Raheem Mir, Malcolm Todd, Raj Pannu

Preamble: Nearly every function that matches the names of the ones defined in the project handout only performs the job of interpreting the representation(s) of the input polynomial(s) in order to:

- Determine the representation of the output polynomial
- Convert the representation(s) of the inputs to dense if needed

before passing them as parameters to one or more helper functions. All of the internal logic is done on polynomials of dense representation and all of the logic is in the helper functions.

1. We chose to perform all internal logic assuming a dense representation of inputted polynomial(s). Each interpreter function converts the representations to dense as needed before passing into helper functions, and converts the final output to sparse if the inputted polynomial(s) were sparsely represented.

2. Each function was implemented with similar methodology. We preferred the dense representation due to its simplicity and accessibility without the need for parsing pair lists. This introduced the need for tracking the power of a term into successive recursions on a sub-list. This is why several helper functions take in some type of offset integer parameter.

    a. is-dense, problem is the same as ensuring the list is a list of numbers.
    b. is-sparse, if we break down the entries into pairs using a helper function we can use the same approach as is-dense.
    c. to-sparse, utilizes a helper function that takes in an integer representing the offset from the beginning of the list as the list gets smaller with recursion. To convert from dense, it cons's a pair of the first element/coefficient and its index/power, then recurses down the list until the base case of null.
    d. to-dense, utilizes a helper function that takes in an integer representing the power of the current term in the dense representation of the polynomial. If the current pair's power in the sparse representation matches that index, cons's the coffecient of that pair into the dense representation, otherwise cons 0 and continue.
    e. degree, simply increments an integer for each index traversed in the dense representation of a polynomial
    f. is-zero, recurses down an increasingly smaller list checking if the first element is = 0. Will only return true if it reaches the end (if every element is 0).
    g. coeff, utilizes a helper function that checks if the index (power) of the first/current term in the dense representation of the list matches the k parameter. Otherwise recurses down the list, decrementing k to account for the smaller list size/degree.
    h. eval, utilizes a helper function that takes in an integer parameter that stores the offset from the beginning of the array, this represents the power of the current term when that information is not accessible from the smaller sub-lists during the recursion down the list. Simply multiplies the coefficient of the current first term by k ^ offset and recurses, incrementing offset.

i. add, utilizes a helper function that cons's the sum of coeffecients of two terms with equivalent power. If one polynomial is larger than the other, cons's the terms of the remaining one.
j. subtract, same logic as add function but subtracting coefficients instead.
k. multiply, utilizes two helper functions that execute two subproblems. Distribute performs the task of multiplying a densely represented polynomial by a scalar coefficient, and multiply-helper utilizes the previously defined add function to add successive distributions (done for each element of p1 by recursively calling multiply helper on rest p1).
l. remainder, the same as quotient except upon finding no possible multiple of p2 to subtract from p1, returns p1 (the current dividend at that stage in the subtraction).
m. gcd, the Euclidean algorithm, utilizing previously defined functions. Namely taking the remainder with the polynomials alternating as dividends and divisor, and distribute to make the leading coefficient 1.

3. There are no limitations, other than the ones specified in the project handout (divisors are not 0, etc..) and no bugs that we are aware of.
4. We tested our functions on a base set of polynomials, with some being factors of each other and some not, in order to holistically test all of the binary operations with cases of both complete/incomplete division for testing quotient, remainder, and gcd.

   We used the difference of squares (x^2-1) = (x+1)(x-1) as simple base case(s) and then combinations of (x^3 + x^2 – x – 1) and (x^2 – 13x – 4) in addition with the previous polynomials for breadth of tests. The test cases are provided below, simply paste them into the Racket program.

```
; ---------- Test Cases ----------


(define A '((-1 0) (1 2)))  ; (x^2-1)
(define B '(1 1))           ; (x+1)
(define C '(-1 1))          ; (x-1)
(define D '(-1 -1 1 1))     ;(x^3 + x^2 - x - 1)
(define E '((-4 0) (-13 1) (1 2)))   ; x^2 - 13x - 4


(displayln "is-sparse?")

(is-sparse? A)
(is-sparse? B)
(is-sparse? C)
(is-sparse? D)
(is-sparse? E)
```

```
(displayln "is-dense?")

(is-dense? A)
(is-dense? B)
(is-dense? C)
(is-dense? D)
(is-dense? E)

(displayln "to-sparse")
(to-sparse A)
(to-sparse B)
(to-sparse C)
(to-sparse D)
(to-sparse E)

(displayln "to-dense")
(to-dense A)
(to-dense B)
(to-dense C)
(to-dense D)
(to-dense E)

(displayln "degree")
(degree A)
(degree B)
(degree C)
(degree D)

(displayln "is-zero?")
(is-zero? A)
(is-zero? B)
(is-zero? C)
(is-zero? D)
(is-zero? E)

(displayln "coeff")
(coeff A 1)
(coeff B 0)
(coeff C 1)
(coeff D 3)
(coeff E 1)

(displayln "eval")
```

```
(eval A 0)
(eval B 1)
(eval C 2)
(eval D 3)
(eval E 4)

(displayln "add")
(add A A) ; (2x^2 - 2)
(add B C) ; (2x)
(add D A) ; x^3 + 2x^2 -x - 2
(add A D)

(displayln "subtract")
(subtract A A) ; 0
(subtract A B) ; x^2 - x -2
(subtract D C) ; x^3 +x^2 -2x
(subtract B D) ; -x^3 -x^2 + 2x + 1
(subtract D E) ; x^3 + 12x + 3

(displayln "multiply")
(multiply B C) ; x^2 - 1
(multiply A B) ; x^3 + x^2 - x - 1
(multiply C A) ; x^3 - x^2 - x + 1
(multiply E (multiply E B)) ; x^5 - 25 x^4 + 135 x^3 + 265 x^2 + 120 x + 16
(multiply D A) ; x^5 + x^4 - 2x^3 -2x^2 + x + 1

(displayln "quotient")
(quotient A A)
(quotient A B) ; x - 1
(quotient A C) ; x + 1
(quotient D A) ; x + 1
(quotient D B) ; x^2 - 1
(quotient D E) ; x + 14
(quotient (multiply E (multiply E B)) E) ; x^3 - 12x^2 - 17x - 4

(displayln "remainder")
(remainder A A)
(remainder D C)
(remainder E A) ; -13x -3
(remainder E B) ; 10
(remainder D E) ; 185x + 55

(displayln "derivative")
(derivative A) ; 2x
```

```
(derivative B) ; 1
(derivative C) ; 1
(derivative D) ; 3x^2 + 2x - 1
(derivative E) ; 2x -13

(displayln "gcd")
(gcd A A) ; x^2 - 1
(gcd A B) ; x + 1
(gcd D A) ; x^2 - 1
(gcd D C) ; x - 1
(gcd E D) ; 1
```