# Examen-febrero-2015-Haskell.pdf

Perejy

Estructuras de Datos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

```haskell
-------------------------------------------------------------------------------
-- Estructuras de Datos. Grado en Informática, IS e IC. UMA.
-- Examen de Febrero 2015.
--
-- Implementación del TAD Deque
--
-- Apellidos:
-- Nombre:
-- Grado en Ingeniería ...
-- Grupo:
-- Número de PC:
-------------------------------------------------------------------------------

module TwoListsDoubleEndedQueue
   ( DEQue
   , empty
   , isEmpty
   , first
   , last
   , addFirst
   , addLast
   , deleteFirst
   , deleteLast
   ) where

import Prelude hiding (last)
import Data.List(intercalate)
import Test.QuickCheck

data DEQue a = DEQ [a] [a]

-- Complexity: O(1)
empty :: DEQue a
empty = DEQ [] []

-- Complexity: O(1)
isEmpty :: DEQue a -> Bool
isEmpty (DEQ [] []) = True
isEmpty _           = False


-- Complexity:  O(1)
addFirst :: a -> DEQue a -> DEQue a
--addFirst x empty = DEQ [x] []
addFirst a (DEQ xs ys) = DEQ (a:xs) ys

-- Complexity:  O(1)
addLast :: a -> DEQue a -> DEQue a
addLast a (DEQ xs ys) = DEQ xs (a:ys)

-- Complexity: O(1)
first :: DEQue a -> a
first (DEQ (x:xs) ys) = x
first (DEQ [] ys) = head(reverse ys)

-- Complexity:  O(1)
last :: DEQue a -> a
last (DEQ xs (y:ys)) = y
last (DEQ xs []) = head(reverse xs)
```

```haskell

-- Complexity: O(log n)
deleteFirst :: DEQue a -> DEQue a
deleteFirst (DEQ [] []) = empty
deleteFirst (DEQ (x:xs) ys) = DEQ xs ys
deleteFirst (DEQ [] (y:ys)) =deleteFirst (DEQ xs' ys')
         where
            xs' = reverse (snd (splitAt (sz) (y:ys)))
            ys' = fst (splitAt (sz) (y:ys))
            sz = div (length (y:ys)) 2

-- Complexity: O(log n)
deleteLast :: DEQue a -> DEQue a
deleteLast (DEQ [] []) = empty
deleteLast (DEQ xs (y:ys)) = DEQ xs ys
deleteLast (DEQ (x:xs) []) = deleteLast (DEQ xs' ys')
         where
            ys' = reverse (snd (splitAt (sz) (x:xs)))
            xs' = fst (splitAt (sz) (x:xs))
            sz = div (length (x:xs)) 2



instance (Show a) => Show (DEQue a) where
   show q = "TwoListsDoubleEndedQueue(" ++ intercalate "," [show x | x <- toList q]
++ ")"

toList :: DEQue a -> [a]
toList (DEQ xs ys) =  xs ++ reverse ys

instance (Eq a) => Eq (DEQue a) where
   q == q' =  toList q == toList q'

instance (Arbitrary a) => Arbitrary (DEQue a) where
   arbitrary =  do
      xs <- listOf arbitrary
      ops <- listOf (oneof [return addFirst, return addLast])
      return (foldr id empty (zipWith ($) ops xs))

```