Assignment Guidelines:

- Solutions to these questions are expected to follow the requirements of the Style Guide (https://www.student.cs.uwaterloo.ca/~cs115/coursenotes1/styleguide.pdf). This includes all relevant design recipe elements, proper use of constants, and proper use of helper functions.
- Submission details:
  - Solutions to these questions must be placed in files a10q1.rkt and a10q2.rkt, respectively, and must be completed in Racket.
  - All solutions must be submitted through MarkUs. Solutions will not be accepted through email.
  - Verify your basic test results using MarkUs to ensure that your files were submitted properly and are readable on MarkUs. Note, however, that passing the basic tests does not guarantee that you will pass all our correctness tests.
- Download the interface file from the course Web page to ensure that all function names are spelled correctly, and each function has the correct number and order of parameters.
- Restrictions:
  - You may only use the built-in functions and special forms introduced in the lecture slides up to and including the module covered by this assignment. A list of these functions can be found on the Assignments web page: https://www.student.cs.uwaterloo.ca/~cs115/#allowed
  - Read each question carefully to see if any additional restrictions apply.
  - Test data for correctness tests will always meet the stated assumptions for consumed values.
- The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.

Plagiarism: The following applies to all assignments in CS115.

All work in CS 115 is to be done individually. The penalty for plagiarism on assignments (first offense) is a mark of 0 on the affected question and 5 marks off the final grade, consistent with School of Computer Science policy. In addition, a letter detailing the offense is sent to the Associate Dean of Undergraduate Studies, meaning that subsequent offenses will carry more severe penalties, up to suspension or expulsion.

To avoid inadvertently incurring this penalty, you should discuss assignment issues with other students only in a very broad and high-level fashion. Do not take notes during such discussions, and avoid looking at anyone else's code, on screen or on paper. If you find yourself stuck, contact the ISA or instructor for help, instead of getting the solution from someone else. Do not consult other books, library materials, Internet sources, or solutions (yours or other people's) from other courses or other terms.

Read more course policies at: https://www.student.cs.uwaterloo.ca/~cs115/#policies

Language level: Intermediate Student with Lambda
Coverage: Module 9
Question 1: Polynomials

A polynomial is a mathematical function of a single unknown \( x \) that takes the form

\[
P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n
\]

for some sequence of numbers \( a_0, a_1, a_2, \ldots, a_n \). We can represent any polynomial in Racket as a \((\text{listof Num})\) containing the numbers \( a_i \) in the order given above, with the extra restriction \( a_n \neq 0 \). The polynomial \( P(x) = 0 \) is represented by the empty list.

(a) We can modify the binary arithmetic expressions of Module 09 so that they can represent polynomials (and other expressions). We need a way to represent the unknown \( x \) as a basic expression, and we don’t need the division operation. We end up with the following structure and data definitions:

```racket
(define-struct pnode (op arg1 arg2))
;; A Polynomial Expression Node (PNode) is a
;; (make-pnode (anyof '+ '- '*) PExp PExp)

;; A Polynomial expression (PExp) is one of:
;; * A Num
;; * 'x
;; * A PNode
```

Create a Racket function \( \text{eval/x} \) that consumes \( \text{ex} \), a \( \text{PExp} \), and \( \text{val} \), a \( \text{Num} \) that represents the current value of \( x \), and produces the result of evaluating the expression when \( \text{val} \) is used for \( x \). For example:

- \( \text{eval/x (make-pnode '+ 11 12) 159} \) => 23
- \( \text{eval/x 'x 66} \) => 66
- \( \text{eval/x (make-pnode '+ (make-pnode '* 'x -7) 3) -2} \) => 17
- \( \text{eval/x (make-pnode '* 'x (make-pnode '+ 'x 1)) 5} \) => 30

The solution can be expressed as a very small modification of the \( \text{eval} \) function shown in class.

(b) Create a Racket function \( \text{poly->exp} \) that consumes a polynomial \( \text{p} \) (represented as a \((\text{listof Num})\) with a nonzero last element) and produces a \( \text{PExp} \) that can be used to evaluate the polynomial for any value of \( x \). For example:

- \( \text{poly->exp empty} \) => 0
- \( \text{poly->exp (list 4 2)} \)
  => (make-pnode '+ 4 (make-pnode '* 2 'x))
- \( \text{poly->exp (list 3 0 1)} \)
  => (make-pnode '+ (make-pnode '* 'x 'x) 3)
Note that there are many possible PExprs that might all represent the same polynomial. For example, the polynomial (list 3 0 1) could just as easily be represented by

\[(\text{make-pnode } '+ 3 (\text{make-pnode } '* 'x 'x))\]

Don’t be alarmed if your results do not match the ones shown above exactly, as long as they’re mathematically equivalent. But this means that it doesn’t make sense to test that poly->exp produces a specific PExpr. Instead, we can test this function indirectly by using eval/x to make sure the PExpr evaluates to the expected result for a number of values of x. Here’s an example:

\[
\begin{align*}
\text{;; } p(x) &= 41 + x + x^2 \\
(\text{define test1 (poly->exp (list 41 1 1)))} \\
\text{;; Test p(0), p(1), p(2), p(3), and p(4) together} \\
(\text{check-expect (map (lambda (x) (eval/x test1 x))}
\quad (\text{list 0 1 2 3 4}))
\quad (\text{list 41 43 47 53 61}))
\end{align*}
\]

Hint: while there are numerous possible solutions to this problem, the most elegant approach is based on Horner’s method, and can be written using straightforward structural recursion on p.

Submit your solution in the file a10q1.rkt.

NOTE: It’s very interesting to work in the other direction too. As a mathematical and programming challenge, think about creating a function exp->poly that consumes a PExpr named ex and produces a (listof Num) that represents the polynomial calculated by ex.

Question 2: Followers

Consider a modified Binary Search Tree where the keys are strings and the values are lists of strings:

\[
\begin{align*}
(\text{define-struct fnode (key val left right))} \\
\text{;; An FNode is a (make-fnode Str (listof Str) FBST FBST)}
\end{align*}
\]

\[
\begin{align*}
\text{;; A Follower Binary Search Tree (FBST) is one of:} \\
\text{;; * empty} \\
\text{;; * (make-fnode Str (listof Str) FBST FBST)} \\
\text{;; Where:} \\
\text{;; * every key in left is less than key (using string<?)} \\
\text{;; * every key in right is greater than key}
\end{align*}
\]
Given a list of strings, we can use an **FBST** to track which strings follow which other strings in the list. Create a Racket function `calc-followers` that consumes `los`, a `(listof Str)`, and produces a **FBST**. Every string in `los` should appear as a key exactly once in the **FBST**. The value associated with the key should be a list of the strings in `los` that occur *immediately* after each occurrence of that key. The tree should be built in the order discussed in lectures (see "Creating a BST" on Slide 32 of Module 9); so, for example, the last string in the list will be the key at the root of the tree. The strings within each `val` field can be in any order, but the `val` fields must not contain any duplicates.

Examples:

```racket
(calc-followers empty)
=> empty
(calc-followers (list "Hi"))
=> (make-fnode "Hi" empty empty empty)
(calc-followers (list "a" "a" "a" "a" "a"))
=> (make-fnode "a" (list "a") empty empty)
(calc-followers (list "b" "a" "d"))
=> (make-fnode "d" empty
          (make-fnode "a" (list "d")
                      empty
                      (make-fnode "b" (list "a")
                                  empty
                                  empty))
               empty)
(calc-followers (list "m" "z" "m" "z" "a" "m"))
=> (make-fnode "m" (list "z")
           (make-fnode "a" (list "m") empty empty)
           (make-fnode "z" (list "a" "m")
                       empty empty))
```

Submit your solution in the file `a10q2.rkt`. 