Homework: DefineLang and FuncLang (Part 1)

1. (3 pt) Write a DefineLang program to do the following: define the ASCII values for the letters 'a' 'b' and 'c'; write DefineLang expressions to compute the ASCII value for the corresponding upper case letter.

Sol

```
(define a 97)
(define b 98)
(define c 99)
(define A (a − 32))
(define B (b − 32))
(define C (c − 32))
```

2. (3 pt) Pythagoras' theorem is a relation between three sides of a right angle triangle. It states that in a right angle triangle (a triangle whose one side is 90 degrees), the square of the hypotenuse (the side opposite the right angle) is equal to the sum of the squares of the other two sides. This can be represented as:

(H*H)=(B*B)+(L*L)

where,
H : length of hypotenuse
B : length of base
L : height of triangle

Define a lambda function isRightAngled such that isRightAngled accepts three parameters height, base, hypotenuse in order and returns true if three sides satisfy Pythagoras' theorem else returns false.

$ (isRightAngled 4 3 5)
#t
$ (isRightAngled 3 4 5)
#t
$ (isRightAngled 5 4 3)
#f

Sol

```
(define isRightAngled
  (lambda (b l h)
    (if (= (+ (* b b) (* l l)) (* h h))
      #t
      #f)))
```

or

```
(define isRightAngled
  (lambda (b l h)
    (= (+ (* b b) (* l l))(* h h))))
```

3. (6 pt) FuncLang programming: recursive functions.

   (a) (3 pt) Compute the factorial of a given number n (n is the input variable).

   (b) (3 pt) Compute the n^{th} number in the fibonacci sequence (n is the input variable).

   Sol (3pt each)

   (a) summarize

```
(define factorial
  (lambda (n)
      (if (= n 1)
        1
        (* (factorial (- n 1)) n))))
```

   (b) fibonacci

```
(define fibonacci
  (lambda (n)
    (if (= n 1)
      1
      (if (= n 2)
        1
        (+ (fibonacci (- n 1)) (fibonacci (- n 2)))))))
```

4. (8 pt) FuncLang programming: list and pair.

   (a) (4 pt) Define a function named *max* that calculates the maximum value of a list,
   $(max (list))
   0
   $ (max (list 1 10 3 14))
   14
   $ (max (list 11 18 31 14))
   31

   (b) (4 pt) Define a function even that returns all the even numbers for a given list. Some example usage is show below,
   $ (even (list))
   ()
   $ even (list 1 2 3 4 5)
   (2 4)
   $ (even (list 5 7))
   ()

   Sol

   (a) (4 pt)

```
(define maxhelper
        (lambda (lst max)
                (if (null? lst)
                        max
                        (if (> (car lst) max)
                                (maxhelper (cdr lst) (car lst))
                                (maxhelper (cdr lst) max )))))
(define max
        (lambda (lst)
        (if (null? lst)
                0
                (maxhelper lst (car lst) ))))
```

(b) (4 pt)

```
(define mod2
        (lambda (n)
                (if (= n 1)
                        1
                        (if (= n 0)
                                0
                                (if (< n 0)
                                        (mod2 (- 0 n))
                                        (mod2 (- n 2)))))))
(define even
        (lambda (l)
        (if (null? l)
                (list)
                (if (= (mod2 (car l)) 0)
                        (cons (car l) (even (cdr l)))
                        (even (cdr l))))))
```

5. (5 pt) FuncLang programming: list and pair.

Using list-related expressions list, car, cdr, caddr, null?, cons provided in FuncLang, write programs that achieve the following purpose and provide one transcript in the space for answer below:

(a) (2 pt) Using list expression define a list named *pairs* that contains a list of 4 pairs: (5,1) (4,6) (8,7) (10,15).

(b) (3 pt) Define a function firstSum that performs an addition on the first element of each pair on the list *pairs*. The function has to extract each element using list-related expressions. Result = 5 + 4 + 8 + 10 = 27.

Sol (4 pt) (a)

```
(define pairs
        (list
                (cons 5 1)
                (cons 4 6)
                (cons 8 7)
                (cons 10 15)))
```

(4 pt) (b)

```
(define firstSum
        (lambda ()
                (+ (car (car pairs)) (car (car (cdr pairs))) (car (car (cdr (cdr pairs)))) (car (car (cdr (cdr (cdr
                        pairs))))))))
```

(4 pt) (c)

```
(define extract7 (lambda () (cdr (car (cdr (cdr pairs))))))
```

6. (8 pt) FuncLang programming: high order functions.

   (a) (4 pt) Rajan's book Exercise 5.4.1 in Ch 5.

   (b) (4 pt) Rajan's book Exercise 5.4.2 in Ch 5.

Sol

(a) (5pt)

```
(define filter
  (lambda (op lst)
    (if (null? lst)
      (list)
      (if (op (car lst))
        (cons (car lst) (filter op (cdr lst)))
        (filter op (cdr lst))))))
```

(b) (5pt)

```
(define foldl
  (lambda (op zero lst)
    (if (null? lst)
      zero
      (foldl op (op (car lst) zero) (cdr lst)))))
```

7. (8 pt) FuncLang programming: high order functions and curried functions.

   (a) (2 pt) Construct a global variable mylist that holds a list of three pairs, (1,3) (4,2) (5,6).

   (b) (4 pt) Write a function apply-on-nth that takes three arguments op, lst, n, where op is a function, lst is a list of pairs, n is an integer. The return value should be the result of applying op on the n-th pair in the list. If n is out of range of the list, return -1. You can assume op is a function valid to accept two arguments.

   Some examples of using apply-on-nth with above mylist variable:
   $ (applyonnth (lambda (a b) (+ a b) mylist 1)
   4 // 1+3
   $ (applyonnth (lambda (a b) (- a b) mylist 2)
   4 // 4-2
   $ (applyonnth (lambda (a b) (- a b) mylist 8)
   - 1 // third parameter out of range
   $ (applyonnth (lambda (a b) (- a b) mylist -1)
   - 1 // third parameter out of range

   (c) (2 pt) Convert the above FuncLang program into the curried form

Sol

(a)

```
(define mylist (list (cons 1 3)
                     (cons 4 2)
                     (cons 5 6)))
```

```
(define applyonnth
        (lambda (op lst n)
                (if (null? lst) -1
                        (if (= n 1)
                                (op (car (car lst))
                                        (cdr (car lst)))
                                (if (< n 0) -1
                                        (applyonnth op (cdr lst) (- n 1)))))))
```

(b)

```
(define applyonnth
        (lambda (op)
                (lambda (lst)
                        (lambda (n)
                                (if (null? lst) -1
                                        (if (= n 1)
                                                (op (car (car lst))
                                                        (cdr (car lst)))
                                                (if (< n 0) -1
                                                        (((applyonnth op) (cdr lst)) (- n 1)))))))))
```