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.

{Añadimos las nuevas operaciones en pseudocódigo, usamos la especificación base, no cambia ni el géneros ni el parámetro formal}

operaciones

```
contar: pila → natural
func contar (p:pila): natural                    {Iterativa }
var n:natural
        n←0
        mientras ¡vacia?(p) hacer
                desapilar(p)
                n←n+1
        finmientras
devolver n
finfunc
{LA PILA SE DESTRUYE/PIERDE}

parcial verFondo: pila →elemento
{la pila tiene que tener datos para poder ver el último}
func verFondo (p:pila):elemento                  {Iterativa }
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
finfunc
{INCLUIR CÓDIGO PARA NO PERDER LA PILA}
```

¿VERSIÓN RECURSIVA?

```
montar: pila pila → pila
func  montar ( p1, p2:pila):pila        {recursivo}
var e:elemento
si vacia?(p2) entonces dev p1
sino    e←cima(p2)
                desapilar(p2)
                dev apilar(e, montar(p1, p2))
finsi
finfunc

proc montar (E/S p1, p2:pila)                {iterativo}
var pi:pila e:elemento
pi←pilaVacia
        mientras  ¡vacia?(p2) hacer
                e←cima(p2)
                desapilar(p2)
                apilar(e,pi)
finmientras

        mientras  ¡vacia?(pi) hacer
                e←cima(pi)
                desapilar(pi)
                apilar(e,p1)
finmientras
finfunc

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 (E/S p:pila, n:natural)                                {iterativo}
        si (n>contar(p)) entonces error (no hay suficientes datos)
        si no   mientras (n>0) hacer
                        desapilar(p)
                        n←n-1
        fmientras
finsi
finproc
```

¿VERSIÓN RECURSIVA?

Ejercicio 2.- 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          {función auxiliar, invierte p1 en p2}
func invertir_aux (p1, p2: pila) dev pila                    {recursiva}
si vacia?(p1) entonces devolver p2
si no
 e←cima(p1)
devolver invertir_aux(desapilar(p1), apilar (e, p2))
finsi
finfunc

func invertir(p:pila) dev pila
Invertir_aux (p, pvacia)
finfunc

parcial mayorquecima:elemento pila→bool
{función auxiliar que comprueba si elto es mayor o igual que la cima de la pila,
parcial porque la pila no puede ser vacía}
func mayorquecima (e:elemento, p:pila) :bool
si e>= cima (p) entonces devolver T
                sino devolver F
finsi
finfunc
```

```
demayoramenor:pila→boolean
func 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
finsi
finfunc

demenoramayor:pila→boolean
func demenoramayor (p:pil): boolean
        var pi:pila
                pi←invertir(p)
                devolver demayoramenor(pi)
finfunc

parcial eliminarfondo: pila→pila      {debe haber al menos un elemento en la pila}
func eliminarfondo (p:pila):pila
var pi:pila
si vacia (p) entonces error(pila vacía)
si no
        pi←invertir(p)
        desapilar(pi)
devolver invertir(pi)
finsi
finfunc
```

Ejercicio 3.- 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.

Espec EJE3_PILA[ENTERO]
usa NATURAL2
parámetro formal
generos 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 (E/S p:pila, n:natural) {recursiva}
var e:entero
si (n>0) **entonces**
    si vacía?(p) **entonces error**(no hay suficientes datos)
    si no
        e←cima(p)
        desapilar(p)
        sacar_en_pos(p, n-1)
        si (n≠1) **entonces** apilar(e, p) **fsi**
    finsi
finsi
finproc

```
proc sacar_en_pos (E/S p:pila, n:natural)                    {iterativa}
var e:entero; p2: pila
si (n>contar(p)) entonces error(no hay suficientes datos)
si no
        p2←pvacía()
        mientras (n>1) hacer
                e←cima(p)
                desapilar(p)
                apilar(e,p2)
                n←n-1
        finmientras
        si (n=1) entonces desapilar(p) fsi
        mientras (no vacía(p2)) hacer
                e←cima(p2)
                desapilar(p2)
                apilar(e,p)
        finmientras
finsi

parcial sacar_entre:pila natural natural→pila
proc sacar_entre (E/S 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)
        finsi
        finsi
finproc
```

Ejercicio 4.- 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
- está?: elemento conjunto→ bool
- vacio?: conjunto→bool
- ver_uno: conjunto→elemento

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

```
func 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
    finsi
    finf
```

```
cima_es_subconjunto: pila_conjuntos→booleano
subconjunto:conjunto conjunto→booleano
        {operación auxiliar que comprueba que todos los elementos de un conjunto están
        en el otro}
func 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
        finsi
finf

subconjunto_de_todos:conjunto pila_conjuntos→booleano
        {operación auxiliar que comprueba que el conjunto dado es subconjunto de todos
        los de la pila.}
func 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
finsi
finfunc
```

parcial cima_es_subconjunto: pila_conjuntos→booleano
{la pila debe tener al menos un elemento para acceder a cima}
func 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)
finsi
finfunc

quitar_en_todos: elemento pila_conjuntos→pila_conjuntos
proc quitar_en_todos (e:elemento, E/S 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)
finsi
finproc

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

unión_todos: pila_conjuntos→conjunto
func 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))
finsi
finfunc

```
quitar_vacios:pila_conjuntos→pila_conjuntos
proc quitar_vacios(E/S 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))
        finsi
finproc
```