Calcular el n-simo *número redondo*. Los *números redondos* son aquellos que son, a la vez, múltiplos de 6, 8 y 9.

Ejercicio 65 *(Números cuadrados perfectos)*

Se dice que un número es *cuadrado perfecto*, si se puede expresar como el cuadrado de otro.

Escribir una función que determine si un número es cuadrado perfecto o no.

Ejemplos:

- $EsCuadradoPerfecto(64) = true$, ya que $64 = 8 \times 8$
- $EsCuadradoPerfecto(7) = false$

Ejercicio 66 *(Cuadrados perfectos y Triangulares)*

Calcular cuántos cuadrados perfectos hay entre 1 y N de manera que sean el doble de un número triangular.

Si la solución obtenida al problema general no es recursiva pero contiene cuantificaciones, proporcionar una solución que sí lo sea. Lo mismo se aplica a todas las funciones accesorias que aparezcan en la resolución del problema.

Ejercicio 67 *(Números primos)*

Se dice que un número es *primo* si solamente es divisible por el 1 o por si mismo.

Determinar si un número es primo o no.

Ejemplos:

- $EsPrimo(7) = true$
- $EsPrimo(8) = falso$

Ejercicio 68 *(N-ésimo número triangular)*

Se dice que un número es *triangular* si se puede expresar como la suma de un numero determinado de numeros naturales empezando en el 1. Determinar el n-ésimo número triangular.

Ejercicio 69 *(Números triangulares perfectos)*

Se dice que un número es *triangular perfecto*, si es triangular y perfecto.
Determinar si un número es triangular perfecto o no.

Ejemplos:

- $EsTriangularPerfecto(6) = true$, ya que $6 = 1 + 2 + 3$ y además $SumaDivisores(6) = 1 + 2 + 3 = 6$
- $EsTriangularPerfecto(7) = false$

Ejercicio 70 *(Número* correcto*)*

Se dice que un número $n \in \mathbb{N}$ es *correcto* si:

- $n < 10$ y par, o bien
- $n \geq 10$, y:
  - $ndiv10$ es correcto, y
  - $(nmod13) + 14$ es correcto.

Determinar si un número es correcto o no.

Ejercicio 71 *(Máximo Común Divisor (MCD))*

Calcular el *máximo común divisor* –abreviado *MCD*– de dos números naturales.

Ejemplos:

- $MCD\ (24, 180) = 12$
- $MCD\ (8, 9) = 1$

Ejercicio 72 *(*Ser par *(versión recursiva)))*

Determinar si un número $n \in \mathbb{N}$ es par, aplicando la siguiente definicion:
Un número $n \in \mathbb{N}$ es **par**, *si cumple:*

- $n = 0$, *o bien*
- $n - 2$ *es par.*

Ejercicio 73 *(Binomio de* Newton*)*

Dados a y $b \in \mathbb{R}$, y $n \in \mathbb{N}$, calcular el binomio de *Newton* definido por la fórmula:

$$(a + b)^n = \sum_{k=0}^{n} \left(\begin{array}{c} a \\ b \end{array} \right) a^{n-k} b^k$$

Problemas con túplas

Ejercicio 74 *(Sumar 1 sg a una hora)*

Devolver la hora resultante de sumarle un segundo a una hora dada como $(horas, minutos, segundos)$.

Ejemplo:

- $Sumar1Segundo\ (11, 59, 59) = (12, 0, 0)$

Ejercicio 75 *(Fecha correcta)*

Determinar si una fecha, dada como una tupla de tres números naturales, es correcta o no. Se considera que una fecha es *correcta* si:

- dia $\in \{1, \ldots, 31\}$
- mes $\in \{1, \ldots, 12\}$

Ejemplos:

- $EsFechaCorrecta\ (32, 9, 1998) = false$
- $EsFechaCorrecta\ (9, 5, 1966) = true$
- $EsFechaCorrecta\ (12, 90, 1198) = false$

Ejercicio 76 *(Convertir segundos a hh/mm/ss)*

Calcular el tiempo en formato de hh/mm/ss equivalente a un tiempo dado como segundos.

Ejemplos:

- $SegundosAHora\ (3666) = (1, 1, 6)$

Ejercicio 77 *(Convertir hh/mm/ss a segundos)*

Calcular el tiempo en segundos equivalente a un tiempo dado como: $hh/mm/ss$.

Ejemplos:

- $HoraASegundos\ (1, 1, 6) = 3666$

Ejercicio 78 *(Números Racionales)*

Calcular la suma, resta, multiplicación y división de dos numeros racionales descritos como una tupla: $(numerador, denom$

Ejercicio 79 *(Baraja de Cartas)*

Definir el tipo *TipoCarta* para una carta de la baraja española. Cada carta se representa como una *figura* y el *palo*. Definir, así mismo, un tipo que represente una mano de cartas.

- Determinar si en una mano de N cartas – $N > 2$ – hay al menos 2 reyes.
- Determinar si en una mano de N cartas – $N > 4$ – se encuentran todos los palos de la baraja.
- Determinar si dado un *palo* determinado, en una mano de N cartas – $N > 2$ – se puede cantar "las 40" –se encuentran el rey y el caballo del *palo*–.

Ejercicio 80 *(Juego del Dominó)*

Definir un tipo *TipoFichaDomino* que permita representar las fichas de dicho juego. Definir, así mismo, un tipo para representar una baza del juego –la colección de fichas que tiene un jugador–, y la partida –las bazas de los cuatro jugadores–.

Se pide:

- Determinar el valor –el número de "puntos"– de una ficha.
- Determinar si en una baza está la ficha "blanca doble".
- Determinar si en una baza está la ficha "seis doble".
- Dado un valor n, determinar el número de fichas de una baza que contienen dicho valor.
- Dado el comienzo de una partida, determinar el jugador que "abre" el domino –será aquel que tenga la ficha "seis doble"–.

Ejercicio 81 *(Números complejos)*

Calcular la suma, resta, multiplicación y división de dos números complejos.

Ejercicio 82 *(Coordenadas en el plano y el espacio)*

Definir el tipo de datos *TipoPunto* para representar puntos en el plano $\mathbb{R}^2$.

- Dado un punto del plano determinar a qué cuadrante pertenece.
- Dados tres puntos del plano determinar si se encuentran en la misma recta.

Definir el tipo de datos *TipoCoordenadas* para representar puntos en $\mathbb{R}^n$ – $n > 1$.

- Dados dos puntos de $\mathbb{R}^n$ determinar si pertenecen al mismo hiperplano.
- Dados dos puntos de $\mathbb{R}^n$ determinar la distancia entre ellos.

Ejercicio 83 *(Posición relativa entre circunferencias)*

Definir el tipo de datos *TipoCircunferencia* para representar circunferencias. Definir, así mismo, el tipo *TipoPosicionRelativa* que represente la posición relativa de 2 circunferencias, las cuales pueden ser: *coincidentes, concentricas, secantes, exteriores, interiores, tangentes exteriores o tangentes interiores.*

- Dadas dos circunferencias, determinar cuál es su posición relativa.

Ejercicio 84 *(Rectas en $\mathbb{R}^2$)*

Utilizando la definición de *TipoPunto*, definir un tipo que permita representar rectas en el plano $\mathbb{R}^2$.

- Dadas dos rectas determinar si son paralelas entre si.
- Dadas dos rectas determinar si se cruzan.

Ejercicio 85 *(Carnet de identidad)*

Definir el tipo de datos *TipoCarnet* que permita representar toda la información relacionada con una persona –nombre, apellidos, dirección, DNI, ...–. Utilizar esta representación para definir un fichero de carnets.

Se pide gestionar un archivo de información, donde se incluyen las siguientes operaciones:

- Insertar un carnet nuevo.
- Eliminar un carnet.
- Consultar un carnet.

Ejercicio 86 *(Gestión de libros)*

Definir el tipo de datos *TipoLibro* para representar toda la información relacionada con un libro. Utilizar esta representación para gestionar una biblioteca.

Se pide gestionar un archivo de información sobre libros donde se incluyen las siguientes operaciones:

- Insertar un libro nuevo.
- Eliminar un libro.
- Consultar un libro.

Ejercicio 87 *(La Tortuga saltarina)*

El lenguaje logo incluye procedimientos para dibujar figuras geométricas a partir del recorrido de una *tortuga* que puede realizar los siguientes movimientos:

TIPO	MOVIMIENTO	EFECTO
Avance	*avanzar n*	avanzar n unidades ($n \geq 0$)
Giro	*derecha, izquierda*	girar 90 o -90 grados

de manera que siempre aparece una operación de giro seguida de otra de avance. Así, un cuadrado de tamaño 4 se haría con el procedimiento siguiente:

```
derecha;
avanzar 4;
derecha;
avanzar 4;
derecha;
avanzar 4;
derecha;
avanzar 4;
```

1. Declarar un tipo *Fig_ Tortuga* que permite representar la generación de figuras con ordenes del lenguaje. Escribir en este tipo el procedimiento del cuadrado anterior.
2. Escribir una función:

```
Fig_Cerrada: Fig_Tortuga -> bool
```

que diga si una figura termina con la tortuga en el punto de partida. Así, el cuadrado es cerrado, pero el procedimiento:

```
izquierda;
avanzar 4;
derecha;
avanzar 4;
izquierda;
avanzar 4;
derecha;
avanzar 4;
```

que pinta una escalera, no lo es.

3. Escribir el axioma de inducción estructural para el tipo *Fig_ Tortuga*.

4. Demostrar, usando el axioma anterior, que *Fig_ Cerrada figura* es cierta si, y sólamente si el número de pasos que se dan en sentido Oeste-Este es igual al número de pasos que se dan en sentido Este-Oeste, y que el número de pasos en sentido Sur-Norte es igual al número de pasos en sentido Norte-Sur. Nótese que si se ha usado una función auxiliar para definir *Fig_ Cerrada* se deberá

   - especificar la propiedad que cumple la función auxiliar.
   - probar esta propiedad.
   - usar esta propiedad como un lema para probar la propiedad general.

Además, vamos a considerar figuras definidas recursivamente, mediante procedimientos con un único parámetro entero. Para ello se cuenta con una instrucción

```
switch <variable>
then   <accion 1>
else   <accion 2>
end
```

con el significado

```
si       <variable> = 0
entonces <accion 1>
si no    <accion 2>
```

y la función **dec** tal que **dec** *n* devuelve *n-1*.
El cuadrado anterior se puede ahora generalizar a:

```
proc cuadrado n;
     derecha;
     avanzar n;
     derecha;
     avanzar n;
     derecha;
     avanzar n;
     derecha;
     avanzar n;
```

Pero podemos escribir ejemplos donde aparezcan llamadas a otro procedimientos o a él mismo (recursivos) como muestra el siguiente ejemplo:

```
proc triangulo n;
     switch n
     then
     else avanzar n;
          derecha;
          avanzar 1;
          derecha;
          avanzar n;
          izquierda;
          avanzar 1;
          triangulo (dec n);
     end
end
```

que produce, para *n=4*:

5. Modificar el tipo anterior para permitir las nuevas instrucciones y las llamadas a procedimientos. Declarar, ademas, un tipo *Prog_ Tortuga* que permita representar programas formados por varios procedimientos definidos como en el ejemplo.

6. Extender la función:

   ```
   Fig_Cerrada: Fig_Tortuga * Prog_Tortuga -> bool
   ```

   que diga si una figura termina con la tortuga en el punto de partida. Así, el cuadrado es cerrado, pero el triángulo anterior no lo es.

7. Discutir que extensiones hay que hacer a la demostración anterior para probar la propiedad sobre *Fig_ Cerrada*.

Ejercicio 88 *(El calendario Juliano)*

Definir un tipo de datos que represente la información de un calendario. El calendario incluye el año al que se refiere, los 12 meses de ese año. A su vez, cada mes se define con los días que correspondan, indicando número y día de la semana.

Se pide además:

- Determinar en que día de la semana cae una fecha.
- Elegido un mes concreto, determinar cuántos lunes tiene ese mes.

Ejercicio 89 *(Partidas de Ajedrez)*

Definir un tipo que represente las piezas del ajedrez y el tablero. Escribir una función *Ataca* que diga si una determinada pieza de ajedrez situada en (x, y) ataca o no a la posición (i, j). Se supone que no hay más piezas sobre el tablero.

Ejercicio 90 *(Unidades de medida británica)*

En el sistema británico de unidades, una yarda tiene 3 pies y un pie 12 pulgadas. Se pide:

- Definir un tipo *brit* que comprenda esas tres unidades.
- Definir una función $conv : brit \rightarrow int$ que transforme a pulgadas medidas en cada una esas unidades. Ej.: $conv\ inch\ 7 \longrightarrow 7, conv\ ft\ 4 \longrightarrow 48, conv\ yd\ 2 \longrightarrow 72$.
- Definir una función $acumlis : brit\ list \rightarrow int$ que transforme a pulgadas una lista de medidas *brit*. Ej.: $acumlis\ [yd\ 3; inch\ 26; ft\ 30; inch\ 23] \longrightarrow 517$.
- Definir una función $convlis : brit\ list \rightarrow brit \times brit \times brit$ que transforme unalista de medidas en unidades británicas en una terna de la forma $(yd\ x, ft\ y, inch\ z)$, con $0 \leq y < 3, 0 \leq z < 12$. Ej.: $convlis[yd\ 3, inch\ 26, ft\ 30, inch\ 23] \longrightarrow (yd\ 14, ft\ 1, inch\ 1)$.

Ejercicio 91 *(Sistema de Ventanas)*

El objetivo de esta práctica es la implementación de algunas de las funciones del núcleo gestor de un sistema gráfico de ventanas. No se trata de realizar las operaciones de gráficos sino el tipo abstracto de datos de la representación interna que el sistema tiene del conjunto de ventanas.

Un tipo *Entorno* representa los mundos de ventanas, procurando que la estructura permita una implementación eficiente de las operaciones primitivas. El entorno contendrá, aparte de las ventanas, cierta información adicional, como una —a lo sumo— ventana activa, posición del cursor, etc. Las ventanas podrán cubrirse unas a otras parcial o totalmente. A efectos de esta práctica consideraremos una ventana simplemente como un rectángulo más una asignación de valores a atributos. Los atributos deben incluir cuestiones como colores de fondo y de figura, nombres de la ventana, procesos asociados, etc. Un tipo *Primitiva* representa tanto a las peticiones (con sus correspondientes parámetros) que se hacen al sistema de ventanas como las operaciones que el sistema genera para la implementación del servicio.

Por ejemplo, el efecto de una primitiva para poner una ventana detrás de las demás podemos ilustrarla por: Habrá que implementar una función

```
Peticion: Primitiva -> Entorno -> Entorno * Resultado * Primitiva list}
```

es decir, una operación sobre el entorno proporciona un nuevo entorno, un resultado, y una serie de operaciones de más bajo nivel. Para simplificar el problema, nos vamos a concentrar en aspectos puramente geométricos. Un posible conjunto de primitivas sería:

Posicion Para un punto en coordenadas de la pantalla debe devolverse la ventana sobre la que tal punto está situado. La aplicación obvia es saber donde está el cursor. Ha de considerarse el caso de que el punto no esté sobre ninguna ventana (esté sobre el fondo). El tipo de datos *entorno* debe estar definido de forma que esta operación se realice de forma eficiente.

Coords Conversión de coordenadas de generales a locales a una ventana dada.

Crear Añadir una ventana al entorno, en una posición dada con unas dimensiones dadas. Debe asegurarse que la nueva ventana no se salga de la pantalla. Inicialmente, la nueva ventana queda sobre las anteriores. La lista de acciones a ejecutar puede incluir el dibujo del rectángulo en cuestión.

Destruir Elimina una ventana del entorno. Además de modificar éste, esta primitiva —al igual que las *back* y *mover*— debe generar operaciones de *descubrir*. Estas operaciones representan los rectángulos que quedan al descubierto en otras ventanas y que hay que redibujar. Es decir, hay que calcular todas las regiones que pasan a ser visibles, y darlas en el orden correcto de redibujado correcto —de atrás a adelante.

Back Mandar una ventana detrás de las otras. Genera operaciones de descubrir.

Front Poner una ventana delante de las demás.

Move Mover una ventana (sin salirse de la pantalla).

Resize Cambiar una ventana de tamaño. Puede generar operaciones de descubrir.

Atrib Cambiar cierto atributo de una ventana. Este puede o no estar definido previamente para dicha ventana.

Activ Convierte cierta ventana en la activa.

Los resultados de las primitivas deben contemplar la posibilidad de generar errores (por ejemplo, cuando nos salimos de la pantalla).

Estúdiese razonadamente la complejidad —en media y en el peor caso— de las operaciones. Tened en cuenta que algunas han de ser realizadas muy a menudo (la de *Posicion* cada vez que el cursor se mueve) y es relevante la eficiencia.

Una posible mejora de esta práctica consiste en considerar un sistema de ventanas jerárquico, donde toda ventana puede ser, en principio, raíz de todo un sistema de ventanas. Para las ventanas hijas el espacio de representación está limitado por el de la ventana madre, y toda operación de movimiento, destrucción, etc, les afecta en la manera correspondiente. La creación de ventanas hijas a una dada implica la herencia de atributos. Como ejemplo, considérese el efecto de mover una ventana que contiene a otras:

Ejercicio 92 *(Fórmula de Euclides-Euler)*

La búsqueda de números perfectos es quizás el más antiguo proyecto no concluido aún por las Matemáticas. Se dice que un número es perfecto cuando su valor es igual a la suma de sus divisores. Por ejemplo:

$$6 = 1 + 2 + 3, \text{ (donde 1, 2 y 3 son los divisores de 6).}$$

Los números perfectos pares están intimamente relacionados con los números primos de la forma $2^n - 1$ (denominados primos de Mersenne), a través de la fórmula de *Euclides-Euler*, la cual se enuncia como sigue:

> *"Por cada número primo de Mersenne* $2^n - 1$*, se tiene que el número par de la forma* $2^{n-1}(2^n - 1)$ *es perfecto. Además, todos los números perfectos pares tienen esta forma."*

Se pide:

1. Programar el objeto infinito *Perfectos_Pares* a partir de la fórmula de Euclides-Euler.
2. Programar la función *N-esimo* que calcule el n-ésimo número perfecto de la sucesión infinita calculada por *Perfectos_Pares*.
3. Programar la función *N-primeros* que calcule la lista con los n-primeros números perfectos de la sucesión infinita calculada por *Perfectos_Pares*.

Problemas con secuencias

Ejercicio 93 *(Primer elemento de una secuencia)*

Calcular el primer elemento de una secuencia de números. Lo mismo en el caso de una serie de caracteres.

Ejemplos:

- $PrimerElemento\ (<1,2,3,4>) = 1$
- $PrimerElemento\ (<12,341,-12,3,4>) = 12$
- $PrimerElemento\ (<'a','b','c'>) = 'a'$

Ejercicio 94 *(Último elemento de una secuencia)*

Calcular el último elemento de una secuencia de números. Igualmente para una serie de caracteres.

Ejemplos:

- $UltimoElemento\ (<1,2,3,4>) = 4$
- $UltimoElemento\ (<12,341,-12,3,4>) = 4$
- $UltimoElemento\ (<'a','b','c'>) = 'c'$

Ejercicio 95 *(Iésimo elemento de una secuencia)*

Calcular el i-ésimo elemento de una secuencia de números. Igualmente para una serie de caracteres.

Ejemplos:

- $IesimoElemento\ (<1,2,3,4>,2) = 2$
- $IesimoElemento\ (<12,341,-12,3,4>,1) = 12$
- $IesimoElemento\ (<'a','b','c'>,3) = 'c'$

Ejercicio 96 *(Posición de un elemento en una secuencia)*

Calcular la posiciíon de la primera aparición de un elemento en una secuencia.

Ejemplos:

- $PosicionElemento\ (<1,2,3,4>,4) = 4$
- $PosicionElemento\ (<12,341,-12,3,4>,12) = 1$
- $PosicionElemento\ (<'a','b','c'>,'b') = 2$

Ejercicio 97 *(Máximo elemento en una secuencia)*

Calcular el máximo valor en una secuencia de números enteros.

Ejemplos:

- $MaximoElemento\ (<1,2,3,4>) = 4$
- $MaximoElemento\ (<12,341,-12,3,4>) = 341$

Ejercicio 98 *(Frecuencia de un elemento en una secuencia)*

Calcular la frecuencia de aparición de un elemento en una secuencia.

Ejemplos:

- $FrecuenciaElemento\ (<1,2,3,4>,2) = 1$
- $FrecuenciaElemento\ (<12,341,12,3,4>,12) = 2$
- $FrecuenciaElemento\ (<'a','b','c','a'>,'a') = 2$

Ejercicio 99 *(Pertenencia de un elemento en una secuencia)*

Determinar si un elemento pertenece o no a una secuencia.

Ejemplos:

- $PerteneceElemento(<1,2,3,4>,2) = true$
- $PerteneceElemento(<12,341,12,3,4>,1) = false$
- $PerteneceElemento(<'a','b','c','a'>,'a') = true$

Ejercicio 100 *(Multiplos de un número en una secuencia)*

Determinar cuántos valores de una serie de números enteros son múltiplos de un número dado.

Ejemplos:

- $Multiplo_K(<1,4,5,6,7,90,76>,2) = 4$
- $Multiplo_K(<-11,41,50,16,72,9,-7>,3) = 2$

Ejercicio 101 *(Suma de elementos de una secuencia)*

Calcular la suma de los elementos de una secuencia de números.

Ejemplos:

- $SumarSerie\ (<1,2,3>) = 6$
- $SumarSerie\ (<10,12,333>) = 355$
- $SumarSerie\ (<4,2,43>) = 49$

Ejercicio 102 *(Producto de los elementos de una secuencia)*

Calcular la producto de los elementos de una serie de números.

Ejemplos:

- $ProductoSerie(<1,2,3>) = 6$
- $ProductoSerie(<10,2,333>) = 6666$
- $ProductoSerie(<4,2,43>) = 344$

Ejercicio 103 *(Media aritmética de los elementos de una secuencia)*

Calcular la media aritmética de una secuencia de números.

Ejemplos:

- $MediaSerie\ (<1,2,3>) = 2$
- $MediaSerie\ (<10,12,33>) = 18.3$
- $MediaSerie\ (<4,2,42>) = 16$

Ejercicio 104 *(Desviación típica de los elementos de una secuencia)*

Calcular la desviación típica de una secuencia de números enteros.

Ejemplos:

- $DesviacionTipicaSerie(<1,2,3>) =?$
- $DesviacionTipicaSerie(<10,12,333>) =?$
- $DesviacionTipicaSerie(<4,2,43>) =?$

Ejercicio 105 *(Moda en una secuencia)*

Calcular la moda de una secuencia de números.

Ejemplos:

- $ModaSerie\ (<1,2,3,3>) = 3$
- $ModaSerie\ (<1,1,10,12,333>) = 1$
- $ModaSerie\ (<4,2,2,2,43>) = 2$

Ejercicio 106 *(Diptóngos en una secuencia de caracteres)*

Se dice que dos vocales forman *diptongo*, si la primera es una vocal fuerte –es decir, a, e o o– y la segunda una vocal débil –i o u–. Calcular el número de diptongos que hay en una secuencia de caracteres.

Ejemplos:

- $Diptongos(<' c',' a',' u',' d',' a',' l' >) = 1$
- $Diptongos(< 10, 12, 333 >) =?$
- $Diptongos(< 4, 2, 43 >) =?$

Ejercicio 107 *(Orden lexicográfico en secuencias de caracteres)*

Dadas dos secuencias de caractéres determinar si la primera es lexicográficamente "menor o igual" que la segunda.

Ejemplos:

- $OrdLexicografico\ (<' h',' o',' l',' a' >, <' h',' o',' l',' a',' s' >) = true$
- $OrdLexicografico\ (<' n',' e',' n',' e' >, <' n',' e',' n',' a' >) = false$
- $OrdLexicografico\ (<' a',' v',' e' >, <' e',' v',' a' >) = true$

Ejercicio 108 *(Sustitución de un elemento en una secuencia)*

Dada una serie S de número naturales, y dos valores N y M, también número natural, calcular la serie resultante de sustituir todas las apariciones de N en la serie S por el valor M.

Ejemplos:

- $SustituirSerie\ (< 1, 2, 3, 4 >, 3, 6) =< 1, 2, 6, 4 >$
- $SustituirSerie\ (< 1, 1, 1, 4 >, 1, 8) =< 8, 8, 8, 4 >$
- $SustituirSerie\ (< 1, 2, 3, 4 >, 5, 9) =< 1, 2, 3, 4 >$

Ejercicio 109 *(Borrado de elementos en una secuencia)*

Dada una serie S de número naturales y un valor N calcular la serie resultante de eliminar todas las apariciones de N en la serie S.

Ejemplos:

- $EliminarSerie\ (< 1, 2, 3, 4 >, 3) =< 1, 2, 4 >$
- $EliminarSerie\ (< 1, 1, 3, 4 >, 1) =< 3, 4 >$
- $EliminarSerie\ (< 1, 2, 3, 4 >, 7) =< 1, 2, 3, 4 >$

Ejercicio 110 *(Números pares en una secuencia)*

Dada una secuencia de número naturales calcular el número de elementos pares de la serie.

Ejemplos:

- $ParesEnSerie\ (<1,2,3,4>) = 2$
- $ParesEnSerie\ (<1,2,8,4>) = 3$
- $ParesEnSerie\ (<1,5,3,7>) = 0$

Ejercicio 111 *(Mayor longitud de $0's$ en una secuencia)*

Dada una serie de número naturales determinar la mayor secuencia de $0's$ que aparece en la secuencia.

Ejemplos:

- $CerosEnSerie(<1,2,0,0,4>) = 2$
- $CerosEnSerie(<1,2,120,10,4>) = 0$
- $CerosEnSerie(<1,0,0,0,4>) = 3$

Ejercicio 112 *(Parejas iguales de números en una secuencia)*

Dada una secuencia de pares de números naturales, determinar el número de pares donde los elementos que forman el par son iguales.

Ejemplos:

- $ParejasEnSerie\ (<(1,2),(2,2),(3,4)>) = 1$
- $ParejasEnSerie\ (<(1,2),(2,3),(3,4)>) = 0$
- $ParejasEnSerie\ (<(1,1),(2,2),(3,3)>) = 3$

Ejercicio 113 *(Tríos iguales de números en una secuencia)*

Dada una secuencia de tríos de números naturales, determinar el número de tríos donde los elementos que forman el trío son iguales.

Ejemplos:

- $TriosEnSerie\ (<(1,2,3),(2,2,2),(3,4,5)>) = 1$
- $TriosEnSerie\ (<(1,2,3),(2,3,4),(3,4,5)>) = 0$
- $TriosEnSerie\ (<(1,1,1),(2,2,2),(3,3,3)>) = 3$

Ejercicio 114 *(Prefijo a partir de un elementos en una secuencia)*

Calcular en una secuencia de números la subsecuencia que precede a la primera aparición de un número dado.

Ejemplos:

- $Prefijo\ (<1,2,3,4>,3) =<1,2>$
- $Prefijo\ (<1,2,3,4>,1) =<>$
- $Prefijo\ (<1,2,3,4>,5) =<>$

Ejercicio 115 *(Sufijo a partir de un elementos en una secuencia)*

Calcular en una secuencia de números la subsecuencia que sigue a la primera aparición de un número dado.

Ejemplos:

- $Sufijo\ (<1,2,3,4>,2) =<3,4>$
- $Sufijo\ (<1,2,3,4>,4) =<>$
- $Sufijo\ (<1,2,3,4>,5) =<>$

Ejercicio 116 *(Invertir una secuencia)*

Invertir el orden en que están dispuestos los elementos de una secuencia.

Ejemplos:

- $Inversa\ (<1,2,3,4>) =<4,3,2,1>$

Ejercicio 117 *(Secuencia* palíndroma*)*

Determinar si una secuencia de caracteres es palíndroma.

Ejemplos:

- $EsPalindroma\ (<'t','u','c','u','t'>) = cierto$
- $EsPalindroma\ (<'t','u','u','t'>) = cierto$
- $EsPalindroma\ (<'t','u','c'>) = falso$

Ejercicio 118 *(Átomos en una secuencia)*

A partir de una secuencia de elementos, calcular la secuencia formada por los elementos de la original sin las repeticiones.

Ejemplos:

- $Atomos\ (<1,4,2,2,3,2,4>=<1,2,3,4>$

Ejercicio 119 *(Elementos de una secuencia en otra)*

Calcular la secuencia formada por los elementos comunes a dos secuencias dadas.

Ejemplos:

- $Interseccion\ (<1,4,2,2>,<3,2,4>) =<4,2>$

Ejercicio 120 *(Prefijos de una secuencia)*

Calcular la coleccion de secuencias que forman todas las que son prefijos de una secuencia dada.

Ejemplos:

- $Prefijos\ (<2,3,1,2>) = (<2>,<2,3>,<2,3,1>)$

Ejercicio 121 *(Ser subsecuencia de una secuencia)*

Determinar si una secuencia es subsecuencia de otra, es decir, si en algun tramo de la segunda aparece la primera.

Ejemplos:

- $EsSubsecuencia\ (<1,4>,<2,3,1,2,4>) = falso$
- $EsSubsecuencia\ (<1,4>,<2,3,1,4,2>) = cierto$

Ejercicio 122 *(Subsecuencias de una secuencia)*

Calcular todas las subsecuencias de una secuencia dada.

Ejemplos:

- $Subsecuencias\ (<1,2,3>) =<<>,<1>,<2>,<3>,<1,2>,<1,3>,<2,3>,<1,2,3>>$

Ejercicio 123 *(Ser permutación de una secuencia)*

Dadas dos secuencias de números, determinar si la primera secuencia es una permutación de la segunda.

Ejemplos:

- $EsPermutacion\ (<1,2,3,4>,<2,4,1,3>) = true$
- $EsPermutacion\ (<1,2,3,4>,<1,4,1,3>) = false$

Ejercicio 124 *(Permutaciones de una secuencia)*

Dada una secuencia de números, determinar todas las secuencias que sean permutaciones de ésta.

Ejemplos:

- $Permutaciones\ (<1,2,3>) =<<1,2,3>,<1,3,2>,<2,1,3>,<2,3,1>,<3,1,2>,<3,2,1>>$

Otros problemas con secuencias

Ejercicio 125 *(Las vueltas en calderilla)*

En ciertos establecimientos existen máquinas en las que al introducir dinero se obtiene un producto y un cambio correcto. Se pide:

- Escribir la función $Calderilla: \ int\ list \times int \rightarrow int\ list$ que dado un sistema de monedas y una cantidad devuelve las monedas del cambio. Por ejemplo
$$Calderilla([100, 50, 25, 10, 5, 1], 127) \longrightarrow [100, 25, 1, 1].$$
- Hacer más realista el problema escribiendo la función *Cajero* que tenga en cuenta las existencias de cada moneda, y devuelva, junto con el dinero, las nuevas existencias. Deben tenerse en cuenta las situaciones en las que no se pueda dar el dinero solicitado y proporcionar los mensajes pertinentes.

Ejercicio 126 *(El justificador de texto)*

Escribir una función que dados una tabla con anchos de letras, un ancho de página y una lista de caracteres devuelva una lista de lineas de texto (listas de caracteres) de modo que no se partan palabras y cada linea quepa en el ancho de página. Así, para el texto

[[‘E‘;‘n‘;‘ ‘;‘u‘;‘n‘;‘ ‘;‘l‘;‘u‘;‘g‘;‘a‘;‘r‘;‘ ‘;‘d‘;‘e‘;‘ ‘;‘l‘;‘a‘;‘ ‘;‘M‘;‘a‘;‘n‘;‘c‘;‘h‘;‘a‘]]

se puede obtener una salida como

[[‘E‘;‘n‘;‘ ‘;‘u‘;‘n‘;‘ ‘;‘l‘;‘u‘;‘g‘;‘a‘;‘r‘;‘ ‘;‘d‘;‘e‘]; [‘l‘;‘a‘;‘ ‘;‘M‘;‘a‘;‘n‘;‘c‘;‘h‘;‘a‘]]

Ejercicio 127 *(Cadenas de caracteres)*

Definir el tipo *Strings* como una lista de caracteres, con las funciones: *Nth_char*, *String_length*, *String_append* (concatenación), *Sub_string* y los operadores de comparación *EQ_string* ($=$), *NEQ_string* ($\neq$), *LE_string* ($\leq$), *LT_string* ($<$), *GE_string* ($\geq$) y *GT_string* ($>$).

Ejercicio 128 *(Secuencia que aproxime a π)*

Calcular una secuencia de tamaño N de la serie numérica que define el número π.

$$\pi = 4 \times \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{2\mathsf{n} - 1}$$

Ejercicio 129 *(Secuencia que aproxime a e)*

Calcular una secuencia de tamaño N de la serie numérica que define el número irracional e.

$$e = \sum_{n=1}^{\infty} \frac{1}{\mathsf{n}!}$$

Ejercicio 130 *(Números de Hamming)*

Calcular N elementos de la secuencia de los números de *Hamming*. El conjunto de los números de *Hamming* se define como sigue:

$$Hamming = \{\ 2^i 3^j 3^k \mid i, j, k \in I\!N\}$$

Ejercicio 131 *(Secuencia de los N primeros números primos)*

Calcular los N primeros números primos.

Ejemplos:

- $NPrimos\ (3) =< 2, 3, 5 >$
- $NPrimos\ (5) =< 2, 3, 5, 7, 11 >$
- $NPrimos\ (6) =< 2, 3, 5, 7, 11, 13 >$

Ejercicio 132 *(Factores primos de un número)*

Dados un número natural N, calcular la secuencia de los números "primos" que dividen a N.

Ejercicio 133 *(Suma binaria de dos secuencias)*

Dados 2 números binarios, representados como secuencias de 0's y 1's, calcular la suma binaria de esos números.

Ejercicio 134 *(Multiplicación binaria de dos secuencias)*

Dados 2 números binarios, representados como secuencias de 0's y 1's, calcular la multiplicación binaria de esos números.

Ejercicio 135 *(Producto escalar de dos secuencias de números)*

Dados 2 secuencias de números del mismo tamaño, calcular el producto escalar de los mismos.

Ejercicio 136 *(Producto vectorial de dos secuencias de números)*

Dados 2 secuencias de números del mismo tamaño, calcular el producto vectorial de los mismos.

Ejercicio 137 *(Suma de polinómios)*

Un polinómio puede describirse, representando sus coeficientes como una secuencia de números. Calcular la suma de dos polinómios, empleando la anterior representación.

Ejercicio 138 *(Producto de polinómios)*

Un polinómio puede describirse, representando sus coeficientes como una secuencia de números. Calcular la suma de dos polinómios, empleando la anterior representación.

Ejercicio 139 *(Suma de Matrices)*

Calcular la suma de dos matrices de números reales, de tamaño $N \times M$.

Ejercicio 140 *(Producto de Matrices)*

Calcular el producto de dos matrices de números reales, de tamaño $N \times M$ y $M \times S$, respectivamente.

Ejercicio 141 *(Matriz Diagonal)*

Calcular el producto de dos matrices de números reales, de tamaño $N \times M$ y $M \times S$, respectivamente.

Ejercicio 142 *(Matriz Triangular Inferior)*

Determinar si una matriz es "*triangular inferior*".

Ejercicio 143 *(Matriz Triangular Superior)*

Determinar si una matriz es "*triangular superior*".

Ejercicio 144 *(Matriz Identidad)*

Determinar si una matriz $N \times N$ es la "*matriz identidad*".

Ejercicio 145 *(Matriz Tridiagonal)*

Determinar si una matriz $N \times N$ es uan matriz "*tridiagonal*".

Ejercicio 146 *(Matriz Tridiagonal por bloques)*

Determinar si una matriz $N \times N$ es uan matriz "*tridiagonal por bloques*".

Ejercicio 147 *(Temperatura)*

Definir un tipo de datos que permita representar, indistintamente, temperaturas en en grados Celsius o Farenheit. Así mismo, resolver el problema de trasladar una temperatura en grados FanrenheitCelsius a la equivalente en grados Celsius, y viceversa. Para ello, se utilizará la siguiente regla de conversion:

"restar 23 a la temperatura Fanrenheit y multiplicar por 5/9."

Ejercicio 148 *(Frutas)*

Definir un tipo que permita representar cualquier tipo de fruta indicando: nombre, contenido de agua, temporada y calorías por gramo. Determinar si dada una determinada fruta se tiene que:

- es *Exotica* –la clasificación de "exótica" es libre–
- es alta o baja en calorías,
- es alta o baja en contenido de agua.

Ejercicio 149 *(Notas Musicales)*

Definir un tipo que permita representar notas musicales. Cada nota musical se define como una tupla: $(figura, frecuencia, serie)$, donde:

- las figuras son: *Redonda, Blanca, Negra, Corchea, Semicorchea, Fusa, Semifusa*,
- las frecuencias son: Do, Re, Mi, Fa, Sol, La, Si, y
- la serie se define como un número $I\!N$.

Las figuras indican la duración de la nota. Cada una de las figuras tiene una duración doble que la situada a su derecha (la Blanca el doble que la Negra, la Negra el doble que la Corche, ...).

Utilizando la definición anterior, definir *TipoCompas* que permita definir una secuencia de notas.

Ejercicio 150 *(Figuras Geométricas)*

Definir el tipo de datos *TipoFigura* para representar figuras geométricas en el plano. Construir las figuras segmento, triángulo y rectángulo.

Ejercicio 151 *(Operaciones con Intervalos)*

En topología un intervalo definido sobre la recta real $\mathbb{R}$ está delimitado por dos números reales, siendo "cerrado" o "abierto", según que el número forme, respectivamente, parte o no del intervalo.

Un conjunto de intervalos se entiende como conjunto de intervalos disjuntos. Debe desarrollarse un programa que halle la unión y la intersección de conjuntos de intervalos.

Se pide:

- Definir un tipo que permita representar segmentos o intervalos en $\mathbb{R}$.
- Definir un tipo que permita representar conjuntos de intervalos en $\mathbb{R}$.
- Dados un intervalo en $\mathbb{R}$ y un número $n \in \mathbb{N}$, determinar si n pertenece al intervalo.
- Dados dos intervalos en $\mathbb{R}$ determinar si los intervalos se "solapan" – si su intersección es no vacía–.
- Dados un conjunto de intervalos en $\mathbb{R}$ y un número $n \in \mathbb{N}$, determinar si n pertenece al conjunto.
- Dados dos conjuntos de intervalos en $\mathbb{R}$ determinar su unión y su interseccion.

Ejercicio 152 *(Codigo de Gray)*

El código de Gray es una alternativa a la codificación de números en binario. La función:

```
Gray: int -> int list
```

tiene las siguientes propiedades:

1. Al igual que en en código binario:

```
Gray 0 = [0] \\
Gray 1 = [1] \\
Gray n = d::Gray (n-1) , d $\in$ \{0, 1\}, n \verb^>^ 1
```

2. *Gray n* y *Gray (n+1)* difieren exactamente en una posición, para cualquier n.

Nótese que, por simplicidad, los generamos en la lista en orden inverso a su lectura de izquierda a derecha. Por ejemplo, el número 1010 lo generamos como la lista *[0; 1; 0; 1]*.

La tabla de los primeros códigos de Gray es:

n	Gray n	binario
0	0	0
1	1	1
2	11	10
3	10	11
4	110	100
5	111	101
6	101	110
7	100	111
8	1100	1000
9	1101	1001

Se pide:

1. Especificar formalmente la función *Gray*.
2. Escribir el código correspondiente a la función *Gray*.
3. Verificar la corrección parcial de la función resultante.

Ejercicio 153 *(Árboles Binarios)*

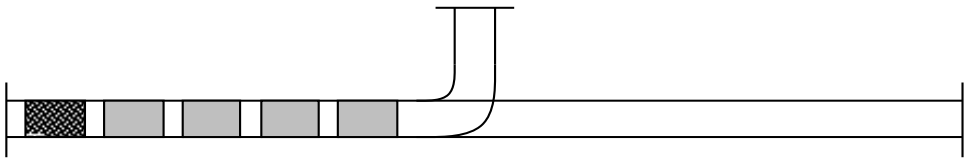
Un árbol binario es o bien un árbol vacío o bien un árbol con un nodo (e información en el mismo) y dos hijos que son árboles binarios.

Definir el tipo *TipoArbolBinario* que represente árboles binarios de números enteros. Definir la función *AplanaArbol* que tome un árbol binario y lo aplane a una lista y una función *InsertarEnArbol que inserte en orden un entero en un árbol binario.*

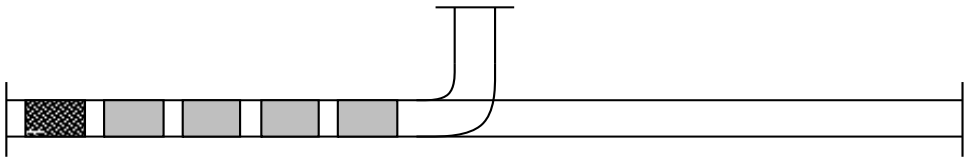
Se puede redefinir TipoArbolBinario para representar árboles binarios que sólo tengan información en las hojas. Escribir la función IgualFrontera que decida si dos árboles de este tipo tienen la misma frontera (lista de los elementos del árbol tomados de izquierda a derecha).

Ejercicio 154 *(La vía muerta)*

Se tiene un tren compuesto por una locomotora y n vagones en la siguiente disposición:



Se trata de llegar a la configuración



Los extremos del tramo largo son de longitud suficiente para dar cabida al tren completo, mientras que el corto es de poco más de un vagón. Un posible conjunto de operaciones es:

- *Mover la locomotora al extremo opuesto,*
- *Poner el cambio de agujas en posición recta,*
- *Poner el cambio de agujas en posición de desvío,*
- *Soltar la i-ésima unión contando a partir de la locomotora y*
- *Reagrupar el tren*

Se pide definir el tipo TipoOperacionesTren y la función $Tren : int \rightarrow Operaciones\_tren\ list$ *que resuelva el problema.*

Ejercicio 155 *(El Buscaminas)*

En el juego del Buscaminas *(ref. MS-Windows 3.0) se opera sobre un tablero de* $N \times M$ *casillas. Sobre dicho tablero hay distribuidas* K minas *en* K *casillas del mismo (con* $K < N \times M$*). Inicialmente, el jugador se enfrenta con un tablero que oculta todas sus casillas y deberá ir indicando al programa la casilla del tablero que quiere "destapar". En la figura 1.a el jugador ha destapado la casilla superior izquierda.*

El juego consiste en destapar todas aquellas casillas del tablero en las que no hay minas. Si en algún momento el jugador destapa alguna de las casillas en donde hay una mina, pierde y se termina el juego. La figura 1.d refleja esta situación `:-(`.

Para que la elección de la casilla no sea absolutamente aleatoria, el programa da "pistas" sobre la situación de las minas en el tablero. Así, siempre que se tiene una celda ya destapada, aparecerá, dentro de la misma, el número de minas que hay en las 8 casillas que la rodean (tal y como se puede observar en la figura 1.b y 1.c)

Se pide:

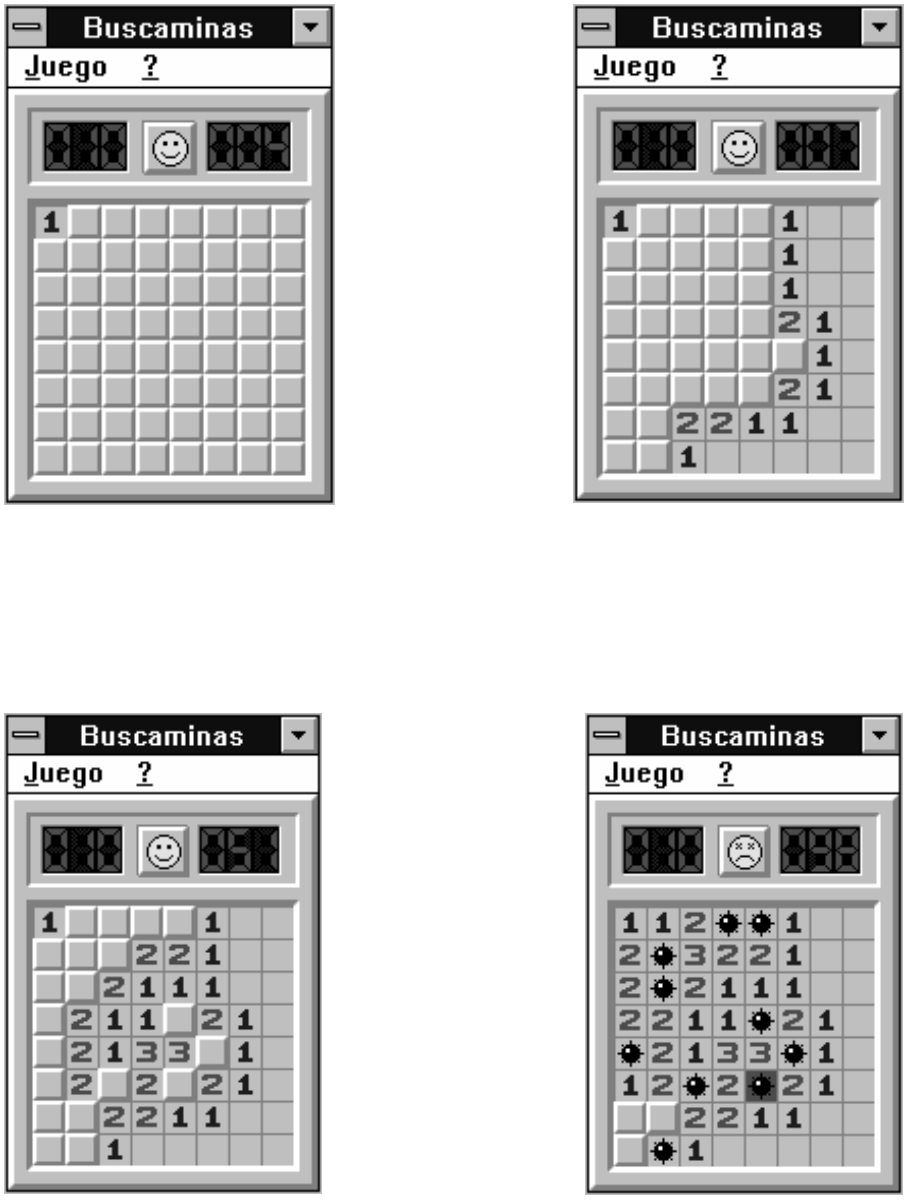


Figura 1: Etapas en el juego del buscaminas

1. *Definir en HOPE un tipo* TipoTablero *que permita representar el juego del buscaminas. Nótese que el tipo* TipoTablero *deberá ser capaz de representar, de alguna manera, toda la información necesaria para jugar sobre (número de minas que rodean a cada celda, etc.).*

   Definir la función TableroInicial*, encargada de construir un tablero de dimensiones $N \times M$, con K minas distribuidas aleatoriamente.*

2. *Programar las funciones* HayMina *y* NuevoTablero*, definidas como sigue:*

   ```
   HayMina: TipoTablero $\times$ TipoCelda -> bool
   ```

   HayMina *nos dice si una celda de un tablero esconde o no una mina.*

   ```
   NuevoTablero: Tablero -> Celda -> Tablero
   ```

   NuevoTablero *evalua la jugada de "destapar" una celda en un tablero y devuelve el tablero resultante (el cual puede ser de exito o fracaso).*

3. *Definir la interacción del juego con el jugador y la función* Jugar *que implemente el juego, permitiendo al usuario determinar el tipo de tablero que quiere y el número de minas que contiene y posteriormente mostrando las jugadas con el apoyo de la función* Dibujar*.*

4. *Definir la función de* Deshacer *que permita deshacer el último paso en la ejecución del juego.*

Ejercicio 156 *(Matrices en memoria)*

multidimensionales. Así, los elementos de una matriz de dimensión $m \times n$ se almacenan secuencialmente en memoria de la siguiente manera:

$$x_{11} \ldots x_{1n} x_{21} \ldots x_{2n} \ \ldots \ x_{m1} \ldots x_{mn}$$

Esto se generaliza para cualquier dimensión. Escribir la función Accede_dir : $int\ list \times int\ list \to int$ tal que dadas las dimensiones de la matriz y las coordenadas de un elemento, proporcione su posición relativa en memoria. Así, $AccedeDireccin\ ([3,4],[2,3]) \longrightarrow 7$.

Ejercicio 157 *(Transacciones económicas)*

Una entidad bancaria gestiona las transacciones económicas *que entre si realizan sus* entidades *colaboradoras. Para la gestión de las transacciones se tiene una operación* GestionTransacciones *que recibe una secuencia de transacciones económicas entre entidades y devuelve otra en la que se almacena cada entidad con su saldo resultante*[1]*.*

- *Cada entidad se representa por un nombre de entidad.*
- *Una transacción consta de un código de la entidad perceptora, otro de la entidad pagadora y la cantidad pagada.*
- *Cualquier entidad que aparezca en la secuencia de entrada debe aparecer una y sólo una vez en la secuencia de salida.*
- *Todas las entidades que aparecen en la secuencia de salida deben aparecer al menos una vez en la de entrada.*
- *El saldo de una entidad se calcula de la siguiente manera:*

$$Saldo = SUM_{perceptora} - SUM_{pagadora}$$

[1] ¡¡Que podría ser negativo!

donde $SUM_{perceptora}$ es la suma de todas las cantidades de las transacciones en que aparece como entidad perceptora, y $SUM_{pagadora}$ es la suma de todas las cantidades en que aparece como pagadora.

Se pide:

- *Definir tipos adecuados para representar transacciones y entidades.*
- *Definir la función* GestionTransacciones *de manera que cumpla los requisitos comentados. La solución propuesta debe intentar minimizar el número de veces que se recorre la secuencia de transacciones de entrada.*

Ejercicio 158 *(Cuadriláteros)*

Un cuadrilátero queda definido por cuatro puntos del plano que son sus vértices. En general, cuatro puntos del plano cartesiano no tienen por qué formar un cuadrilátero. Se pide:

- *Definir un tipo adecuado para representar cuadriláteros.*
- *Una función* EsCuadrilátero *que recibe un argumento del tipo anterior cuyos puntos son todos distintos y determina si forman un cuadrilátero. Cuatro puntos P_1, P_2, P_3, P_4 del plano forman un cuadrilátero si ninguna de las rectas formadas por los puntos (P_1, P_2), (P_2, P_3), (P_3, P_4) y (P_4, P_1) coinciden.*
- *Una función* ClaseCuadrilatero *que recibe un cuadrilátero sujeto a la precondición* EsCuadrilátero *y devuelve el tipo de cuadrilátero que es (*cuadrado, rectángulo, rombo, romboide, trapecio y trapezoide*).*

Para resolver este ejercicio es recomendable modelar rectas del plano y definir operaciones para saber si dos rectas son coincidentes y si son paralelas.

Ejercicio 159 *(Problemas con fechas)*

Cuando una ley modifica ciertos criterios generales es preciso, en la toma de ciertas decisiones administrativas, decidir cuantos expedientes están dentro de unas determinadas fechas de aplicación de la ley antigua.

Especificar un tipo que permita representar fechas. Definir una función Menor que permita comparar dos fechas, por ejemplo:

- Menor $(<<15$ *de Abril de* $1995>>, <<1$ *de Septiembre de* $1995>>) = cierto$
- Menor $(<<15$ *de Abril de* $1995>>, <<1$ *de Enero de* $1994>>) = falso$

Definir, después, la función FechasEnIntervalo *que decida cuántas fechas de una serie de ellas están dentro de un intervalo. El intervalo está marcado por dos fechas, como se ve en el ejemplo:*

```
FechasEnIntervalo ([«8 de Junio de 1994»,
                    «20 de Abril de 1995»,
                    «30 de Mayo de 1995»,
                    «29 de Octubre de 1995»], «1 de Enero de 1995», «1 de Agosto de 1995») = 2
```

Nota.- *Coméntese cómo influiría sobre la solución el que las fechas de la serie de entrada estuvieran ordenadas.*

Ejercicio 160 *(Generación de números aleatorios)*

La generación de números "pseudo-aleatorios" consiste en la obtención de una secuencia $\{S_i\}_{i=1...n}$ de números siguiendo una regla recurrente de construcción. Uno de estos métodos es el llamado de "los cuadrados medios", el cual se define de la siguiente manera:

- *Se tiene un valor inicial, o* semilla, *$S_0 \in \mathbb{N}$ de cuatro dígitos.*
- *Dado un número S_i de la secuencia, el siguientes valor de la secuencia se calculan de la siguiente manera:*

> "Se eleva S_i al cuadrado –esto da un número N–, y se toma como S_{i+1} el número formado por los dígitos: $3^o, 4^o, 5^o y 6^o$ del número N."

Por ejemplo:

$S_0 = 2398$ $(semilla)$ $N_0 = (2398)^2 = 5750404$

$S_1 = 5040$ $N_1 = (5040)^2 = 25401600$

$S_2 = 4016$ $N_2 = (4016)^2 = 16128256$

$S_3 = 1282$ $N_3 = (1282)^2 = 1643524$

$S_4 = 4352$ $N_4 = (4352)^2 = 18939904$

$\vdots$

$S_n = X$, *tal que* X *es algún* S_i.

La secuencia pseudo-aleatoria generada con este método no es infinita, llegando un momento en que se repite alguno de sus elementos. Se pide:

- *Dada una semilla* S_0*, calcular el tamaño de la secuencia máxima a partir de esa semilla.*
- *Dada una semilla* S_0*, calcular la secuencia máxima que puede obtenerse a partir de esa semilla.*

Ejercicio 161 *(Dias repetidos)*

A lo largo de un curso un alumno ha ido a clase de forma no uniforme. Al final de un mes se han registrado los días de la semana en los cuales asistío a clase.
Se pide:

- *Definir* TipoDiasSemana *que represente los días de la semana.*
- *Calcular la frecuencia de asistencia para cada día de la semana. Por ejemplo:*

FrecAsistencia *(<Lunes, Martes, Jueves,*
Lunes, Miercoles, Jueves, Viernes,
Martes, Miercoles, Jueves,
Lunes, Miercoles, Jueves >) =

< (Lunes, 3), (Martes, 2), (Miercoles, 3), (Jueves, 4), (Viernes, 1) >

- *Determinar en cuál de los días de la semana asistió más veces.*

Ejercicio 162 *(Codificación de texto)*

Se quiere "encriptar" mensajes descritos como secuencias de caracteres. Para ello se define una operación Encriptar, *que toma una secuencia de caracteres y devuelve otra del mismo tamaño. El proceso de encriptación utiliza una "clave" k la cual es un numero natural* $k \leq 10$*. El proceso de encriptación sigue la siguiente regla:*

- *Cada caracter c de la secuencia original se codifica en la secuencia resultado como el caracter d que cumple la siguiente relación:*

$$ord(d) = (ord(c) + k) mod 28$$

donde ord es la función:

$$ord : char -> num$$

que calcula el ordinal –código ASCII– de un caracter.

Por ejemplo, en el caso del ejemplo siguiente:

$$[`E`,`n`,` `,`u`,`n`,` `,`l`,`u`,`g`,`a`,`r`,` `,`d`,`e`,` `,`l`,`a`,` `,`M`,`a`,`n`,`c`,`h`,`a`]$$

el resultado de la encriptación sería el siguiente:

$$[`E`,`n`,` `,`u`,`n`,` `,`l`,`u`,`g`,`a`,`r`,` `,`d`,`e`,` `,`l`,`a`,` `,`M`,`a`,`n`,`c`,`h`,`a`]$$

Se pide:

- *Definir* TipoMensaje *que permita representar mensajes de caracteres.*
- *Definir la función* Encriptar *que resuelva el problema de la encriptación descrito arriba.*
- *Definir la función* Desencriptar *que deshaga la encriptación realizada por* Encriptar.

Ejercicio 163 *(Busqueda dicotómica)*

Dado un número $n \in \mathbb{N}$ determinar si n contiene un dígito determinado d.

Si consideramos que ahora el número n tiene sus dígitos ordenados de forma creciente –pudiéndose repetir varios, por ejemplo: 111235567779*–, se puede plantear una solución más efiente para el problema mediante "busqueda dicotómica".*

Ejercicio 164 *(Código Morse)*

En telegrafía, los mensajes que se desean enviar se traducen primero a un código Morse*, el cual se define mediante un alfabeto muy sencillo ($\bullet, \square$). Cada caracter del alfabeto se codifica como una secuencia única de $\bullet$ y $\square$, de tal manera que ninguna de estas codificaciones es subsecuencia de otra codificación. De esta manera, los mensajes se pueden codificar y descodificar sin confusión. A continuación se propone un ejemplo de codificación en código Morse.*

Se pide:

- *Definir* TipoCodigoMorse *que permita representar codificaciones en este código.*
- *Definir la función* Codificar *que resuelva el problema de la codificación de una cadena de caracteres a código Morse.*
- *Definir la función* Decodificar *que deshaga la encriptación realizada por* Codificar.

Ejercicio 165 *(Conversión a unidades británicas)*

En el sistema británico de unidades, una yarda *tiene 3* pies *y un pie 12* pulgadas. *Se pide:*

- *Definir un tipo $TipoMedidaBrit$ que comprenda esas tres unidades.*
- *Definir una función $Convertir : TipoMedidaBrit \to \mathbb{N}$ que transforme a pulgadas medidas en cada una esas unidades.*
- *Definir una función $Acumulacion : list(TipoMedidaBrit) \to \mathbb{N}$ que transforme a pulgadas una lista de medidas $TipoMedidaBrit$.*
- *Definir una función $ConvertirEnLista : list(TipoMedidaBrit) \to TipoMedidaBrit \times TipoMedidaBrit \times TipoMedidaBrit$ que transforme una lista de medidas en unidades británicas en una terna de la forma $(yardas, pies, pulgadas)$, con $0 \leq yardas < 3, 0 \leq pies < 12$.*

Ejercicio 166 *(Juego de Dados)*

En el juego de los dados se tiran 2 dados y se suman los valores obtenidos. A esta suma se le denomina "resultado" de la tirada.

Se pide:

- *Definir* TipoTirada *que permita representar una tirada de 2 dados.*
- *Definir la función* FrecuenciaResultado *que dada una secuencia de tiradas S, determine la frecuencia de los resultados obtenidos en las tiradas.*
- *Definir la función* VecesGanadas *que calcule el número de veces que el resultado de la tirada ha sido 7 en una secuencia de tiradas.*

Ejercicio 167 *(Operaciones con intervalos)*

En topología un intervalo definido sobre la recta real $\mathbb{R}$ está delimitado por dos números reales, siendo "cerrado" o "abierto", según que el número forme, respectivamente, parte o no del intervalo.

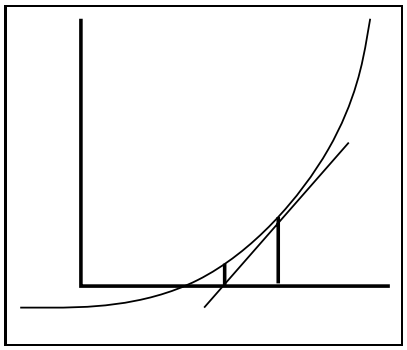
Un conjunto de intervalos se entiende como conjunto de intervalos disjuntos. Debe desarrollarse un programa que halle la unión y la intersección de conjuntos de intervalos.

Se pide:

- *Definir un tipo que permita representar segmentos o intervalos en $\mathbb{R}$.*
- *Definir un tipo que permita representar conjuntos de intervalos en $\mathbb{R}$.*
- *Dados un intervalo en $\mathbb{R}$ y un número $n \in \mathbb{N}$, determinar si n pertenece al intervalo.*
- *Dados dos intervalos en $\mathbb{R}$ determinar si los intervalos se "solapan" – si su intersección es no vacía–.*
- *Dados un conjunto de intervalos en $\mathbb{R}$ y un número $n \in \mathbb{N}$, determinar si n pertenece al conjunto.*
- *Dados dos conjuntos de intervalos en $\mathbb{R}$ determinar su unión y su interseccion.*

Ejercicio 168 *(Método de Newton)*

Realizar una función de orden superior que sirva para aplicar el método de Newton de búsqueda de ceros de una función diferenciable:



"Iterar la transformación x' = x - (f(x) / f'(x)) hasta que el error entre un valor y el siguiente sea menor que 0.0001. f'(x) es la derivada de f en el punto x."

Ejercicio 169 *(Juego de la ruleta)*

*En el "juego de la ruleta" los jugadores hacen sus apuestas poniendo dinero sobre una o varias casillas. Cada casilla está definida por un número –del 0 al 49– y por un color –*blanco, rojo o negro*–. La casilla asociada al número 0 tiene el color* blanco*, mientras que las otras se reparten los colores de la siguiente manera:*

- rojo *para los números pares, y*
- negro *para los números impares.*

El croupier *lanza una bola a la ruleta y se obtiene el valor de una casilla –número y color–.*
La tabla de premios se establece de la siguiente manera:

- *Si la bola ha caido en el número 0, los jugadores pierden el dinero apostado.*
- *Si la bola ha caido en un número* $\neq 0$*, se tiene que:*
  - *Si el jugador hizo una apuesta sobre ese número, recibe como premio 47 veces la apuesta realizada.*
  - *Si el jugador hizo la apuesta sobre una casilla cuyo color coincide con el del número donde cayó la bola el jugador recupera la apuesta.*
  - *En otro caso, el jugador pierde las apuestas realizadas*

Se pide:

- *Definir* TipoCasilla *que permita representar una casilla del tapete.*
- *Definir* TipoApuesta *que permita representar la colección de apuestas de un jugador.*
- *Dada una apuesta y el resultado de la ruleta habido, calcular el valor de las* Ganancias *de un jugador.*

Ejercicio 170 *(Número Handel-Klandel)*

Dado un número $n \in \mathbb{N}$*, cuya representación decimal sea:* $n = d_1 d_2 \ldots d_k$*, se dice que es un* número Handel-Klandel *si se cumple que* $\exists i \in \mathbb{N}, 1 \leq i \leq k$*, tal que:*

$$\sum_{n=1}^{i} d_n = \sum_{n=i+1}^{k} d_n$$

Se pide determinar si un número es Handel-Klandel.

Ejercicio 171 *(Código Morse)*

En telegrafía, los mensajes que se desean enviar se traducen primero a un código Morse*, el cual se define mediante un alfabeto muy sencillo (*$\bullet, \square$*). Cada caracter del alfabeto se codifica como una secuencia única de* $\bullet$ *y* $\square$*, de tal manera que ninguna de estas codificaciones es subsecuencia de otra codificación. De esta manera, los mensajes se pueden codificar y descodificar sin confusión. A continuación se propone un ejemplo de codificación en código Morse.*
Se pide:

- *Definir* TipoCodigoMorse *que permita representar codificaciones en este código.*
- *Definir la función* Codificar *que resuelva el problema de la codificación de una cadena de caracteres a código Morse.*
- *Definir la función* Decodificar *que deshaga la encriptación realizada por* Codificar.

Ejercicio 172 *(Números naturales largos)*

Impleméntese un tipo de datos que permita representar números naturales de cualquier número de cifras y algunas de sus operaciones.

Se pide:

1. *Definir un tipo de datos* `TipoNatural` *que permita representar números naturales de longitud no limitada y la funciíon* Convertir, *que convierte un valor n de tipo num en su representación equivalente en el tipo* `TipoNatural`. *(NOTA: Presta atención a qué representación adoptáis porque de ello depende la facilidad con que realicéis el resto de operaciones).*
2. *Realizar la función de comparación* `MenorIgual`.
3. *Realizar para este tipo las operaciones* `Suma` *y* `Resta`. *(NOTA: La resta sólo debe contemplar el caso en el que está definida en naturales, es decir, el primer operando mayor o igual que el segundo.).*

Ejercicio 173 *(Distacia entre dos listas)*

La distancia entre dos listas se define como el mínimo valor absoluto de la diferencia entre un elemento de la primera lista con un elemento de la segunda. Para encontrar una solución se ha optado por usar una función auxiliar DistanciaValorLista. *La especificación y solución (que se puede considerar verificada) de ambas funciones es la siguiente:*

FUNCIÓN DistanciaListas: Lista ($\mathbb{N}$) $\times$ Lista ($\mathbb{N}$) $\rightarrow \mathbb{N}$
PRE: lista1 $\neq$ [ ] $\wedge$ lista2 $\neq$ [ ]
DistanciaListas (lista1, lista2)
POST: resultado = min $\{|x - y|$ | Pertenece (x, lista1) , Pertenece (y, lista2)$\}$
SOL: resultado = min {DistanciaValorLista (x, lista2) | Pertenece (x, lista1)}

FUNCIÓN DistanciaValorLista: $\mathbb{N}$ $\times$ Lista ($\mathbb{N}$) $\rightarrow \mathbb{N}$
PRE: lista $\neq$ [ ]
DistanciaValorLista (x, lista)
POST/SOL: resultado = min $\{|x - y|$ | Pertenece (y, lista)$\}$

Escribir una función HOPE que implemente la solución descrita para DistanciaListas. *En dicha implementación puede suponerse que se dispone del código de la función* DistanciaValorLista, *de acuerdo a la siguiente cabecera:*

```
dec DistanciaValorLista : num # list (num) -> num;
. . .
```

por lo que el alumno no tiene que escribirlo. Si se utiliza cualquier otra función auxiliar, el alumno deberá proporcionar su código a no ser que sea una de las estudiadas en clase.

Calcular la complejidad de la función resultante. Se asume que la complejidad de la función `DistanciaValorLista` *es* $T_{DVL}(n) = 6n$.

Ejercicio 174 *(Pertenencia a Intervalos)*

Escribir la especificación *(pre y postcondición) de una función que resuelva el siguiente problema:*

> *En la topología de la recta real* $\mathbb{R}$, *un* intervalo *se define mediante sus límites inferior y superior. Cada límite se define como un número real más la propiedad de ser "cerrado" o "abierto", según que el número sea parte o no del intervalo.*
>
> *El problema consiste en obtener, a partir de un* conjunto *de intervalos* I *y un valor* $x \in \mathbb{R}$, *el* subconjunto *de intervalos de* I *que contienen a* x.

La función seguirá el siguiente esquema, que es únicamente lo que se espera que el alumno aporte como resolución del ejercicio:

FUNCIÓN IntervalosContienenPunto: Conjunto (TipoIntervalo) × $\mathbb{R}$ → Conjunto (TipoIntervalo)
PRE: cierto
IntervalosContienenPunto (I, x)
POST: resultado = $\{inter \mid inter \in$ listaI.$Esta(x, inter)\}$
DONDE: TipoClaseLimite = abierto | cerrado
TipoLimite = Limite (TipoClaseLimite × $\mathbb{R}$)
TipoIntervalo = Intervalo (TipoLimite × TipoLimite)
$Esta(x,$ Intervalo$(l_1, l_2)) = Mayor(x, l_1) \wedge Menor(x, l_2)$
$Mayor(x,$ Limite$(clase, l)) = x > l \vee (x = l \wedge clase =$ cerrado$)$
$Menor(x,$ Limite$(clase, l)) = x < l \vee (x = l \wedge clase =$ cerrado$)$

Ejercicio 175 *(Anagramas)*

Un anagrama de una lista de caracteres se forma cambiando de sitio algunos de sus elementos. Por ejemplo, "ROMA"es un anagrama de "AMOR". El problema consiste en encontrar el patrón *de un anagrama dado, que es la indicación del nuevo lugar que ocupan las letras de la lista original en el anagrama. Por ejemplo, si asumimos que las letras "AMOR"están numeradas según su posición del 1 al 4, el anagrama "ROMA"tiene el patrón [4, 3, 2, 1].*

Especificar esta función, teniendo en cuenta que no tiene sentido usarla si realmente la segunda lista no es un anagrama de la primera, es decir, si no tienen exactamente los mismos elementos. Para simplificar el problema puede suponerse que en las listas no hay repeticiones. Se asume que las funciones

FUNCIÓN Enésimo: $\mathbb{N}$ × Lista (Tipo) → Tipo
FUNCIÓN Posición: Tipo × Lista(Tipo) → $\mathbb{N}$

que, respectivamente, nos dicen el elemento que ocupa una cierta posición en una lista y la posición que ocupa un elemento en una lista, son conocidas y pueden usarse libremente en la especificación.

La contestación seguirá exactamente este esquema:

FUNCIÓN PatrónAnagrama: Lista (Caracter) × Lista (Caracter) → Lista ($\mathbb{N}$)
PRE: Longitud (lista) = Longitud (anagrama) ∧
$\forall \alpha, \beta \in \{1..$Longitud(lista)$\}. \alpha \neq \beta \longrightarrow$ (Enésimo (α, lista) ≠ Enésimo (β, lista) ∧
$\forall \gamma \in$ Caracter.$(gamma \in$ lista $\longleftrightarrow gamma \in$ anagrama)
PatrónAnagrama (lista, anagrama)
POST: Longitud (resultado) = Longitud (lista) ∧
$\forall \alpha \in \{1..$Longitud(resultado)$\}$. (Enésimo (Enésimo (α, resultado), anagrama) = Enésimo (α, lista))
o, equivalentemente, esta última parte puede sustituirse por
$\forall \alpha \in \{1..$Longitud(resultado)$\}$. Enésimo (α, resultado) = Posición (Enésimo (α, anagrama), lista)

DONDE: