

```

-----
--
-- (c) 2000 Alexander Vivian Hugh McPhail. All Rights Reserved
--
-----
{-----}
module Set (
  Set,
  --
  emptySet,    -- Set a
  singletonSet, -- Eq a => a -> Set a
  --
  listToSet,   -- Eq a => [a] -> Set a
  setToList,   -- Eq a => Set a -> [a]
  --
  isEmptySet,  -- Eq a => Set a -> Bool
  isElement,   -- Eq a => a -> Set a -> Bool
  isSubSet,    -- Eq a => Set a -> Set a -> Bool
  isEqual,     -- Eq a => Set a -> Set a -> Bool
  --
  unionSet,    -- Eq a => Set a -> Set a -> Set a
  diff,        -- Eq a => Set a -> Set a -> Set a
  inter,       -- Eq a => Set a -> Set a -> Set a
  --
  insertSet,   -- Eq a => a -> Set a -> Set a
  removeSet,   -- Eq a => a -> Set a -> Set a
  --
  filterSet,   -- Eq a => Set a -> (a -> Bool) -> Set a
  --
  cardSet,     -- Set a -> Integer
  --
  first        -- Set a -> a
) where
{-----}
import List(insert)
{-----}
type Size = Integer

data Set a = Set Size [a]
{-----}
instance Show a => Show (Set a) where
  show (Set 0 _) = show "{}"
  show (Set _ l) = "{" ++ doList l ++ "}" where
    doList []      = ""
    doList [x]    = show x
    doList (x:xs) = show x ++ "," ++ (doList xs)

instance Eq a => Eq (Set a) where
  s1 == s2 = isEqual s1 s2
{-----}
emptySet :: Set a
emptySet = (Set 0 [])

singletonSet :: Eq a => a -> Set a
singletonSet v = (Set 1 [v])
{-----}
listToSet :: Eq a => [a] -> Set a
listToSet []      = emptySet
listToSet [x]    = singletonSet x
listToSet (x:xs) = check x (listToSet xs) where
  check x s@(Set n l)
    | isElement x s = s
    | otherwise     = (Set (n+1) (x:l))

```

```

setToList :: Eq a => Set a -> [a]
setToList (Set _ t) = t
{-----}
isEmptySet :: Set a -> Bool
isEmptySet (Set 0 _) = True
isEmptySet _ = False
isElement :: Eq a => a -> Set a -> Bool
isElement e (Set 0 _) = False
isElement e (Set _ t) = elem e t

isSubSet :: Eq a => Set a -> Set a -> Bool
isSubSet (Set 0 _) _ = True
isSubSet _ (Set 0 _) = False
isSubSet (Set r1 l1) (Set r2 l2)
  | r1 > r2 = False
  | otherwise = check l1 l2 where
    check [] = True
    check (x:xs) l = (elem x l) && (check xs l)

isEqual :: Eq a => Set a -> Set a -> Bool
isEqual s1@(Set n1 l1) s2@(Set n2 l2)
  | n1 == n2 = l1 == l2
  | otherwise = False
{-----}
unionSet :: Eq a => Set a -> Set a -> Set a
unionSet s1@(Set 0 _) s2 = s2
unionSet s1 s2@(Set 0 _) = s1
unionSet s1@(Set n1 l1) s2@(Set n2 l2)
  | n1 < n2 = doInsert l1 s2
  | otherwise = doInsert l2 s1 where
    doInsert [] s = s
    doInsert (x:xs) s@(Set n t)
      | elem x t = doInsert xs s
      | otherwise = doInsert xs (Set (n+1) (x:t))

diff :: Eq a => Set a -> Set a -> Set a
diff s1@(Set 0 _) s2 = s1
diff s1 s2@(Set 0 _) = s1
diff s1 s2 = filterSet (\x -> not (isElement x s2)) s1

inter :: Eq a => Set a -> Set a -> Set a
inter s1@(Set 0 _) s2 = s1
inter s1 s2@(Set 0 _) = s2
inter s1@(Set n1 _) s2@(Set n2 _)
  | n1 < n2 = filterSet (\x -> isElement x s2) s1
  | otherwise = filterSet (\x -> isElement x s1) s2
{-----}
insertSet :: Eq a => a -> Set a -> Set a
insertSet e s
  | isElement e s = s
  | otherwise = unionSet (singletonSet e) s
removeSet :: Eq a => a -> Set a -> Set a
removeSet e s
  | isElement e s = diff s (singletonSet e)
  | otherwise = s
{-----}
filterSet :: Eq a => (a -> Bool) -> Set a -> Set a
filterSet _ s1@(Set 0 _) = s1
filterSet f s1@(Set _ l) = listToSet (filter f l)
{-----}
cardSet :: Set a -> Integer
cardSet (Set n _) = n
{-----}
first :: Set a -> a

```

```
first (Set 0 _) = error "Set:(first ): empty set"
first (Set _ (x:xs)) = x
{-----}
```