Programming Principles

Homework No. 4 [25 points]

Due: November 26, 2018 10:30AM

Write your report in English. Submit your racket file for the problems that are specified in the problem description. When "run" button is pushed, your interaction window must show your testing results. If you need to write your solution on the paper as specified, write on the paper. Submit your Racket file (in a single file) to TA (email: 204606131@qq.com), so that TA can test your program. Your file name must be "PP_hw4*_your_student_id_number.rkt*". Submit your answer sheet (if any) at the class. Don't forget documenting your programs. Make the first comment of your program be your student id and name, so that TA can find who you are.

1. [5 points] Write a procedure **`diagonal`** that takes a list of sentences(lists) as its argument. It should return a sentence(list) containing the first word of the first sentence, the second word of the second sentence, and so on. (Assume the sentences are long enough; don't add error checks.)

```
> (diagonal '((she loves you)(tell me why)(I want to hold your hand)))
  (she me to)

; diagonal : list of lists -> list
; Given the list of sentences(list), pick the first word from the
; first sentence, the second word from the second sentence and so on.
; test:
;(diagonal '((I want) (You have a book) (He needs medicine to drink)))
;        -> (I have medicine)

; Your program must start with the following lines
(define (diagonal sentences)
     (if (null? sentences)

         ; You must fill in this part

      )
 )
```

2. [5 points] We will write a procedure **`every-nth`** that accepts two arguments: **`num`** is an integer and **`list-of-sents`** is a list of a certain number of lists. **`every-nth`** selects the **`num`**-th elements from

each list in **list-of-sents** and returns the list of selected elements. Assume that all lists in **list-of-sents** have more than **num** elements. Consider the first element as the 0-th. Examples:

(every-nth 2 '((a b c d) (e f g h))) gives **(c g).**

(every-nth 1 '((a b c) (d e f) (g h i))) gives **(b e h).**

```
(define (every-nth num list-of-sents)
        ; You must fill in this part
)
```

3. [5 points] Write a function **(transform n fun x)** that takes an integer **n**, a function **fun** mapping one argument ***y*** to **fun(*y*)**, and an argument **x** for the function **fun**. **(transform n fun x)** returns a list **(x, fun(x), fun²(x), …, fun⁽ⁿ⁻¹⁾(x))**. For example, **(transform 3 square 2)** returns **(2 4 16)** since the list of squares of the previous element is **(2 2² 4²)**.
   The implementation uses **helper** as a helper function. Complete the following **helper** procedure.

```
(define (transform n fun x)
   (define (helper temp count)
      (if
        ; You must fill in this part
       )
    )
    (helper x 0 ))
```

4. [10 points] A set is represented as a list of elements, some of which may be repeated. For example, **(define aset '(2 1 2 3 4 2))** creates a set, **aset**, with six elements having four unique values, since there are three repetitions of **2**. You can identify each space in the skeleton functions below, and must complete them with the function name in the list below. Fill in the space with the given function name (only one function per blank) that fits in. You may use the same function more than once. If no given function fits, write out your own function name in the space below so that the program works. Complete the program and run your test.

1. car	**2. cdr**	**3. filter-iter**	**4. map**
5. equal?	**6. cadr**	**7. cons**	**8. trans2pairs**
9. null?	**10. append**	**11. count-occurrence**	**12. accumulate**
13. +	**14. list**	**15. (lambda (x) x)**	**16. filter-list**

A. First, you write a procedure, **count-occurrence,** that takes an element, **elm**, and a list, **aset**, and returns the number of elements of **aset** that are equal to **elm**. For example,
(count-occurrence 3 '(3 5 2 3 3 6 5 4 3)) returns 4.

```
(define (count-occurrence elm aset)
   (if ( _ _   aset)
       0
       ( ___  ( _ _   elm (cdr aset))
              (if  ( _ _   elm ( _ _ aset)) 1 0))))
```

B. Second, you write a procedure, **filter-list**, which takes an item, **elm**, and a list, **aset**, and returns a new list constructed of all the elements that are in **aset** but are not equal to **elm**, in the same order as they occur in **aset**. You can write it with a higher-ordered function, **filter**, but you are going to implement it in the linear-iterative version to demonstrate your knowledge on topics of CS220. For example,
(filter-list 3 '(3 5 2 3 3 6 5 4 3)) returns **(5 2 6 5 4)**.

```
(define (filter-list elm aset)
  (define (filter-iter lst result)
    (cond (( ___  lst) result)
          (( ___ ( ___ lst) elm) ( ___ ( ___  lst) result))
          (else ( ___ ( ___ lst) ( ___ result ( ___ ( ___ lst)))))))
  (filter-iter aset '() ))
```