Patrick Charles prc9219
CMPS 450
Assignment 1
Fall 15

Functional Programming

I.1 Description of Assignment Goals

To familiarize students with the functional programming concepts and teach students how to change their mentality from imperative programming to functional.

II.1 Language and platform

II.2 Scheme functions

Written in notepad++ v6.5.5 on Windows 10 and run on MIT-GNU Scheme v9.2

II.3 Haskell functions

Written in notepad++ v6.5.5 on Windows 10 and run on GHCi v7.10.2

III.1 Description of Functions

Overview: Functions to be written in the functional languages Scheme and Haskell designed to familiarize students with functional programming concepts and syntax.

III.1.1 Tail Recursive list

Description: Function receives a list and accumulator value initialized to zero. If the list is empty, the zero value of the accumulator is returned. Otherwise, the functions recursively loops through the list incrementing the accumulator until the end of the list is reached
an arbitrary list.

III.1.2 Maximum/ Minimum of a list of integers

Description:
minimum value function: receives a list of integers and tests if the head is less than the min value so far. If so, the function calls itself with the tail of the list and the new min value and goes through the list again.
Maximum value function: receives a list of integers and tests if the head is greater than the max value so far. If so, the

function calls itself with the tail of the list and the new min value and goes through the list again.
Minmax function: is called with 2 parameters which are function calls to the minimum and maximum value functions and returns the values.

III.1.3 Collect user input until zero is reached

Description: reads user input into a list until a zero is reached in which case, it will stop reading user input and output the list of values entered. The list is created by taking each user input and using the 'cons' function to add it to the next value.

III.1.4 Collect user inputted List and Output Min/Max of List

Description: reads user input into a list until a zero is reached in which case, it will stop reading user input and output the list of values entered. The list is created by taking each user input and using the 'cons' function to add it to the next value. After a list is created, the minimum and maximum value functions are applied to the list and the minimum value and maximum value of the list are outputted

III.1.5 Mergesort

Description: receives a list which is split into two lists using split functions. The lists are then sorted recursively based on iterating through the values to determine the least value of the lists and ordered as such. After the lists are sorted, the two lists are merged using, merge function and sorted again.

III.1.6 Nth Increment

Description: takes in 2 variables as parameters and simply adds the second variable to the first and returns the value

IV.1 Source code for Scheme functions

IV.1.1 Tail recursive list

filename: tailRecursiveList.scm

```
;This program will write a tail-recursive function to
```

```
;compute the length of an arbitrary list.

(define counter (lambda (L result)
; function receives a list and accumulator
(if (null? L) result
;if list is empty, return the accumulator
(counter(cdr L) (+ 1 result)
;calls itself and goes through list incrementing
;accumulator each time
))))

(define count( lambda (L) (counter L 0)))
;function loops through list adding to accumulator until null
(0)
;and returns the result
```

IV.1.2 Maximum/ Minimum of a list of integers

filename: maxMinList.scm

```
;compute the maximum and minimum of a list of integers

(define findmin (lambda (L min1)
;takes a list and a result

    (cond ((null? L) min1)
     ;if list empty return 1st element/head

          ((< (car L) min1) (findmin (cdr L)(car L)))
          ;test if head of list is < min value so far
          ;if so, calls itself with the tail of the list and
          ;the new min value and goes through entire list

          (else(findmin (cdr L) min1)))))
          ;if headNOT < min value so far, calls itself with
          ;the tail of list and initial min1 value

(define min1 (lambda (L) (findmin L (car L))))
;takes a list and returns min value using findmin function

(define findmax (lambda (L max1)
;takes a list and a result

    (cond ((null? L) max1)
     ;if list empty return 1st element/head
        ((> (car L) max1) (findmax (cdr L)(car L)))
          ;test if head of list is > max value so far
          ;if so, calls itself with the tail of the list and
```

```
          ;the new min value and goes through entire list

          (else(findmax (cdr L) max1)))))
          ;if headNOT > max value so far, calls itself with
          ;the tail of list and initial min1 value

(define max1 (lambda (L) (findmin L (car L))))
;takes a list and returns max value using findmax function

(define minmax(lambda (L) (list (min1 L) (max1 L))))
;returns a list of minimum value and maximum value using min/max
functions
```

IV.1.3 Collect user input until zero is reached

filename: inputUntilZero.scm

```
;a function that collects integers from the user until a 0 is
encountered and
;returns them in a list in the order they were input

(define ( zero ) (lambda (tempList)
     (let ((input (read)))
     ;read input into list

       (cond (( = input 0 ) (reverse tempList))
       ;if zero, stop reading, return inputted values and
reverse

         (else (zero ( cons tempList )))))))
         ;calls function again for input and adds inputted value
to
         ;list with 'cons' function
```

IV.1.4 Collect user inputted List and Output Min/Max of List

filename: inputZeroMinMax.scm

```
;input a 0-ended list of integers;
;print the list in the order entered and print the maximum
;and minimum of the list


(define getmaxmin(lambda() (minmax ( zero ))))

;uses minmax function which finds returns the max/min of a list
;and zero which inputs a list of user date until a zero is
reached
```

```
;minmax takes zero and a empty list as parameters.
;zero returns a list of user input until zero is input
;and then the minmax function outputs a list of the max and min
of
;the user input
```

V.1.5 Mergesort

filename: mergesort.scm

IV.1.6 Nth increment

filename: nthincrement.scm

```
;function takes in an integer and returns an nth increment
function
;which increments parameters by n

(define nthinc (lambda (n)(lambda(x) (+ n x))))

;example    ((nthinc 3 ) 2 ) = 5
;where 3 is lambda (n) and 2 is lambda (x)
;it just take the 2 and adds it to the n and returns the value

;it evaluates the parenthesis to see if there is another
parameter
; and if there is, it is added to the original one given to the
;function
```

IV.2 Source code for Haskell functions

IV.2.1 Tail recursive list

filename: tailRecursiveList.hs

```
-- This program will write a tail-recursive function to
-- compute the length of an arbitrary list.

len :: [a] -> Int -> Int
len [] acc = acc
len (x : xs) acc = len xs $! (1 + acc)
```

IV.2.2 Maximum/ Minimum of a list of integers

filename: maxmin.hs

```
max1 :: Ord a => [a] -> a
max1 [] = undefined
max1 [x] = x
max1 (x1 : x2 : xs)
  | x1 > x2 = max1 (x1 : xs)
  | otherwise = max1 (x2 : xs)

min1 :: Ord a => [a] -> a
min1 [] = undefined
min1 [x] = x
min1 (x1 : x2 : xs)
  | x1 < x2 = min1 (x1 : xs)
  | otherwise = min1 (x2 : xs)


--minmax [x] = min1[x] max1[x]
```

IV.2.3 Collect user input until zero is reached

filename: inputUntilZero.hs

not attempted

IV.2.4 Collect user inputted List and Output Min/Max of List

filename: inputZeroMinMax.hs

```
max1 :: Ord a => [a] -> a
max1 [] = undefined
max1 [x] = x
max1 (x1 : x2 : xs)
  | x1 > x2 = max1 (x1 : xs)
  | otherwise = max1 (x2 : xs)

min1 :: Ord a => [a] -> a
min1 [] = undefined
min1 [x] = x
min1 (x1 : x2 : xs)
  | x1 < x2 = min1 (x1 : xs)
  | otherwise = min1 (x2 : xs)


--minmax [x] = min1[x] max1[x]
```

IV.2.5 Mergesort

filename: mergesort.hs

```
merge :: (Ord a) => [a] -> [a] -> [a]
merge [] xs = xs
```

```
merge xs [] = xs
merge (x:xs) (y:ys)
    | (x <= y) =  x : merge xs ( y : ys)
    | otherwise = y : merge (x :xs) ys

split :: [a] -> ([a], [a])
split [] = ([], [])
split [x] = ([x], [])
split ( x : y : xys) = (x : xs, y : ys)
    where (xs, ys) = split xys

mergesort :: (Ord a) => [a] -> [a]
mergesort xs
    | (length xs) > 1 = merge (mergesort ls) (mergesort rs)
    | otherwise = xs
    where (ls, rs) = split xs
```

IV.2.6 Nth increment

filename: nthincrement.hs

```
--function takes in an integer and returns an nth increment
function
--which increments parameters by n
--it evaluates the parenthesis to see if there is another
parameter
-- and if there is, it is added to the original one given to the
--function

nthinc :: Int -> Int -> Int
nthinc x y =  x + y
```

V.1 Test Case for Scheme function

V.1.1 Tail recursive list

1. Load file
2. Define a list
3. Test ‘counter’ function with list as parameter and accumulator as second parameter initialized to 0
4. Test ‘count’ function with list as parameter

```
1 ]=> (load "tailrecursivelist.scm")

;Loading "tailrecursivelist.scm"... done
;Value: count

1 ]=> (define list1 ( list 1 2 3 4 5 6 7 8 9 10))

;Value: list1

1 ]=> (counter list1 0)

;Value: 10

1 ]=> (count list1)

;Value: 10
```

V.1.2 Maximum/ Minimum of a list of integers

1. Load file
2. Define a list
3. Use 'min' function with list as parameter
4. Use 'max' function with list as parameter
5. Use 'minmax' function with list as parameter

```
1 ]=> (load "maxMinList.scm")

;Loading "maxminlist.scm"... done
;Value: minmax

1 ]=> (define list1(list -34 3 2 66 -43 888))

;Value: list1

1 ]=> (min list1)

;Value: -43

1 ]=> (max list1)

;Value: -43

1 ]=> (minmax list1)

;Value 11: (-43 -43)
```

V.1.3 V.2.3 Collect user input until zero is reached

1. Load file
2. Run 'zero' function
3. Input list of integers and eventually enter zero

```
1 ]=> (load "inputuntilzero.scm")

;Loading "inputuntilzero.scm"... done
;Value: zero

1 ]=> (zero)
3
5
7
89
345
9754
0

;Value 11: (3 5 7 89 345 9754)
```

V.1.4 Collect user inputted List and Output Min/Max of List

V.1.5 Mergesort

1. Load 'mergesort.scm'
2. Define a list to mergesort
3. Run mergesort on list

```
1 ]=> (load "mergesort.scm")

;Loading "mergesort.scm"... done
;Value: mergesort

1 ]=> (define list1 (list -54 0 45 23 765 98 65 -9 -45 -420 2345))

;Value: list1

1 ]=> (mergesort list1)

;Value 12: (-420 -54 -45 -9 0 23 45 65 98 765 2345)
```

V.1.6 Nth increment

1. Load file

2. Use nthinc function with 2 integers as paramters

```
1 ]=> ((nthinc -3 ) 2)

;Value: -1

1 ]=> ((nthinc 1234 ) -1234)

;Value: 0

1 ]=> ((nthinc 10 ) 25)

;Value: 35

1 ]=>
```

V.2 Test Case for Haskell Functions

VI.2.1 Tail recursive list

1. Call function with a list as 1st parameter and accumulator set to 0 as second parameter

```
Ok, modules loaded: Main.
*Main> :t len
len :: [a] -> Int -> Int
*Main> len [1..10] 0
10
*Main> len [[1..10],[11..20],[21..30]] 0
3
*Main>
```

VI.2.2 Maximum/ Minimum of a list of integers

1. Test min1 function with list of integers
2. Test max1 function with list of integers
3. Test minmax function with list of integers

```
Ok, modules loaded: Main.
*Main> min1 [-4,3,456,-98,56]
-98
*Main> max1 [-2,-7,-6,6]
6
*Main> min1 [9,8,6,5,4,3,2,-16,-17]
-17
*Main> max1 [12,24,36,48,60,72,84,96]
96
*Main>
```

VI.2.3 Collect user input until zero is reached

VI.2.4 Collect user inputted List and Output Min/Max of List

VI.2.5 Mergesort

filename:

1. Create list
2. Split the list using function 'list'
3. Mergesort list using function 'mergesort'

```
*Main> let list = [11,76,34,6,2,777,87,345,3,9,14]
*Main> list
[11,76,34,6,2,777,87,345,3,9,14]
*Main> split list
([11,34,2,87,3,14],[76,6,777,345,9])
*Main> mergesort list
[2,3,6,9,11,14,34,76,87,345,777]
*Main>
```

VI.2.6 Nth increment

filename:

1. Call function with 1st parameter as number to incremented and 2nd number to add to the first

```
*Main> :t nthinc
nthinc :: Int -> Int -> Int
*Main> nthinc (-10) 10
0
*Main> nthinc 2 3
5
*Main> nthinc 3 (-45)
-42
*Main>
```