Fall 2025
Clarke, Garbary, Morland, Petrick, Roh, Watson

Assignment: 04

Due:	Tuesday, October 7, 2025 9:00 pm
Coverage:	L07
Language level:	Beginning Student
Allowed recursion:	Second version
Files to submit:	`listfun.rkt`, `morelistfun.rkt`, `horner.rkt`, `spells.rkt`

Assignment policies:

- Make sure you read the official assignment post on **Piazza**.
- You may not use functions or language constructs from lectures after the "coverage" lecture listed above.
- Functions and symbols must be named **exactly** as they are written in the assignment questions. You may define helper functions, if needed.
- You must provide a purpose, contract, and appropriate test cases for all required functions, i.e. those we explicitly ask you to write.

Here are the assignment questions you need to solve and submit.

1. **(10%)** Complete the required stepping problems for A04 at:

   `https://www.student.cs.uwaterloo.ca/~cs135/stepping/`

   You should refer to the instructions from A01 Question 1 for the stepper question instructions.

2. **(20%)** Below are some problems requiring recursion on lists:

   (a) **(2%)** Write a function `double-plus-one` that

   - consumes a list of numbers, and
   - produces a transformed list where each number is doubled, then 1 is added.

   For example:

   ```
   (check-expect (double-plus-one (cons 5 (cons -3 (cons 0 (cons 12
      empty)))))
       (cons 11 (cons -5 (cons 1 (cons 25 empty)))))
   ```

   (b) **(8%)** Write a function `symbol-sandwich` that

   - consumes a list of symbols, and

- produces a new list where each symbol is "sandwiched" between the symbol `'bread`
- if the consumed list is empty, `symbol-sandwich` produces `empty`.

For example:

```
(check-expect (symbol-sandwich empty) empty)
(check-expect (symbol-sandwich (cons 'ham (cons 'cheese (cons
   'lettuce empty))))
    (cons 'bread (cons 'ham (cons 'bread (cons 'cheese (cons 'bread
       (cons 'lettuce (cons 'bread empty)))))))))
(check-expect (symbol-sandwich (cons 'bread empty))
    (cons 'bread (cons 'bread (cons 'bread empty))))
```

(c) **(10%)** Write a function `shuffle-rock-paper-lizard-spock` that consumes a list of symbols and produces a new list where:

- every `'rock` is changed to `'paper`
- every `'paper` is changed to `'lizard`
- every `'lizard` is changed to `'rock`
- every `'spock` is removed

Note: Your function should handle any symbol (not just those listed above).

For example:

```
(check-expect
   (shuffle-rock-paper-lizard-spock (cons 'rock (cons 'paper (cons
      'lizard (cons 'spock (cons 'therock empty))))))
   (cons 'paper (cons 'lizard (cons 'rock (cons 'therock empty)))))
```

Read all requirements carefully and stick to them! Submit your code for the above problems in the file `listfun.rkt`.

3. **(40%)** Below are some more problems requiring recursion on lists. These problems are connected, where the last problem requires the solution to the first three parts:

(a) Write a function `negate` that consumes a list of numbers and produces a new list where each number has been negated.

For example:

```
(check-expect (negate (cons 100 (cons -10 (cons 0 empty))))
                      (cons -100 (cons 10 (cons 0 empty))))
```

(b) Write a function `count-down` that consumes a natural number and produces a list of natural numbers counting down from that number to 0 (inclusive).

For example:

```
(check-expect (count-down 9) (cons 9 (cons 8 (cons 7 (cons 6 (cons 5
   (cons 4 (cons 3 (cons 2 (cons 1 (cons 0 empty))))))))))
```

(c) Write a function `add-constant` that consumes a number and a list of numbers, and produces a new list where the given number has been added to each element of the list.

For example:

```
(check-expect (add-constant 1/2 (cons 10 (cons 4 empty)))
                                (cons 21/2 (cons 9/2 empty)))
```

(d) Write a function `count-up` that consumes a natural number and produces a list of natural numbers counting up from 0 to that number (inclusive). You **must** use **only** the three functions you wrote above (`negate`, `count-down`, and `add-constant`).

For example:

```
(check-expect (count-up 4)
              (cons 0 (cons 1 (cons 2 (cons 3 (cons 4 empty))))))
```

Again, stick to the rules. Submit your code for the above problems in the file `morelistfun.rkt`.

4. **(20%)** A young wizard is preparing the spells that they will memorize for a day of adventuring! In D&D, different spells consume different amounts of spell slots, and wizards have a limited number of spell slots available each day.

   Note: This is a simplified version of how D&D spell slots actually work!

   Here are the spells available and their spell slot costs:

Spell	Spell Slots Required
`'light`	0
`'mage-hand`	0
`'magic-missile`	1
`'shield`	1
`'fireball`	2
`'invisibility`	2
`'teleport`	3
`'meteor-swarm`	3

(a) Create data definitions for a:

- **Spell** - A spell is represented by one of the spell symbols from the table above.
- **Spellbook** - a list of spells the wizard will memorize.

(b) Write the predicate function `valid-spellbook?` that consumes two arguments:

- `available-slots`: the number of spell slots the wizard has.
- `spellbook`: a proposed spellbook with the spells the wizard will memorize.

The function should produce `true` if the wizard has enough spell slots to memorize all the spells in their spellbook, `false` otherwise.

For example, if a wizard has 3 spell slots and wants to memorize `'magic-missile`, `'shield`, and `'light`, they should be able to do so (costing $1+1+0=2$ spell slots). However, if they want to memorize `'fireball` and `'invisibility` with only 2 spell slots available, they cannot (it would cost $2+2=4$ spell slots).

```
(check-expect (valid-spellbook? 3 (cons 'magic-missile
                                        (cons 'shield
                                              (cons 'light empty))))
                                           true)
(check-expect (valid-spellbook? 2 (cons 'fireball
                                        (cons 'invisibility empty)))
                                           false)
(check-expect (valid-spellbook? 1 empty) true)
```

A wizard can memorize a spell more than once, as long as they have the available spell slots, like in this example:

```
(check-expect (valid-spellbook? 5 (cons 'fireball
                                        (cons 'light
                                              (cons 'light empty)))) true)
```

Submit your code in the file `spells.rkt`.

5. **(10%)** We can represent a polynomial in a variable x

$$a_0 + a_1 x + \cdots + a_n x^n$$

as a non-empty list `(cons a-0 (cons a-1 ... (cons a-n empty)...))`. Note the order of the coefficients. You may assume that `a-n` $\neq 0$, provided the list is non-empty. In addition, we interpret the `empty` list as the polynomial 0.

Write a function `eval-poly` that consumes a list of numbers (representing the coefficients of a polynomial) and a value for x, and produces the result of evaluating the given polynomial at the given value of x.

For example, `(eval-poly (cons 1.4 (cons 4 (cons 0 (cons 2 empty)))) 3)` should produce the value $1.4+4(3)+0(3^2)+2(3^3)=67.4$.

A *delta* of 0.0001 should be acceptable for `check-within` tests.

To earn marks for this question, you may **not** use the built-in `expt` operator, or any other exponentiation operation, and you may **not** use more than n multiplications and n additions, where n is the degree of the polynomial.

Hint: Look up *Horner's rule*.

Submit your code in the file `horner.rkt`.

This concludes the list of questions that you should submit. Don't forget to always check the basic test results after making a submission.
