Name:

CS 443 — Lisp to Forth Compiler

Mattox Beckman

Objectives

You came to this class to learn how to write a compiler. Let's write one today and be done with it. At least for today.

- Consider some issues in representing and manipulating expressions in multiple languages simultaneously.
- Implement translator to convert Lisp style expressions to Forth style.

Given Code

Here is some Clojure code to take a sequence of Forth operations and execute them.

```
(ns forth
  (require [clojure.edn :as edn]))

(defn execute-symbol
  "Use `symbol` to perform an operation.  Returns a vector
  of the new input and stack, intended to be applied to `eval-forth`."
  [symbol input stack]
  (let [a (first stack)
        b (second stack)
        nu-stack (rest (rest stack))
        nu-input (rest input)]
    (case symbol
      + [nu-input (cons (+ a b) nu-stack)]
      - [nu-input (cons (- a b) nu-stack)]
      * [nu-input (cons (* a b) nu-stack)])))

(defn eval-forth
  ([input] (eval-forth input []))
  ([input stack]
   ;; When the input is done the answer is TOS
   (if (empty? input)
     stack
     ;; Otherwise, read the first symbol...
     (let [first-symbol (first input)]
       (if (= (class first-symbol) clojure.lang.Symbol)
         ;; Execute symbols
         (apply eval-forth (execute-symbol first-symbol input stack))
         ;; Push everthing else, hoping they're numerics.
         (eval-forth (rest input) (cons first-symbol stack)))))))
```

Here is a sample run:

```
forth> (eval-forth '[10 32 +])
(42)
```

Your Work

- Review the given code with a classmate to verify understanding and correctness.
- Write a compiler that takes LISP expressions and converts them into FORTH expressions. Call it `lisp-to-forth`.