CSCI3136
Assignment 6

Instructor: Alex Brodsky

Due: 3:00pm, Friday, March 10, 2014

Consider the grammar in Figure 1 where the terminal `int` denotes an integer and the terminal `id`

$$
\begin{array}{rcl}
S & \rightarrow & Atoms \\
Atoms & \rightarrow & \epsilon \\
Atoms & \rightarrow & Atom\ Atoms \\
Atom & \rightarrow & \texttt{'}\ Atom \\
Atom & \rightarrow & List \\
Atom & \rightarrow & \texttt{id} \\
Atom & \rightarrow & \texttt{int} \\
List & \rightarrow & (\ Atoms\ )
\end{array}
$$

Figure 1: A simplified grammar for Scheme.

denotes an identifier. The only other terminals in this grammar are the quote `'`, the left parenthesis `(`, and the right parenthesis `)`.

Scheme, a derivative of Lisp, is a list based language where all programs are represented by lists. For example, a simple Scheme program

```
( + 1 2 )
```

adds two numbers together and prints out the result. Note: a scheme interpreter, `mzscheme`, is available on `bluenose`. Feel free to give it a try. A program, which is zero or more atoms is executed by *evaluating* each of the atoms. An atom is either an integer, an identifier, or a list. The evaluation of an integer is the integer, and the evaluation of an identifier, for now, is simply the identifier. The evaluation of a quoted atom is simply the atom itself. E.g., the evaluation of `' ( 1 2 3 )` is `( 1 2 3 )`.

A list is evaluated by evaluating the first atom of the list and treating the result as a function. Then each of the remaining atoms of the list is evaluated, and passed as parameters to the function. For example, the evaluation of `( + 1 2 )`, treats + as a function, and passes 1 and 2 to the function. Applying + to 1 and 2 yields the result 3. Whereas, the evaluation of `( + 1 ( * 2 3 ) )`, evaluates + and 1 as before, but then evaluates `( * 2 3 )`, yielding 6, which is then passed to the + function, which yields the result of 7.

1. **[10 marks]** As a Scheme program is being parsed, it is useful to generate a linked list representation of the program, where the nodes in the list are atoms, which can be integers, identifiers, or lists themselves. Suppose you were provided with the operations

   $L \leftarrow list()$ which creates and returns a new empty linked list.
   $L \leftarrow prepend(a, L)$ which creates a new node, stores atom a in it, prepends it to list L and returns the modified list.

   Give a synthetic attribute grammar (S-Grammar) for the grammar above that will construct a linked-list representation of a program as it is being parsed using the provided grammar.

2. **[10 marks]** A Scheme program that does not define its own functions can easily be evaluated as it is parsed. Assume you are provided with the function

   $b \leftarrow eval(a)$ which takes an atom as a parameter, evaluates it, and returns the result of the evaluation, which is also an atom. If an error occurred during evaluation, an error is raised.

   Extend the synthetic attribute grammar from Question 1 to an inherited attribute grammar (L-grammar) that evaluates the program as it is being parsed.. I.e., after an atom is parsed, it should be evaluated. Note: you will use the linked-lists created by the attribute grammar from Question 1, but instead of the original atom a being stored, its evaluation $eval(a)$ should be stored. Recall, that a quoted atom should *not* be evaluated, which is why an L-grammar, instead of an S-grammar is needed in this case.

3. **[30 marks]** A recursive descent parser, called `uscm.py`, (written in Python) that parses programs using the above context-free grammar is provided as part of this assignment. Using this parser, or your own version of the parser (implemented in a language of your choice) implement the attribute grammar that you created in Question 2. Your parser will need to build the linked list representation to evaluate the program as it is being parsed. Note: it is impossible to evaluate the program without a list representation. The output of your augmented parser should be the evaluation of the program. The *eval*() function performs the following operation:

   - If the atom is an integer or an id, the atom is returned as is.
   - If the atom is a quoted atom, the quote is stripped off, and the remaining atom is returned. Note, the remaining atom is *not* evaluated.
   - If the atom is a list, *eval*()
     - Checks that the list is not empty.
     - Checks that the first item in the list is an id that denotes a valid operation.
     - Checks whether the remaining atoms in the list, which become arguments to the operation, are what the operation expects. E.g., the + operation expects one or more integers.
     - Performs the aforementioned operation.
     - Returns an atom, representing the result of the operation.
     - If the operation is not valid or cannot be applied to the arguments, the message `Evaluation Error` is outputted and the parser should exit.

The *eval*() function in your parser should recognize the following operations:

Identifier	Arguments	Operation
+	1+ integers	Sum the integers, e.g., `( + 1 2 3 )` evaluates to `6`
-	1+ integers	Subtract the remaining integers from the first, e.g., `( - 1 2 3 )` evaluates to `-4`
***	1+ integers	Multiply the integers, e.g., `( * 2 3 4 )` evaluates to `24`
/	1+ integers	Divide the first integer by each of the remaining integers, e.g., `( / 12 2 3 )` evaluates to `2`
car	nonempty list	return the head of the list e.g., `( car ' ( a b c ) )` evaluates to `a`
cdr	nonempty list	remove the head and return the rest of the list, e.g., `( cdr ' ( a b c ) )` evaluates to `( b c )`
list	0+ atoms	return a list of the atoms e.g., `( list a b c )` evaluates to `( a b c )`
cons	atom, list	prepend atom to list e.g., `( cons a ' ( b c ) )` evaluates to `( a b c )`

The parser should print out the result of the evaluation in the same format as the input. E.g., the evaluation of `( + 1 2 3 ) ' ( 1 2 3 )` would yield `6 ( 1 2 3 )`

You may implement your parser in any language that you wish, but it must be runnable on the `bluenose` server. The parser should take input from `stdin` and output to `stdout`. The input to the parser will comprise a stream of tokens, with the tokens being separated by white-space. Each token will be one of quote `'`, left parenthesis `(`, right parenthesis `)`, nonnegative integer, or identifier.

A test solution (in the form of a Java class file: `SchemeEval.class`) is provided for you to test your new parser. The sample solution takes about 100 lines of Java code.

Since the choice of language is up to you, you must provide a standard script called `runme.sh` to run your parser. See previous assignment for an example.

To submit this part of the assignment please use SVN as well as in hard-copy. On the hard-copy please note the login id of the person who submitted the programming portion of the assignment for the group. Please see the Resource or Assignment page of the course website for how to use SVN. Remember, you need to include a script called `runme.sh` that will let the marker run your code.

4. [**Bonus 5 marks**] In Scheme the `let` operation allows the programmer to define variables, set them to a given value, and use them within the `let` evaluation. For example,

```
( let ( ( x 5 ) ( y 7 ) ) ( * x y ) )
```

evaluates to `35`. Extend the L-grammar from Question 2 to implement this functionality. You may assume that you have access to a map (hashtable) that has the standard operations *put* and *get*.

CSCI3136: Assignment 6

Winter 2014

Student Name	Login ID	Student Number	Student Signature

	Mark	
Question 1		/10
Question 2		/10
Question 3		/30
Functionality	/15	
Structure	/15	
Question 4 (Bonus)		/5
Total		**/50**

Comments:

Assignments are due by 3:00pm on the due date before class and must include this cover page. Assignment *must* be submitted into the assignment boxes on the second floor of the Goldberg CS Building (by the elevators).

Plagiarism in assignment answers will not be tolerated. By submitting their answers to this assignment, the authors named above declare that its content is their original work and that they did not use any sources for its preparation other than the class notes, the textbook, and ones explicitly acknowledged in the answers. Any suspected act of plagiarism will be reported to the Facultys Academic Integrity Officer and possibly to the Senate Discipline Committee. The penalty for academic dishonesty may range from failing the course to expulsion from the university, in accordance with Dalhousie Universitys regulations regarding academic integrity.