CIS 552
Advanced Programming

Fall 2021

Today: Monday, Oct 4th

- HW #3 available, due Thursday, Oct 7th
- Today
  - Short XML discussion (TAs)
  - QuickCheck Q&A time

```
mapXMLTree ::
  (String -> Int -> [SimpleXML] -> [SimpleXML]) ->
  SimpleXML ->
  [SimpleXML]
mapXMLTree f = mapXMLTree' f 0

mapXMLTree' ::
  (String -> Int -> [SimpleXML] -> [SimpleXML]) ->
  Int ->
  SimpleXML ->
  [SimpleXML]
mapXMLTree' f depth x = case x of
  PCDATA val -> [PCDATA val]
  Element key children -> f key depth next
where
   next = foldr (\x ys -> mapXMLTree' f (depth + 1) x ++ ys) [] children

line :: [SimpleXML] -> [SimpleXML]
line next = next ++ [Element "br" []]

boldLine :: [SimpleXML] -> [SimpleXML]
boldLine next = [Element "b" next, Element "br" []]

headerDepth :: Int -> [SimpleXML] -> [SimpleXML]
headerDepth depth next = [Element ("h" ++ show depth) next]

section :: String -> [SimpleXML] -> [SimpleXML]
section key next = [Element key next]

headedSection :: String -> Int -> [SimpleXML] -> [SimpleXML]
headedSection header depth next =
  Element ("h" ++ show depth) [PCDATA header] : next

pass :: [SimpleXML] -> [SimpleXML]
pass = id

playMapper :: String -> Int -> [SimpleXML] -> [SimpleXML]
playMapper key depth next = case key of
  "PLAY" -> section "body" next
  "TITLE" -> headerDepth depth next
  "PERSONAE" -> headedSection "Dramatis Personae" 2 next
  "ACT" -> pass next
  "SCENE" -> pass next
  "SPEECH" -> pass next
  "PERSONA" -> line next
  "SPEAKER" -> boldLine next
  "LINE" -> line next
  _ -> pass next

formatPlay :: SimpleXML -> SimpleXML
formatPlay tree = Element "html" (mapXMLTree playMapper tree)
```

```
type DList a = [a] -> [a]

-- | cons appends a single element to the front of the DList
cons :: a -> DList a -> DList a
cons = (.) . (:)

-- | a new line element represented by html tag <br/>
newLine :: DList SimpleXML
newLine = (Element "br" [] :)

-- | foldXML recursively applies the supplied function to the input list
-- and produces a DList of SimpleXMLs
foldXML :: (SimpleXML -> DList SimpleXML) -> [SimpleXML] -> DList SimpleXML
foldXML f = foldr (\x xs -> f x . xs) id

-- | transformTitle takes an element and a tag,
-- and outputs an HTML element of with the provided tag and original content
transformTag :: SimpleXML -> ElementName -> SimpleXML
transformTag (Element _ str) tag = Element tag str
transformTag str tag = Element tag [str]

-- | writeLine takes a SimpleXML element with a single string element
-- and returns a Dlist with two elements - an element with the string
-- followed by a line break
writeLine :: SimpleXML -> DList SimpleXML
writeLine (Element _ [PCDATA line]) = cons (PCDATA line) newLine
writeLine _ = id

-- | transformTitleList takes a SimpleXML element with a list
-- of elements where the first element is the title, and the remainder
-- of the list represents the contents, and ell elements are written as
-- separate lines
transformTitleLines :: SimpleXML -> DList SimpleXML
transformTitleLines (Element _ (title : lines)) =
  (transformTag title "b" :) . newLine . foldXML writeLine lines
transformTitleLines _ = id

-- | transformTitledXML takes a SimpleXML element with a list
-- of elements where the first element is the title, and the remainder
-- of the list are elements that need to be recursively transformed further
-- using the function provided as a parameter
transformTitledXML :: String -> (SimpleXML -> DList SimpleXML) -> SimpleXML ->
DList SimpleXML
transformTitledXML tag f (Element _ (title : xs)) = cons (transformTag title tag)
(foldXML f xs)
transformTitledXML _ _ _ = id

-- | transformTitledXML takes a SimpleXML element with a list
-- of elements where the first element is the title, and the remainder
-- of the list are elements that need to be recursively transformed further
-- using the function provided as a parameter
transformTitledXML :: String -> (SimpleXML -> DList SimpleXML) -> SimpleXML ->
DList SimpleXML
transformTitledXML tag f (Element _ (title : xs)) = cons (transformTag title tag)
(foldXML f xs)
transformTitledXML _ _ _ = id

-- | base function for the transforming the play from XML to valid HTML format
-- the function adds the html and body tags, and calls helper methods
-- to transform the list of persons and the acts in the play
formatPlay :: SimpleXML -> SimpleXML
formatPlay (Element "PLAY" (title : personae : acts)) =
  Element "html" [Element "body" body]
  where
body = cons (transformTag title "h1") (transformPersonae personae . foldXML
transformAct acts) []
transformAct = transformTitledXML "h2" transformScene
transformScene = transformTitledXML "h3" transformTitleLines
formatPlay _ = PCDATA ""
```

Arbitrary Typeclass

```
class Arbitrary a where
    arbitrary :: Gen a    -- generator for arbitrary elements of a
    shrink :: a -> [a]    -- we'll talk about this on Wednesday

type Gen a                          -- generator type
sample' :: Gen a -> IO [a]          -- generate some random values
instance Functor Gen                -- fmap :: (a -> b) -> Gen a -> Gen b
instance Monad Gen                  -- return :: a -> Gen a
                                    -- (>>=) :: Gen a -> (a -> Gen b) -> Gen b
```

Testable Typeclass

quickCheck :: Testable prop => prop -> IO ()

Testable Bool

(Arbitrary a, Show a, Testable prop) => **Testable** (a -> prop)

```
-- to quickCheck (f :: a -> Bool), generate 100 random "a"s
-- check if each "f a" returns true or false
-- If ever false, print a as counterexample
```

Property type

instance Testable Property

(==>) :: Testable prop => Bool -> prop -> Property

not a || b

a ==> b