The ADT ‘Stack’

```
module Stack ( Stack, emptyStack, isEmptyStack, push, pop, top ) where

-------------------------------------------------------------------------------
-- I N T E R F A C E  :  P U B L I C
-------------------------------------------------------------------------------

-- Stack a : a last-in first-out collection of items of type ‘a’

-------------------------------------------------------------------------------

-- emptyStack : the empty stack

emptyStack :: Stack a

-------------------------------------------------------------------------------

-- isEmptyStack s : is stack ‘s’ empty ?

isEmptyStack :: Stack a -> Bool

-------------------------------------------------------------------------------

-- push x s : the stack formed by placing item ‘x’ onto the top of stack ‘s’

push :: a -> Stack a -> Stack a

-------------------------------------------------------------------------------

-- pop s : the stack formed by removing its top item
--         from the non-empty stack ‘s’

pop :: Stack a -> Stack a

-------------------------------------------------------------------------------

-- top s : the top item of the non-empty stack ‘s’

top :: Stack a -> a

-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
-- I M P L E M E N T A T I O N  :  P R I V A T E
-------------------------------------------------------------------------------

data Stack a = EmptyStack | Push a ( Stack a )

-------------------------------------------------------------------------------

emptyStack = EmptyStack

-------------------------------------------------------------------------------

isEmptyStack EmptyStack = True
isEmptyStack _          = False

-------------------------------------------------------------------------------

push x s = Push x s

-------------------------------------------------------------------------------

pop ( Push _ s ) = s  -- crashes on empty stack

-------------------------------------------------------------------------------

top ( Push x _ ) = x  -- crashes on empty stack

-------------------------------------------------------------------------------

{-
-------------------------------------------------------------------------------
-- A L T E R N A T E   I M P L E M E N T A T I O N
-------------------------------------------------------------------------------

type Stack a = [ a ]

-------------------------------------------------------------------------------

emptyStack = [ ]

-------------------------------------------------------------------------------

isEmptyStack = null

-------------------------------------------------------------------------------

push x s = x : s

-------------------------------------------------------------------------------

pop = tail  -- crashes on empty stack

-------------------------------------------------------------------------------

top = head  -- crashes on empty stack

-------------------------------------------------------------------------------
-}
```

The ADT ‘STACK’

```
module STACK ( module Stack, listToStack, stackToList, invertStack ) where

import Stack

-------------------------------------------------------------------------------
-- I N T E R F A C E  :  P U B L I C  :  all exports of module 'Stack', plus :
-------------------------------------------------------------------------------

-- listToStack xs : the stack composed of the items of list 'xs',
--                  arranged so that the first item of 'xs' is on top

listToStack :: [ a ] -> Stack a

-------------------------------------------------------------------------------

-- stackToList s : the list composed of the items of stack 's',
--                 arranged so that the top item of 's' is first

stackToList :: Stack a -> [ a ]

-------------------------------------------------------------------------------

-- invertStack s : a copy of the stack 's' with items in inverted order

invertStack :: Stack a -> Stack a

-------------------------------------------------------------------------------
-- I M P L E M E N T A T I O N  :  P R I V A T E
-------------------------------------------------------------------------------

listToStack [ ]       = emptyStack
listToStack ( x : xs ) = push x ( listToStack xs )

-------------------------------------------------------------------------------

stackToList s = if isEmptyStack s then [ ]
                                  else top s : stackToList ( pop s )

-------------------------------------------------------------------------------

invertStack s = pour s emptyStack

-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

-- pour s1 s2 : the stack 's2' with an inverted copy of the stack 's1' on top

pour :: Stack a -> Stack a -> Stack a

pour s1 s2 = if isEmptyStack s1 then s2
                                else pour ( pop s1 ) ( push ( top s1 ) s2 )

-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------

$ ghci Stack.hs
GHCi, version 7.4.1: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
[1 of 1] Compiling Stack            ( Stack.hs, interpreted )
Ok, modules loaded: Stack.

Stack> isEmptyStack emptyStack
True

Stack> isEmptyStack ( push 'a' emptyStack )
False

Stack> top ( pop ( push 'b' ( push 'a' emptyStack ) ) )
'a'

Stack> top emptyStack
*** Exception: Stack.hs:86:1-20: Non-exhaustive patterns in function top

Stack> top ( pop emptyStack )
*** Exception: Stack.hs:82:1-20: Non-exhaustive patterns in function pop

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

$ ghci STACK.hs
GHCi, version 7.4.1: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
[1 of 2] Compiling Stack            ( Stack.hs, interpreted )
[2 of 2] Compiling STACK            ( STACK.hs, interpreted )
Ok, modules loaded: STACK, Stack.

STACK> stackToList ( listToStack [ 1 .. 5 ] )
[1,2,3,4,5]

STACK> stackToList ( invertStack ( listToStack [ 1 .. 5 ] ) )
[5,4,3,2,1]

STACK> stackToList ( invertStack ( listToStack
                                     [ "Yoda", "is", "indeed", "wise" ] ) )
["wise","indeed","is","Yoda"]

-------------------------------------------------------------------------------
```