# Homework 01

Graded

**Student**

Giancarlos Marte

**Total Points**

**100 / 100 pts**

**Autograder Score**
**100.0 / 100.0**

**Passed Tests**

Exercise 1 (4/4)
Exercise 1 (partial) (3/3)
Exercise 2 (4/4)
Exercise 2 (partial) (3/3)
Exercise 3 (7/7)
Exercise 3 (partial) (7/7)
Exercise 4. bst-insert (1) (2/2)
Exercise 4. bst-insert (2) (2/2)
Exercise 4. bst-insert (3) (8/8)
Exercise 4. tree (2/2)
Exercise 4. tree-leaf (2/2)
Exercise 4. tree-left (2/2)
Exercise 4. tree-right (2/2)
Exercise 4. tree-set-left (2/2)
Exercise 4. tree-set-right (2/2)
Exercise 4. tree-set-value (2/2)
Exercise 4. tree-value (2/2)
Exercise 5.a. lambda? (1) (3/3)
Exercise 5.a. lambda? (2) (3/3)
Exercise 5.a. lambda? (3) (1/1)
Exercise 5.a. lambda? (4) (2/2)
Exercise 5.a. lambda? (5) (3/3)
Exercise 5.b. lambda-params (2/2)
Exercise 5.c. lambda-body (2/2)
Exercise 5.d. apply? (4/4)
Exercise 5.e. apply-func, Exercise 4.f. apply-args (4/4)
Exercise 5.g. define? (1/1)
Exercise 5.h. define-basic? (1) (1/1)
Exercise 5.h. define-basic? (2) (1/1)
Exercise 5.h. define-basic? (3) (2/2)
Exercise 5.h. define-basic? (4) (2/2)
Exercise 5.i. define-func? (1) (3/3)
Exercise 5.i. define-func? (2) (3/3)
Exercise 5.i. define-func? (3) (3/3)
Exercise 5.i. define-func? (4) (4/4)

**Question 2**

**Adjustments** **0** / 0 pts

✔ **– 0 pts** No adjustment

**– 5 pts** Late penalty

Autograder Results

Exercise 1 (4/4)

Exercise 1 (partial) (3/3)

Exercise 2 (4/4)

Exercise 2 (partial) (3/3)

Exercise 3 (7/7)

Exercise 3 (partial) (7/7)

Exercise 4. bst-insert (1) (2/2)

Exercise 4. bst-insert (2) (2/2)

Exercise 4. bst-insert (3) (8/8)

Exercise 4. tree (2/2)

Exercise 4. tree-leaf (2/2)

Exercise 4. tree-left (2/2)

Exercise 4. tree-right (2/2)

Exercise 4. tree-set-left (2/2)

Exercise 4. tree-set-right (2/2)

Exercise 4. tree-set-value (2/2)

Exercise 4. tree-value (2/2)

Exercise 5.a. lambda? (1) (3/3)

Exercise 5.a. lambda? (2) (3/3)

Exercise 5.a. lambda? (3) (1/1)

Exercise 5.a. lambda? (4) (2/2)

Exercise 5.a. lambda? (5) (3/3)

Exercise 5.b. lambda-params (2/2)

Exercise 5.c. lambda-body (2/2)

Exercise 5.d. apply? (4/4)

Exercise 5.e. apply-func, Exercise 4.f. apply-args (4/4)

Exercise 5.g. define? (1/1)

Exercise 5.h. define-basic? (1) (1/1)

Exercise 5.h. define-basic? (2) (1/1)

Exercise 5.h. define-basic? (3) (2/2)

Exercise 5.h. define-basic? (4) (2/2)

Exercise 5.i. define-func? (1) (3/3)

Exercise 5.i. define-func? (2) (3/3)

Exercise 5.i. define-func? (3) (3/3)

Exercise 5.i. define-func? (4) (4/4)

**Submitted Files**

**hw1.rkt** Download

```
#lang racket
#|
   ===> PLEASE DO NOT DISTRIBUTE THE SOLUTIONS PUBLICLY <===

  We ask that solutions be distributed only locally -- on paper, on a
  password-protected webpage, etc.

  Students are required to adhere to the University Policy on Academic
  Standards and Cheating, to the University Statement on Plagiarism and the
  Documentation of Written Work, and to the Code of Student Conduct as
  delineated in the catalog of Undergraduate Programs. The Code is available
  online at: http://www.umb.edu/life_on_campus/policies/code/

                * * * ATTENTION! * * *

  Every solution submitted to our grading server is automatically compared
  against a solution database for plagiarism, which includes every solution
  from every student in past semesters.

  WE FOLLOW A ZERO-TOLERANCE POLICY: any student breaking the Code of Student
  Conduct will get an F in this course and will be reported according to
  Section II Academic Dishonesty Procedures.

|#

;; Please, do not remove this line and do not change the function names,
;; otherwise the grader will not work and your submission will get 0 points.
(provide (all-defined-out))


(define ex1 (/
             (*
              (/ 15 10) 7)
             (*
              (- 11 7) 15)))
(define ex2
  (list
   (/
    (*
     (/ 15 10) 7)
    (*
     (- 11 7) 15))
   (/
    (* 3/2 7)
    (*
     (- 11 7) 15))
```

```
   (/ 21/2
      (*
       (- 11 7) 15))
   (/ 21/2
      (* 4 15))
   (/ 21/2 60)
   7/40))

(define (ex3 x y)
            (>=
             (+
              (+ 2 15)
              (- x 5))
             (+
              (* 6 12) x)))


;; Constructs a tree from two trees and a value
(define (tree left value right) (list left value right))
;; Constructs a tree with a single node
(define (tree-leaf value) (tree null value null))

;; Accessors
(define (tree-left self) (car self))
(define (tree-value self) (car (cdr self)))
(define (tree-right self) (car (cdr (cdr self))))

;; Copies the source and updates one of the fields
(define (tree-set-value self value) (tree (car self) value (car (cdr (cdr self)))))
(define (tree-set-left self left) (tree left (car (cdr self)) (car (cdr (cdr self)))))
(define (tree-set-right self right) (tree (car self) (car (cdr self)) right))

;; Function that inserts a value in a BST
(define (bst-insert self value)
  (cond [(null? self) (tree null value null)]
        [(equal? value (car (cdr self))) (tree-set-value self value)]
        [(< value (car (cdr self))) (tree-set-left self (bst-insert (car self) value))]
        [else (tree-set-right self (bst-insert (car (cdr (cdr self))) value))]))

;; lambda
(define (lambda? node)
  (cond [(not (list? node)) #f]
        [(< (length node) 3) #f]
        [(not (equal? 'lambda (car node))) #f]
        [(not (list? (car (cdr node)))) #f]
        [(not (andmap symbol? (car (cdr node)))) #f]
        [else #t]))
(define (lambda-params node) (car (cdr node)))
(define (lambda-body node) (cdr (cdr node)))
```

```
96
97  ;; apply
98  (define (apply? l)
99    (cond [(null? l) #f]
100         [(not (list? l)) #f]
101         [else #t]))
102 (define (apply-func node) (car node))
103 (define (apply-args node) (cdr node))
104
105 ;; define
106 (define (define? node)
107   (cond
108     [(or
109       (define-basic? node)
110       (define-func? node)) #t]
111     [else #f]))
112
113 (define (define-basic? node)
114   (cond
115     [(and
116       (not (null? node))
117       (list? node)
118       (equal? (length node) 3)
119       (equal? (car node) 'define)
120       (symbol? (car (cdr node)))) #t]
121     [else #f]))
122
123 (define (define-func? node)
124   (cond
125     [(and
126       (not (null? node))
127       (list? node)
128       (>= (length node) 3)
129       (equal? (car node) 'define)
130       (list? (car (cdr node)))
131       (not (null? (car (cdr node))))
132       (andmap symbol? (car (cdr node)))) #t]
133     [else #f]))
134
135
136
137
138
```