Ejercicio 1.- Extender la especificación PILA[ELEMENTO] del tipo pila visto en clase añadiendo las siguientes operaciones (pueden ser parciales):

- contar: pila → natural, para ver cuántos elementos tiene la pila.
- fondo: pila → elemento, que consulta el elemento más profundo de la pila.
- montar: pila pila → pila, para poner la segunda pila encima de la primera (respetando el orden de los elementos).
- quitar: pila natural → pila, que quita tantos elementos de la pila como indica el parámetro natural; por ejemplo, quitar(p, 3) eliminaría tres elementos de la pila.

Extendemos la especificación de las pilas de la siguiente manera:
espec EJ1_PILA[ELEMENTO] {la especificación del ejercicio 1}
usa PILA[ELEMENTO]
{usamos la especificación base, no cambia ni el géneros ni el parámetro formal, añadimos las nuevas operaciones en pseudocódigo}

operaciones

contar: pila → natural
fun contar (p:pila): natural
var n:natural n←0
    mientras !vacia?(p) **hacer**
        Desapilar(p)
        n←n+1
    fmientras
devolver n
ffun

parcial fondo: pila →elemento {la pila tiene que tener datos }
fun fondo (p:pila):elemento
var e:elemento
    si vacía?(p) **entonces error**(pila vacía)
    mientras (no vacía?(p)) **hacer** {al menos hay un elto en la pila}
        e←cima(p)
        desapilar(p)
    fmientras
    devolver e
ffun

Jesús Lázaro
Mª José Domínguez

montar: pila pila → pila
proc montar (p1, p2:pila) {recursivo}
var e:elemento
si !vacia?(p2) **entonces**
e←cima(p2)
desapilar(p2)
montar(p1, p2)
apilar (e, p1)
fsi
fproc

invertir:pila→pila {visto en clase}
proc montar (p1, p2:pila) {Iterativo}
var pi:pila e:elemento
pi←invertir(p2)
mientras ¡vacia?(pi) **hacer**
e←cima(pi)
desapilar(pi)
apilar(e,p1)
fmientras
fproc

parcial quitar: pila natural → pila
{esta operación es parcial porque la pila tiene que tener como poco tantos elementos como se quieren quitar}
proc quitar (p:pila, n:natural)
si (n>contar(p)) **entonces error**(no hay suficientes datos)
sino
mientras (n>0) **hacer**
desapilar(p)
n←n-1
fmientras
fsi
fproc

Jesús Lázaro
Mª José Domínguez

Ejercicio 2.- (Examen del Grado en Ingeniería en Informática, Noviembre 2012) Suponiendo conocida la especificación PILA[ELEMENTO], y suponiendo que el TAD de elemento tiene una operación _≤_*: elemento elemento → bool*, que comprueba si un elemento es menor o igual que otro, crear operaciones para:

- contar cuántos elementos hay en una pila.
- obtener la inversa de una pila.
- comprobar si los elementos de una pila fueron introducidos en orden de mayor a menor (el mayor debería estar en el fondo de la pila, y el menor en la cima).
- comprobar si los elementos de una pila fueron introducidos en orden de menor a mayor (el menor debería estar en el fondo de la pila, y el mayor en la cima).
- eliminar el elemento que se encuentre en el fondo de la pila.

```
invertir_aux: pila pila →pila          {procedimiento auxiliar, invierte p2 en p1}
proc invertir_aux (p1, p2: pila)
        si !vacia?(p2) entonces
                apilar(cima(p2), p1))
                desapilar(p2)
                invertir_aux(p1, p2)
        fsi
fproc

invertir:pila→pila
fun invertir(p:pila):pila
        Invertir_aux (pvacia, p)
ffun

parcial mayorquecima:elemento pila→bool
{f. auxiliar que comprueba si la cima de la pila es mayor o igual que elto. Es parcial
porque la pila no puede ser vacia}
fun mayorquecima (e:elemento, p:pila) :bool
        si e>= cima (p) entonces devolver T
                        sino devolver F
        fsi
ffun
```


Jesús Lázaro
Mª José Domínguez

```
demayoramenor:pila→boolean
fun demayoramenor (p:pila):bool
var e1: elemento
        si vacia(p) entonces Devolver T
        sino    e1←cima(p)
                desapilar(p)
                si  mayorquecima (e1, p)  entonces
                                          devolver demayoramenor(p)
                                          sino devolver F
        fsi
ffun


demenoramayor:pila→boolean
fun demenoramayor (p:pil): boolean
        var pi:pila
                pi←invertir(p)
                devolver demayoramenor(pi)
ffun

parcial eliminarfondo: pila→pila          {debe haber al menos un elemento en la
pila}
proc eliminarfondo (p:pila)
var pi:pila
        si vacia (p) entonces error(pila vacía)
        sino
                pi←invertir(p)
                desapilar(pi)
                devolver invertir(pi)
        fsi
ffun
```

Ejercicio 3.- *(Examen del Grado en Ingeniería de Computadores, Noviembre 2012).* Dar la especificación del TAD básico PILA[ENTERO]. Extender dicha especificación con operaciones adicionales para (pueden ser parciales):

- sacar_en_pos: pila natural → pila, que elimina el número entero que se encuentra en la posición indicada por el parámetro natural, siendo la posición número uno la cima; por ejemplo, sacar_en_pos(p,2) debería quitar el dato que está justo debajo de la cima de p.

- sacar_entre: pila natural natural → pila, que elimina de la pila todos los enteros que se encuentren entre las posiciones indicadas por los parámetros naturales; así, sacar_entre(p, 2, 4) quitaría los elementos que están en las posiciones 2, 3 y 4.

spec EJE3_PILA[ENTERO]
usa NATURAL2
parámetro formal
    espec entero
fparametro
generos pila
operaciones (vistas en clase)
pvacia:pila→pila
apilar:pila entero→pila
parcial desapilar.pila→pila
parcial cima:pila→entero
vacia?:pila→booleano

operaciones que extienden la especificación
parcial sacar_en_pos: pila natural→pila
{Debe haber al menos n enteros en la pila}
proc sacar_en_pos (p:pila, n:natural)
var e:entero
    si (n>0) **entonces**
        si vacía?(p) **entonces error**(no hay suficientes datos)
        sino
            e←cima(p)
            desapilar(p)
            sacar_en_pos(p, n-1)
            si (n≠1) **entonces** apilar(e, p) **fsi**
        fsi
    fsi
fproc
parcial sacar_entre:pila natural natural→pila
proc sacar_entre (p:pila, n,m:natural)
    si n=m **entonces** sacar_en_pos (p, n) {el posible error lo detecta sacar_en_pos}
    si no **si** n<m **entonces**
            sacar_en_pos (p,m)
            sacar_entre(p, n, m-1)
        fsi
    fsi
fproc

Jesús Lázaro
Mª José Domínguez

Ejercicio 4.- *(Examen del Grado en Ingeniería Informática, Noviembre 2011)* Se conoce el TAD CONJUNTO[ELEMENTO] para representar los datos *conjunto* de *elemento* con las siguientes operaciones:

- Ø: → conjunto
- insertar: elemento conjunto → conjunto
- borrar: elemento conjunto → conjunto
- ver_uno: conjunto→elemento
- está?: elemento conjunto→ bool
- vacio?: conjunto→bool

así como la especificación necesaria para PILAS[CONJUNTO[ELEMENTO]] (las pilas que están formadas por conjuntos). Añadir operaciones para:

- comprobar si un elemento está en todos los conjuntos de la pila.
- comprobar si todos los conjuntos de la pila tienen, al menos, los mismos elementos que el conjunto de la cima.
- quitar un elemento de todos los conjuntos de la pila.
- crear un único conjunto con todos los elementos de los conjuntos de la pila.
- quitar todos los conjuntos vacíos de la pila.

{operaciones nuevas}

esta_en_todos: elemento pila_conjuntos→booleano
fun esta_en_todos (e:elemento, pc:pila_conjuntos):booleano
    si vacia?(pc) **entonces** devolver T
    sino si esta?(e, cima(pc)) **entonces**
        desapilar(pc)
        devolver esta_en_todos(e, pc)
        sino devolver F
    fsi
ffun

```
subconjunto:conjunto conjunto→booleano
        {operación auxiliar que comprueba que todos los elementos de un conjunto están
        en el otro}
fun subconjunto (c1,c2:conjunto):booleano
var e:elemento
        si vacio (c1) entonces devolver T
        sino e←ver_uno(c1)
                si esta?(e, c2)  entonces
                        borrar(e,c1)
                        devolver subconjunto (c1, c2)
                        sino devolver F
        fsi
ffun

subconjunto_de_todos:conjunto pila_conjuntos→booleano
        {operación auxiliar que comprueba que el conjunto dado es subconjunto de todos
        los de la pila.}
fun subconjunto_de_todos (c:conjunto, pc:pila_conjuntos):booleano
        si vacio(pc) entonces devolver T
        sino si subconjunto(c, cima(pc)) entonces
                desapilar(pc)
                devolver subconjunto_de_todos (c, pc)
                sino devolver F
        fsi
ffun

parcial cima_es_subconjunto: pila_conjuntos→booleano
                        {la pila debe tener al menos un conjunto para acceder a cima}
fun cima_es_subconjunto (pc:pila_conjuntos):booleano
var c:conjunto
        si vacía?(pc) entonces error(Pila vacía)
        si no
                c← cima (pc)
                desapilar(pc)
                devolver subconjunto_de_todos(c, pc)
        fsi
ffun
```

quitar_en_todos: elemento pila_conjuntos→pila_conjuntos
proc quitar_en_todos (e:elemento, pc:pila_conjuntos)
var cc:conjunto
    si !vacia(pc) **entonces**
        cc←cima(pc)
        desapilar(pc)
        quitar_en_todos(e, pc)
        apilar(quitar (e, cc), pc)
    fsi
fproc

unión: conjunto conjunto→conjunto
{función auxiliar que devuelve la unión de dos conjuntos dados, solución en ejercicios TAD}

unión_todos: pila_conjuntos→conjunto
fun unión_todos(pc:pila_conjuntos):conjunto
var cc:conjunto
    si vacia?(pc) **entonces** devolver Ø
    sino cc←cima(pc)
        desapilar(pc)
        devolver union(cc, unión_todos(pc))
    fsi
ffun

quitar_vacios:pila_conjuntos→pila_conjuntos
proc quitar_vacios(pc:pila_conjuntos)
var cc:conjunto
    si !vacia(pc) **entonces**
        si vacio?(cima(pc)) **entonces**
            desapilar(pc)
            quitar_vacios(pc)
        sino
            cc←cima(pc)
            desapilar(pc)
            apilar(cc, quitar_vacios(pc))
    fsi
fproc

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
```


Jesús Lázaro
Mª José Domínguez

```
            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
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.- *(Examen del Grado en Ingeniería Informática, Noviembre 2010)* 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 (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
ffun

contar:colac→natural
quitar_mitad:colac→colac
proc quitar_mitad (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.- *(Examen del Grado en Sistemas de Información, Noviembre 2012)* 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(cb:cola_bool)
        mientras (no vacía?(cb)) ∧ eq(primero(cb),F) hacer
                desencolar(cb)
        fmientras
fproc

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

proc cambiarvalores (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
                devolver dato1
        fsi
```