# Cmput 325 Assignment 2

**Update Feb 2: fixed typo in Question 2, (0 \* Exp) changed to (\* Exp 0). Added test cases 2.8, 3.9. 3.10 in [a2-public-tests.lisp](https://webdocs.cs.ualberta.ca/~mmueller/courses/325-Winter-2019/assignment/a2-public-tests.lisp) to cover more variations of multiply by zero.**

**This assignment is due Feb 25 at 23:55pm. Submit through the button on eClass. NO LATE SUBMISSIONS.   
This assignment should be submitted as a single text file named assignment2.lisp .**

**All our examples, which are also our public test cases, are in file [a2-public-tests.lisp](https://webdocs.cs.ualberta.ca/~mmueller/courses/325-Winter-2019/assignment/a2-public-tests.lisp). Also see the comments in that file.**

## Overview

**In this assignment, you implement a restricted form of the classic problem of simplifying arithmetic expressions. You first implement two simplification operations, then a general algorithm to transform such expressions into a normal form.**

**The main restriction is that we only deal with the operations +, -, and \*, and that the arguments are only integers and a single variable symbol x. Therefore, all expressions eventually simplify to a [polynomial](https://en.wikipedia.org/wiki/Polynomial) inx, with integer coefficients.**

### TYPES OF EXPRESSIONS IN THIS ASSIGNMENT

**We have two main types of expressions in this assignment: Assignment 2 expressions or A2Expr are built as explained above, and polynomials in x or PExpr are represented in a specific short form.**

#### A2Expr - a more formal definition

* **An integer is an A2Expr**
* **x is an A2Expr**
* **If E1 and E2 are A2Expr, then (+ E1 E2), (- E1 E2) and (\* E1 E2) are A2Expr**
* **Nothing else is an A2Expr**

## #1 (2 marks)

**Write a Lisp function:**

**(remove-identities E)**

**The input E can be any valid A2Expr. This function should replace any term of the form (+ 0 Exp), (+ Exp 0), (\* 1 Exp), (\* Exp 1) by Exp. It should apply this simplification recursively to all nested expressions. It should not change E in any other way, and return the simplified A2Expr.**

**See remove-identities Examples in public tests.**

## #2 (2 marks)

**Write a Lisp function:**

**(simplify-zeroes E)**

**The input E can be any valid A2Expr. This function should replace any term of the form (\* 0 Exp), (\* Exp 0) or (- Exp Exp) by 0. It should apply this simplification recursively into all nested expressions. It should not change E in any other way, and return the simplified A2Expr.**

**See simplify-zeroes Examples in public tests.**

## #3 (1 mark + 1 possible bonus mark)

### #3.1 (1 MARK)

**Write a Lisp function:**

**(simplify E)**

**This function should repeatedly call   
1. remove-identities, followed by   
2. simplify-zeroes,   
until no more simplification is possible in either step 1 or 2.**

**See simplify Examples in public tests.**

## #4 (2 marks)

**Write a Lisp function:**

**(normalize P)**

**The input P is an arbitrary PExpr. The output is the normal form of P. See the normalize examples in public tests.**

## #5 (5 marks)

**Write a Lisp function:**

**(polynomial E)**

**The input E is an arbitrary A2Expr. The output is the equivalent PExpr of E in normal form. First implement the three helper functions in 5.1 and 5.2.**

### #5.1 (1 MARK)

**Write two Lisp functions:**

**(poly-add P1 P2)**

**(poly-subtract P1 P2)**

**For these two functions, the inputs P1 and P2 are PExpr in normal form. The output should be the sum (for poly-add) or difference (for poly-subtract) of the two PExpr, also in normal form. See poly-add, poly-subtract examples in public tests.**

**Hints:   
normalize is your friend...   
If n is an integer, then (- n) computes its negative value. Note the space.**

### #5.2 (2 MARKS)

**Write a Lisp function:**

**(poly-multiply P1 P2)**

**Again, the inputs P1 and P2 as well as the output are PExpr in normal form. See poly-multiply examples in public tests.**

**Hints: To compute the product of two PExpr, "multiply them out" and normalize the result.   
Example for illustration only in math-like syntax:   
(x+1)\*(x-1) = x^2 - x + x - 1 = x^2 + (-1 + 1)x -1 = x^2 - 1   
Real examples in PExpr for the assignment: see public tests.**

**The function cartesian in sample code [list-functions.lisp](https://webdocs.cs.ualberta.ca/~mmueller/courses/325-Winter-2019/code/lisp/list-functions.lisp) is an example with similar structure, which may help you with the recursion over both P1 and P2.**

### #5.3 (2 MARKS)

**Implement polynomial, using the three helper functions.**

**Hint: use recursion, with the base cases:   
1. integer n - represent by (n . 0)   
2. atom x - represent by (1 . 1)**

**See polynomial examples in public tests.**

## #6 (2 marks) Printing a PExpr in normal form

**Write a Lisp function:**

**(print-pexpr P)**

**The input P is a PExpr in normal form. Output a string representing P in the following "common sense" format:**

* **Print terms in the form cx^n, where c and n are integer**
* **Print " + " (space plus space) or " - " (space minus space) between the terms, depending on whether the next term has a positive or negative coefficient**
* **Print terms in sorted order from highest to lowest exponent**
* **Do not print the 1 if the coefficient is 1, except for the constant term**
* **Print -, not -1 if the coefficient is -1, except for the constant term**
* **Do not print \*x^0 for a constant term**
* **Print x instead of x^1**
* **Print 0 if the PExpr is nil.**

**See print-pexpr Examples in public tests, and [string functions](https://webdocs.cs.ualberta.ca/~mmueller/courses/325-Winter-2019/assignment/a2.html" \l "string-functions).**

1. remove-identities

; basic tests for +

(test-case '1.1.1 (remove-identities '(+ x 0)) 'x)

(test-case '1.1.2 (remove-identities '(+ 0 x)) 'x)

(test-case '1.1.3 (remove-identities '(+ 0 0)) 0)

(test-case '1.1.4 (remove-identities '(+ x x)) '(+ x x))

(test-case '1.1.5 (remove-identities '(+ (+ x 0) 0)) 'x)

(test-case '1.1.6 (remove-identities '(+ (+ 0 0) 0)) 0)

(test-case '1.1.7 (remove-identities '(+ (+ 5 0) 1)) '(+ 5 1))

; basic tests for -

(test-case '1.2.1 (remove-identities '(- x 0)) 'x)

(test-case '1.2.2 (remove-identities '(- 0 x)) '(- 0 x))

(test-case '1.2.3 (remove-identities '(- 0 0)) 0)

(test-case '1.2.4 (remove-identities '(- x x)) '(- x x))

(test-case '1.2.5 (remove-identities '(- (- x 0) 0)) 'x)

(test-case '1.2.6 (remove-identities '(- (- 0 0) 0)) 0)

(test-case '1.2.7 (remove-identities '(- (- 5 0) 1)) '(- 5 1))

; basic tests for \*

(test-case '1.3.1 (remove-identities '(\* x 1)) 'x)

(test-case '1.3.2 (remove-identities '(\* 1 x)) 'x)

(test-case '1.3.3 (remove-identities '(\* 1 1)) 1)

(test-case '1.3.4 (remove-identities '(\* x x)) '(\* x x))

(test-case '1.3.5 (remove-identities '(\* (\* x 1) 1)) 'x)

(test-case '1.3.6 (remove-identities '(\* (\* 1 1) 1)) 1)

(test-case '1.3.7 (remove-identities '(\* (\* 5 1) 0)) '(\* 5 0))

; basic tests for mixing +, -, \*

(test-case '1.4.1 (remove-identities '(+ 0 (\* x 1))) 'x)

(test-case '1.4.2 (remove-identities '(+ (+ 0 0) (\* 5 1))) 5)

(test-case '1.4.3 (remove-identities '(- 10 (+ 0 (\* 0 1)))) 10)

(test-case '1.4.4 (remove-identities '(- 10 10)) '(- 10 10))

(test-case '1.4.5 (remove-identities '(- 10 10)) '(- 10 10))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; 2. tests for simplify-zeroes

(test-case 2.1 (simplify-zeroes 0) 0)

(test-case 2.2 (simplify-zeroes '(\* x 0)) 0)

(test-case 2.3 (simplify-zeroes '(\* x (+ 0 0)))

'(\* x (+ 0 0)))

; It cannot simplify (+ 0 0) to 0

(test-case 2.4 (simplify-zeroes '(\* x (\* 0 0))) 0)

; It can simplify this

(test-case 2.5 (simplify-zeroes '(- x x)) 0)

(test-case 2.6 (simplify-zeroes '(- (\* 5 (+ 2 x)) (\* 5 (+ 2 x)))) 0)

(test-case 2.7 (simplify-zeroes '(- (\* 5 (+ 2 x)) (\* 5 (+ x 2))))

'(- (\* 5 (+ 2 x)) (\* 5 (+ x 2))))

; It does not simplify, (+ 2 x) and (+ x 2) look different

(test-case 2.8 (simplify-zeroes '(\* 0 x)) 0)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; 3. tests for simplify

;Some examples where one type of simplification enables the other.

;All the examples from question 1 and 2 should still be simplified

;at least as much as before, too.

(test-case 3.1 (simplify 0) 0)

(test-case 3.2 (simplify '(\* x 0)) 0)

(test-case 3.3 (simplify '(\* x (+ 0 0))) 0)

; remove-identities reduces this to (\* x 0). Then simplify-zeroes does the rest.

(test-case 3.4 (simplify '(- x x)) 0)

(test-case 3.5 (simplify '(- (\* 5 (+ 2 x)) (\* 5 (+ 2 x)))) 0)

(test-case 3.6 (simplify '(- (\* 5 (+ 2 x)) (\* 5 (+ x 2))))

'(- (\* 5 (+ 2 x)) (\* 5 (+ x 2))))

(test-case 3.7 (simplify '(\* (+ 5 0) (- 9 9))) 0)

(test-case 3.8 (simplify '(+ (+ 5 0) (- 4 9)))

'(+ 5 (- 4 9))

)

(test-case 3.9 (simplify '(\* 0 x)) 0)

(test-case 3.10 (simplify '(\* (- 9 9) (+ 5 0))) 0)

(test-case 3.11 (simplify '(+ (+ 1 0) (\* 0 (+ 0 x)))) 1)

(test-case 3.12 (simplify '(+ (+ 0 0) (+ 1 (- x x)))) 1)

(test-case 3.13 (simplify '(+ 1 (- x x))) 1)

(test-case 3.14 (simplify '(+ (- (+ x (\* 5 (- 9 (+ 9 (\* 0 x))))) x) (- (+ x (\* 5 (- 9 (+ 9 (\* 0 x))))) x))) 0)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; 4. tests for normalize

(test-case 4.1 (normalize '((5 . 10) (3 . 4) (7 . 0)))

'((5 . 10) (3 . 4) (7 . 0)))

(test-case 4.2 (normalize '((5 . 2) (7 . 0) (3 . 1)))

'((5 . 2) (3 . 1) (7 . 0)))

(test-case 4.3 (normalize '((5 . 3) (0 . 2) (3 . 1) (7 . 0)))

'((5 . 3) (3 . 1) (7 . 0)))

(test-case 4.4 (normalize '((0 . 0))) NIL)

(test-case 4.5 (normalize '((5 . 2) (7 . 0) (1 . 11111) (-9 . 2) (3 . 0)

(3 . 1) (-1 . 11111) (0 . 1)))

'((-4 . 2) (3 . 1) (10 . 0)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; 5. tests for polynomial

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; 5.1 tests for poly-add

(test-case '5.1.1

(poly-add '((5 . 2) (3 . 1) (7 . 0))

'((5 . 2) (3 . 1) (7 . 0)))

'((10 . 2) (6 . 1) (14 . 0))

)

(test-case '5.1.2

(poly-add '((5 . 2) (3 . 1) (7 . 0)) nil)

'((5 . 2) (3 . 1) (7 . 0))

)

(test-case '5.1.3

(poly-add '((-9 . 0)) '((5 . 0)))

'((-4 . 0))

)

(test-case '5.1.4

(poly-add '((5 . 2) (3 . 1) (7 . 0)) '((5 . 3) (3 . 1) (7 . 0)))

'((5 . 3) (5 . 2) (6 . 1) (14 . 0))

)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; 5.1 tests for poly-subtract

(test-case '5.1.5

(poly-subtract '((5 . 2) (3 . 1) (7 . 0))

'((5 . 2) (3 . 1) (7 . 0)))

NIL

)

(test-case '5.1.6

(poly-subtract '((-9 . 0)) '((5 . 0)))

'((-14 . 0))

)

(test-case '5.1.7

(poly-subtract '((5 . 2) (3 . 1) (7 . 0))

'((5 . 3) (3 . 1) (7 . 0)))

'((-5 . 3) (5 . 2))

)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; 5.2 tests for poly-multiply

(test-case '5.2.1 (poly-multiply nil '((5 . 3) (0 . 2) (3 . 1) (7 . 0)))

'NIL

)

(test-case '5.2.2 (poly-multiply '((5 . 2) (3 . 1) (7 . 0)) '((-10 . 0)))

'((-50 . 2) (-30 . 1) (-70 . 0))

)

(test-case '5.2.3 (let ((P '((5 . 2) (3 . 1) (7 . 0))))

(poly-multiply P P)

)

'((25 . 4) (30 . 3) (79 . 2) (42 . 1) (49 . 0))

)

(test-case '5.2.4 (let\* ((P '((1 . 1) (1 . 0))))

(poly-multiply P P))

'((1 . 2) (2 . 1) (1 . 0))

)

(test-case '5.2.5

(let\* ((P '((1 . 1) (1 . 0)))

(P2 (poly-multiply P P)))

(poly-multiply P2 P2)

)

'((1 . 4) (4 . 3) (6 . 2) (4 . 1) (1 . 0))

)

(test-case '5.2.6

(let\* ((P '((1 . 1) (1 . 0)))

(P2 (poly-multiply P P))

(P4 (poly-multiply P2 P2))

(P8 (poly-multiply P4 P4))

(P16 (poly-multiply P8 P8)))

(poly-multiply P16 P16)

)

'((1 . 32) (32 . 31) (496 . 30) (4960 . 29) (35960 . 28) (201376 . 27)

(906192 . 26) (3365856 . 25) (10518300 . 24) (28048800 . 23) (64512240 . 22)

(129024480 . 21) (225792840 . 20) (347373600 . 19) (471435600 . 18)

(565722720 . 17) (601080390 . 16) (565722720 . 15) (471435600 . 14)

(347373600 . 13) (225792840 . 12) (129024480 . 11) (64512240 . 10)

(28048800 . 9) (10518300 . 8) (3365856 . 7) (906192 . 6) (201376 . 5)

(35960 . 4) (4960 . 3) (496 . 2) (32 . 1) (1 . 0))

)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; 5.3 tests for polynomial

(test-case '5.3.1

(polynomial -42)

'((-42 . 0))

)

(test-case '5.3.2

(polynomial 'x)

'((1 . 1))

)

(test-case '5.3.3

(polynomial '(+ x 1))

'((1 . 1) (1 . 0))

)

(test-case '5.3.4

(polynomial '(+ 1 x))

'((1 . 1) (1 . 0))

)

(test-case '5.3.5

(polynomial '(+ (+ 5 0) (- 4 9)))

nil

)

(test-case '5.3.6

(let\* ((P '(+ 1 x))

(P2 (list '\* P P))

(P4 (list '\* P2 P2))

)

(polynomial P4)

)

'((1 . 4) (4 . 3) (6 . 2) (4 . 1) (1 . 0))

)

(test-case '5.3.7

(polynomial '(- (\* (+ x 5) (- x 5) ) (\* x x)))

'((-25 . 0))

)

(test-case '5.3.8

(polynomial '(- (\* 5 (+ 2 x)) (\* 5 (+ x 2))))

NIL

)

(test-case '5.3.9

(let\* ((P '(+ 1 x))

(P2 (list '\* P P))

(P4 (list '\* P2 P2))

(P8 (list '\* P4 P4))

(P16 (list '\* P8 P8))

(P32 (list '\* P16 P16))

)

(polynomial P32)

)

'((1 . 32) (32 . 31) (496 . 30) (4960 . 29) (35960 . 28) (201376 . 27)

(906192 . 26) (3365856 . 25) (10518300 . 24) (28048800 . 23) (64512240 . 22)

(129024480 . 21) (225792840 . 20) (347373600 . 19) (471435600 . 18)

(565722720 . 17) (601080390 . 16) (565722720 . 15) (471435600 . 14)

(347373600 . 13) (225792840 . 12) (129024480 . 11) (64512240 . 10)

(28048800 . 9) (10518300 . 8) (3365856 . 7) (906192 . 6) (201376 . 5)

(35960 . 4) (4960 . 3) (496 . 2) (32 . 1) (1 . 0))

)

(test-case '5.3.10

(polynomial '(+ 5 5))

'((10 . 0))

)

(test-case '5.3.11

(polynomial '(- 5 5))

nil

)

(test-case '5.3.12

(polynomial '(\* 2 3))

'((6 . 0))

)

(test-case '5.3.13

(polynomial '(+ (+ x 1) x))

'((2 . 1) (1 . 0))

)

(test-case '5.3.14

(polynomial '(- (+ (+ (- (\* (\* (\* x x) (\* x x)) (\* 3 x))

(\* (\* -2 x) (\* x x))) (\* (\* 2 (\* 2 2)) (\* x x))) x)

(+ (\* 2 (\* 2 (\* 2 2))) 1)))

'((3 . 5) (2 . 3) (8 . 2) (1 . 1) (-17 . 0))

)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; 6 tests for print-pexpr

(test-case 6.1

(print-pexpr '((23 . 0)))

"23"

)

(test-case 6.2

(print-pexpr '((1 . 1) (-5 . 0)))

"x - 5"

)

(test-case 6.3

(print-pexpr '((5 . 3) (5 . 1)))

"5x^3 + 5x"

)

(test-case 6.4

(print-pexpr '((-1 . 10) (-23 . 0)))

"-x^10 - 23"

)

(test-case 6.5

(print-pexpr '((-1 . 1) (-1 . 0)))

"-x - 1"

)

(test-case 6.6

(print-pexpr '((-1 . 0)))

"-1"

)

(test-case 6.7

(print-pexpr nil)

"0"

)

(test-case 6.8

(print-pexpr '((1 . 100)))

"x^100"

)

; Examples of BAD output of print-pexpr:

; 1 + 1 (cannot happen with input in normal form)

; x + x^2 (wrong order)

; 1x (just write x)

; -1x (just write -x)

; x + 0 (no + 0, cannot happen with input in normal form)

; 0x + 5 (just write 5, cannot happen with input in normal form)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Finally, tests that combine polynomial and print-pexpr

(test-case 'combined.1

(let

((P (polynomial '(- (+ (+ (- (\* (\* (\* x x) (\* x x)) (\* 3 x))

(\* (\* -2 x) (\* x x))) (\* (\* 2 (\* 2 2)) (\* x x))) x)

(+ (\* 2 (\* 2 (\* 2 2))) 1))))

)

(print-pexpr P)

)

"3x^5 + 2x^3 + 8x^2 + x - 17"

)

(test-case 'combined.2

(let

((P (polynomial '(- (\* (+ x 5) (- x 5) ) (\* x x))))

)

(print-pexpr P)

)

"-25"

)

(test-case 'combined.3

(let

((P (polynomial '(- (\* 5 (+ 2 x)) (\* 5 (+ x 2)))))

)

(print-pexpr P)

)

"0"

)

(test-case 'combined.4

(let\* ((P '(+ 1 x))

(P2 (list '\* P P))

(P4 (list '\* P2 P2))

(P8 (list '\* P4 P4))

(P16 (list '\* P8 P8))

(P32 (list '\* P16 P16))

(P32-PExpr (polynomial P32))

)

(print-pexpr P32-PExpr)

)

"x^32 + 32x^31 + 496x^30 + 4960x^29 + 35960x^28 + 201376x^27 + 906192x^26 + 3365856x^25 + 10518300x^24 + 28048800x^23 + 64512240x^22 + 129024480x^21 + 225792840x^20 + 347373600x^19 + 471435600x^18 + 565722720x^17 + 601080390x^16 + 565722720x^15 + 471435600x^14 + 347373600x^13 + 225792840x^12 + 129024480x^11 + 64512240x^10 + 28048800x^9 + 10518300x^8 + 3365856x^7 + 906192x^6 + 201376x^5 + 35960x^4 + 4960x^3 + 496x^2 + 32x + 1"

)

(test-case 'combined.5

(let\* ((P '(- 1 x))

(P2 (list '\* P P))

(P4 (list '\* P2 P2))

(P8 (list '\* P4 P4))

(P16 (list '\* P8 P8))

(P32 (list '\* P16 P16))

(P32-PExpr (polynomial P32))

)

(print-pexpr P32-PExpr)

)

"x^32 - 32x^31 + 496x^30 - 4960x^29 + 35960x^28 - 201376x^27 + 906192x^26 - 3365856x^25 + 10518300x^24 - 28048800x^23 + 64512240x^22 - 129024480x^21 + 225792840x^20 - 347373600x^19 + 471435600x^18 - 565722720x^17 + 601080390x^16 - 565722720x^15 + 471435600x^14 - 347373600x^13 + 225792840x^12 - 129024480x^11 + 64512240x^10 - 28048800x^9 + 10518300x^8 - 3365856x^7 + 906192x^6 - 201376x^5 + 35960x^4 - 4960x^3 + 496x^2 - 32x + 1"

)