Haskell Programming Assignment: Various Computions

Abstract:

This assignment provided us with a basic understanding of Haskell. Haskell is a very powerful functional programming language. This assignment helped us understand how list processing works in Haskell by performing recursive list processing and using higher-order functions.

Task 1 - Mindfully Mimicking the Demo

```
ghci> :set prompt ">>> "
>>> length [2,3,5,7]
4
>>> words "need more coffee"
["need","more","coffee"]
>>> unwords ["need", "more", "coffee"]
"need more coffee"
>>> reverse "need more coffee"
"eeffoc erom deen"
>>> reverse ["need", "more", "coffee"]
["coffee","more","need"]
>>> head ["need", "more", "coffee"]
"need"
>>> tail ["need", "more", "coffee"]
["more","coffee"]
>>> last ["need", "more", "coffee"]
"coffee"
>>> init ["need", "more", "coffee"]
["need","more"]
>>> take 7 "need more coffee"
"need mo"
>>> drop 7 "need more coffee"
"re coffee"
>>> ( \x -> length x > 5 ) "Friday"
True
```

```
>>> ( \x -> length x > 5 ) "uhoh"
False
>>> ( \x -> x /= ' ' ) 'Q'
True
>>> ( \x -> x /= ' ' ) ' '
False
>>> filter ( \x -> x /= ' ' ) "Is the Haskell fun yet?"
"IstheHaskellfunyet?"
```

Task 2 - Numeric Function Definitions

Function definition:

```
squareArea a = a * a
circleArea a = pi * a * a
blueAreaOfCube a = (6 * squareArea a) - (6 * circleArea x)
where x = 0.25 * a
paintedCube1 1 = 0
paintedCube1 a = 6 * squareArea p
where p = a - 2
paintedCube2 1 = 0
paintedCube2 a = 12 * (a - 2)
```

DEMO:

```
ghci> :set prompt ">>> "
>>> :load ha
[1 of 1] Compiling Main ( ha.hs, interpreted )
Ok, one module loaded.
>>> squareArea 10
100
>>> squareArea 12
144
>>> circleArea 10
314.1592653589793
>>> circleArea 12
452.3893421169302
>>> blueAreaOfCube 10
482.19027549038276
```

```
>>> blueAreaOfCube 12
694.3539967061512
>>> blueAreaOfCube 1
4.821902754903828
>>> map blueAreaOfCube [1..3]
[4.821902754903828,19.287611019615312,43.39712479413445]
>>> paintedCube1 1
0
>>> paintedCube1 2
0
>>> paintedCube1 3
6
>>> map paintedCube1 [1..10]
[0,0,6,24,54,96,150,216,294,384]
>>> paintedCube2 1
0
>>> paintedCube2 2
0
>>> paintedCube2 3
12
>>> map paintedCube2 [1..10]
[0,0,12,24,36,48,60,72,84,96]
>>> :quit
```

Task 3 - Puzzlers

Function Definition:

```
reverseWords charString = unwords ( reverse ( words charString ) )
averageWordLength charString = fromIntegral ( length ( filter ( \x ->
x /= ' ') charString ) ) /
fromIntegral( length ( words charString ) )
```

DEMO:

```
ghci> :set prompt ">>> "
>>> :load ha
[1 of 1] Compiling Main ( ha.hs, interpreted )
Ok, one module loaded.
>>> reverseWords "appa and baby yoda are the best"
"best the are yoda baby and appa"
>>> reverseWords "want me some coffee"
"coffee some me want"
>>> averageWordLength "appa and baby yoda are the best"
3.5714285714285716
>>> averageWordLength "want me some coffee"
4.0
```

Task 4 - Recursive List Processors

Function Definition:

```
list2set [ ] = [ ]
list2set (x:xs) =
if((x `elem` xs) == False)
then x : list2set xs
else list2set xs

isPalindrome [ ] = True
isPalindrome (x:[ ]) = True
isPalindrome (x:xs) =
if(x == last xs)
then isPalindrome ( init xs )
else False
collatz :: Int -> [Int]
collatz 1 = [1]
collatz a =
if(a `mod` 2 == 0)
then a : collatz ( a `div` 2 )
else a : collatz ( ( 3 * a ) + 1 )
```

DEMO:

```
ghci> :set prompt ">>> "
>>> :load ha
[1 of 1] Compiling Main ( ha.hs, interpreted )
Ok, one module loaded.
>>> list2set [1,2,3,2,3,4,3,4,5]
[1,2,3,4,5]
>>> list2set "need more coffee"
"ndmr cofe"
>>> isPalindrome ["coffee", "latte", "coffee"]
True
>>> isPalindrome ["coffee", "latte", "espresso", "coffee"]
False

>>> isPalindrome [1,2,5,7,11,13,11,7,5,3,2]
False
>>> isPalindrome [2,3,5,7,11,13,11,7,5,3,2]
True
>>> collatz 10
[10,5,16,8,4,2,1]
>>> collatz 11
[11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
>>> collatz 100
[100,50,25,76,38,19,58,29,88,44,22,11,34,17,52,26,13,40,20,10,5,16,8,4
,2,1]
```

Task 5 - List Comprehensions

Function definition:

```
count e list = length [ x | x <- list, x == e ]
freqTable list = [(x, count x list) | x <- list2set list]
```

```
DEMO:
ghci> :set prompt ">>> "
>>> :load ha
[1 of 1] Compiling Main ( ha.hs, interpreted )
Ok, one module loaded.
>>> count 'e' "need more coffee"
5
>>> count 4 [1,2,3,2,3,4,3,4,5,4,5,6]
3
>>> freqTable "need more coffee"
[('n',1),('d',1),('m',1),('r',1),('
',2),('c',1),('o',2),('f',2),('e',5)]
>>> freqTable [1,2,3,2,3,4,3,4,5,4,5,6]
[(1,1),(2,2),(3,3),(4,3),(5,2),(6,1)]
>>> :quit
```

Task 6 - Higher Order Functions

Function definition:

```
tgl n = foldl (+) 0 [1..n]
triangleSequence n = map (tgl) [1..n]
vowelCount letters = length ( filter ( \x -> x == 'a' || x == 'e' || x
== 'i' || x == 'o' || x == 'u' ) letters )
lcsim fm p list = map fm ( filter ( \x -> p x ) list )
```

DEMO:

```
ghci> :set prompt ">>> "
>>> :load ha
[1 of 1] Compiling Main ( ha.hs, interpreted )
Ok, one module loaded.
>>> tgl 5
15
>>> tgl 10
55
```

```
>>> triangleSequence 10
[1,3,6,10,15,21,28,36,45,55]
>>> triangleSequence 20
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210]
>>> vowelCount "cat"
1
>>> vowelCount "mouse"
3
>>> lcsim tgl odd [1..15]
[1,6,15,28,45,66,91,120]
>>> animals = ["elephant", "lion", "tiger", "orangatan", "jaguar"]
>>> lcsim length (\w -> elem ( head w ) "aeiou") animals
[8,9]
```

Task 7 - An Interesting Statistic: nPVI

Task 7a-Demo

```
ghci> :set prompt ">>> "
>>> :load npvi
[1 of 1] Compiling Main ( npvi.hs, interpreted )
Ok, one module loaded.
>>> a
[2,5,1,3]
>>> b
[1,3,6,2,5]
>>> c
[4,4,2,1,1,2,2,4,4,8]

>>> u
[2,2,2,2,2,2,2,2,2,2]
>>> x
[1,9,2,8,3,7,2,8,1,9]
```

Task 7b-Demo

```
>>> pairwiseValues a
[(2,5),(5,1),(1,3)]
>>> pairwiseValues b
[(1,3),(3,6),(6,2),(2,5)]
>>> pairwiseValues c
[(4,4),(4,2),(2,1),(1,1),(1,2),(2,2),(2,4),(4,4),(4,8)]
>>> pairwiseValues u
[(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2)]
>>> pairwiseValues x
[(1,9),(9,2),(2,8),(8,3),(3,7),(7,2),(2,8),(8,1),(1,9)]
```

Task 7c-Demo

```
>>> pairwiseDifferences a
[-3,4,-2]
>>> pairwiseDifferences b
[-2,-3,4,-3]
>>> pairwiseDifferences c
[0,2,1,0,-1,0,-2,0,-4]
>>> pairwiseDifferences u
[0,0,0,0,0,0,0,0,0]
>>> pairwiseDifferences x
[-8,7,-6,5,-4,5,-6,7,-8]
```

Task 7d-Demo

```
>>> pairwiseSums a
[7,6,4]
>>> pairwiseSums b
[4,9,8,7]
>>> pairwiseSums c
[8,6,3,2,3,4,6,8,12]
>>> pairwiseSums u
[4,4,4,4,4,4,4,4,4]
>>> pairwiseSums x
[10,11,10,11,10,9,10,9,10]
```

Task 7e-Demo

```
>>> pairwiseHalves [1..10]
[0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0]
>>> pairwiseHalves u
[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
>>> pairwiseHalves x
[0.5,4.5,1.0,4.0,1.5,3.5,1.0,4.0,0.5,4.5]
```

Task 7f-Demo

```
>>> pairwiseHalfSums a
[3.5,3.0,2.0]
>>> pairwiseHalfSums b
[2.0,4.5,4.0,3.5]
>>> pairwiseHalfSums c
[4.0,3.0,1.5,1.0,1.5,2.0,3.0,4.0,6.0]
>>> pairwiseHalfSums u
[2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0]
>>> pairwiseHalfSums x
[5.0,5.5,5.0,5.5,5.0,4.5,5.0,4.5,5.0]
```

Task 7g-Demo

```
>>> pairwiseTermPairs a
[(-3,3.5),(4,3.0),(-2,2.0)]
>>> pairwiseTermPairs b
[(-2,2.0),(-3,4.5),(4,4.0),(-3,3.5)]
>>> pairwiseTermPairs c
[(0,4.0),(2,3.0),(1,1.5),(0,1.0),(-1,1.5),(0,2.0),(-2,3.0),(0,4.0),(-4
,6.0)]
>>> pairwiseTermPairs u
[(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.
0)]
>>> pairwiseTermPairs x
[(-8,5.0),(7,5.5),(-6,5.0),(5,5.5),(-4,5.0),(5,4.5),(-6,5.0),(7,4.5),(
-8,5.0)]
```

Task 7h-Demo

```
>>> pairwiseTerms a
[0.8571428571428571,1.3333333333333333,1.0]
>>> pairwiseTerms b
[1.0,0.6666666666666666,1.0,0.8571428571428571]
>>> pairwiseTerms c
[0.0,0.6666666666666666,0.6666666666666666,0.0,0.6666666666666666,0.0,
0.666666666666
6666,0.0,0.6666666666666666]
>>> pairwiseTerms u
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
>>> pairwiseTerms x
[1.6,1.2727272727272727,1.2,0.9090909090909091,0.8,1.1111111111111112,
1.2,1.5555555555
555556,1.6]
```

Task 7i-Demo

```
>>> nPVI a
106.34920634920636
>>> nPVI b
88.09523809523809
>>> nPVI c
37.03703703703703
>>> nPVI u
0.0
>>> nPVI x
124.98316498316497
```

Task 8 - Historic Code: The Dit Dah Code

Subtask 8a Demo

```
ghci> :set prompt ">>> "
>>> :load ditdah.hs
[1 of 1] Compiling Main ( ditdah.hs, interpreted )
Ok, one module loaded.
>>> dit
"-"
>>> dah
"---"
>>> dit +++ dah
"- ---"
>>> m
('m',"--- ---")
>>> g
('g',"--- --- -")
>>> h
('h',"- - - -")
>>> symbols
[('a',"- ---"),('b',"--- - - -"),('c',"--- - --- -"),('d',"--- -
-"),('e',"-"),('f',"- - --- -"),('g',"--- --- -"),('h',"- -
- -"),('i',"- -"),('j',"- --- --- ---"),('k',"--- - ---"),('l',"- ---
- -"),('m',"--- ---"),('n',"--- -"),('o',"--- ---
---"),('p',"- --- --- -"),('q',"--- --- - ---"),('r',"- ---
-"),('s',"- - -"),('t',"---"),('u',"- - ---"),('v',"- - -
---"),('w',"- --- ---"),('x',"--- - - ---"),('y',"--- - ---
---"),('z',"--- --- - -")]
```

Subtask 8b Demo

```
ghci> :set prompt ">>> "
>>> :load ditdah
[1 of 1] Compiling Main ( ditdah.hs, interpreted )
Ok, one module loaded.
>>> assoc 'c' symbols
('c',"--- - --- -")
```

```
>>> assoc 'z' symbols
('z',"--- --- - -")
>>> find 't'
"___"
>>> find 'a'
"- ___"
>>>
```

Subtask 8c Demo

```
ghci> :set prompt ">>> "
>>> :load ditdah
[1 of 1] Compiling Main ( ditdah.hs, interpreted )
Ok, one module loaded.
>>> addletter "a" "b"
"a    b"
>>> addword "happy" "holiday"
"happy     holiday"
>>> droplast3 "kritika"
"krit"
>>> droplast7 "abcdefghijkl"
"abcde"
```

Subtask 8d Demo

```
ghci> :set prompt ">>> "
>>> :load ditdah
[1 of 1] Compiling Main ( ditdah.hs, interpreted )
Ok, one module loaded.
>>> encodeletter 'm'
"___ ___"
>>> encodeletter 'w'
"- ___ ___"
>>> encodeletter 's'
"- - -"
>>> encodeword "yay"
"___ - ___ ___ - ___ ___ - ___ ___"
>>> encodeword "good"
"___ ___ - ___ ___ ___ ___ ___ ___ ___ - -"
>>> encodeword "haskell"
```

```
"- - - - - --- - - - --- - --- - - --- - - - --- - -"
>>> encodemessage "need more coffee"
"--- - - - --- - - --- --- --- --- --- - --- - - --- - --- - --- ---
--- - - --- - - - --- - - -"
>>> encodemessage "computer science is the best"
"--- - --- - --- --- --- --- --- - --- --- - - - --- --- - - --- - - -
- --- - --- - - - - --- - --- -
--- - - - - - - - --- - - - - - --- - - - - - - - - ---"
>>> encodemessage "good happy"
"--- --- - --- --- --- --- --- --- --- - - --- - - - --- --- - - - - -
---"
```