COMP 333

Project #3: Polynomials

3/17/2015

Yiana Chang

Shant Hairapetian

#lang racket

;Yiana Chang and Shant Hairapetian

;March 17, 2015

;COMP333 Project 3 - Polynomials

(define coeff (lambda (t)

(car t)

)

)

(define expon (lambda (t)

(last t))

)

(define printTerm (lambda (t)

(if (= (coeff t) 0) (display "0")

(begin

(let\*

([c (coeff t)]

[e (expon t)])

(cond

((= c 1) (display ""))

((= c -1) (display "-"))

(else (display c)))

(cond

((= e 0) (display ""))

((= e 1) (display "x"))

(else

(display "x^")

(display e)))

)

)

)

)

)

(define printpoly (lambda (p)

(cond

((null? p) p)

((= (length p) 1)

(printTerm (car p)))

(else

(begin

(if (= (caar p) 0)

(printpoly (cdr p))

(begin

(if (and (= (length (cdr p)) 1) (= (caadr p) 0))

(printTerm (car p))

(begin

(printTerm (car p))

(display " + ")

(printpoly (cdr p))))

)

)

)

)

)

)

)

(define evalpoly (lambda (p v)

(if (= (length p) 1)

(\* (coeff (car p)) (expt v (expon (car p))))

(+ (\* (coeff (car p)) (expt v (expon (car p))))

(evalpoly (cdr p) v)))

)

)

(define GT (lambda (t1 t2)

(if (> (expon t1) (expon t2)) #t #f))

)

(define EQExp? (lambda (t1 t2)

(if (= (expon t1) (expon t2)) #t #f))

)

(define addTerm (lambda (t1 t2)

(list (+ (coeff t1) (coeff t2)) (expon t1)))

)

(define simplify (lambda (p)

(simplifyRec (sort p GT)))

)

(define simplifyRec (lambda (p)

(if (<= (length p) 1) (list (car p))

(let\*(

[t1 (car p)]

[t2 (cadr p)]

)

(if (EQExp? t1 t2 )

(if (> (length(cddr p)) 0)

(simplifyRec (append (list (addTerm t1 t2)) (cddr p)))

(append (list (addTerm t1 t2))))

(append (list (car p)) (simplifyRec (cdr p)))))

)

)

)

(define addpoly (lambda (p1 p2)

(simplify (append p1 p2)))

)

(define subtractpoly (lambda (p1 p2)

(simplify (append p1 (map (lambda (x) (list (\* (car x) -1) (cadr x))) p2))))

)

(define multiplyterms (lambda (t1 t2)

(list (\* (coeff t1)(coeff t2)) (+ (expon t1)(expon t2))))

)

(define multiplytermpoly (lambda (t1 p1)

(simplify (map (lambda (x) (multiplyterms t1 x)) p1)))

)

(define multiplypolyRec (lambda (p1 p2 acc)

(if (= (length p1) 0)

acc

(multiplypolyRec (cdr p1) p2 (append acc (multiplytermpoly (car p1) p2))))

)

)

(define multiplypoly (lambda (p1 p2)

(let\* ([acc '()])

(simplify (multiplypolyRec p1 p2 acc)))

)

)

Welcome to DrRacket, version 6.1.1 [3m].

Language: racket; memory limit: 128 MB.

> (define p1 '( ( 3 2) (8 3) (7 1) (4 0)))

> (define p2 '( ( 5 2) (6 4) (-9 3) (4 0)))

> (define p3 '(( 3 2) (-8 1) ( 5 2 ) ( 6 1) ( 6 2)))

> (define p4 '( (5 0) ( 2 3) ( -3 1) ))

> (define p5 '( (2 1) ( 1 0)))

> (printpoly p3)

3x^2 + -8x + 5x^2 + 6x + 6x^2

> (printpoly (simplify p3))

14x^2 + -2x

> (evalpoly p3 3)

120

> (printpoly (addpoly p1 p2))

6x^4 + -x^3 + 8x^2 + 7x + 8

> (evalpoly (addpoly p1 p2) -2)

130

> (printpoly (subtractpoly '( ( 3 2) (5 1) (1 0)) '( (6 0) ( 4 1) ( 3 2))))

x + -5

> (define a ( multiplytermpoly '( 3 1) p5))

> (evalpoly a 1.5)

18.0

> (printpoly( multiplypoly p4 p5))

4x^4 + 2x^3 + -6x^2 + 7x + 5

> (evalpoly ( multiplypoly p4 p5) 1.25)

18.046875

> (subtractpoly p3 p3)

'((0 2) (0 1))

> (printpoly (subtractpoly p3 p3))

0

>