Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Guadalajara

TC2037 - Implementación de métodos computacionales - Gpo 820

Evidencia #3: Implementación de un simulador de un almacén automatizado (segunda parte)

Karen Priscila Navarro Arroyo | A01641532

07/06/2023

1. Estructura de datos

La estructura de cada inventario o carrusel está escrito en cada archivo de texto de la carpeta invent. La estructura es una lista principal, la cual hace de 'carrusel' la cual contiene maps listas, las cuales serían las 'filas' y cada una de estas filas contiene las listas de producto en forma (nombre_de_producto cantidad precio) (desde ahora haré referencia a este último formato de lista en específico cuando hable de **producto**) un ejemplo es la imagen de abajo. Como muestra, el programa original utiliza diez inventarios, cada uno con nombres de producto y cantidad de filas y columnas diferentes (pueden verse todas a la vez en el archivo informacion.html)

ID inventario	Inventario (invent/*)	Posición (id del producto en ventanilla) (pos/*)
1	*invent/1.txt* (((a01 64 10) (b01 64 20) (c01 64 30) (d01 64 40) (e01 64 50) (f01 64 60)) ((a02 64 10) (b02 64 20) (c02 60 30) (d02 64 40) (e02 64 50) (f02 64 60)) ((a03 64 10) (b03 0 20) (c03 61 30) (d03 64 40) (e03 64 50) (f03 64 60)) ((a04 64 10) (b04 64 20) (c04 64 30) (d04 64 40) (e04 64 50) (f04 64 60)) ((a05 64 10) (b05 64 20) (c05 5 30) (d05 59 40) (e05 64 50) (f05 64 60)) ((a06 64 10) (b06 64 20) (c06 44 30) (d06 64 40) (e06 64 50) (f06 64 60)) ((a07 60 10) (b07 64 20) (c07 64 30) (d07 64 40) (e07 64 50) (f07 64 60)) ((a08 64 10) (b08 64 20) (c08 64 30) (d08 64 40) (e08 64 50) (f08 64 60)) ((a09 64 10) (b09 64 20) (c09 64 30) (d09 34 40) (e09 64 50) (f09 64 60)) ((a10 64 10) (b10 64 20) (c10 64 30) (d10 64 40) (e10 64 50) (f10 64 60)) ((a11 64 10) (b11 64 20) (c11 64 30) (d11 64 40) (e11 64 50) (f11 64 60)) ((a12 64 10) (b12 64 20) (c12 64 30) (d12 64 40) (e12 64 50) (f12 64 60)) ((a13 64 10) (b13 64 20) (c13 64 30) (d13 64 40) (e13 11 50) (f13 64 60)) ((a14 64 10) (b14 64 20) (c14 64 30) (d14 30 40) (e14 64 50) (f14 64 60)) ((a15 64 10) (b15 64 20) (c15 64 30) (d15 64 40) (e15 64 50) (f15 64 60)) ((a16 64 10) (b16 64 20) (c16 64 30) (d16 1 40) (e16 30 50) (f16 64 60)) ((a17 64 10) (b17 64 20) (c17 64 30) (d17 64 40) (e17 64 50) (f17 64 60)) ((a18 64 10) (b18 64 20) (c18 64 30) (d18 64 40) (e18 64 50) (f18 64 60)) ((a19 64 10) (b19 64 20) (c19 30 30) (d19 30 40) (e19 30 50) (f19 64 60)) ((a20 64 10) (b20 64 20) (c20 64 30) (d20 64 40) (e20 64 50) (f20 64 60)) ((a21 64 10) (b21 64 20) (c21 64 30) (d21 64 40) (e21 64 50) (f21 64 60)) ((a22 64 10) (b22 64 20) (c22 64 30) (d22 64 40) (e22 64 50) (f22 64 60)))	*pos/1.txt* 101

Cada uno de estos inventarios tiene un ID, el cual lo identificará a lo largo de la ejecución del programa. Esta ID se genera en función de la cantidad de carruseles y el órden en el que se escriban. Por ejemplo, el carrusel de esta primera imagen tiene en ID 1, ya que es el primero de la lista, así cada carrusel es independiente de los otros.

A la par, cada carrusel tiene su propia lista de IDs para cada producto. A diferencia de la evidencia pasada, en este programa no fue necesario crear una función que genere la lista completa de IDs para cada carrusel, pero funciona de la misma manera.

El programa no incluye esta función, pero para mostrarlo de forma gráfica, la lista de IDs de los productos del inventario 1 es el de la segunda imagen con la función:

```
(defn id [idn] ; Genera un MAP con un ID para cada
elemento de inventario con ID idn
  (map (fn [x] (map (fn [y] (+ (* x 100) y)) (range 1
(+ 1 (col idn))))) (range 1 (+ 1 (fil idn)))))
```

```
clj:user:>
; IDs de productos de carrusel 1
(id 1)
((101 102 103 104 105 106)
 (201 202 203 204 205 206)
 (301 302 303 304 305 306)
 (401 402 403 404 405 406)
 (501 502 503 504 505 506)
 (601 602 603 604 605 606)
 (701 702 703 704 705 706)
 (801 802 803 804 805 806)
 (901 902 903 904 905 906)
 (1001 1002 1003 1004 1005 1006)
 (1101 1102 1103 1104 1105 1106)
 (1201 1202 1203 1204 1205 1206)
 (1301 1302 1303 1304 1305 1306)
 (1401 1402 1403 1404 1405 1406)
 (1501 1502 1503 1504 1505 1506)
 (1601 1602 1603 1604 1605 1606)
 (1701 1702 1703 1704 1705 1706)
 (1801 1802 1803 1804 1805 1806)
 (1901 1902 1903 1904 1905 1906)
 (2001 2002 2003 2004 2005 2006)
 (2101 2102 2103 2104 2105 2106)
 (2201 2202 2203 2204 2205 2206))
```

2. Diseño del autómata

El autómata está estrechamente ligado al id, pues cada una de estas ids representa un estado del autómata, las cuales cambian de estado con la función `Mover-carrusel`, que recibe como entrada una lista con una letra para cada movimiento; A, B, I, D para arriba, abajo, izquierda y derecha respectivamente, además del ID del carrusel al que se debe aplicar esta función. El estado actual del autómata es el producto que se encuentra en ventanilla, el cual se guarda en la carpeta pos en su respectivo archivo de texto (el ID del carrusel también sirve para identificar a su archivo de posición).

El estado actual del autómata puede obtenerse con `(getPos id)`, donde el parámetro es el id del inventario deseado, el cual devuelve el id del producto. También puede modificarse con `(setPos n id)` donde n es el nuevo estado e id el identificador del inventario. También puede mostrarse en el formato del producto con `(getProd id)`. Por ejemplo, el estado por defecto del autómata es 101 para cada inventario, id correspondiente al primer producto del inventario. Tomando como ejemplo el inventario 2:

```
clj:user:> (getPos 2)
101
clj:user:> (getProd 2)
(g01 59 10)
```

ID inventario	Inventario (invent/*)	Posición (id del producto en vent
2	*invent/2.txt* (((g01 59 10) (h01 64 20) (i01 64 30) (j01 64 40) (k01 64 50) (l01 64 60)) ((g02 64 10) (h02 64 20) (i02 64 30) (j02 64 40) (k02 2 50) (l02 64 60))	*pos/2.txt* 101

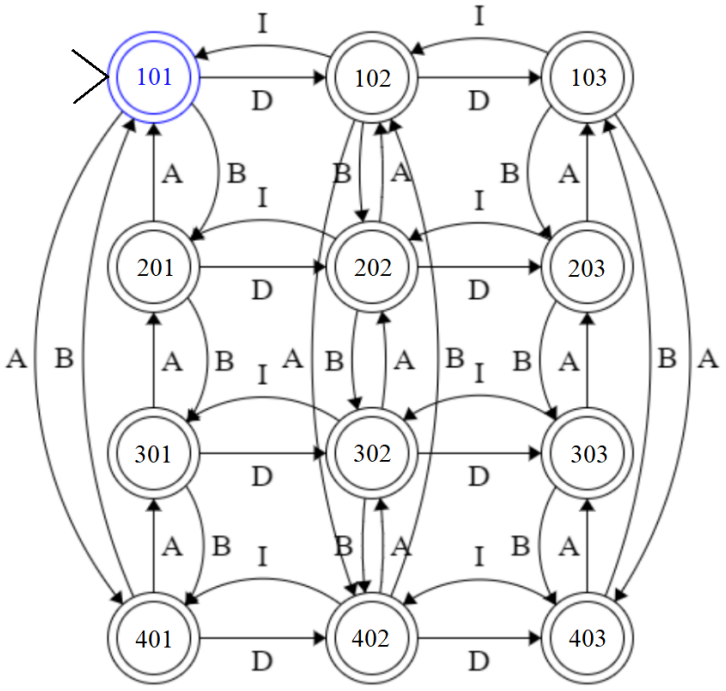
Entiéndase todos los datos referentes al autómata como

Estado actual del autómata = **información de pos/*.txt** = **producto en ventanilla** = ID = `(getPos)` `(getProd)`

No se confunda el id de producto, los cuales abarcan desde el 101 hasta (cantidad de filas*100 + cantidad de columnas) con id de inventario. Generalmente, el id que pide gran parte de las funciones del código como último parámetro hace referencia a este último,

La misma información en diferentes formatos es importante para el funcionamiento y ejecución del programa.

Gráficamente, el autómata para un inventario de 4 filas y 3 columnas estaría representado de la siguiente manera (debido a que el que muestro al principio es demasiado grande con 22 filas y 6 columnas):



Aunque el tamaño del autómata varía en función del inventario, el principio es el mismo; se puede subir o bajar una cantidad indefinida de veces, pero solo puede moverse a la izquierda o derecha dentro del rango de columnas que tenga (cantidad máxima de movimientos a la Izq o Der = cantidad de columnas - 1). Como se muestra en el autómata, el inventario es horizontalmente circular, Es decir, una fila arriba de `((101 102 103 104 105 106)` está `(2201 2202 2203 2204 2205 2206)`.

3. Manejo y ejecución del autómata

Internamente, la función Mover-carrusel, para ejecutar el autómata, funciona con otras cuatro funciones para mover el lugar/cambiar el estado del autómata. Cada una de estas funciona matemáticamente con el id de la posición actual (getPos), se hace la operación correspondiente al movimiento y se modifica el estado con (setPos).

```
82  (defn Arr [id]                              ; Sube una fila
83    (if (and (<= (getPos id) (+ 100 (col id))) ; Si pertenece a la primer fila,
84             (>= (getPos id) 101))             ; sube a la última
85      (setPos (+ (getPos id) (* (- (fil id) 1) 100)) id)
86      (setPos (- (getPos id) 100) id)))
87
88  (defn Baj [id]                                        ; Baja una fila
89    (if (and (<= (getPos id) (+ (* (fil id) 100) (col id))) ; Si pertenece a la última fila,
90             (>= (getPos id) (* (fil id) 100)))           ; baja a la primera
91      (setPos (- (getPos id) (* (- (fil id) 1) 100)) id)
92      (setPos (+ (getPos id) 100) id)))
93
94  (defn Izq [id]                                          ; Mueve un lugar a la izquierda
95    (if (< (Integer/parseInt (str (last (str (getPos id))))) 2) ; Verifica si está en rango para moverse
96      (print "ERROR: No se puede mover más a Izq\n")
97      (setPos (- (getPos id) 1) id)))
98
99  (defn Der [id]                                                ; Mueve un lugar a la derecha
100   (if (= (Integer/parseInt (str (last (str (getPos id))))) (col id)) ; Verifica si está en rango para moverse
101     (print "ERROR: No se puede mover más a Der\n")
102     (setPos (+ (getPos id) 1) id)))
```

Considerando

- setPos = nueva posición (ID)
- getPos = posición actual (ID)
- fil = cantidad total de filas (en este caso 22)
- col = cantidad total de columnas (en este caso 6)
- id = id de inventario (en este caso 1)

Cada función puede describirse de la siguiente manera:

funct	*Funcionamiento*	*Ejemplo (valores de ref. del inventario original)*
Arr	Si getPos está en la primer fila setPos = getPos + ((fila-1)*100) Sino setPos = getPos - 100	 2201 = 101 + ((22-1)*100) 305 = 405 - 100
Baj	Si getPos está en la última fila setPos = getPos - ((fil-1)*100) Sino setPos = getPos + 100	 103 = 2203 - ((22-1)*100) 502 = 402 + 100
Izq	Si (2 ≤ columna de getPos < (col+1)) setPos = getPos - 100 Sino ERROR	 105 = 106 - 1 ERROR = 101 - 1

Der	Si (1 ≤ columna de getPos < col) setPos = getPos + 1 Sino ERROR	404 = 403 + 1 ERROR = 406 + 1

La función principal de movimiento `Mover-carrusel`, al funcionar de manera recursiva, por cada "iteración" debe leer la entrada y al mismo tiempo ejecutarla, pues el estado del autómata también va cambiando durante la ejecución. Por eso mismo, esta parte del programa es estrictamente secuencial.

```
; Función principal de movimiento. Lee y ejecuta el autómata en el inventario id
(defn Mover-carrusel [ent id] ; ent es una lista ej. '(A A A A D). id es el ID de su carrusel/inventario
  (cond (empty? ent)
        (do (print "En ventanilla-> ")
            (print (getProd id)))
        (= (first ent) 'A)
        (do (Arr id)
            (print "A^  ") (print (getProd id)) (newline)
            (Mover-carrusel (rest ent) id))
        (= (first ent) 'B)
        (do (Baj id)
            (print "Bv  ") (print (getProd id)) (newline)
            (Mover-carrusel (rest ent) id))
        (= (first ent) 'I)
        (do (Izq id)
            (print "I<  ") (print (getProd id)) (newline)
            (Mover-carrusel (rest ent) id))
        (= (first ent) 'D)
        (do (Der id)
            (print "D>  ") (print (getProd id)) (newline)
            (Mover-carrusel (rest ent) id))
        :else (do (print "ERROR: Entrada no aceptada") nil)))
```

Si en algún punto llega a ocurrir un error en la entrada, el autómata se queda en la última posición en la que la entrada fue aceptada. Por ejemplo, cuando getPos = 101, `(Mover-carrusel '(D D D D D D) 1)` marcará un error en la última entrada, pues ya no hay espacio para seguir más a la derecha.

También, durante la ejecución de esta función, se van imprimiendo los productos que pasaron por ventanilla para llegar a la posición final.

```
clj:user:> (getPos 1)
101
clj:user:> (Mover-carrusel '(D D D D D D) 1)
D>  (b01 64 20)
D>  (c01 64 30)
D>  (d01 64 40)
D>  (e01 64 50)
D>  (f01 64 60)
ERROR: No se puede mover más a Der
D>  (f01 64 60)
En ventanilla-> (f01 64 60)
nil
```

4. Mutabilidad de los datos

Considerando que un elemento es mutable cuando no hay que modificar el código fuente para soportar el cambio.

Elemento del simulador	**¿Es mutable?**	**Restricción**	**Notas**
Cantidad de filas	Sí	Tener mínimo 2 pero no exceder las 99.	Como todas las transacciones están ligadas a la id y se manejan por fórmulas que utilizan el 100 como número fijo (véase la parte de funciones de movimiento en la sección anterior), habría que modificar este dato o generar otra función que de este número base según el tamaño del inventario.
Cantidad de columnas	Sí		
Nombre de los productos	Sí	-	El nombre del producto no afecta el funcionamiento de ninguna manera pues, aunque los nombres declarados en los archivos de texto puedan seguir cierto patrón según posición, no es necesario seguir ninguna regla para nombrar los productos.
Cantidad máxima por producto	No	-	Se considera 64 como cantidad máxima para cada producto, por lo que habría que modificar esta cantidad en el código fuente, generar otra lista con los valores máximos de cada producto o modificar la estructura de los datos de carrusel.
Precio de productos	Sí	-	Este dato puede ser cualquier cantidad. En **producto** es el último elemento de la lista
Inventarios de los archivos de texto	Sí	Apegarse al formato de listas	El inventario, aunque varíe en tamaño, nombres o precios, debe seguir la estructura de listas del carrusel, de otra forma el simulador no podrá ejecutarse.
`entrada.txt`	Sí	Llamar a las funciones correctamente	Para poder utilizar el simulador, las funciones deben llamarse igual a como se escribirían en la terminal.

Casos especiales

Donde un dato puede variar pero con una mínima variación en el código fuente. Consideré necesario agregar esta sección ya que tuve algunos problemas con la creación, lectura y manejo de archivos posiblemente por la extensión que se utilizó de Visual Studio Code para correr Clojure, por lo que me vi en la necesidad de hacer las funciones relacionadas a archivos en la forma en que se hizo.

Elemento del simulador	Notas
Cantidad de inventarios	El programa sí puede manejar una cantidad de inventarios indefinida, pero deben agregarse las direcciones a la lista 'inventarios' del código fuente, así como la dirección de archivo de posición en la lista 'posiciones' `(def inventarios ; lista con las direcciones de los inventarios (list "CLOJURE/invent/1.txt" "CLOJURE/invent/2.txt" "CLOJURE/invent/3.txt" "CLOJURE/invent/4.txt" "CLOJURE/invent/5.txt" "CLOJURE/invent/6.txt" "CLOJURE/invent/7.txt" "CLOJURE/invent/8.txt" "CLOJURE/invent/9.txt" "CLOJURE/invent/10.txt"))` `(def posiciones ; lista con las direcciones de las posiciones de cada carrusel (list "CLOJURE/pos/1.txt" "CLOJURE/pos/2.txt" "CLOJURE/pos/3.txt" "CLOJURE/pos/4.txt" "CLOJURE/pos/5.txt" "CLOJURE/pos/6.txt" "CLOJURE/pos/7.txt" "CLOJURE/pos/8.txt" "CLOJURE/pos/9.txt" "CLOJURE/pos/10.txt"))` Estas son las diez listas que se usan por defecto, pero podrían estar en cualquier dirección y en cualquier cantidad.

5. Paralelización de procesos

El uso más destacable de paralelización se encuentra en la función (main), la cual sería la principal para ejecutar todas las transacciones ingresadas en el archivo de texto (entrada)
Esta función se llama en (ejecutar), que realiza cuatro funciones: Primero realiza las transacciones del archivo de entrada (llamadas con esta función (main)), una vez hechos los cambios en los archivos de texto, lamma las tres funciones finales; (poco-inventario), (valor-total-almacen) y (top10).

```
(defn main []
  (def a (clojure.string/split-lines (slurp "CLOJURE/entrada.txt")))
  (defn separarPorInventario [lst]
    (->> lst
         (group-by last)
         (vals)))
  (let [comm (separarPorInventario (map #(read-string %) a))]
    (pmap (fn [a] (map (fn [b] (eval b)) a)) comm)))
```

Para que (main) convierta las instrucciones del archivo de entrada en hilos para cada carrusel, sigue el siguiente proceso:

1. Esta primer función 'a' definida dentro de main lee el contenido de entrada.txt y lo devuelve como una lista de strings

Código	Salida
`(def a (clojure.string/split-lines (slurp "CLOJURE/entrada.txt")))`	`clj:CLOJURE.ev3-V14:> a` `["(retirar 1 1)"` `"(agregar 't02 1 4)"` `"(retirar 'r01 1 4)"` `"(retirar 1 10)"` `"(retirar 1 2)"` `"(retirar 'j02 1 2)"` `"(retirar 'g03 1 2)"` `"(retirar 'h04 1 2)"`

2. Una vez con esta lista, debe cambiarse al formato de una lista normal y acomodarla por inventarios

Código

```
(map #(read-string %) a)
```

Salida

```
clj:CLOJURE.ev3-V14:> (map #(read-string %) a)
((retirar 1 1)
 (agregar 't02 1 4)
 (retirar 'r01 1 4)
 (retirar 1 10)
 (retirar 1 2)
 (retirar 'j02 1 2)
 (retirar 'g03 1 2)
 (retirar 'h04 1 2)
```

```
(defn separarPorInventario
[lst]
    (->> lst
         (group-by last)
         (vals)))
```

```
clj:CLOJURE.ev3-V14:> (separarPorInventario
(map #(read-string %) a))
([(retirar 1 1)]
 [(agregar 't02 1 4) (retirar 'r01 1 4)]
 [(retirar 1 10)]
 [(retirar 1 2)
  (retirar 'j02 1 2)
  (retirar 'g03 1 2)
  (retirar 'h04 1 2)])
```

3. Ahora, con esta última salida, lo usamos en un let como variable llamada ‘comm’ para que lo tome el pmap y genere un hilo para cada parte de esta lista. En este caso, como he manejado diez inventarios, el máximo de hilos que pueden generarse son diez. Crear un hilo para cada carrusel es útil ya que los datos mutables del código son las posiciones y los inventarios, por lo que, al ser independientes uno de otro, no se generen errores en las transacciones al reescribir los datos de inventario.

```
(let [comm (separarPorInventario (map
#(read-string %) a))]
    (pmap (fn [a] (map (fn [b] (eval
b)) a)) comm))
```

Aquí aparecen las salidas de cada transacción de entrada.txt

```
clj:CLOJURE.ev3-V14:>
(let [comm (separarPorInventario (map #(read-string %) a))]
    (pmap (fn [a] (map (fn [b] (eval b)) a)) comm))


_____________________________
CARRUSEL/INVENTARIO 1 --------------

Retirado    ->  (a01 32 10)
(("")
_____________________________
CARRUSEL/INVENTARIO 4 --------------
Bv  (r02 13 20)
D>  (s02 64 30)
D>  (t02 64 40)
En ventanilla-> (t02 64 40)
Agregado     ->  (t02 64 40)
ADVERTENCIA: Se agregó la cantidad máxima
_____________________________
CARRUSEL/INVENTARIO 4 --------------
A^  (t01 64 40)
I<  (s01 64 30)
I<  (r01 32 20)
En ventanilla-> (r01 32 20)
Retirado     ->  (r01 31 20)
(nil "")
```

6. Ventajas y desventajas de desarrollar el simulador en Clojure

Mis principales desventajas fueron los tipos de datos y el manejo de archivos. Algunas veces trataba de leer una dirección de un archivo de texto, evaluar el carrusel y realizar las transacciones y reescribir este carrusel, lo cual, primero no quería leer el texto de el primer archivo como una dirección, y después el resultado que escribía en el archivo de texto era algo como clojure.lang.LazySeq@60f4bd24. Al final pude resolver este problema escribiendo las direcciones de los archivos de texto directamente en el programa en la función inventarios, aunque me habría gustado hacerlo como originalmente lo había pensado.
También tuve problemas utilizando funciones que evaluaran listas con maps, problema que no tuve con Racket.
Por otra parte, la paralelización es la ventaja principal de este lenguaje, de tal forma que los procesos sean más rápidos y eficientes, aunque también es algo peligroso de manejar, especialmente cuando debía imprimirse en consola exactamente lo que estaba pasando en procesador. Fuera de eso, los resultados finales (los cambios en los inventarios de los archivos de texto) se escribieron como debían.

7. Código

ev3_V14.clj

```
(ns CLOJURE.ev3-V14)
(require '[clojure.java.io :as io])
;_____________________________________________________________________________
_____________________________________________________
; 1 - Lectura, declaración y manejo de archivos
-------------------------------------------------------------------------------
----------------

(def inventarios ; lista con las direcciones de los inventarios
  (list
   "CLOJURE/invent/1.txt"
   "CLOJURE/invent/2.txt"
   "CLOJURE/invent/3.txt"
   "CLOJURE/invent/4.txt"
   "CLOJURE/invent/5.txt"
   "CLOJURE/invent/6.txt"
   "CLOJURE/invent/7.txt"
   "CLOJURE/invent/8.txt"
   "CLOJURE/invent/9.txt"
   "CLOJURE/invent/10.txt"))

(def posiciones ; lista con las direcciones de las posiciones de cada carrusel
```

```
  (list
   "CLOJURE/pos/1.txt"
   "CLOJURE/pos/2.txt"
   "CLOJURE/pos/3.txt"
   "CLOJURE/pos/4.txt"
   "CLOJURE/pos/5.txt"
   "CLOJURE/pos/6.txt"
   "CLOJURE/pos/7.txt"
   "CLOJURE/pos/8.txt"
   "CLOJURE/pos/9.txt"
   "CLOJURE/pos/10.txt"))

;_____________________________________________________________________
_____________________________________________
; 1.1 - Set y Get
-------------------------------------------------------------------------
-----------------------------------------

; n=iteración, xy=tope de iteraciones (depende del valor de fila/columna o ID del
elemento buscado). SIEMPRE DEBE LLAMARSE CON n=1
(defn avanza [n xy lista] ; funcion auxiliar para búsquedas que impliquen recorrer
una lista
  (if (= n xy) (first lista) (avanza (+ n 1) xy (rest lista))))

(defn getInv [id]     ; dado el ID, devuelve la dirección del carrusel con es ID
buscado
  (avanza 1 id inventarios))

(defn posicion [id]   ; dado el ID, devuelve la dirección del carrusel con es ID
buscado
  (avanza 1 id posiciones))

(defn inventario [id] ; lee el carrusel id (int) y lo devuelve como lista
  (with-open [archivo (io/reader (getInv id))] ; EJ.: (inventario 1) regresa el
contenido del inventario 1 como lista
    (read-string (apply str (line-seq archivo)))))

(defn getPos [id]     ; lee el carrusel id (int) y lo devuelve como lista
  (with-open [archivo (io/reader (posicion id))] ; EJ.: (inventario 1) regresa el
contenido del inventario 1 como lista
    (read-string (apply str (line-seq archivo)))))

(defn setPos [p id]   ; cambia la posición p del carrusel dir
```

```
  (with-open [out (io/writer (posicion id))]
    (spit out p)))

(map #(setPos 101 %) (range 1 (+ 1 (count posiciones)))) ; Asigna la posición 101 a
todos los carruseles por defecto

(defn getProd [id]          ; Regresa el producto de (inventario id)
correspondiente al estado actual de su autómata
  (avanza 1 (mod (getPos id) 100)                                    ;
(mod (getPos) 100)             -> col
          (avanza 1 (int (Math/floor (/ (getPos id) 100))) (inventario id)))) ;
(Math/floor (/ (getPos) 100))  -> fil

(defn lugar [id] ; Imprime en qué carrusel se está haciendo la transacción
  (newline) (print "_____________________________________") (newline)
  (print "CARRUSEL/INVENTARIO" id "--------------") (newline))

;_____________________________________________________________________________
_____________________________________________
; 2 - Creación y manejo de Identificadores
-------------------------------------------------------------------------------
----------------------

(defn col [id] ; Da un entero con la cantidad total de filas del carrusel id
  (count (first (inventario id)))) ; Ej.: (col 1) regresa la cantidad de filas del
inventario id

(defn fil [id] ; Da un entero con la cantidad total de columnas del carrusel id
  (count (inventario id))) ; Ej.: (fil 1) regresa la cantidad de columnas del
inventario 1

;_____________________________________________________________________________
_____________________________________________
; 3 - Funciones de movimiento. Lee y ejecuta el autómata
-------------------------------------------------------------------------------
--------
; Mover-carrusel Recibe la entrada y la posición actual en el carrusel
-------------------------------------------------------------------------------
; A^  Bv  I<  D>
-------------------------------------------------------------------------------
-----------------------------------------------

(defn Arr [id]                          ; Sube una fila
```

```
  (if (and (<= (getPos id) (+ 100 (col id))) ; Si pertenece a la primer fila,
           (>= (getPos id) 101))             ; sube a la última
    (setPos (+ (getPos id) (* (- (fil id) 1) 100)) id)
    (setPos (- (getPos id) 100) id)))

(defn Baj [id]                                          ; Baja una fila
  (if (and (<= (getPos id) (+ (* (fil id) 100) (col id))) ; Si pertenece a la
última fila,
           (>= (getPos id) (* (fil id) 100)))           ; baja a la primera
    (setPos (- (getPos id) (* (- (fil id) 1) 100)) id)
    (setPos (+ (getPos id) 100) id)))

(defn Izq [id]                                              ; Mueve un lugar a la
izquierda
  (if (< (Integer/parseInt (str (last (str (getPos id))))) 2) ; Verifica si está en
rango para moverse
    (print "ERROR: No se puede mover más a Izq\n")
    (setPos (- (getPos id) 1) id)))

(defn Der [id]                                                  ; Mueve un
lugar a la derecha
  (if (= (Integer/parseInt (str (last (str (getPos id))))) (col id)) ; Verifica si
está en rango para moverse
    (print "ERROR: No se puede mover más a Der\n")
    (setPos (+ (getPos id) 1) id)))

; Función principal de movimiento. Lee y ejecuta el autómata en el inventario id
(defn Mover-carrusel [ent id] ; ent es una lista ej. '(A A A A D). id es el ID de
su carrusel/inventario
  (cond (empty? ent)
        (do (print "En ventanilla-> ")
            (print (getProd id)))
        (= (first ent) 'A)
        (do (Arr id)
            (print "A^  ") (print (getProd id)) (newline)
            (Mover-carrusel (rest ent) id))
        (= (first ent) 'B)
        (do (Baj id)
            (print "Bv  ") (print (getProd id)) (newline)
            (Mover-carrusel (rest ent) id))
        (= (first ent) 'I)
        (do (Izq id)
            (print "I<  ") (print (getProd id)) (newline)
```

```
            (Mover-carrusel (rest ent) id))
        (= (first ent) 'D)
        (do (Der id)
            (print "D>  ") (print (getProd id)) (newline)
            (Mover-carrusel (rest ent) id))
        :else (do (print "ERROR: Entrada no aceptada") nil)))

;_____________________________________________________________________________
_____________________________________________________
; 4 - Funciones de búsqueda por nombre
------------------------------------------------------------------------------
-------------------------
; dado el nombre y lista (se debe llamar inicialmente con (inventario)), regresa el
elemento. Ej, (buscar 'a03 (inventario 1)) -> (a03 64 10)
; Si el nombre dado no existe, regresa una lista vacía
------------------------------------------------------------------------------
---------

(defn buscar2 [nombre subl] ; Función auxiliar de (buscar)
  (cond (empty? subl) '()
        (= nombre (first (first subl))) (first subl)
        :else (buscar2 nombre (rest subl))))

(defn buscar [nombre lista]
  (cond (empty? lista) (do (print "El producto no existe") '())
        (empty? (buscar2 nombre (first lista)))
        (buscar nombre (rest lista))
        :else (buscar2 nombre (first lista))))

(defn auxGetID [prod lista n] ; Función auxiliar de getID
  (if (= prod (first lista)) n ; Regresa índiec neto del producto
      (auxGetID prod (rest lista) (+ n 1))))

(defn getID [prod ID] ; Regresa la ID del producto en el id. [prod = (nomProd cant
precio) ]
  (let [a (auxGetID prod (apply concat (inventario ID)) 1)
        b (int (Math/floor (/ a (col ID))))]
    (+ (if (= b (/ a (col ID))) (* b 100) (* (+ b 1) 100))
       (if (= (- a (* b (col ID))) 0) (col ID) (- a (* b (col ID)))))))

;_____________________________________________________________________________
_____________________________________________________
; 5 - camino/racorrer automáticamente el carrusel
```

```
;-------------------------------------------------------------------------------
-------------
; Busca el camino más corto entre el producto en ventanilla y el que se desea
retirar/agregar ----------------------------------------------------

(defn camino [pos2 id] ; recibe el ID del segundo producto y el id de su inventario
  (lugar id)
  (defn fila [a] (int (Math/floor (/ a 100))))  ; Fila del producto dada su ID
  (defn colm [b] (mod b 100))                   ; Columna del product
  (def minFila (if (<= (getPos id) pos2)
                 (list (getPos id) pos2)      ; Cuál de los dos productos tiene
menor ID = cuál está más "arriba"
                 (list pos2 (getPos id))))    ; (car minFila) es el de menor ID y
(cdr minFila) el mayor

  ; Genera la lista con los movimientos del carrusel (Ej. (genMov 3 'A) -> '(A A
A))
  (defn genMov [n m]
    (if (= n 0) '()
        (concat (list m) (genMov (- n 1) m)))) ; n=cantidad, m= letra de mov. (A B
I D)
  (Mover-carrusel
   (concat (if (<= (abs (- (fila (last minFila)) (fila (first minFila))))         ;
Qué es más corto, bajar
                   (abs (- (+ (fila (first minFila)) (fil id)) (fila (last
minFila))))) ; o subir
             (genMov (abs (- (fila (last minFila)) (fila (first minFila))))
                     (if (= (getPos id) (first minFila)) 'B 'A))
             (genMov (abs (- (+ (fila (first minFila)) (fil id)) (fila (last
minFila))))
                     (if (= (getPos id) (first minFila)) 'A 'B))) ; Parte para
subir o bajar
           (genMov (abs (- (colm (getPos id)) (colm pos2))) (if (<= (colm (getPos
id)) (colm pos2)) 'D 'I))) id)) ; Parte para Der o Izq



;_____________________________________________________________________________
_____________________________________
; 6 - Retirar y agregar productos
;-------------------------------------------------------------------------------
-----------------------------
; Los movimientos necesarios para llegar al producto donde se desea realizar la
transacción se efectúan automáticamente ---------------------------
```

```
; nom->nombre, cant->cantidad, s(1->sacar, 2->meter), id->id de inventario
(defn error1 [] (print "ADVERTENCIA: Se retiró la cantidad máxima"))
(defn error2 [] (print "ADVERTENCIA: Se agregó la cantidad máxima"))
(defn retirar? [y cant] (if (> (second y) cant)          (- (second y) cant) 0))
(defn agregar? [y cant] (if (<= (+ (second y) cant) 64) (+ (second y) cant) 64))
(defn actualInvent [nom cant s id] ; Genera el inventario con los cambios
  (map (fn [x]
         (map (fn [y]
                (if (= nom (first y))
                  (list (first y) (if (= s 1) (retirar? y cant) (agregar? y cant))
(last y))
                  y))
              x))
       (inventario id)))
(defn editarProd [nom cant s id] ; < Reescribe el inventario con la información
nueva
  (spit (getInv id) (prn-str (actualInvent nom cant s id))))

(defn retirar [& args]
  (cond
    (= 2 (count args))
    (do (lugar (last args))
        (editarProd (first (getProd (last args))) (first args) 1 (last args))
        (println "\nRetirado    -> " (getProd (last args)))
        (if (= (second (getProd (last args))) 0)
          (error1) ""))
    :else
    (do (camino (getID (buscar (first args) (inventario (last args))) (last args))
(last args))
        (editarProd (first args) (second args) 1 (last args))
        (println "\nRetirado     -> " (getProd (last args)))
        (if (= (second (getProd (last args))) 0)
          (error1) ""))))

(defn agregar [& args]
  (cond
    (= 2 (count args))
    (do (lugar (last args))
        (editarProd (first (getProd (last args))) (first args) 2 (last args))
        (println "\nAgregado     -> " (getProd (last args)))
        (if (= (second (getProd (last args))) 64)
          (error2) ""))
```

```
    :else
    (do (camino (getID (buscar (first args) (inventario (last args))) (last args))
(last args))
        (editarProd (first args) (second args) 2 (last args))
        (println "\nAgregado     -> " (getProd (last args)))
        (if (= (second (getProd (last args))) 64)
          (error2) ""))))

;_____________________________________________________________________
_____________________________________________________
; 7 - Valor total de inventario / poco-inventario  / Top 10
-------------------------------------------------------------------------------
----
; Funciones principales.

; 7.1.1 Devuelve el valor total del inventario id
(defn valor-total-inventario [id]
  (apply + (map #(* (second %) (last %)) (apply concat (inventario id)))))

; 7.1 Imprime una lista con los vvalores totales de cada inventario/carrusel y
devuelve la sumatoria de estos valores
(defn valor-total-almacen []
  (defn a [] (pmap #(valor-total-inventario %) (range 1 (+ 1 (count
inventarios))))) ; Se evalúan todos los valores totales de cada inventario en
paralelo
  (print "Valor total de carruseles:")
  (print (map (fn [x y] [(newline) (str "  Carrusel " x ": $ " y)]) (range 1 (+ 1
(count inventarios))) (a)))
  (newline) (print "\nValor total del Almacen: $" (apply + (a))) (newline))

; 7.2.1.1 Evalúa qué productos tienen poco inventario (<12). Genera una lista con
los nombres de los productos para mostrar los detalles
(defn poco-invent-format [id] ; Para un inventario a la vez
  (map #(if (< (second %) 12)
          (buscar (first %) (inventario id))
          '())
       (apply concat (inventario id))))

; 7.2.1.2 Formato para imprimir los datos de un producto
(defn formato [a id] ; a -> producto (nom cant<12 precio)
  (let [n (getID a id)]
    (newline)
    (print "nombre:-----" (first a)) (newline)
```

```
    (print "  cantidad: " (second a)) (newline)
    (print "  ID:" id "-" n) (newline)
    (print "  fila:" (int (Math/floor (/ n 100))))
    (print ", colm:" (mod n 100))
    (print ", carrusel:" id) (newline)
    0))

; 7.2.1 Imprime la lista de los productos con poco inventario en inventario id
(defn poco-inventario-id [id] ; Imprime los datos de los productos con poca
cantidad (<12) en el inventario id
  (lugar id)
  (pr (doall (map #(if (not (empty? %)) (formato % id) 0)
                  (poco-invent-format id)))))

; 7.2 Imprime la lista de todos los productos del almacén con poco inventario o 0
si no hay problema con el producto
(defn poco-inventario [] ;Imprime los datos de los productos con poca cantidad
(<12) en todos los inventarios
  (print "ADVERTENCIA productos con poca o nula existencia") (newline)
  (pr (doall (map #(poco-inventario-id %) (range 1 (+ 1 (count inventarios)))))))

; 7.3.1 Imprime los n valores más altos de la lista
(defn top [lista n] ;lista-> lista de los valores totales y si ID de inventario
  (if (= 1 n)
    (print n "- Carrusel" (last (last lista)) ": $" (first (last lista)))
    (do (print n "- Carrusel" (last (last lista)) ": $" (last (last lista)))
        (top (rest lista) (- n 1)))))

; 7.3 Top 10% de inventarios con mayor valor total
(defn top10 []
  (print "\n\nTop 10% de inventarios con mayor valor total") (newline)
  (top (sort-by first (map list (map #(valor-total-inventario %) (range 1 (+ 1
(count inventarios))))
                           (range 1 (+ 1 (count inventarios)))))
       (int (Math/floor (/ (count inventarios) 10)))))

;_____________________________________________________________________
_____________________________________________________

; 8 - Ejecución
--------------------------------------------------------------------------------
------------------------------------------------
; Lee el archivo de entrada 'entrada.txt', lo ejecuta y al final muestra el valor
total del almacén [7.1], Productos con poco inventario [7.2] y el top 10 de
```

```
carruseles con más valor [7.3]

(defn main []
  ; Regresa el contenido de entrada.txt como lista de strings
  (def a (clojure.string/split-lines (slurp "CLOJURE/entrada.txt")))

  ; Separa la entrada según el inventario, así cada inventario tiene su popia lista
sin importar cuantas transacciones se hagan en cada uno
  (defn separarPorInventario [lst]
    (->> lst
         (group-by last) ; El último argumento de cada función es el id de su
inventario
         (vals)))

  ; (map #(read-string %) a) devuelve la lista de strings a como lista normal
  (let [comm (separarPorInventario (map #(read-string %) a))]
    ; Se genera un hilo para cada carrusel, el cual sigue sus propias transacciones
    (pmap (fn [a] (map (fn [b] (eval b)) a)) comm)))

(defn ejecutar []
  (pr (doall (main)))
  (println "\n\nTRANSACCIONES COMPLETADAS. Generando totales...")
  (Thread/sleep 3500)
  (poco-inventario)
  (valor-total-almacen)
  (top10))
```

8. Uso e interfaz

El simulador puede manejarse escribiendo primero todas las transacciones / llamada de las funciones en el archivo entrada.txt y después de correr el programa llamar a la función `(ejecutar)`. Esto ejecutará todas las transacciones escritas en el en archivo de texto y al final ejecutar las dos funciones finales; Productos con poco inventario `(poco-inventario)`, valor total de cada inventario y todo el almacén `(valor-total-almacen)` y top 10% carruseles con mayor valor `(top10)`.

También puede utilizarse directamente la terminal del programa, si se desea ejecutar una transacción a la vez, solo que las funciones de valor total del inventario y los productos con poco inventario deberán llamarse manualmente.

El archivo entrada.txt debe contener entradas válidas para las transacciones y no tener saltos de línea vacíos entre éstas.

9. Pruebas

Parte 1

Para esta primera sección utilizo una misma ejecución, por lo que los valores de inventario se van modificando. Muestra el funcionamiento de las funciones que se solicitaban para realizar transacciones (agregar y retirar producto, valor total del almacén, Top inventarios con más valor y Productos con poco inventario o inventario nulo).

Para mayor facilidad, usaré la página 'informacion.html' para visualizar los datos de los archivos de texto. Esta página está incluída en los documentos del .zip del proyecto.

①	**Agregar producto** (con tres argumentos)
	Agrega el producto con el nombre del primer argumento, la cantidad del segundo en el inventario del tercero. El carrusel se mueve automáticamente desde el producto en ventanilla hasta el producto con el nombre especificado y después realiza el retiro. Los productos que pasaron por la ventanilla para llegar a la posición del producto especificado se imprimen con su respectivo movimiento.

Ejecución

```
clj:CLOJURE.ev3-V14:> (agregar 'd19 5 1)


_____________________________________
CARRUSEL/INVENTARIO 1 --------------
A^  (a22 64 10)
A^  (a21 64 10)
A^  (a20 64 10)
A^  (a19 64 10)
D>  (b19 64 20)
D>  (c19 30 30)
D>  (d19 30 40)
En ventanilla-> (d19 30 40)
Agregado     ->  (d19 35 40)
""
clj:CLOJURE.ev3-V14:>
```

Imagen de referencia de inventario + pos

invent/1.txt

```
(((a01 31 10) (b01 64 20) (c01 64 30) (d01 64 40) (e01 64 50) (f01 64 60))
((a02 64 10) (b02 64 20) (c02 60 30) (d02 64 40) (e02 64 50) (f02 64 60))
((a03 64 10) (b03 0 20) (c03 61 30) (d03 64 40) (e03 64 50) (f03 64 60))
((a04 64 10) (b04 64 20) (c04 64 30) (d04 64 40) (e04 64 50) (f04 64 60))
((a05 64 10) (b05 64 20) (c05 5 30) (d05 59 40) (e05 64 50) (f05 64 60))
((a06 64 10) (b06 64 20) (c06 44 30) (d06 64 40) (e06 64 50) (f06 64 60))
((a07 60 10) (b07 64 20) (c07 64 30) (d07 64 40) (e07 64 50) (f07 64 60))
((a08 64 10) (b08 64 20) (c08 64 30) (d08 64 40) (e08 64 50) (f08 64 60))
((a09 64 10) (b09 64 20) (c09 64 30) (d09 34 40) (e09 64 50) (f09 64 60))
((a10 64 10) (b10 64 20) (c10 64 30) (d10 64 40) (e10 64 50) (f10 64 60))
((a11 64 10) (b11 64 20) (c11 64 30) (d11 64 40) (e11 64 50) (f11 64 60))
((a12 64 10) (b12 64 20) (c12 64 30) (d12 64 40) (e12 64 50) (f12 64 60))
((a13 64 10) (b13 64 20) (c13 64 30) (d13 64 40) (e13 11 50) (f13 64 60))
((a14 64 10) (b14 64 20) (c14 64 30) (d14 30 40) (e14 64 50) (f14 64 60))
((a15 64 10) (b15 64 20) (c15 64 30) (d15 64 40) (e15 64 50) (f15 64 60))
((a16 64 10) (b16 64 20) (c16 64 30) (d16 1 40) (e16 30 50) (f16 64 60))
((a17 64 10) (b17 64 20) (c17 64 30) (d17 64 40) (e17 64 50) (f17 64 60))
((a18 64 10) (b18 64 20) (c18 64 30) (d18 64 40) (e18 64 50) (f18 64 60))
((a19 64 10) (b19 64 20) (c19 30 30) (d19 35 40) (e19 30 50) (f19 64 60))
((a20 64 10) (b20 64 20) (c20 64 30) (d20 64 40) (e20 64 50) (f20 64 60))
((a21 64 10) (b21 64 20) (c21 64 30) (d21 64 40) (e21 64 50) (f21 64 60))
((a22 64 10) (b22 64 20) (c22 64 30) (d22 64 40) (e22 64 50) (f22 64 60)))
```

pos/1.txt

```
1904
```

②	**Agregar producto** (con dos argumento)
	Agregará otra cantidad del producto en ventanilla en inventario elegido. En este caso, el primer producto (id 101) del inventario 2. Como la casilla está llena, aparecerá el mensaje de error.

Ejecución

```
clj:CLOJURE.ev3-V14:> (agregar 10 2)


_____________________________________
CARRUSEL/INVENTARIO 2 --------------

Agregado     ->  (g01 64 10)
ADVERTENCIA: Se agregó la cantidad máxima
nil
clj:CLOJURE.ev3-V14:>
```

Imagen de referencia de inventario + pos

invent/2.txt

```
((g01 64 10) (h01 64 20) (i01 64 30) (j01 64 40) (k01 64 50) (l01 64 60))
((g02 64 10) (h02 64 20) (i02 64 30) (j02 60 40) (k02 2 50) (l02 64 60))
((g03 60 10) (h03 64 20) (i03 64 30) (j03 64 40) (k03 64 50) (l03 64 60))
((g04 64 10) (h04 60 20) (i04 64 30) (j04 64 40) (k04 64 50) (l04 64 60))
((g05 64 10) (h05 64 20) (i05 60 30) (j05 6 40) (k05 64 50) (l05 64 60))
((g06 64 10) (h06 60 20) (i06 64 30) (j06 64 40) (k06 64 50) (l06 64 60))
((g07 64 10) (h07 64 20) (i07 59 30) (j07 64 40) (k07 60 50) (l07 64 60))
((g08 64 10) (h08 64 20) (i08 64 30) (j08 64 40) (k08 64 50) (l08 64 60))
((g09 64 10) (h09 64 20) (i09 64 30) (j09 64 40) (k09 64 50) (l09 64 60))
((g10 64 10) (h10 64 20) (i10 64 30) (j10 64 40) (k10 64 50) (l10 64 60))
((g11 64 10) (h11 64 20) (i11 64 30) (j11 64 40) (k11 64 50) (l11 64 60))
((g12 64 10) (h12 64 20) (i12 64 30) (j12 64 40) (k12 64 50) (l12 64 60))
((g13 64 10) (h13 64 20) (i13 64 30) (j13 64 40) (k13 64 50) (l13 64 60))
((g14 64 10) (h14 64 20) (i14 64 30) (j14 64 40) (k14 64 50) (l14 64 60))
((g15 64 10) (h15 64 20) (i15 64 30) (j15 64 40) (k15 64 50) (l15 64 60))
((g16 64 10) (h16 64 20) (i16 64 30) (j16 64 40) (k16 64 50) (l16 64 60))
((g17 64 10) (h17 64 20) (i17 5 30) (j17 64 40) (k17 64 50) (l17 64 60))
((g18 64 10) (h18 64 20) (i18 64 30) (j18 64 40) (k18 64 50) (l18 64 60))
((g19 64 10) (h19 64 20) (i19 64 30) (j19 64 40) (k19 64 50) (l19 64 60))
((g20 64 10) (h20 64 20) (i20 64 30) (j20 64 40) (k20 64 50) (l20 64 60))
((g21 64 10) (h21 64 20) (i21 64 30) (j21 64 40) (k21 64 50) (l21 64 60))
((g22 64 10) (h22 64 20) (i22 64 30) (j22 64 40) (k22 64 50) (l22 64 60)))
```

pos/2.txt

```
101
```

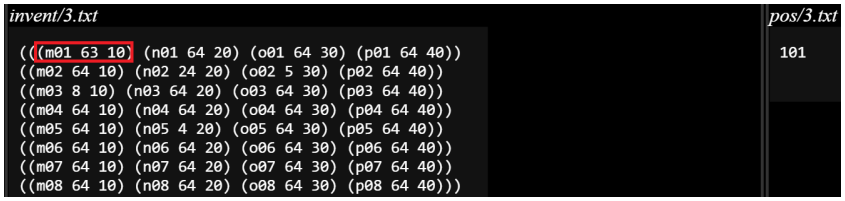
③ **Retirar producto** (con dos argumentos)

Retirará la cantidad de 1 del producto en ventanilla (id 101, el primero) del inventario 3.

Ejecución

```
clj:CLOJURE.ev3-V14:> (retirar 1 3)


_____________________________________
CARRUSEL/INVENTARIO 3 --------------

Retirado    ->  (m01 63 10)
""
clj:CLOJURE.ev3-V14:>
```

Imagen de referencia de inventario



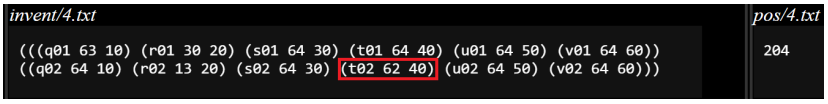
④ **Retirar producto** (con tres argumentos)

Retirará la cantidad del producto en inventario indicado.

Ejecución

```
clj:CLOJURE.ev3-V14:> (retirar 't02 2 4)


_____________________________________
CARRUSEL/INVENTARIO 4 --------------
Bv  (q02 64 10)
D>  (r02 13 20)
D>  (s02 64 30)
D>  (t02 64 40)
En ventanilla-> (t02 64 40)
Retirado     ->  (t02 62 40)
""
clj:CLOJURE.ev3-V14:>
```

Imagen de referencia de inventario



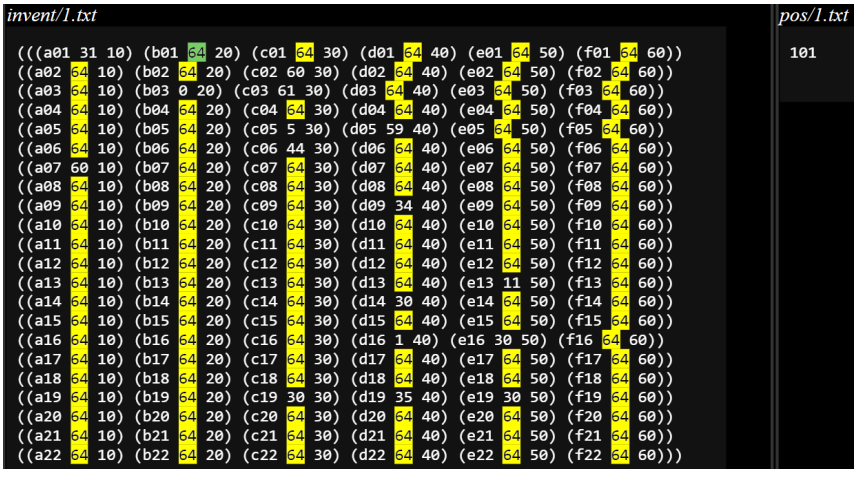
⑤ **Poco inventario**

Esta función mostrará todos los productos con una cantidad menor a 12 en inventario. Aplica para todos los carruseles de almacén. (Como el resultado es un poco largo, solo mostraré el resultado de los cuatro primeros inventarios. Los valores 64 están resaltados con amarillo para que sea más fácil leer el inventario)

Ejecución

```
clj:CLOJURE.ev3-V14:> (poco-inventario)
ADVERTENCIA productos con poca o nula existencia


_____________________________________
CARRUSEL/INVENTARIO 1 --------------

nombre:----- b03
  cantidad:  0
  ID: 1 - 302
  fila: 3, colm: 2, carrusel: 1

nombre:----- c05
  cantidad:  5
  ID: 1 - 503
  fila: 5, colm: 3, carrusel: 1

nombre:----- e13
  cantidad:  11
  ID: 1 - 1305
  fila: 13, colm: 5, carrusel: 1

nombre:----- d16
  cantidad:  1
  ID: 1 - 1604
```

Imagen de referencia de inventario



	`` fila: 16, colm: 4, carrusel: 1 (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ____________________________________ CARRUSEL/INVENTARIO 2 ------------- nombre:----- k02 cantidad: 2 ID: 2 - 205 fila: 2, colm: 5, carrusel: 2 ``` ``` nombre:----- j05 cantidad: 6 ID: 2 - 504 fila: 5, colm: 4, carrusel: 2 nombre:----- i17 cantidad: 5 ID: 2 - 1703 fila: 17, colm: 3, carrusel: 2 (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ____________________________________ CARRUSEL/INVENTARIO 3 ------------- nombre:----- o02 cantidad: 5 ID: 3 - 203 fila: 2, colm: 3, carrusel: 3 nombre:----- m03 cantidad: 8 ID: 3 - 301 fila: 3, colm: 1, carrusel: 3 nombre:----- n05 cantidad: 4 ID: 3 - 502 fila: 5, colm: 2, carrusel: 3 (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ____________________________________ CARRUSEL/INVENTARIO 4 ------------- (0 0 0 0 0 0 0 0 0 0 0 0) ```	*invent/2.txt* ``` (((g01 64 10) (h01 64 20) (i01 64 30) (j01 64 40) (k01 64 50) (l01 64 60)) ((g02 64 10) (h02 64 20) (i02 64 30) (j02 60 40) (k02 2 50) (l02 64 60)) ((g03 60 10) (h03 64 20) (i03 64 30) (j03 64 40) (k03 64 50) (l03 64 60)) ((g04 64 10) (h04 60 20) (i04 64 30) (j04 64 40) (k04 64 50) (l04 64 60)) ((g05 64 10) (h05 64 20) (i05 60 30) (j05 6 40) (k05 64 50) (l05 64 60)) ((g06 64 10) (h06 60 20) (i06 64 30) (j06 64 40) (k06 64 50) (l06 64 60)) ((g07 64 10) (h07 64 20) (i07 59 30) (j07 64 40) (k07 60 50) (l07 64 60)) ((g08 64 10) (h08 64 20) (i08 64 30) (j08 64 40) (k08 64 50) (l08 64 60)) ((g09 64 10) (h09 64 20) (i09 64 30) (j09 64 40) (k09 64 50) (l09 64 60)) ((g10 64 10) (h10 64 20) (i10 64 30) (j10 64 40) (k10 64 50) (l10 64 60)) ((g11 64 10) (h11 64 20) (i11 64 30) (j11 64 40) (k11 64 50) (l11 64 60)) ((g12 64 10) (h12 64 20) (i12 64 30) (j12 64 40) (k12 64 50) (l12 64 60)) ((g13 64 10) (h13 64 20) (i13 64 30) (j13 64 40) (k13 64 50) (l13 64 60)) ((g14 64 10) (h14 64 20) (i14 64 30) (j14 64 40) (k14 64 50) (l14 64 60)) ((g15 64 10) (h15 64 20) (i15 64 30) (j15 64 40) (k15 64 50) (l15 64 60)) ((g16 64 10) (h16 64 20) (i16 64 30) (j16 64 40) (k16 64 50) (l16 64 60)) ((g17 64 10) (h17 64 20) (i17 5 30) (j17 64 40) (k17 64 50) (l17 64 60)) ((g18 64 10) (h18 64 20) (i18 64 30) (j18 64 40) (k18 64 50) (l18 64 60)) ((g19 64 10) (h19 64 20) (i19 64 30) (j19 64 40) (k19 64 50) (l19 64 60)) ((g20 64 10) (h20 64 20) (i20 64 30) (j20 64 40) (k20 64 50) (l20 64 60)) ((g21 64 10) (h21 64 20) (i21 64 30) (j21 64 40) (k21 64 50) (l21 64 60)) ((g22 64 10) (h22 64 20) (i22 64 30) (j22 64 40) (k22 64 50) (l22 64 60))) ``` *pos/2.txt*: 101 *invent/3.txt* ``` (((m01 63 10) (n01 64 20) (o01 64 30) (p01 64 40)) ((m02 64 10) (n02 24 20) (o02 5 30) (p02 64 40)) ((m03 8 10) (n03 64 20) (o03 64 30) (p03 64 40)) ((m04 64 10) (n04 64 20) (o04 64 30) (p04 64 40)) ((m05 64 10) (n05 4 20) (o05 64 30) (p05 64 40)) ((m06 64 10) (n06 64 20) (o06 64 30) (p06 64 40)) ((m07 64 10) (n07 64 20) (o07 64 30) (p07 64 40)) ((m08 64 10) (n08 64 20) (o08 64 30) (p08 64 40))) ``` *pos/3.txt*: 101 *invent/4.txt* ``` (((q01 63 10) (r01 30 20) (s01 64 30) (t01 64 40) (u01 64 50) (v01 64 60)) ((q02 64 10) (r02 13 20) (s02 64 30) (t02 62 40) (u02 64 50) (v02 64 60))) ``` *pos/4.txt*: 204

⑥ Valor total de inventario

Función auxiliar de Valor total Almacén. Es el valor total para un inventario en particular. En este caso, el valor total del inventario 5.

```
clj:CLOJURE.ev3-V14:> (valor-total-inventario 4)
25090
clj:CLOJURE.ev3-V14:>
```

invent/4.txt

```
(((q01 63 10) (r01 30 20) (s01 64 30) (t01 64 40) (u01 64 50) (v01 64 60))
((q02 64 10) (r02 13 20) (s02 64 30) (t02 62 40) (u02 64 50) (v02 64 60)))
```

⑦ Valor total de almacén

Muestra los valores totales de cada inventario e imprime la sumatoria de éstos.

```
clj:CLOJURE.ev3-V14:> (valor-total-almacen)
Valor total de carruseles:(

[nil   Carrusel 1: $ 277940]
[nil   Carrusel 2: $ 287660]
[nil   Carrusel 3: $ 46860]
[nil   Carrusel 4: $ 25090]
[nil   Carrusel 5: $ 106860]
[nil   Carrusel 6: $ 284490]
[nil   Carrusel 7: $ 47580]
[nil   Carrusel 8: $ 280500]
[nil   Carrusel 9: $ 275480]
[nil   Carrusel 10: $ 111060])

Valor total del Almacen: $ 1743520
nil
clj:CLOJURE.ev3-V14:>
```

⑧	**Top10**
	Imprime el 10% de los almacenes con mayor valor total en orden. Como estoy utilizando diez carruseles se imprimirá uno, pero funciona para el 10% de la cantidad que sea de inventarios.
	Ejecución `clj:CLOJURE.ev3-V14:> (top10)` `Top 10% de inventarios con mayor valor total` `1 - Carrusel 2 : $ 287660` `nil` `clj:CLOJURE.ev3-V14:>`

Parte 2

A partir de aquí se realizan las ejecuciones con el archivo de texto 'entrada.txt' y la función (ejecutar).
Como los resultados son muy largos, por lo que solo mostraré el inicio y el fin de lo que se imprima en consola

⑨	**Agregar y retirar en un mismo inventario [1]**	
	entrada.txt `(agregar 1 1)` `(agregar 'e19 5 1)` `(agregar 'e13 70 1)` `(agregar 'f22 1 1)` `(agregar 'd14 1 1)` `(agregar 'd16 20 1)` `(retirar 1 1)` `(retirar 'a22 1 1)` `(retirar 'b22 1 1)` `(retirar 'c22 1 1)` `(retirar 'd22 1 1)` `(retirar 'e22 1 1)` `(retirar 'f22 1 1)`	consola `clj:CLOJURE.ev3-V14:> (ejecutar)` `((` `CARRUSEL/INVENTARIO 1 ----------` `Agregado -> (a01 32 10)` `CARRUSEL/INVENTARIO 1 ----------` `A^ (a22 64 10)` `A^ (a21 64 10)` `A^ (a20 64 10)` `A^ (a19 64 10)` `D> (b19 64 20)` `D> (c19 30 30)` `D> (d19 35 40)` `D> (e19 30 50)` `En ventanilla-> (e19 30 50)` `Agregado -> (e19 35 50)` `CARRUSEL/INVENTARIO 1 ----------` `A^ (e18 64 50)` `A^ (e17 64 50)` `A^ (e16 30 50)` `A^ (e15 64 50)` `A^ (e14 64 50)` `A^ (e13 11 50)` `En ventanilla-> (e13 11 50)` `Agregado -> (e13 64 50)` `ADVERTENCIA: Se agregó la cantidad máxima` […]

```
nombre:----- e519
  cantidad:  3
  ID: 10 - 1902
  fila: 19, colm: 2, carrusel: 10

nombre:----- f519
  cantidad:  11
  ID: 10 - 1903
  fila: 19, colm: 3, carrusel: 10

nombre:----- d525
  cantidad:  4
  ID: 10 - 2501
  fila: 25, colm: 1, carrusel: 10
(0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

[nil   Carrusel 1: $ 281440]
[nil   Carrusel 2: $ 287660]
[nil   Carrusel 3: $ 46860]
[nil   Carrusel 4: $ 25090]
[nil   Carrusel 5: $ 106860]
[nil   Carrusel 6: $ 284490]
[nil   Carrusel 7: $ 47580]
[nil   Carrusel 8: $ 280500]
[nil   Carrusel 9: $ 275480] [nil   Carrusel 10: $ 111060])

Valor total del Almacen: $ 1747020


Top 10% de inventarios con mayor valor total
1 - Carrusel 2 : $ 287660
  nil
clj:CLOJURE.ev3-V14:>
```

Inventario original

invent/1.txt

```
(((a01 31 10) (b01 64 20) (c01 64 30) (d01 64 40) (e01 64 50) (f01 64 60))
((a02 64 10) (b02 64 20) (c02 60 30) (d02 64 40) (e02 64 50) (f02 64 60))
((a03 64 10) (b03 0 20) (c03 61 30) (d03 64 40) (e03 64 50) (f03 64 60))
((a04 64 10) (b04 64 20) (c04 64 30) (d04 64 40) (e04 64 50) (f04 64 60))
((a05 64 10) (b05 64 20) (c05 5 30) (d05 59 40) (e05 64 50) (f05 64 60))
((a06 64 10) (b06 64 20) (c06 44 30) (d06 64 40) (e06 64 50) (f06 64 60))
((a07 60 10) (b07 64 20) (c07 64 30) (d07 64 40) (e07 64 50) (f07 64 60))
((a08 64 10) (b08 64 20) (c08 64 30) (d08 64 40) (e08 64 50) (f08 64 60))
((a09 64 10) (b09 64 20) (c09 64 30) (d09 34 40) (e09 64 50) (f09 64 60))
((a10 64 10) (b10 64 20) (c10 64 30) (d10 64 40) (e10 64 50) (f10 64 60))
((a11 64 10) (b11 64 20) (c11 64 30) (d11 64 40) (e11 64 50) (f11 64 60))
((a12 64 10) (b12 64 20) (c12 64 30) (d12 64 40) (e12 64 50) (f12 64 60))
((a13 64 10) (b13 64 20) (c13 64 30) (d13 64 40) (e13 11 50) (f13 64 60))
((a14 64 10) (b14 64 20) (c14 64 30) (d14 30 40) (e14 64 50) (f14 64 60))
((a15 64 10) (b15 64 20) (c15 64 30) (d15 64 40) (e15 64 50) (f15 64 60))
((a16 64 10) (b16 64 20) (c16 64 30) (d16 1 40) (e16 30 50) (f16 64 60))
((a17 64 10) (b17 64 20) (c17 64 30) (d17 64 40) (e17 64 50) (f17 64 60))
((a18 64 10) (b18 64 20) (c18 64 30) (d18 64 40) (e18 64 50) (f18 64 60))
((a19 64 10) (b19 64 20) (c19 30 30) (d19 35 40) (e19 30 50) (f19 64 60))
((a20 64 10) (b20 64 20) (c20 64 30) (d20 64 40) (e20 64 50) (f20 64 60))
((a21 64 10) (b21 64 20) (c21 64 30) (d21 64 40) (e21 64 50) (f21 64 60))
((a22 64 10) (b22 64 20) (c22 64 30) (d22 64 40) (e22 64 50) (f22 64 60)))
```

pos/1.txt

```
101
```

Inventario después de transacciones

invent/1.txt

```
(((a01 32 10) (b01 64 20) (c01 64 30) (d01 64 40) (e01 64 50) (f01 64 60))
((a02 64 10) (b02 64 20) (c02 60 30) (d02 64 40) (e02 64 50) (f02 64 60))
((a03 64 10) (b03 0 20) (c03 61 30) (d03 64 40) (e03 64 50) (f03 64 60))
((a04 64 10) (b04 64 20) (c04 64 30) (d04 64 40) (e04 64 50) (f04 64 60))
((a05 64 10) (b05 64 20) (c05 5 30) (d05 59 40) (e05 64 50) (f05 64 60))
((a06 64 10) (b06 64 20) (c06 44 30) (d06 64 40) (e06 64 50) (f06 64 60))
((a07 60 10) (b07 64 20) (c07 64 30) (d07 64 40) (e07 64 50) (f07 64 60))
((a08 64 10) (b08 64 20) (c08 64 30) (d08 64 40) (e08 64 50) (f08 64 60))
((a09 64 10) (b09 64 20) (c09 64 30) (d09 34 40) (e09 64 50) (f09 64 60))
((a10 64 10) (b10 64 20) (c10 64 30) (d10 64 40) (e10 64 50) (f10 64 60))
((a11 64 10) (b11 64 20) (c11 64 30) (d11 64 40) (e11 64 50) (f11 64 60))
((a12 64 10) (b12 64 20) (c12 64 30) (d12 64 40) (e12 64 50) (f12 64 60))
((a13 64 10) (b13 64 20) (c13 64 30) (d13 64 40) (e13 64 50) (f13 64 60))
((a14 64 10) (b14 64 20) (c14 64 30) (d14 31 40) (e14 64 50) (f14 64 60))
((a15 64 10) (b15 64 20) (c15 64 30) (d15 64 40) (e15 64 50) (f15 64 60))
((a16 64 10) (b16 64 20) (c16 64 30) (d16 20 40) (e16 30 50) (f16 64 60))
((a17 64 10) (b17 64 20) (c17 64 30) (d17 64 40) (e17 64 50) (f17 64 60))
((a18 64 10) (b18 64 20) (c18 64 30) (d18 64 40) (e18 64 50) (f18 64 60))
((a19 64 10) (b19 64 20) (c19 30 30) (d19 35 40) (e19 35 50) (f19 64 60))
((a20 64 10) (b20 64 20) (c20 64 30) (d20 64 40) (e20 64 50) (f20 64 60))
((a21 64 10) (b21 64 20) (c21 64 30) (d21 64 40) (e21 64 50) (f21 64 60))
((a22 63 10) (b22 63 20) (c22 63 30) (d22 63 40) (e22 63 50) (f22 63 60)))
```

pos/1.txt

```
2206
```

⑩ Agregar y retirar en un mismo inventario [2]

Esta ejecución será igual que la pasada, pero ahora en inventario 2, esto con el fin de disminuir su valor lo suficiente para que este inventario deje de ser el más valioso.

Entrada

```
(agregar 1 2)
(agregar 'k19 5 2)
(agregar 'k13 70 2)
(agregar 'l22 1 2)
(agregar 'j14 1 2)
(agregar 'j16 1 2)
(retirar 1 2)
(retirar 'g22 64 2)
(retirar 'h22 64 2)
(retirar 'i22 64 2)
(retirar 'j22 64 2)
(retirar 'k22 64 2)
(retirar 'l22 64 2)
(retirar 'l21 64 2)
```

Consola

```
clj:CLOJURE.ev3-V14:> (ejecutar)
((
____________________________________
CARRUSEL/INVENTARIO 2 -------------

Agregado     ->  (g01 64 10)
ADVERTENCIA: Se agregó la cantidad máxima
____________________________________
CARRUSEL/INVENTARIO 2 -------------
A^  (g22 64 10)
A^  (g21 64 10)
A^  (g20 64 10)
A^  (g19 64 10)
D>  (h19 64 20)
D>  (i19 64 30)
D>  (j19 64 40)
D>  (k19 64 50)
En ventanilla-> (k19 64 50)
Agregado     ->  (k19 64 50)
ADVERTENCIA: Se agregó la cantidad máxima
____________________________________
CARRUSEL/INVENTARIO 2 -------------
A^  (k18 64 50)
A^  (k17 64 50)
A^  (k16 64 50)
A^  (k15 64 50)
A^  (k14 64 50)
A^  (k13 64 50)
En ventanilla-> (k13 64 50)
Agregado     ->  (k13 64 50)
ADVERTENCIA: Se agregó la cantidad máxima
____________________________________
CARRUSEL/INVENTARIO 2 -------------
```

```
(retirar 'l20 64 2)
(retirar 'l19 64 2)
(retirar 'l18 64 2)
(retirar 'l17 64 2)
(retirar 'l16 64 2)
(retirar 'l15 64 2)
(retirar 'l14 64 2)
(retirar 'l13 64 2)
(retirar 'l12 64 2)
(retirar 'l11 64 2)
(retirar 'l10 64 2)
(retirar 'l09 64 2)
(retirar 'l08 64 2)
(retirar 'l07 64 2)
(retirar 'l06 64 2)
(retirar 'l05 64 2)
(retirar 'l04 64 2)
(retirar 'l03 64 2)
(retirar 'l02 64 2)
(retirar 'l01 64 2)
```

[…]

```
nombre:----- e519
  cantidad:  3
  ID: 10 - 1902
  fila: 19, colm: 2, carrusel: 10

nombre:----- f519
  cantidad:  11
  ID: 10 - 1903
  fila: 19, colm: 3, carrusel: 10

nombre:----- d525
  cantidad:  4
  ID: 10 - 2501
  fila: 25, colm: 1, carrusel: 10
(0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

[nil   Carrusel 1: $ 281440]
[nil   Carrusel 2: $ 193540]
[nil   Carrusel 3: $ 46860]
[nil   Carrusel 4: $ 25090]
[nil   Carrusel 5: $ 106860]
[nil   Carrusel 6: $ 284490]
[nil   Carrusel 7: $ 47580]
[nil   Carrusel 8: $ 280500]
[nil   Carrusel 9: $ 275480] [nil   Carrusel 10: $ 111060])

Valor total del Almacen: $ 1652900


Top 10% de inventarios con mayor valor total
1 - Carrusel 6 : $ 284490
nil
clj:CLOJURE.ev3-V14:>
```

Inventario original

invent/2.txt

```
(((g01 64 10) (h01 64 20) (i01 64 30) (j01 64 40) (k01 64 50) (l01 64 60))
((g02 64 10) (h02 64 20) (i02 64 30) (j02 60 40) (k02 2 50) (l02 64 60))
((g03 60 10) (h03 64 20) (i03 64 30) (j03 64 40) (k03 64 50) (l03 64 60))
((g04 64 10) (h04 60 20) (i04 64 30) (j04 64 40) (k04 64 50) (l04 64 60))
((g05 64 10) (h05 64 20) (i05 60 30) (j05 6 40) (k05 64 50) (l05 64 60))
((g06 64 10) (h06 60 20) (i06 64 30) (j06 64 40) (k06 64 50) (l06 64 60))
((g07 64 10) (h07 64 20) (i07 59 30) (j07 64 40) (k07 60 50) (l07 64 60))
((g08 64 10) (h08 64 20) (i08 64 30) (j08 64 40) (k08 64 50) (l08 64 60))
((g09 64 10) (h09 64 20) (i09 64 30) (j09 64 40) (k09 64 50) (l09 64 60))
((g10 64 10) (h10 64 20) (i10 64 30) (j10 64 40) (k10 64 50) (l10 64 60))
((g11 64 10) (h11 64 20) (i11 64 30) (j11 64 40) (k11 64 50) (l11 64 60))
((g12 64 10) (h12 64 20) (i12 64 30) (j12 64 40) (k12 64 50) (l12 64 60))
((g13 64 10) (h13 64 20) (i13 64 30) (j13 64 40) (k13 64 50) (l13 64 60))
((g14 64 10) (h14 64 20) (i14 64 30) (j14 64 40) (k14 64 50) (l14 64 60))
((g15 64 10) (h15 64 20) (i15 64 30) (j15 64 40) (k15 64 50) (l15 64 60))
((g16 64 10) (h16 64 20) (i16 64 30) (j16 64 40) (k16 64 50) (l16 64 60))
((g17 64 10) (h17 64 20) (i17 5 30) (j17 64 40) (k17 64 50) (l17 64 60))
((g18 64 10) (h18 64 20) (i18 64 30) (j18 64 40) (k18 64 50) (l18 64 60))
((g19 64 10) (h19 64 20) (i19 64 30) (j19 64 40) (k19 64 50) (l19 64 60))
((g20 64 10) (h20 64 20) (i20 64 30) (j20 64 40) (k20 64 50) (l20 64 60))
((g21 64 10) (h21 64 20) (i21 64 30) (j21 64 40) (k21 64 50) (l21 64 60))
((g22 64 10) (h22 64 20) (i22 64 30) (j22 64 40) (k22 64 50) (l22 64 60)))
```

pos/2.txt

```
101
```

Inventario después de transacciones

invent/2.txt

```
(((g01 64 10) (h01 64 20) (i01 64 30) (j01 64 40) (k01 64 50) (l01 0 60))
((g02 64 10) (h02 64 20) (i02 64 30) (j02 60 40) (k02 2 50) (l02 0 60))
((g03 60 10) (h03 64 20) (i03 64 30) (j03 64 40) (k03 64 50) (l03 0 60))
((g04 64 10) (h04 60 20) (i04 64 30) (j04 64 40) (k04 64 50) (l04 0 60))
((g05 64 10) (h05 64 20) (i05 60 30) (j05 6 40) (k05 64 50) (l05 0 60))
((g06 64 10) (h06 60 20) (i06 64 30) (j06 64 40) (k06 64 50) (l06 0 60))
((g07 64 10) (h07 64 20) (i07 59 30) (j07 64 40) (k07 60 50) (l07 0 60))
((g08 64 10) (h08 64 20) (i08 64 30) (j08 64 40) (k08 64 50) (l08 0 60))
((g09 64 10) (h09 64 20) (i09 64 30) (j09 64 40) (k09 64 50) (l09 0 60))
((g10 64 10) (h10 64 20) (i10 64 30) (j10 64 40) (k10 64 50) (l10 0 60))
((g11 64 10) (h11 64 20) (i11 64 30) (j11 64 40) (k11 64 50) (l11 0 60))
((g12 64 10) (h12 64 20) (i12 64 30) (j12 64 40) (k12 64 50) (l12 0 60))
((g13 64 10) (h13 64 20) (i13 64 30) (j13 64 40) (k13 64 50) (l13 0 60))
((g14 64 10) (h14 64 20) (i14 64 30) (j14 64 40) (k14 64 50) (l14 0 60))
((g15 64 10) (h15 64 20) (i15 64 30) (j15 64 40) (k15 64 50) (l15 0 60))
((g16 64 10) (h16 64 20) (i16 64 30) (j16 63 40) (k16 64 50) (l16 0 60))
((g17 64 10) (h17 64 20) (i17 5 30) (j17 64 40) (k17 64 50) (l17 0 60))
((g18 64 10) (h18 64 20) (i18 64 30) (j18 64 40) (k18 64 50) (l18 0 60))
((g19 64 10) (h19 64 20) (i19 64 30) (j19 64 40) (k19 64 50) (l19 0 60))
((g20 64 10) (h20 64 20) (i20 64 30) (j20 64 40) (k20 64 50) (l20 0 60))
((g21 64 10) (h21 64 20) (i21 64 30) (j21 64 40) (k21 64 50) (l21 0 60))
((g22 0 10) (h22 0 20) (i22 0 30) (j22 0 40) (k22 0 50) (l22 0 60)))
```

pos/2.txt

```
106
```

⑪ Agregar y retirar en todos los inventarios

Entrada

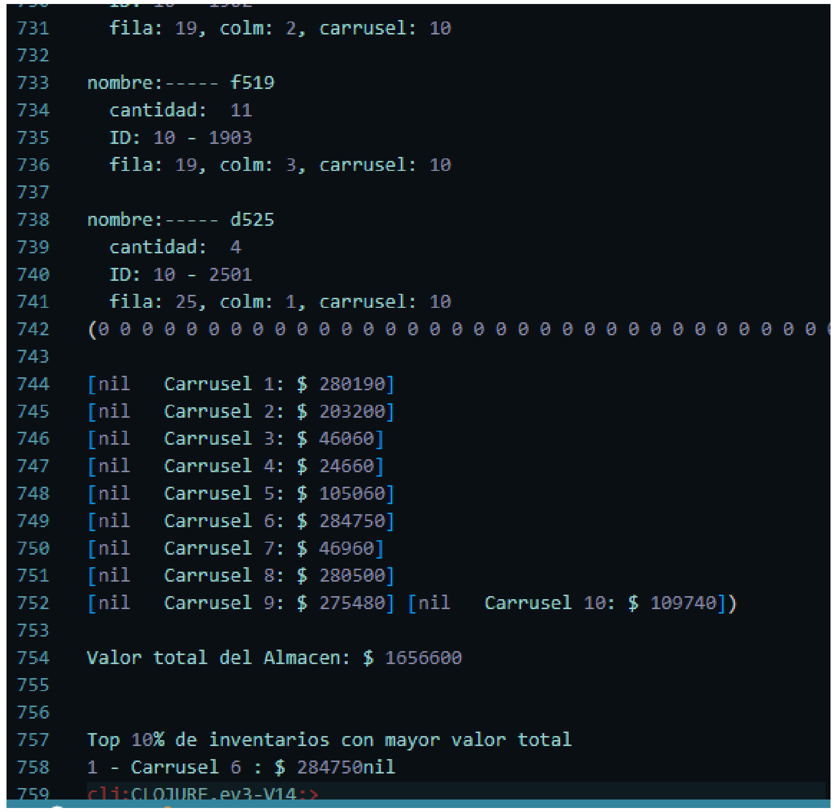
```
(retirar 1 2)
(retirar 'k19 5 2)
(retirar 'k13 70 2)
(retirar 'l22 1 2)
(retirar 'j14 1 2)
(retirar 'j16 1 2)
(agregar 1 2)
(agregar 'g22 64 2)
(agregar 'h22 64 2)
(agregar 'i22 64 2)
(agregar 'j22 64 2)
(agregar 'k22 64 2)
(agregar 'l22 64 2)
(retirar 1 1)
(retirar 'a19 5 1)
(retirar 'b13 70 1)
(retirar 'c22 1 1)
```

Consola

```
clj:CLOJURE.ev3-V14:> (ejecutar)
((
___________________________________
CARRUSEL/INVENTARIO 2 --------------

Retirado    ->  (l22 63 60)


___________________________________
CARRUSEL/INVENTARIO 2 --------------
A^  (l21 0 60)
A^  (l20 0 60)
A^  (l19 0 60)
I<  (k19 59 50)
En ventanilla-> (k19 59 50)
Retirado     ->  (k19 54 50)


___________________________________
CARRUSEL/INVENTARIO 2 --------------
A^  (k18 64 50)
A^  (k17 64 50)
A^  (k16 64 50)
A^  (k15 64 50)
A^  (k14 64 50)
A^  (k13 0 50)
En ventanilla-> (k13 0 50)
Retirado     ->  (k13 0 50)
ADVERTENCIA: Se retiró la cantidad máxima

___________________________________
```

```
(retirar 'd14 1 1)
(retirar 'e16 1 1)
(agregar 1 1)
(agregar 'f22 64 1)
(agregar 'e22 64 1)
(agregar 'd22 64 1)
(agregar 'b22 64 1)
(agregar 'a22 64 1)
(agregar 'c22 64 1)
(agregar 'm03 70 3)
(agregar 'n05 70 3)
(retirar 'p08 65 3)
(retirar 70 4)
(agregar 'r02 10 4)
(retirar 'b501 10 5)
(retirar 'b502 10 5)
(retirar 'b503 10 5)
(retirar 'b504 10 5)
(retirar 'b505 10 5)
(retirar 'b506 10 5)
(retirar 'b507 10 5)
(retirar 'b508 10 5)
(retirar 'b509 10 5)
(retirar 'b510 10 5)
(agregar 100 6)
(agregar 'a607 4 6)
(retirar 'a622 30 6)
(retirar 100 7)
(retirar 'f501 4 10)
(retirar 'f502 4 10)
(retirar 'f503 4 10)
(retirar 'f504 4 10)
(retirar 'f505 4 10)
(retirar 'f506 4 10)
(retirar 'f507 4 10)
(retirar 'f508 4 10)
(retirar 'f509 4 10)
(retirar 'f510 4 10)
(retirar 'f511 4 10)
```

[…]


```
  fila: 19, colm: 2, carrusel: 10

nombre:----- f519
  cantidad:  11
  ID: 10 - 1903
  fila: 19, colm: 3, carrusel: 10

nombre:----- d525
  cantidad:  4
  ID: 10 - 2501
  fila: 25, colm: 1, carrusel: 10
(0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

[nil   Carrusel 1: $ 280190]
[nil   Carrusel 2: $ 203200]
[nil   Carrusel 3: $ 46060]
[nil   Carrusel 4: $ 24660]
[nil   Carrusel 5: $ 105060]
[nil   Carrusel 6: $ 284750]
[nil   Carrusel 7: $ 46960]
[nil   Carrusel 8: $ 280500]
[nil   Carrusel 9: $ 275480] [nil   Carrusel 10: $ 109740])

Valor total del Almacen: $ 1656600


Top 10% de inventarios con mayor valor total
1 - Carrusel 6 : $ 284750nil
```

9. Reflexión

Si había pensado que sería relativamente fácil adaptarme a este lenguaje después de Racket, me equivoqué. En estructura es muy similar, pero con Clojure tuve problemas desde cómo usarlo; no funcionaba en replit, no pude cambiar el directorio raíz en la extensión de VS, pero después de todo pude terminar este programa. Como ya había mencionado antes, también tuve problemas con la paralelización, manejo de datos, creación y escritura de archivos, entre otros. A pesar de todo, aprendí muchas cosas utilizando este lenguaje, después de pasar todos estos problemas, al final pude resolverlos y cómo manejarlos.