Ejercicio 1.- Se quiere diseñar el TAD fecha, que contiene día, mes y año (en ese orden) como número naturales y que debe estar formado por las siguientes operaciones:

- /_/_/_/: natural natural natural→fecha
- Esanterior?: fecha fecha→bool, determina si una fecha es anterior a otra.
- Diasiguiente: fecha →fecha, deviuelve la fecha resultante de aumentar la fecha dda en un día.
- Distancia: fecha fecha→natural, calcula cuantos días de diferencia hay entre dos fechas.
- Pasardias: fecha natural→fecha, devuelve la fecha resultante tras aumentar fecha en un númeo de días.
- Coincidentes?: fecha fecha→bool, determina si dos fechas caen en el mismo día de la semana (lunes, martes, ..).

Especificar el TAD fecha, incluyendo posibles operaciones auxiliares. No es necesario considerar los años bisiestos, y puede utilizarse libremente cualquier constante natural si necesidad de definirla.

Solución:

espec FECHA

usa NATURAL, BOOL

generos fecha

operaciones

var a, m, d:natural

parcial díasdelmes: natural -->natural {Operación auxiliar}

parcial /_/_/_/: natural natural natural→fecha {Operación generadora}

es_anterior?: fecha, fecha→bool

Diasiguiente:fecha→fecha

Iguales?: fecha, fecha→bool

Distancia:fecha fecha→natural

Pasardias: fecha natural→fecha

Coincidentes?: fecha fecha→bool

ecuaciones

fun díasdelmes (m:natural):natural

si (m>0) y (m<=12) **entonces**

    si (m=2) **entonces Devolver** 28

    si no

```
            si (m= 1) V (m=3) V (m=5) V (m=7) V (m=8) V (m=10) V (m=12)
            entonces Devolver 31
            si no si (m=4) V(m=6) V(m=9) V(m=11)
                  entonces Devolver 30
      si no Devolver 0 {el mes m no es correcto}
      fsi
ffun

fun /_/_/_/ (d,m,a: natural):fecha
      si (m>0) ^ (m<=12) ^ (d>0) ^ (d<=diasdelmes (m)) entonces
                                                Devolver /d/m/a/
      fsi
ffun

fun  es_anterior? (/d1/m1/a1/, /d2/m2/a2/: fecha):boolean
      si  (a1*10000+m1*100+d1)  <  (a2*10000+m2*100+d2)
                                                entonces Devolver T
      si no Devolver F
      fsi
ffun

fun diasiguiente (/d1/m1/a1/:fecha):fecha
      si  (d1< diasdelmes(m1)) entonces Devolver /d1+1/m1/a1/
            si no si (d1=diasdelmes) y (m1<12)  entonces
                                          Devolver /d1/m1+1/a1/
                  si no si (d1=diasdelmes) y (m1=12) entonces
                                          Devolver /d1/m1/a1+1/
      fsi
ffun

fun iguales? (/d1/m1/a1/,/d2/m2/a2/:fecha): bool
```

```
        si (d1=d2) ^ (m1=m2) ^ (a1=a2) entonces Devolver T
        sino Devolver F
        fsi
ffun

fun distancia  (f1,f2:fecha) : natural
        si iguales? (f1,f2) entonces Devolver 0
                si no si es_anterior? (f1, f2) entonces
                                Devolver  1+ distancia(diasiguiente (f1), f2)
                        si no   Devolver distancia (f2, f1)
        fsi
ffun

fun pasardias (f:fecha,n:natural):fecha
        si n=0 entonces Devolver f
        si no   Devolver pasardias (diasiguiente(f), n-1)
        fsi
ffun

fun coincidentes? (f1, f2:fecha):bool
        si distancia(f1,f2)=0 entonces Devolver T
        si no si distancia (f1,f2)<7 entonces Devolver F
                si no si es_anterior?(f1, f2) entonces
                                Devolver conincidentes?(pasardias (f1,7), f2)
                        si no   Devolver conincidentes?( f1, pasardias (f2,7))
        fsi
ffun
fespec
```

Ejercicio 2.- Extender la especificación BOOLEANOS del tipo *bool* añadiendo las operaciones lógicas de implicación, equivalencia y disyunción exclusiva.

Solución:

espec BOOLEANOS

generos bool

operaciones

T: → bool

F: → bool {operaciones generadoras T y F}

¬: bool → bool

_∧_: bool bool → bool

_∨_: bool bool → bool

_ implic _:bool bool→bool {o tb. implic:bool bool→bool}

dobimplic:bool bool→bool

Notor: bool→bool

var

x: bool

fun *not* (b:bool):bool

si b **entonces Devolver** F

si no Devolver T

fsi

ffun

fun *and* (b1, b2:bool):bool

si b1 **entonces Devolver** b2

si no Devolver F

fsi

ffun

fun *or* (b1, b2:bool):bool

si b1 **entonces Devolver** T

si no Devolver b2

fsi

ffun

fun *implic* (b1, b2 :bool) :bool

Devolver or(not (b1), b2)

ffun

o también, usando la notación tradicional:

fun *implic* (b1, b2 :bool) :bool

Devolver (not b1) or b2

ffun

fun *dobimplic* (b1 ,b2:bool) bool

Devolver (b1 implic b2) and (b2 implic b1)}

ffun

fun *notor* (b1, b2:bool):bool

Devolver not (dobimplic (b1, b2))

ffun

fespec

Ejercicio 3.- Diseñar un Tipo Abstracto de Datos para trabajar con conjuntos de elementos (el tipo exacto de los elementos no se conoce, así que será un *parámetro formal* y puede suponerse una operación _==_: elemento elemento → bool), considerando que en un conjunto cada elemento aparece un sola vez. El TAD deberá tener las siguientes operaciones:

- es_vacio?: conjunto→bool, decide si un conjunto está vacío.
- coger: conjunto → elemento, devuelve uno de los elementos del conjunto.
- insertar: elemento conjunto → conjunto, para añadir un elemento a un conjunto (si se inserta un elemento a un conjunto que ya lo contiene no vuelve a ponerse);
- borrar: elemento conjunto → conjunto, para quitar un elemento de un conjunto (si se quiere quitar un elemento que no está, el conjunto no debe cambiar);
- _ ∩ _: conjunto conjunto → conjunto, hace la intersección de dos conjuntos;
- _ ∪ _: conjunto conjunto → conjunto, para hacer la unión de dos conjuntos;
- cardinal: conjunto → natural, que cuenta cuántos elementos tiene un conjunto;

- está?: elemento conjunto → bool, para comprobar si un elemento ya pertenece a un conjunto o no.

Para escribir la especificación en pseudocódigo de las operaciones indicadas se supondrá que ya disponemos de las operaciones que dependen directamente de la implementación: es_vacio, coger un elemento del conjunto, insertar y borrar un elemento.

Solución:

espec CONJUNTO [elem]

usa NATURALES, BOOLEANOS

generos conjunto

parámetro formal

espec elem

fparametro

operaciones

{todas las del enunciado }

var
e:elemento
c, c1,c2:conjunto

func está? (e:elemento, c:conjunto):bool
var pertenece:bool
uno:elemento
pertenece←F
mientras !vacio (c) **y** ¡pertenece **hacer**
uno←coger(c)
si e !=uno entonces borrar (uno, c)
si no pertenece← T
finsi
fmientras
si pertenece **entonces Devolver** T
si no Devolver F
finfunc

```
fun ∩ (c1, c2:conjunto):conjunto
        var ci:conjunto
        ci ⟵ vacio()
        mientras !es_vacio? (c1) hacer
                si esta? (coger (c1), c2) entonces
                        insertar (coger (c1), ci)
                fsi
                borrar (coger(c1), c1)

        fmientras
        Devolver ci
ffun

fun ∪ (c1, c2:conjunto):conjunto
        var cu:conjunto
        cu⟵ c1
        mientras !es_vacio? (c2) hacer
                insertar (coger (c2), cu)
                borrar (coger(c2), c2)
        fmientras
        Devolver cu
ffun
```

O también, versión recursiva:

```
fun ∪ (c1,c2:conjunto):conjunto

        si es_vacio?(c2) entonces Devolver c1

        si no Devolver ∪(insertar((coger(c2),c1), borrar (coger(c2), c2))

        fsi

ffun


fun cardinal (c:conjunto):natural

        var card:natural

        card ⟵ 0

        mientras !es_vacio? (c) hacer

                card ⟵ card+1

                quitar (coger(c), c)

        fmientras
```

Devolver card

ffun

O también, versión recursiva:

fun cardinal (c:conjunto):natural

si es_vacio**?** (c) **entonces Devolver** 0

sino Devolver 1+ cardinal (quitar (coger (c), c)

fsi

ffun

fespec

Ejercicio 5.-Especificar las cadenas finitas sobre un alfabeto dado como parámetro. Operaciones:

- Crear cadena vacía (depende de la implementación).
- Decidir si es vacía (depende de la implementación).
- Añadir un elemento por la izquierda (depende de la implementación).
- Consultar el elemento más a la izquierda (depende de la implementación).
- Eliminar el elemento más a la izquierda o primero (depende de la implementación).
- Generar una cadena unitaria formada por un elemento dado.
- Longitud.
- Consultar el elemento más a la derecha
- Eliminar el elemento más a la derecha o último.
- Decidir si elemento pertenece a cadena
- Si dos cadenas son iguales

Para escribir la especificación en pseudocódigo de las operaciones indicadas se supondrá que ya disponemos de las operaciones que dependen directamente de la implementación: Cadena Vacía, insertar por la izquierda, eliminar primero, ver primero y decidir si es vacía.

Solución:

espec CADENA[elem]

usa BOOLEANO, NATURALES

generos cadena

parámetro formal

espec elem

fparametro

operaciones

Vacia: →cadena

Es_vacia?: cadena →bool

insIzq: cadena elto →cadena

Unitaria: elto→cadena

Longitud: cadena→natural

parcial ver_ultimo:cadena→elto {la cadena no puede ser vacia, operación parcial}

Quitar_ultimo: cadena→cadena

Pertenece: elto cadena→bool

Iguales: cadena cadena→bool

ecuaciones

fun Vacia():cadena

Devolver [] **{**cadena vacía}

ffun

fun unitaria (e:elto):cadena

Devolver insIzq (e, [])

ffun

fun longitud (c:cadena):natural

var lon:natural

mientras ! (es_vacia?(c)) **hacer**

lon←lon+1

quitarPrimero(c)

fmientras

ffun

```
fun ver_ultimo(c:cadena):elto
    var ult:elto
    mientras ! (es_vacia?(c)) hacer
            ult←verPrimero(c)
            quitarPrimero(c)
    fmientras
    Devolver ult
Ffun

fun quitar_ultimo (c:cadena):cadena
    si longitud(c)=1 entonces Devolver []
    sino Devolver insIzq (verPrimero(c), quitar_ultimo(quitarPrimero(c)))
    fsi
ffun

fun pertenece (e:elto, c:cadena):bool
    si es_vacia?(c) entonces Devolver F
    sino    si e=ver_primero(c) entonces Devolver T
            sino Devolver pertenece(e, quitar_primero(c))
    fsi
ffun

fun iguales(c1,c2:cadena):bool
    si es_vacia?(c1) ^ es_vacia?(c2) entonces Devolver T
    sino    si es_vacia?(c1) V es_vacia?(c2) entonces Devolver F
            sino Devolver iguales (quitarPrimero(c1), quitarPrimero(c2))
    fsi
ffun
fespec
```

Ejercicio 6.-Una caja de seguridad tiene un cerrojo formado por tres ruedas, cada una de ellas con los dígitos 0 a 9, de forma que las ruedas pueden avanzar su valor de manera independiente sin afectar a las otras. La caja solo se puede abrir si los valores de las tres ruedas coinciden con su código secreto.

Suponiendo disponible la especificación de los naturales ampliada con la multiplicación de naturales (operación _*_: natural natural → natural), se pieden los sisguinets TAD´s y operaciones:

- Especificar el TAD rueda con las operaciones necesarias y diseñar una representación para el TAD cerrojo.
- avanzar1: cerrojo→ cerrojo, avanzar2 avanzar3, que pasan al siguiente valor de alguna de las ruedas del cerrojo (primera, segunda, tercera respectivamente)
- valor_cerrojo:cerrojo→natural, obtiene el número formado por los dígitos de las tres ruedas del cerrojo.
- cambiar:cerrojo natural→ cerrojo que gira las ruedas hasta que el valor del cerrojo coincida con natural

Solución:

espec RUEDAS
usa NATURALES
generos rueda

operaciones

parcial |_|:natural→rueda { operación generadora}
sig: rueda →rueda

ecuaciones
fun sig (|n|:rueda):rueda
si n<9 **entonces Devolver** |n+1|
si no Devolver |0|
fsi
ffun

fespec

espec CERROJOS

usa NAURALES, RUEDA
generos cerrojo

operaciones

_-_-_:rueda rueda rueda →cerrojo {operación generadora}
avanzar1→cerrjo:cerrojo
avanzar2→cerrjo:cerrojo
avanzar3→cerrjo:cerrojo

```
valor_cerrojo: cerrojo→natural
uno_mas:cerrojo:cerrojo
cambiar:cerrojo natural→cerrojo

ecuaciones
var   n1, n2, n3:natural

fun avanzar1 (|n1|-|n2|-|n3|:cerrojo):cerrojo
  Devolver sig(|n1|)-|n2|-|n3|
ffun

fun avanzar2 (|n1|-|n2|-|n3|:cerrojo):cerrojo
  Devolver |n1|-sig(|n2|)-|n3|
ffun

fun avanzar3 (|n1|-|n2|-|n3|:cerrojo):cerrojo
  Devolver |n1|-|n2|-sig(|n3|)
ffun


fun valor_cerrojo ((|n1|_|n2|_|n3|:cerrojo):natural
    Devolver (n1*100)+(n2*10)+n3
ffun


fun unomas ((|n1|_|n2|_|n3|:cerrojo)
    si (n3<9) entonces Devolver avanzar3 (|n1|_|n2|_|n3|)
    sino si (n2 <9) entonces Devolver avanzar2(|n1|_|n2|_|0|)
            sino  si (n1 <9) Devolver avanzar1(|n1|_|0|_|0|)
                    sino Devolver |0|_|0|_|0|
    fsi
ffun

fun cambiar ((|n1|_|n2|_|n3|:cerrojo, n:natural):cerrojo
    si valor_cerrojo=n entonces |n1|_|n2|_|n3|
    sino Devolver cambiar (unomas (|n1|_|n2|_|n3|), n)
    fsi
ffun


{La otra opción es no hacerlo recursivo, y usar variables auxiliares}
fun cambiar (c:cerrojo, n: natural):cerrojo
    var aux: cerrojo
    aux ← c {esta flecha es el símbolo de la asignación en pseudocódigo}
    mientras (valor_cerrojo(aux) != n) hacer
            aux ← unomas(aux)
    fmientras
    Devolver aux
ffun
fespec
```