```haskell
module NavesEspaciales (Componente(Contenedor, Motor, Escudo, Cañón), NaveEspacial(Módulo, Base), Dirección(Babor, Estribor), TipoPeligro(Pequeño, Grande, Torpedo), Peligro, foldNave, capacidad, poderDeAtaque, puedeVolar, mismoPotencial, mayorCapacidad, transformar, impactar, maniobrar, pruebaDeFuego, componentesPorNivel, dimensiones) where

data Componente = Contenedor | Motor | Escudo | Cañón deriving (Eq, Show)

data NaveEspacial = Módulo Componente NaveEspacial NaveEspacial | Base Componente deriving Eq

data Dirección = Babor | Estribor deriving Eq

data TipoPeligro = Pequeño | Grande | Torpedo deriving Eq

type Peligro = (Dirección, Int, TipoPeligro)

instance Show NaveEspacial where
  show = ("\n" ++) . (padNave 0 0 False)

padNave nivel acum doPad (Base c) = (if doPad then pad (4*nivel + acum) else "") ++ show c
padNave nivel acum doPad (Módulo x i d) = (if doPad then pad (4*nivel + acum) else "") ++ show x ++
                      pad 4 ++ padNave (nivel+1) (acum+l) False i ++ "\n" ++
                      padNave (nivel+1) (acum+l) True d where l = length $ show x

pad :: Int -> String
pad i = replicate i ' '

--Ejercicio 1
foldNave :: (Componente -> b) -> (Componente -> b -> b -> b) -> NaveEspacial -> b
foldNave fBase fModulo (Base comp) = fBase comp
foldNave fBase fModulo (Módulo comp subNave1 subNave2) = fModulo comp (foldNave fBase fModulo subNave1) (foldNave fBase fModulo subNave2)

--Ejercicio 2

recorrerNaveDevolviendo :: Componente -> NaveEspacial -> Int
recorrerNaveDevolviendo cmpnt = foldNave (\comp -> if comp == cmpnt then 1 else 0) fModulo
                where fModulo = (\comp recursionIzq recursionDer -> if (comp == cmpnt) then 1 + recursionIzq + recursionDer  else recursionIzq + recursionDer)

capacidad :: NaveEspacial -> Int
capacidad = recorrerNaveDevolviendo Contenedor

poderDeAtaque :: NaveEspacial -> Int
poderDeAtaque = recorrerNaveDevolviendo Cañón

poderDeDefensa :: NaveEspacial -> Int
poderDeDefensa = recorrerNaveDevolviendo Escudo

aceleracion :: NaveEspacial -> Int
aceleracion = recorrerNaveDevolviendo Motor

puedeVolar :: NaveEspacial -> Bool
puedeVolar = (0 <).(recorrerNaveDevolviendo Motor)

mismoPotencial :: NaveEspacial -> NaveEspacial -> Bool
mismoPotencial nave1 nave2 = foldr (\comp recu -> recu && mismaCantidad comp ) True [Contenedor, Motor, Escudo, Cañón]
                             where mismaCantidad comp = recorrerNaveDevolviendo comp nave1 == recorrerNaveDevolviendo comp nave2

-- Ejercicio 3
mayorCapacidad :: [NaveEspacial] -> NaveEspacial
mayorCapacidad = foldr1 (\nave1 recu -> if capacidad nave1 > capacidad recu then nave1 else recu)
```

```haskell

-- Ejercicio 4
transformar :: (Componente -> Componente) -> NaveEspacial -> NaveEspacial
transformar = mapNave

mapNave :: (Componente -> Componente) -> NaveEspacial -> NaveEspacial
mapNave f = foldNave (\comp ->  Base (f comp)) fModulo
                where  fModulo = (\comp subNaveNuevaIzq subNaveNuevaDer->  Módulo (f
                comp) subNaveNuevaIzq subNaveNuevaDer)


-- Ejercicio 5
-- El esquema foldNave no es adecuado para esta funcion ya que 'impactar' recorre la
nave parcialmente
-- y sólo aplica la funcion 'realizarImpacto' a la sub-nave/nivel que corresponde.
-- En cambio, si utilizaramos foldNave la funcion 'realizarImpacto' se aplicaria
recursivamente a todas
-- las sub-naves perdiendo el concepto de Direccion y Nivel que especifica el ejercicio.

dameSubNave :: Dirección -> NaveEspacial -> NaveEspacial
dameSubNave dir (Módulo c subNave1 subNave2)
                                        | dir == Babor = subNave1
                                        | dir == Estribor = subNave2

dameRaiz :: NaveEspacial -> Componente
dameRaiz nave = case nave of
                            (Módulo c subNave1 subNave2) -> c
                            Base c -> c

realizarImpacto :: TipoPeligro -> NaveEspacial -> NaveEspacial
realizarImpacto tipo nave = case tipo of
                                        Pequeño -> if dameRaiz nave == Escudo then nave
                                        else Base Contenedor
                                        Grande -> Base Contenedor
                                        -- Otra posibilidad para el caso 'tipo ==
                                        Grande' es si la nave tiene poderDeAtaque > 0
                                        entonces
                                        -- es equivalente a 'realizarImpacto Pequeño
                                        nave' evitando hacer este chequeo en la funcion
                                        'impactar',
                                        -- quedando mas prolijo, pero implicaria usar
                                        recursion explicita.
                                        Torpedo -> Base Contenedor

impactar :: Peligro -> NaveEspacial -> NaveEspacial
impactar (dir, nivel, tipo) (Base comp)
        | nivel == 0 = if (tipo == Grande) && (poderDeAtaque (Base comp) > 0) then
        realizarImpacto Pequeño (Base comp)
                                                                else realizarImpacto tipo
                                                                (Base comp)
        | otherwise = Base comp

impactar (dir, nivel, tipo) (Módulo comp subNaveIzq subNaveDer)
        | nivel == 0 = if tipo == Grande && (poderDeAtaque (Módulo comp subNaveIzq
        subNaveDer)) > 0 then realizarImpacto Pequeño (Módulo comp subNaveIzq subNaveDer)
                                                                else realizarImpacto tipo
                                                                (Módulo comp subNaveIzq
                                                                subNaveDer)
        | otherwise = case dir of
                                Babor -> Módulo comp (recu subNaveIzq) subNaveDer
                                Estribor -> Módulo comp subNaveIzq (recu subNaveDer)
            where recu subNave = impactar (dir, nivel-1, tipo) subNave

-- Ejercicio 6
maniobrar :: NaveEspacial -> [Peligro] -> NaveEspacial
maniobrar nave ps = (foldr f id ps) nave
                        where f = \p recu -> \naveImpactada -> recu (impactar p
                        naveImpactada)

```

```haskell
-- Ejercicio 7
pruebaDeFuego :: [Peligro] -> [NaveEspacial] -> [NaveEspacial]
pruebaDeFuego ps = filter (\nave -> puedeVolar (maniobrar nave ps))

-- Ejercicio 8
componentesPorNivel :: NaveEspacial -> Int -> Int
componentesPorNivel nave n = (foldNave (\_ -> \n -> if n == 0 then 1 else 0) fModulo
nave) n
                                  where fModulo = \_ recIzq recDer -> \n -> if n == 0
                                  then 1 else (recIzq (n-1)) + (recDer (n-1))

ancho :: NaveEspacial -> Int
ancho nave = foldr (\n recu -> max (componentesPorNivel nave n) recu) 0 [0..altura nave]

altura :: NaveEspacial -> Int
altura = foldNave (const 1) (\_ recIzq recDer -> 1 + max recIzq recDer)

dimensiones :: NaveEspacial -> (Int, Int)
dimensiones nave = (altura nave, ancho nave)

```