Computer Science & Engineering Department
I. I. T. Kharagpur

Principles of Programming Languages: CS40032

Elective

Assignment – 3: Haskell, Scheme and Lisp *Marks: 30*

Solutions

Haskell

1. Func and Func1 are functions which takes multiple arguments. **[1]**

   ```
   ghci> Func1 a b c (Func 2 5)
   ```

   Explain the execution order of the following function. Hint: Curried Function

   Begin Solution
   ((((Func1 a) b) c) ((Func 2) 5))
   End Solution

2. What is the position of (2,2) in **[1]**

   ```
   [ (i,k-i) | k <- [0..],i <- [0..k] ]?
   ```

   Remember that list positions start with 0
   Begin Solution
   Answer: 12
   It is clear that this list is infinite. However, instead of one list, it could be viewed as a concatenation of an infinite number of lists, one for each value of k. To make that precise, we could split

   ```
   [(i,k-i) | k <- [0..], i <- [0..k]]
   as concat [[(i, k-i) | i <- [0 .. k]] | k <- [0..]].
   ```

   That is to say, we could view them as the concat of [(0,0)], [(0,1),(1,0)], [(0,2),(1,1),(2,0)], [(0,3),(1,2),(2,1),(3,0)].So, for each value of k,we have k+1 elements in each of these sublists.
   In particular, we are looking for (i, k-i) == (2, 2): solving for k, this means k==4. So, for the sublists before this particular k, we had 1 + 2 + 3 + 4 elements in total. In this particular sublist, the position of (2,2) is the third element.Hence, the actual position of (2, 2) in the given list is 13. Or if we started counting from 0, 12
   End Solution

   .

3. What is the output of the following **[4]**

   a)
   ```
   ghci> [x*y | x <- [1..5], y <- take 2 (cycle [1,2]) ]
   ```

   b)
   ```
   ghci> let lst = ["cat", "dog", "ant", "pen"]
   ghci> map reverse lst
   ```

   c)
   ```
   ghci> let xxs = [[1,3,5,2,3,1,2,4,5],
                    [4..8],
                    [1,2,4,2,1,6,3,1,3,2,3,6]]
   ghci> [ [ x | x <- xs, odd x ] | xs <- [1,7,4,8,2]:xxs]
   ```

   d)
   ```
   ghci> [3,2,1] > [2,10,200]
   ```

Begin Solution

```
a) [1,2,2,4,3,6,4,8,5,10]
b) ["tac","god","tna","nep"]
c) [[1,7],[1,3,5,3,1,5],[5,7],[1,1,3,1,3,3]]
d) True
```

End Solution

4. Write a function to insert an element in a list at desired index in Haskell using recursion.(Assume the desired index to be within the length of the list.) **[2]**

```
ghci> insertElement 'k' ['a', 'b', 'c', 'd'] 2 => "abkcd"
```

Begin Solution

```
insertElement :: a -> [a] -> Int -> [a]
insertElement x ys     0 = x:ys
insertElement x (y:ys) n = y:insertElement x ys (n-1)
```

End Solution

5. Write a function extractNonUppercase which extracts the uppercase letters from a string. extractNonUppercase "TYuiJ" would produce "TYJ". Mention the type of the function. **[2]**

Begin Solution

```
extractNonUppercase :: [Char] -> [Char]
extractUppercase st = [ c | c <- st, c `elem` ['A'..'Z']]
```

End Solution

Scheme

6. Find the output of the following statements. Explain the intermediate steps. **[4]**

```
a)  (car (cdr '(a b c d e f)))

b)  (let ((x 9))
        (* x
            (let ((x (/ x 3)))
                (+ x x))))
c) (quote (quote cons))
d) (map  (lambda (ls) (cons 'a ls))  '((b c) (a) ()))
```

Begin Solution

```
a) b
b) 54
c) quote cons
d) ((a b c) (a a) (a))
```

End Solution

7. Write a Scheme function to compute the sum of the numbers in a list. Return 0, if the list is empty. **[2]**

Begin Solution

```
(define (sum ls)
  (if (null? ls) 0
      (+ (car ls) (sum (cdr ls)))))
```

End Solution

8. Write Scheme functions using lambda expressions and conditionals to implement: **[4]**

```
A) Counting the number of elements in a list.
   For example (Counter '(a b c b a)) returns 5
B) Logical "AND" operator
```

Begin Solution

```
a)
(define mycounter (lambda (L)
(cond
((null? L) 0)
(else (add 1 (mycounter (cdr L)))) )))
b) (define  and  (lambda (s1  s2)(cond  (s1  s2)  (else#f) )))
```

End Solution

Lisp

9. Write a LISP program using lambda expression for a function that takes a list and an integer as parameter and returns the list with the elements multiplied by that integer. **[5]**

Begin Solution

```
(defun lamdaMult(n list)
    (mapcar #'(lambda (x) (* x n)) list))
Eg: (write (lamdaMult '3 '(5 7 1 8))) => (15 21 3 24)
```

End Solution

10. Define a power function (Power x y implies x^y) using recursion constructs of LISP **[5]**

Begin Solution

```
(defun powerxy ( x y )
(cond ( ( zerop y ) 1)
( t (* x ( powerxy x (- y 1 ) ) ) )
) )
```

End Solution