Eddie Gurnee
E01020445
COSC 341
4/09/2015

COSC 341
Project 6
Haskell: Learn You Far More Haskell for You and Your Friends for Greatest Good

```haskell
module Haskell_Mini_Project
( count
, substring
, get_last
, zip_pairs
, permute
, ackermann
) where

import           Assist_Lib

-- 1 --
count item = counter 0
  where
    counter curr [] = curr
    counter curr (x:xs)
      | item == x = counter (curr + 1) xs
      | not (elem item xs) = curr
      | otherwise = counter curr xs

count' item = length . filter (== item)

count'' item [] = 0
count'' item (x:xs)
  | item == x = 1 + count'' item xs
  | otherwise = count'' item xs

-- 2 --
substring xs ys = subHelp 0 xs ys

subHelp curIndex (x:xs) (y:ys)
  | x == y = checkMore xs ys
  | not (elem x ys) = -1
  | otherwise = subHelp (curIndex + 1) (x:xs) ys
  where
    checkMore [] _ = curIndex
    checkMore (ix:ixs) (iy:iys)
      | ix == iy = checkMore ixs iys
      | not (elem ix iys) = -1
      | otherwise = subHelp (curIndex + 1) (x:xs) ys

-- 3 --
get_last [] = error "empty list"
get_last [x] = x
get_last (_:x:xs) = get_last (x:xs)

-- 4 --
zip_pairs [] _ = []
zip_pairs _ [] = []
zip_pairs (x:xs) (y:ys) = mergeTuples x y : zip_pairs xs ys
```

```
  where
    mergeTuples (x1, x2) (y1, y2) = (x1, x2, y1, y2)


-- 5 --
permute [] = [[]]
permute xs = [x:ys | x <- xs, ys <- permute (trimmedList x xs)]
  where trimmedList target (t:trimmed) =
                  if target == t
                    then trimmed
                    else t : trimmedList target trimmed

-- 6 --
ackermann m n
  | m == 0 = n + 1
  | m > 0 && n == 0 = ackermann (m - 1) 1
  | m > 0 && n > 0 = ackermann (m - 1) (ackermann m (n - 1))
  | otherwise = error "must have positive values"

ackermann' m n p
  | p == 0 = m + n
  | n == 0 && p == 1 = 0
  | n == 0 && p == 2 = 1
  | n == 0 = m
  | otherwise = ackermann' m (ackermann' m (n - 1) p) (p - 1)
```

```haskell
module Assist_Lib
( tail'
, head'
, init'
, last'
, store
, pack
, isSquare
, isVowel
, isConsonant
, someNumbers
, someLetters
, trimListOfKey
, scatter
, gather
) where

import           System.Random

tail' :: [a] -> [a]
tail' [] = error "empty list"
tail' (_:theList) = theList

head' :: [a] -> a
head' [] = error "empty list"
head' (theHead:_) = theHead

init' :: [a] -> [a]
init' [] = error "empty list"
init' [x] = []
init' (x:y:xs) = x : init' (y:xs)

last' :: [a] -> a
last' [] = error "empty list"
last' [x] = x
last' (_:x:xs) = last' (x:xs)

deleteMap = randoms (mkStdGen 1) :: [Bool]
someLetters = randomRs ('a', 'z' ) (mkStdGen 1) :: [Char]
someNumbers = randomRs (-100, 100) (mkStdGen 1) :: [Int]

store :: [a] -> [(a, Bool)]
store [] = []
store theData = zip theData deleteMap

pack :: [(a, Bool)] -> [(a, Bool)]
pack dataBlocks = [ validData
      | validData <- dataBlocks,
        not $ snd validData]
```

```haskell
isSquare :: (Integral a) => a -> Bool
isSquare num = num == (^2) (floor $ sqrt $ fromIntegral num)

isVowel :: Char -> Bool
isVowel c = elem c ['a', 'e', 'i', 'o', 'u']

isConsonant :: Char -> Bool
isConsonant c = not $ isVowel c

trimListOfKey :: (Eq a) => a -> [(a, b)] -> [(a, b)]
trimListOfKey key = filter (\x -> fst x /= key)

scatter :: (Integral a) => a -> [b] -> [(a, [b])]
scatter numberOfCores sharableData =
      combine possibleCores $ distributedData numberOfCores sharableData
  where
    possibleCores = [0..numberOfCores-1]
    distributedData numberOfCores = zip (cycle possibleCores)

combine [] _ = []
combine (coreNum:remainingCores) [] = (coreNum, []) : combine remainingCores []
combine (coreNum:remainingCores) all@(x:xs) =
      grabAllData coreNum all :
            combine remainingCores (removeExtraData coreNum xs)

grabAllData key theData = (key, [snd tuple | tuple <- theData, fst tuple == key])
removeExtraData key = filter (\x -> fst x /= key)

gather :: (Num a) => [(a, [b])] -> (a, [b])
gather allDistributedData = (numCores, recombineData allDistributedData)
  where
    numCores = fst (last allDistributedData) + 1
    recombineData [] = []
    recombineData (first:allTheRest) = snd first ++ recombineData allTheRest
```

```haskell
module Sample_Tests
( testAll
) where

import           Haskell_Mini_Project

testAll = test1 && test2 && test3
          && test4 && test5 && test6

test1 = (count 2 [2,3,2,1,2,5]) == 3
test2 = (substring [2,3,4] [0,1,2,3,4,3,4]) == 2
test3 = (substring [1,7] [0,1,2,3,1,7,6]) == 4
test4 = (get_last [1,2,3,5]) == 5
test5 = (zip_pairs [(1,2),(3,4),(5,6)] [(100,101),(200,201)]) ==  [(1,2,100,101),
(3,4,200,201)]
test6 = (permute [1,2,3]) == [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```