CS 152 / SE 152
Programming Language Paradigms

Spring Semester 2014

Department of Computer Science
San Jose State University
Prof. Ron Mak

Assignment #5

Assigned: Wednesday, April 2
Due: Wednesday, April 16 at 11:59 pm
Team assignment, 100 points max

Scheme parser and scanner

Use Java to write a parser and a scanner for Scheme.

Your scanner should recognize Scheme tokens. It should print each source input line after it reads the line. For each token, your scanner should print the token string and the token type (symbol, number, keyword, etc.), one token per output line. It should ignore comments.

Symbol table

The parser should enter each symbol into a symbol table. It should not enter keywords.. For this assignment, the attributes of each symbol table entry can be null. Also for this assignment, you won't deal with scope, so you'll need only one symbol table.

What is a symbol? It's anything that can be bound to a value at run time. It's anything that returns **`#t`** when the predicate **`symbol?`** is applied to it:

```
(symbol? 'alpha) ➔ #t
(symbol? '+)     ➔ #t
(symbol? #\r)    ➔ #f
(symbol? 132)    ➔ #f
(symbol? #t)     ➔ #f
```

Therefore, enter **`alpha`** and other identifiers into the symbol table. Enter **`+`** into the symbol table. But do not enter characters, numbers, or boolean values.

Special symbols

Special symbols are the punctuation marks of Scheme. They are not symbols and therefore should not go into the symbol table. Special symbols include:

```
( ) [ ] { } ; , . " ' # \
```

Numbers

Your scanner should recognize both integer and real numbers. It can simply call either one a “number”. Numbers do not go into the symbol table.

Keywords

Your scanner should recognize Scheme keywords. They do not go into the symbol table.

```
and begin begin0 break-var case cond cycle define
  delay delay-list-cons do else extend-syntax for
freeze if lambda let letrec let* macro object-maker
   or quote repeat safe-letrec set! stream-cons
            variable-case while wrap
```

Lists

Your parser should translate lists by building the intermediate code (binary parse trees).

After your parser has completed parsing a top-level list and built the intermediate code and the symbol table, your back end should take over.

Backend

Your back end should walk the binary parse tree and print the list with the proper parentheses. The printed list does not need to be “pretty” but it must be correct. You can start each sublist on a new line after an indent.

For example:

```
(define proc
  (lambda (a b)
    (let ((sum (+ a (func b))))
      sum)))
```

can print as

```
(define proc
  (lambda
    (a b)
      (let
        (
          (sum
            (+ a
              (func b)))) sum)))
```

Your back end should also print the contents of the symbol table in alphabetical order. (Hint: Use a Java **`TreeMap`** instead of a **`Hashtable`** or **`Hashmap`** for the symbol table.)

Clear the intermediate code and the symbol table before parsing the next top-level list.

Program structure

Ensure that your code is properly structured by putting your classes into three packages: **`frontend`**, **`intermediate`**, and **`backend`**.

Input file

Parse the two top-level lists in following file **`input.lisp`**:

```
; Find the derivative of polynomial poly with repect to variable var.
; The polynomial must be in canonical infix form.
(define deriv
  (lambda (poly var)
    (let* ((terms (terminize poly)) ; "terminize" the polynomial
           (deriv-term              ; local procedure deriv-term
             (lambda (term)
               (cond
                 ((null? term) '())
                 ((not (member? var term)) '(0))           ; deriv = 0
                 ((not (member? '^ term)) (upto var term)) ; deriv = coeff
                 (else (deriv-term-expo term var))         ; handle exponent
             )))
           (diff (map deriv-term terms)))   ; map deriv-term over the terms
      (remove-trailing-plus (polyize diff)) ; finalize the answer
)))

; Convert an infix polynomial into a list of sublists,
; where each sublist is a term.
(define terminize
  (lambda (poly)
    (cond
      ((null? poly) '())
      (else (cons (upto '+ poly) (terminize (after '+ poly))))
)))
```

You may assume that the input file does not contain any syntax errors.

Your program can read the input file as standard input, or a command-line parameter can be the file path.

What to turn in

Each team turns in one zip file containing

- The source directory containing all your Java source files.
- A text file of the output from your program.

Name your zip file after your team, such as **`SuperProgrammers.zip`**.
Email to: ron.mak@sjsu.edu. Your subject line should say

`CS 152 Assignment #5` ***team name***

CC all the team members so that I can do a "Reply all" when I send out your score.

This is a team assignment. Each member of the team will receive the same score.

Do not wait until the last minute to start this assignment!