Homework Solutions: DefineLang and FuncLang

Learning Objectives:

1. Write programs in DefineLang, FuncLang
2. Get familiar with the concepts of recursive functions, high order functions, currying

Instructions:

- Total points: 41 pt.
- Early deadline: Mar 3 (Wed) at 11:59 PM; Regular deadline: Mar 5 (Fri) at 11:59 PM (you can continue working on the homework till TA starts to grade the homework).
- Write Definelang and Funclang programs for the following questions and submit them in one pdf file.
- You can reuse any functions you write in this homework to answer questions.
- Use the Funclang interpreter provided in hw4code.zip to test the correctness of your programs. Follow the steps in the homework 2 tutorial to setup the interpreter. You can also use Racket to test your funclang programs.
- How to submit:
  - Submit your programs to Canvas under Assignments, Homework 4.
  - Please provide the entire solution in one pdf file.

Questions:

1. (3 pt) [DefineLang programming] Define a constant `faraday` with the usual value of 96454.56. Define a constant `n` with a value of 5. Using the definition of `faraday` and `n`, convert 5 Faradays to Coulombs. Recall that the formula for Faraday to Coulomb conversion is faraday * n.

   Sol:

```
(define faraday 96454.56)
(define n 5)
(define Faraday_to_Coulumb (* faraday n))
```

2. (3 pt) [FuncLang programming] The Greatest common divisor (GCD) of two numbers a and b is defined as follows:

   if a > b then (gcd a b) is gcd of a - b and b
   else if a < b then (gcd a b) is gcd of a and b - a
   otherwise, it is a.

Write a FuncLang program `gcd` that computes the greatest common divisor according the above definition. Example scripts:

```
$ (gcd 4 2)
2
$ (gcd 12 15)
3
```

Sol:

```
(define gcd
        (lambda (a b)
                (if (= a b)
                        a
                        (if (> a b)
                                (gcd (- a b) b)
                                (gcd a (- b a))
                        )
                )
        )
)
```

3. (12 pt) [FuncLang with list programming] FuncLang programming: list.

   (a) (4 pt) Write a function `len` that gives the length of a given list L.

   ```
   $ (len (list))
   0
   $ (len (list 1 10 3 14))
   4
   ```

   Sol:

   ```
   (define len
           (lambda (lst)
                           (if (null? lst)
                                   0
                                       ( + 1 (len (cdr lst)))
                   )
               )
   )
   ```

   (b) (4 pt) Write a function `unique` that removes duplicate elements in a list and returns a list of unique elements.

   ```
   $ (unique (list))
   ()
   $ (unique (list 1 10 3 14))
   (1 10 3 14)
   $ (unique (list 11 18 31 18))
   (11 18 31)
   ```

Sol:

```
(define member
        (lambda (item lst)
                (if (null? lst)
                         #f
                         (if (= item (car lst))
                                  #t
                                  (member item (cdr lst))
                         )
                )
        )
)

(define unique
        (lambda (lst)
                (if (null? lst)
                         (list)
                         (if (member (car lst) (cdr lst))
                                   (unique (cdr lst))
                                   (cons (car lst) (unique (cdr lst)))
                          )
                )
        )
)
```

(c) (4 pt) Write a function `pairup` that constructs an association list from a list of keys and a list of values.

$ (pairup (list 10 9 8) (list 1 2 3))
((10 1) (9 2) (8 3))
$ (pairup (list 10 9) (list 1 2 3))
()
$ (pairup (list 10 9 9) (list 1 2))
((10 1) (9 2))
$ (pairup (list 10 9 8) (list 1 2))
()

Sol:

```
(define pairup
        (lambda (lst1 lst2)
                 (if (null? lst1)
                           (list)
                           (if(null? lst2)
                                    (list)
                                    (let ((x (unique lst1)) (y (unique lst2)))
                                    (if (= (len x) (len y))
                                             (cons (cons (car x) (car y))
                                             (pairup (cdr x) (cdr y)))
                                             (list)
                                     ))
                            )
                  )
         )
)
```

4. (5 pt) [FuncLang with list and pair programming] FuncLang programming: list and pair.

   (a) (2 pt) Using list expression define a list named `authorandtitle` that contains a list of 3 pairs: ("C. S. Lewis", "The Last Battle") ("Charles Dickens", "A Christmas Carol") ("Arthur C. Clarke", "Rama").
   Sol:

```
(define authorandtitle (list (cons "C.S.␣Lewis" "The␣Last␣Battle")
(cons "Charles␣Dickens" "A␣Christmas␣Carol")
(cons "Arthur␣C.␣Clarke" "Rama")))
```

   (b) (3 pt) Write a function, `getbooks` that takes `authorandtitle` string pairs and returns a single list of only the books.
   $ (getbooks authorandtitle)
   ("The Last Battle" "A Christmas Carol" "Rama")

   Sol:

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

5. (8 pt) [High order function programming] Given the following definitions of *pair* and *apair*

   (define pair (lambda (fst snd) (lambda (op) (if op fst snd))))
   (define apair (pair 2 3))

   (a) (2 pt) Explain what is apair? **Sol**:
   apair is a function which returns the desired element of the (pair 2 3). It returns the first element of the (pair 2 3) i.e 2, when given #t and the second when given #f as an argument.

   (b) (2 pt) Modify *pair* to support arithmetic between two elements of *apair*

   Sol:

```
(define pair
        (lambda (fst snd)
                (lambda (op)
                        (if (= op 0) (+ fst snd)
                                (if (= op 1) (- fst snd)
                                        (if (= op 2) (* fst snd)
                                                (if (= op 3) (/ fst snd)
                                                    (if (= op 4) (if (= fst snd) #t #f)
                                                     0)
                                                )
                                        )
                                )
                        )
                )
        )
)
```

(c) (4 pt) Write a FuncLang program to determine if the two elements of *apair* are equivalent.

Sol:

```
(define eq
        (lambda (p) (p 4)
          )
)
```

6. (8pt) [High order function and currying] FuncLang programming: high order functions and curried functions.

(a) (2 pt) Construct two global variables list1 and list2: list1 holds three pairs, (1,3) (4,2) (5,6); and list2 holds three pairs (2,6), (4,2) (1,3)

Sol:

```
(define list1 (list (cons 1 3) (cons 4 2) (cons 5 6)))
(define list2 (list (cons 2 6) (cons 4 2) (cons 1 3)))
```

(b) (4 pt) Write a function processlists that takes three arguments op, list1, list2 where op is a function that takes two pairs as parameters, and list1 and list2 are the two lists of pairs. The return value should be the result of applying op on each pair of list1 and list2.

Some examples of using processlists with above list1 and list2 global variables:

```
$ (processlists add list1 list2)
((3,9) (8,4) (6,9))
$ (processlists subtract list1 list2)
((-1,-3) (0,0) (4,3))
$ (processlists multiply list1 list2)
((2,18) (16,4) (5,18))
$ (processlists common list1 list2)
(()(4,2)())
$ ((processlists diff list1 list2)
((1,3)()(5,6))
$ (processlists diff list2 list1)
((2,6)()(1,3))
```

Sol:

```
(define add
        (lambda (pair1 pair2)
                (if (null? pair1)
                        pair2
                        (if (null? pair2)
                                pair1
                                (cons (+ (car pair1) (car pair2)) (+ (cdr pair1) (cdr pair2)))
                        )
                 )
         )
)

(define subtract
        (lambda (pair1 pair2)
                (if (null? pair2)
                       pair1
                       (if (null? pair1)
                                (cons (- 0 (car pair2)) (- 0 (cdr pair2)))
                                (cons (- (car pair1) (car pair2)) (- (cdr pair1) (cdr pair2)))
                        )
                 )
         )
)
(define multiply
        (lambda (pair1 pair2)
                (if (null? pair1)
                       pair2
                       (if (null? pair2)
                                pair1
                                (cons (* (car pair1) (car pair2)) (* (cdr pair1) (cdr pair2)))
                        )
                 )
         )
)

(define common
        (lambda (pair1 pair2)
                (if (= (car pair1) (car pair2))
                        (if (= (cdr pair1) (cdr pair2))
                             pair1
                            (list))
                        (list))))

(define diff
        (lambda (pair1 pair2)
                (if (= (car pair1) (car pair2))
                        (if (= (cdr pair1) (cdr pair2))
                             (list)
                            pair1)
                        pair1)))




(define processlists (lambda (op lst1 lst2)
                  (if (null? lst1)
                      lst2
                      (if (null? lst2)
                          lst1
                          (cons (op (car lst1) (car lst2))(processlists op (cdr lst1) (cdr lst2)))))))
```

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

Sol:

```
(define processlists (lambda (op)
                 (lambda (lst1)
                     (lambda (lst2)
                          (if (null? lst1)
                               lst2
                                                           (if (null? lst2)
                                                   lst1
                                                 (cons (op (car lst1) (car lst2))(((processlists op) (cdr
                                                    lst1)) (cdr lst2)))))))))
```

The function call will look like this : (((processlists add) list1) list2)