```haskell
-- PROPERTY TEST ------------------------------------------S


-- Fold L
genStack :: [a] -> Stack a
genStack []     = Empty
genStack (x:xs) = Stack (genStack xs) x

stackToL :: Stack a -> [a]
stackToL Empty       = []
stackToL (Stack s a) = a : stackToL s

-- test with + - * div
qcFoldl :: Eq b => (b -> a -> b) -> b -> [a] -> Bool
qcFoldl f z ls =
  l == c
  where
    s = genStack ls
    c = foldl f z ls
    l = stackFoldl f z s

qcFoldlDiv :: Int -> [Int] -> Property
qcFoldlDiv z ls =
  z /= 0 && all (\a -> a /= 0) ls ==> qcFoldl (div) z ls

{-

*Stack> quickCheck $ qcFoldl (+)
+++ OK, passed 100 tests.
*Stack> quickCheck $ qcFoldl (-)
+++ OK, passed 100 tests.
*Stack> quickCheck $ qcFoldl (*)
+++ OK, passed 100 tests.
*Stack> quickCheck $ qcFoldlDiv
+++ OK, passed 100 tests; 40 discarded.

-}
```

```haskell
-- Requires Commutative Opeartions +, *
qcFoldl1 :: Eq a => (a -> a -> a) -> a -> [a] -> Bool
qcFoldl1 f z ls =
  l' == r' && c == l'
  where
    s = genStack ls
    l' = stackFoldl f z s
    r' = stackFoldr f z s
    c = foldl f z ls

{-

*Stack> quickCheck $ qcFoldl1 (*)
+++ OK, passed 100 tests.
*Stack> quickCheck $ qcFoldl1 (+)
+++ OK, passed 100 tests.
*Stack> quickCheck $ qcFoldl1 (-)
*** Failed! Falsified (after 4 tests and 6 shrinks):
0
[1]

-}

-- Fold R
-- test with + - *
qcFoldr :: Eq b => (a -> b -> b) -> b -> [a] -> Bool
qcFoldr f z ls =
  l == c
  where
    s = genStack ls
    c = foldr f z ls
    l = stackFoldr f z s

{-

*Stack> quickCheck $ qcFoldr (+)
+++ OK, passed 100 tests.
*Stack> quickCheck $ qcFoldr (-)
+++ OK, passed 100 tests.
*Stack> quickCheck $ qcFoldr (*)
+++ OK, passed 100 tests.

-}
```

```
111 --commutative operations +, *
112 qcFoldr1 :: Eq b => (a -> b -> b) -> b -> [a] -> Bool
113 qcFoldr1 f z ls = l' ==
114   l && c == l
115   where
116     s' = genStack $ reverse ls
117     s = genStack ls
118     c = foldr f z ls
119     l = stackFoldr f z s
120     l' = stackFoldr f z s'
121
122 {-
123
124 Ok, one module loaded.
125 *Stack> quickCheck $ qcFoldr1 (+)
126 +++ OK, passed 100 tests.
127 *Stack> quickCheck $ qcFoldr1 (*)
128 +++ OK, passed 100 tests.
129 *Stack> quickCheck $ qcFoldr1 (-)
130 *** Failed! Falsified (after 4 tests and 6 shrinks):
131 0
132 [0,1]
133
134 -}
135
136 -- Zip
137 qcZip :: [a] -> [b] -> Bool
138 qcZip ps qs =
139   s == spq && s == sqp
140     where
141       p   = genStack ps
142       q   = genStack qs
143       pq  = stackZip p q
144       spq = stackLen pq 0
145       qp  = stackZip q p
146       sqp = stackLen qp 0
147       s   = minimum [length ps, length qs]
148
149 {-
150
151 *Stack> quickCheck $ qcZip
152 +++ OK, passed 100 tests.
153
154 -}
155
156
157
158
159
160
161
162
163
164
165
```

```haskell
qcZipUnzip :: (Eq a1, Eq a2) => [a1] -> [a2] -> Bool
qcZipUnzip ls ms =
  gsl == l' && m' == gsm && m'' == gsm && l'' == gsl
  where
    sx =  if length ls <= length ms
          then length ls
          else length ms
    sl = take sx ls
    sm = take sx ms
    gsl = genStack sl
    gsm = genStack sm
    l = genStack ls
    m = genStack ms
    lm = stackZip l m
    ml = stackZip m l
    (m'', l'') = stackUnzip ml
    (l', m') = stackUnzip lm
    stackUnzip Empty = (Empty, Empty)
    stackUnzip (Stack s (u,v)) =  let (u',v') = stackUnzip s
                                  in (Stack u' u, Stack v' v)

-- > quickCheck $ qcZipUnzip
-- +++ OK, passed 100 tests.


-- Map
qcMap :: Eq b => (a -> b) -> [a] -> Bool
qcMap f ls =
  genStack (map f ls) == stackMap f (genStack ls)

qcMapDiv :: Int -> [Int] -> Property
qcMapDiv d ls =
  d /= 0 ==> qcMap (`div` d) ls

{-
> quickCheck $ \n -> qcMap (n +)
+++ OK, passed 100 tests.
> quickCheck $ \n -> qcMap (n -)
+++ OK, passed 100 tests.
> quickCheck $ \n -> qcMap (n *)
+++ OK, passed 100 tests.
*Stack> quickCheck $ qcMapDiv
+++ OK, passed 100 tests; 13 discarded.
-}
```

```haskell
-- length of stack remains same after map, zip
stackLen :: Stack a -> Int -> Int
stackLen Empty z       = z
stackLen (Stack s a) z = stackLen s (z+1)

-- inverse and length
qcMap1 :: Eq a => (a -> b) -> (b -> a) -> [a] -> Bool
qcMap1 f f' ls =
  length ls == m' && m' == lm && mx == s
  where
    s  = genStack ls
    m  = stackMap f s
    mx = stackMap f' m
    lm = stackLen s 0
    m' = stackLen m 0

{- apply f then apply inverse of f i.e. f'

*Stack> quickCheck $ \n -> qcMap1 (n +) ((-n) +)
+++ OK, passed 100 tests.

-}

-- more ideas
-- maping with (+ 0) or (* 1) results in the same stack (identity)

-- PROPERTY TEST ----------------------------------------E
```