

```

~\Downloads\Ejercicios-tipo-control\Ejercicio-Roulette\TwoListsDock.hs

1  -- =====
2  -- ===== Puntuación máxima 1.50 =====
3  -- =====
4  -- Un Dock es una secuencia de elementos en el que hay uno destacado (sign)
5  -- Las operaciones disponibles para un dock son:
6  -- Borrar el elemento destacado, añadir un nuevo elemento delante o detrás del destacado,
7  -- cambiar el elemento destacado al elemento anterior o al siguiente,
8  -- saber si la secuencia es vacía o si el destacado es el primero o el último
9  -- y crear un dock a partir de una lista
10 {- !!IMPORTANTE!! !!IMPORTANTE!! !!IMPORTANTE!! !!IMPORTANTE!! !!IMPORTANTE!!
11
12     Añadir a LinearQueue.hs la siguiente función para encolar un elemento al comienzo de
13     la lista;
14
15     enqueueFront :: a -> Queue a -> Queue a
16     enqueueFront x Empty = Node x Empty
17     enqueueFront x q = Node x q
18 -}
19
20 module DataStructures.Dock.TwoListsDock (
21     empty,           -- :: Dock a
22     isEmpty,         -- :: Dock a -> Bool
23     sign,            -- :: Dock a -> a
24     isFirst,          -- :: Dock a -> Bool
25     isLast,           -- :: Dock a -> Bool
26     left,             -- :: Dock a -> Bool
27     right,            -- :: Dock a -> Dock a
28     delete,           -- :: Dock a -> Dock a
29     insertL,          -- :: a -> Dock a -> Dock a
30     insertR,          -- :: a -> Dock a -> Dock a
31     listToDock        -- :: [a] -> Dock a
32 ) where
33
34 import Data.List(intercalate)
35 import qualified DataStructures.Queue.LinearQueue as S
36 import Test.QuickCheck
37
38 -- Vamos a implementar el dock con dos listas:
39 -- En la primera lista estarán los elementos anteriores al destacado siendo
40 -- el mas cercano al destacado el que está en la cima.
41 -- En la segunda lista, el destacado será la cima e seguirán el resto siendo
42 -- el de la derecha del destacado el que está el siguiente en la lista.
43 -- INVARIANTES:
44 -- Siempre hay un elemento destacado (salvo en el dock vacío).
45 -- Siempre mantendremos el elemento destacado en la cima de la segunda lista.
46
47 data Dock a = D (S.Queue a) (S.Queue a) deriving Eq
48 -- Ejemplo. Si tenemos el dock: 1 2 3 <4> 5 donde el elemento destacado es el 4, se
49 -- presentará por el dato
50 -- D S.enqueue 1 $ S.enqueue 2 $ S.enqueue 3 S.empty S.enqueue 5 $ S.enqueue 4 S.empty
51 -- Su show será TwoStackDock(1,2,3,<4>,5)
52 -- y las filas se verán así:
53 -- 3
54 -- 2 4 <- el destacado siempre en la cima de la segunda fila
55 -- 1 5
56 -- -----

```

```

56 -- NOTA: Todas las operaciones con filas están cualificadas con S: S.Queue, S.enqueue,
57 S.dequeue, etc.
58 sample1 = D (S.enqueue 1 $ S.enqueue 2 $ S.enqueue 3 S.empty) (S.enqueue 5 $ S.enqueue 4
59 S.empty)
60 -- =====
61 -- Ejercicio 1 (0.05 ptos.)
62 -- Crea una dock vacío
63 empty :: Dock a
64 empty = D S.empty S.empty
65 -- Ejercicio 2 (0.05 ptos.)
66 -- Determina si un dock está vacío
67 isEmpty :: Dock a -> Bool
68 isEmpty (D si sd) = S.isEmpty si && S.isEmpty sd
69 -- =====
70 -- Ejercicio 3 (0.10 ptos.)
71 -- Devuelve el elemento destacado. Error si está vacío
72 sign :: Dock a -> a
73 sign d@(D si sd) = S.first sd
74 {-}
75 Prelude (Dock.hs)> sign sample1
76 4
77 -- =====
78 -- Ejercicio 4 (0.10 ptos.)
79 -- Devuelve cierto si el destacado es el primero (o está vacía)
80 isFirst :: Dock a -> Bool
81 isFirst (D si sd)
82   | (S.isEmpty si) && (not (S.isEmpty sd)) = True
83   | (S.isEmpty si) && (S.isEmpty sd) = True
84   | otherwise = False
85 {-}
86 Prelude (Dock.hs)> isFirst sample1
87 False
88 -- =====
89 -- Ejercicio 5 (0.10 ptos.)
90 -- Devuelve cierto si el destacado es el último (o está vacía)
91 isLast :: Dock a -> Bool
92 isLast a@(D si sd) = not(S.isEmpty sd) && S.isEmpty (S.dequeue sd)
93           || (isEmpty (D si sd))
94 {-}
95 Prelude (Dock.hs)> isLast sample1
96 False
97 -- =====
98 -- Ejercicio 6 (0.10 ptos.)
99 -- Cambia el elemento destacado que pasa a ser el de la izquierda.
100 -- Si está vacío o el destacado es el primero lo deja igual
101 left :: Dock a -> Dock a
102 left d@(D si sd)
103   | isEmpty d      = D S.empty S.empty
104   | S.isEmpty si    = d
105   | otherwise       = D (S.dequeue si) (S.enqueueFront (S.first si) sd)
106 {-}
107 Prelude (Dock.hs)> left sample1
108 TwoStackDock(1,2,<3>,4,5)
109 Prelude (Dock.hs)> left $ left sample1
110 TwoStackDock(1,<2>,3,4,5)
111 Prelude (Dock.hs)> left $ left $ left sample1
112 TwoStackDock(<1>,2,3,4,5)
113

```

```

114 Prelude (Dock.hs)> left $ left $ left $ left sample1
115 TwoStackDock(<1>,2,3,4,5)
116 Prelude (Dock.hs)> isFirst $ left $ left $ left sample1
117 True
118 {-}
119 --- =====
120 -- Ejercicio 7 (0.25 ptos.)
121 -- El elemento destacado pasa a ser el de la derecha.
122 -- Si está vacío o el destacado es el último lo deja igual
123 right :: Dock a -> Dock a
124 right d@(D si sd)
125   | isEmpty d      = D S.empty S.empty
126   | S.isEmpty (S.dequeue sd) = d
127   | otherwise        = D (S.enqueueFront (S.first sd) si) (S.dequeue sd)
128 {-}
129 Prelude (Dock.hs)> right sample1
130 TwoStackDock(1,2,3,4,<5>)
131 Prelude (Dock.hs)> right $ right sample1
132 TwoStackDock(1,2,3,4,<5>)
133 Prelude (Dock.hs)> isLast $ right sample1
134 True
135 {-}
136 --- =====
137 -- Ejercicio 8 (0.25 ptos.)
138 -- Elimina el objeto destacado. El destacado pasa a ser el siguiente.
139 -- Si no hay siguiente pasa a ser el anterior.
140 -- Si queda vacía no hay destacado.
141 -- Error si está vacía
142 delete :: Dock a -> Dock a
143 delete d@(D si sd)
144   | isEmpty d = error "Esta vacia"
145   | not(S.isEmpty si) && S.isEmpty (S.dequeue sd) = D (S.dequeue si) (S.enqueue (S.first si) (S.dequeue sd))
146   | otherwise = D si (S.dequeue sd)
147 {-}
148 Prelude (Dock.hs)> delete sample1
149 TwoStackDock(1,2,3,<5>)
150 Prelude (Dock.hs)> delete $ delete sample1
151 TwoStackDock(1,2,<3>)
152 Prelude (Dock.hs)> delete $ delete $ delete sample1
153 TwoStackDock(1,<2>)
154 Prelude (Dock.hs)> delete $ delete $ delete $ delete sample1
155 TwoStackDock(<1>)
156 Prelude (Dock.hs)> delete $ delete $ delete $ delete $ delete sample1
157 TwoStackDock()
158 {-}
159 --- =====
160 -- Ejercicio 9 (0.15 ptos.)
161 -- inserta el elemento a la izquierda del destacado y este elemento pasa a ser el destacado
162 insertl :: a -> Dock a -> Dock a
163 insertl x (D si sd) = D si (S.enqueueFront x sd)
164 --- =====
165 -- Ejercicio 10 (0.15 ptos.)
166 -- inserta el elemento a la derecha del destacado y este elemento pasa a ser el destacado
167 insertr :: a -> Dock a -> Dock a
168 insertr x d@(D si sd) = D (S.enqueueFront (S.first sd) si) (S.enqueueFront x (S.dequeue sd))
169 {-}
170 Prelude (Dock.hs)> insertl 8 sample1
171 TwoStackDock(1,2,3,<8>,4,5)

```

```

172 | Prelude (Dock.hs)> insertr 9 $ insertl 8 sample1
173 | TwoStackDock(1,2,3,8,<9>,4,5)
174 | Prelude (Dock.hs)> insertr 9 $ right sample1
175 | TwoStackDock(1,2,3,4,5,<9>)
176 | Prelude (Dock.hs)> insertl 5 empty
177 | TwoStackDock(<5>)
178 | -}
179 | -- =====
180 | -- Ejercicio 11 (0.15 ptos.)
181 | -- genera un dock con los elementos de la lista. El destacado será el primero de la lista
182 | listToDock :: [a] -> Dock a
183 | listToDock = foldr insertl empty
184 | {-
185 | Prelude (Dock.hs)> listToDock [1..5]
186 | TwoStackDock(<1>,2,3,4,5)
187 | -}
188 |
189 | instance (Show a) => Show (Dock a) where
190 |     show d@(D si sd) | isEmpty d = "TwoStackDock()"
191 |     show (D si sd) = "TwoStackDock("++(intercalate "," (reverse (aux si))) ++ (if ei then
192 |         "<" else ">") ++ y ++(if ed then ">" else "<") ++ (intercalate "," ys)++ ")"
193 |             where
194 |                 aux s
195 |                     | S.isEmpty s = []
196 |                     | otherwise = show x : aux s'
197 |             where
198 |                 x = S.first s
199 |                 s' = S.dequeue s
200 |                 (y:ys) = aux sd
201 |                 ei = S.isEmpty si
202 |                 ed = S.isEmpty (S.dequeue sd)
203 |
204 | instance Arbitrary a => Arbitrary (Dock a) where
205 |     arbitrary = do
206 |         xs <- listOf arbitrary
207 |         return (foldr insertl empty xs)

```