# Homework 03

Graded

**Student**

Giancarlos Marte

**Total Points**

**100 / 100 pts**

**Autograder Score**

**52.0 / 52.0**

**Passed Tests**

Exercise 1. min-from (1/1)
Exercise 2. count (1/1)
Exercise 3. sum (1/1)
Exercise 4. occurrences (1/1)
Exercise 5. prefix (1/1)
Exercise 6. interleave (1/1)
Exercise 7. intersperse (1/1)
Exercise 8. apply (1) (4/4)
Exercise 8. apply (2) (4/4)
Exercise 8. define-basic (1) (2/2)
Exercise 8. define-basic+apply (1) (3/3)
Exercise 8. define-func (1) (5/5)
Exercise 8. define-func (2) (3/3)
Exercise 8. define-func+lambda+apply (1) (3/3)
Exercise 8. lambda (1) (4/4)
Exercise 8. lambda (2) (4/4)
Exercise 8. lambda+apply (1) (3/3)
Exercise 8. numbers (5/5)
Exercise 8. variables (5/5)

**Question 2**

## Adjustments

**48** / 48 pts

✔ **– 0 pts** Good use of abstract list functions

**– 6 pts** Missing one use of abstract list function

**– 13 pts** Missing two uses of abstract list functions

**– 7 pts** (6) Not a recursive function

**– 24 pts** Partial credit

**– 3 pts** `intersperse` should not check accumulator

Autograder Results

Exercise 1. min-from (1/1)

Exercise 2. count (1/1)

Exercise 3. sum (1/1)

Exercise 4. occurrences (1/1)

Exercise 5. prefix (1/1)

Exercise 6. interleave (1/1)

Exercise 7. intersperse (1/1)

Exercise 8. apply (1) (4/4)

Exercise 8. apply (2) (4/4)

Exercise 8. define-basic (1) (2/2)

Exercise 8. define-basic+apply (1) (3/3)

Exercise 8. define-func (1) (5/5)

Exercise 8. define-func (2) (3/3)

Exercise 8. define-func+lambda+apply (1) (3/3)

Exercise 8. lambda (1) (4/4)

Exercise 8. lambda (2) (4/4)

Exercise 8. lambda+apply (1) (3/3)

Exercise 8. numbers (5/5)

Exercise 8. variables (5/5)

**Submitted Files**

hw3.rkt  1 Comment  Download

```
#lang racket
#|
        ######################################################
        ###  PLEASE DO NOT DISTRIBUTE SOLUTIONS PUBLICLY  ###
        ######################################################

 Copy your solution of HW1 as file "hw1.rkt". The file should be in the same
 directory as "hw2.rkt" and "ast.rkt".
|#
(require "ast.rkt")
(require "hw1.rkt")
(require rackunit)
(provide (all-defined-out))
;; ^^^^^ DO NOT CHANGE ANY CODE ABOVE THIS LINE ^^^^^


;; Exercise 1
;; min-from: Real (Listof Real) -> Real
(define (min-from n l)
  (foldl min n l))

;; Exercise 2
;; count: (Listof X) -> Real
(define (count l)
  (define (c-step x accum)
    (+ 1 accum))
  (foldl c-step 0 l))

;; Exercise 3
;; sum: (Listof Real) -> Real
(define (sum l)
  (define (sum-step x accum)
    (+ x accum))
  (foldl sum-step 0 l))

;; Exercise 4
;; occurences: X (Listof X) -> Real
(define (occurrences n l)
  (define (occ-step x accum)
    (cond
      [(equal? x n) (+ 1 accum)]
      [else (+ 0 accum)]))
  (foldl occ-step 0 l))

;; Exercise 5
;; prefix: X (Listof X) -> (Listof X)
```

```
(define (prefix s l)
  (match l
    ['() empty]
    [(cons f r)
     (define (pre-step x accum)
       (cons (string-append s x) accum))
     (foldr pre-step empty l)]))

;; Exercise 6
;; interleave: (Listof X) (Listof Y) -> (Listof Z)
(define (interleave l1 l2)
  (match l1
    [(list)
      l2]
    [(cons h1 l1)
     (cons h1 (interleave l2 l1))]))

;; Exercise 7
;; intersperse: (Listof X) X -> (Listof X)
(define (intersperse l v)
  (match l
    ['() empty]
    [(cons f r)
     (define (inter-step x accum)
       (cons x (cons v accum)))
     (foldl cons empty (foldl inter-step (list f) r))]))
```

**Instructor** | 04/20 at 7:41 pm

`(foldl cons empty _)` is just `(reverse _)`

```

;; Exercise 8
;; parse-ast: Quote -> Struct
(define (parse-ast node)
  (define (make-define-func node)
    (r:define
     (parse-ast (first (first (rest node))))
     (r:lambda
      (map parse-ast (rest (first (rest node))))
      (map parse-ast (rest (rest node))))))

  (define (make-define-basic node)
    (r:define
     (parse-ast (second node))
     (parse-ast (third node))))

  (define (make-lambda node)
```

```
90    (r:lambda
91     (map parse-ast (first (rest node)))
92     (map parse-ast (rest (rest node)))))
93
94  (define (make-apply node)
95    (r:apply
96     (parse-ast (first node))
97     (map parse-ast (rest node))))
98
99  (define (make-number node) (r:number node))
100 (define (make-variable node) (r:variable node))
101
102 (cond
103   [(define-basic? node) (make-define-basic node)]
104   [(define-func? node) (make-define-func node)]
105   [(symbol? node) (make-variable node)]
106   [(real? node) (make-number node)]
107   [(lambda? node) (make-lambda node)]
108   [else (make-apply node)]))
109
```