E**jercicio 6.-** Suponiendo disponible _==_: elemento elemento → bool, que determina si dos datos de tipo elemento son iguales, extender la especificación del tipo cola vista en las clases de teoría con las siguientes operaciones (pueden ser parciales)

- contar: cola → natural, para ver cuántos elementos hay en la cola.
- último: cola → elemento, que devuelve el dato que está en última posición.
- invertir: cola → cola, que da la vuelta a los elementos de una cola.
- iguales?: cola cola → bool, que comprueba si dos colas son iguales (mismos elementos y en las mismas posiciones).
- simétrica?: cola → bool, para ver si la cola tiene los mismos datos en los dos sentidos (de primero a último y viceversa).

{Solo se muestran las operaciones nuevas, no el TAD completo}

var c, c2: cola x, y: elemento

Operaciones {Añadimos las nuevas operaciones, contar e invertir se han visto en clase}

parcial último: cola → elemento

```
fun ultimo (c):elemento                {recursiva}
var ult:elemento
        si vacía?(c) entonces error(Cola vacía)
        sino            ult ←primero(c)
        desencolar(c)
                si vacia?(c) entonces devolver ult
                sino devolver ultimo(c)
                fsi
        fsi
ffun

fun ultimo (c):elemento  {iterativa}
var e:elemento
        si vacía?(c) entonces error(Cola vacía)
        sino     e←primero(c)
                desencolar(c)
                mientras (no vacia?(c)) hacer
                        e←primero(c)
                        desencolar(c)
                fmientras
                devolver e
        fsi
```

```
ffun


fun iguales (c1,c2:cola):booleano                {recursiva}
si cvacia?(c1) ∧ cvacia?(c2) entonces devolver T
sino
        si cvacia?(c1) ∨ cvacia? (c2) entonces devolver F
        sino  si primero(c1) ==primero(c2)
              entonces
                    desencolar(c1)
                    desencolar(c2)
                    devolver iguales (c1, c2)
              sino devolver F
              fsi
        fsi
ffun
func iguales (c1,c2:cola):booleano    {iterativa}
      si vacia?(c1) ∧ vacia?(c2) entonces devolver T
      sino    si vacia?(c1) ∨ vacia? (c2) entonces devolver F
              sino
                    mientras (no vacía?(c1)) ∧ (no vacia?(c2)) ∧
                                (primero(c1) ==primero(c2))
                      hacer
                                desencolar(c1)
                                desencolar(c2)
                    fmientras
                    si vacia?(c1) ∧ vacia?(c2) entonces devolver T
                    sino devolver F
                                {si alguna no está vacía se han encontrado eltos diferentes}
                    fsi
              fsi
      fsi
ffun
```

fun simétrica (c:cola) booleano

var ci:cola ci←inversa(c)

devolver (iguales(c, ci)) **ffun**

Ejercicio 7.- Especificar el TAD colas de caracteres (se tienen las generadoras constantes para todas las letras del alfabeto y también está disponible una operación de orden para ver si una letra es anterior a otra _≤_: caracter caracter → bool , pero el resto de las posibles operaciones auxiliares hay que especificarlas), añadiendo operaciones:

- concatenar dos colas de caracteres,;
- mezclar alternativamente los elementos de dos colas de caracteres (no tienen que ser necesariamente de la misma longitud);
- quitar la primera mitad (redondeando la cantidad a la baja si es necesario) de la cola;
- comprobar si la cola está ordenada alfabéticamente;
- ver si la cola representa una palabra, entendiendo por palabra una sucesión de caracteres que no tiene dos vocales o dos consonantes seguidas.

concatenar: colac colac →colac

proc concatenar (E/S c1, c2:colac) {añade los elementos de c2 al final de c1}

mientras ¡cvacia?(c2) **hacer**

encolar (primero(c2), c1)

desencolar(c2)

fmientras

fproc

fun mezcla (c1, c2:colac):colac
var cm:colac
cm←cola_vacia
mientras (no vacia?(c1)) ∧ (no vacia?(c2)) **hacer**
encolar(primero(c1), cm)
encolar(primero(c2), cm) desencolar (c1)
desencolar (c2)
fmientras
concatenar(cm, c1)

```
        concatenar(cm, c2)
        fsi
        devolver cm
ffun


fun mezcla (c1, c2:colac):colac                    {recursiva}
var car1, car2: caracter
        si cvacia?(c1) entonces devolver (c2)
        sino  si cvacia?(c2) entonces devolver(c1)
                sino  car1←primero(c1)
                      car2←primero(c2)
                      desencolar(c1)
                      desencolar (c2)
                      devolver
                          concatenar (encolar(car2 (encolar (car1, cola_vacia))),
                                                  mezcla(c1,c2))
                fsi
         fsi
ffun
 contar:colac→natural quitar_mitad:colac→colac
proc quitar_mitad (E/S c:colac)
var i, n:natural  n←contar(c) i←nDIV2
        mientras i>0 hacer
                desencolar(c) i←i-1
        fmientras
ffun
```

palabra**:** colac→booleano

```
fun palabra (c:colac):booleano
var car:carácter var
es_palabra:booleano
es_palabra←T
        si vacia?(c) entonces devolver T
        si no
                car←primero(c)
                desencolar(c) mientras es_palabra ∧ (no vacia?(c)) hacer
                si vocal(car) = vocal(primero(c)) entonces es_palabra ←F si no
car←primero(c) desencolar (c)
                        fsi
                fmientras
        fsi
        devolver es_palabra
ffun
```

ordenada**:** colac→booleano

```
fun ordenada (c:colac):booleano
 var car:carácter si vacia?(c) entonces
devolver T sino
                car←primero(c)
                desencolar(c)
                mientras (!vacía?(c)) ∧ (car<primero(c)) hacer
                        car←primero(c)
                        desencolar(c)
                fmientras
        fsi
        devolver vacía?(c)
ffun
```

Ejercicio 8.- Usando las especificaciones BOOLEANOS y COLA[BOOLEANOS], crear operaciones:

- contar cuántos elementos están a TRUE en la cola,
- eliminar los elementos FALSE que se encuentren al comienzo de la cola,
- eliminar todos los elementos FALSE que se encuentren en la cola,
- cambiar de valor todos los elementos de la cola,
- obtener la disyunción exclusiva (operación XOR), y
- conseguir el valor lógico resultante de evaluar una cola mediante el operador XOR.

Nota: Se utilizará la operación *eq* para ver si dos elementos son iguales.

```
fun contarT(cb:cola_bool):natural
var p:booleano
        si cvacia?(cb) entonces Devolver 0
        sino  p←primero(cb)
                desencolar(cb)
                si eq(p, T) entonces Devolver 1+ contarT(cb)
                sino Devolver contarT(cb)
                fsi
        fsi
ffun

fun
contarT(cb:cola_bool):natural
var p:booleano          n←0
        mientras ¡cvacia?(cb) hacer
                 si eq(primero(cb), T)
                        entonces n←n+1
                 fsi
                desencolar(cb)
        fmientras
        devolver n
ffun
```

```
proc quitarprimerosF (E/S cb:cola_bool)
        mientras (no vacía?(cb)) ∧ eq(primero(cb),F) hacer
        desencolar(cb)
        fmientras
fproc

proc quitarF(E/S cb)
 var n:natural
        n←contarT(cb)
        cb←cola_vacia
        mientras n>0 hacer
                encolar(T, cb)
                n←n-1
        fmientras
fproc

proc cambiarvalores (E/S cb:cola_bool)
var     dato:booleano cbaux:cola_bool
        cbaux←cola_vacia
        mientras ¡vacia(cb) hacer
                dato←primero(cb)
                desencolar(cb)
                si eq(dato, T) entonces      encolar(F, cbaux)
                               sino          encolar(T, cbaux)
                fsi
        fmientras
        cb←cbaux
fproc

fun xor (b1, b2:booleano):booleano devolver
        (no eq(b1,b2))
ffun
```

```
xorcolab: colab→booleano
fun xorcolab (cb:colab):booleano
var dato1: booleano var dato2: booleano
    si vacía?(cb) entonces error(Cola vacía)
    si no   dato1←primero(cb)
            desencolar(cb)
            mientras (no vacia?(cb)) hacer
                dato2←primero(cb)
                desencolar(cb)
                dato1←xor(dato1,dato2)
            fmientras
    fsi
    devolver dato1
finfunc
```

Ejercicio 9.-Escribir en pseudocódigo la operación mezclar dos colas del ejercicio 8 considerando la implementación basada en memoria dinámica.

```
func mezclar (c1, c2:cola):cola
{se destruyen las colas y se copian los datos de las colas en posiciones de
memoria nuevas}
var cm:cola
reservar(cm.primero)
    cm.ultimo=cm.primero

mientras ¡ (vacia(c1)) ^!( vacia(c2)) hacer  {mezclamos elementos de ambas colas}
enlace-cola borrado
    borrado=c1.primero
    cm.ultimo^.valor=c1.primero^.valor
    c1.primero ← c1.primero ^.sig
    borrado^sig=nil
    delete borrado {liberamos memoria]
    reservar(cm.ultimo^.sig)
    cm.ultimo← cm.ultimo^.sig
    borrado=c2.primero
    cm.ultimo^.valor= c2.primero ^.valor
    c2.primero ← c2.primero^.sig
    borrado^sig=nil
    delete borrado
finmientras
```

{falta ahora lo que quede de c1 o de c2}

mientras no (vacia(c1)) **hacer**
reservar(cm.ultimo^.sig)
cm.ultimo← cm.ultimo^.sig
cm.ultimo^.valor= c1.primero ^.valor c1.primero ← c1.primero ^.sig
finmientras

mientras no(vacia(c2)) **hacer**
reservar(cm.ultimo^.sig)
cm.ultimo← cm.ultimo^.sig
cm.ultimo^.valor= c2.primero ^.valor c2.primero ← c2.primero^.sig
finmientras

cm.ultimo^.sig← nil
devolver cm
finfunc

```
func mezclar (c1, c2:cola):cola          {destruye las colas}
var cm:cola
cm.primero←c1.primero
cm.ultimo←c1.primero
c1.primero←c1.primero^.sigue
mientras no (vacia(c1)) ^ no ( vacia(c2)) hacer
        cm.ultimo^.sigue←c2.primero
        c2.primero←c2.primero^.sigue
                cm.ultimo←cm.ultimo^.sigue
        cm.ultimo^.sigue←c1.primero
        c1.primero←c1.primero^.sigue
finmientras
```

si no (vacia(c2)) **entonces** {quedan nodos en c2}
cm.ultimo^.sigue←c2.primero
cm.ultimo←c2.ultimo
finsi

si no (vacia(c1)) entonces {quedan nodos en c1}
cm.ultimo←c1.ultimo
finsi

cm.ultimo^.sigue←nil {indicamos el fin de la cola mezcla}
c1.ultimo←nil
c2.ultimo←nil
devolver (cm)
finfunc