CSCI3136
Assignment 9

Instructor: Alex Brodsky

Due: 9:00am, Wednesday, July 30, 2019

In this assignment we will further expand the Splat Evaluator to handle closures. We will add the several features to our evaluator including, `set` and `lambda` expressions, the ability to call functions, and three built in functions to work with lists.

The `set` expression is used to modify variables in the current scope. The syntax for `set` is

`set` *id* = *expression*

where the *id* denotes a bound identifier that is in scope and *expression* is the expression to be evaluated whose result is then bound to the *id*. The result of the `set` expression is the previous value that *id* was bound to. For example, the expressions below yield 42, 42, and 37.

```
def x = 42
set x = 37
x
```

The `lambda` operator allows us to create closures (functions with a reference environment) that can then be invoked. I.e., just like you can define and use methods in Java or functions in C, we can define and use closures in Splat. The syntax for `lambda` comprises two parts:

`lambda (` *param1, param2, ...* `)` { *expression1 expression2 ...* }

where the first part is a comma-separated list of zero or more parameters, which are identifiers, and the second part is the body of the function, comprising one or more expressions. The expressions are evaluated when the closure is invoked, and the result of the evaluation is the result of the last expression in the body. Typically, `lambda` expressions are used in conjunction with `def` or `let`.

In Example 1 (Figure 1), the first expression defines a symbol *area* and binds it to a closure that computes the area given the length (l) and width (w). The second expression defines the symbol *square* and binds it to a closure that computes the square area, given a width, by invoking the *area* closure. The next two expressions simply invoke the *area* and *square* closures. The fifth expression binds the symbol *execute* to a closure that takes two parameters: a closure (c) and a value (a), and evaluates the closure, passing the argument a to it twice. The last expression is an invocation of the closure bound to *execute*. The evaluation of these expressions is: `2142 49 2601`.

In Example 2, the expression comprises a `let`, which binds x to 42 and then evaluates a second `let` expression, which binds *foo* to a closure that divides x by the passed argument (a), then evaluates three expression: a third `let` expression, which binds x to 21 and then invokes *foo* with argument 3; a `set` expression, which changes the value of x to 14; and another invocation of `foo` with argument 7. The evaluation of this expressions is 2.

Example 1

```
def area = lambda (l, w) {
  l * w
}

def square = lambda (w) {
  area(w, w)
}

area(51, 42)
square(7)

def execute = lambda (c, a) {
  c(a, a)
}

execute(area, 51)
```

Example 2

```
let x = 42 {
  let foo = lambda (a) {
              x / a
            }
  {
    let x = 21 {
      foo(3)
    }
    set x = 14
    foo(7)
  }
}
```

Figure 1: Examples of `lambda`, `set`, and function calls.

Note: the output for the evaluation of a lambda expression is just `<lambda>`, i.e, The output of the following program is `<lambda>`.

```
lambda ( ) {
  42
}
```

1. [**10 marks**] Starting with either your own solution to Assignment 8 or the provided solution (`splat.3.py`), add the following productions to your parser/evaluator:

$$
\begin{aligned}
\mathit{EXPR} &\rightarrow \mathit{SET} \\
&\rightarrow \mathit{LAMBDA} \\
\mathit{SET} &\rightarrow \text{'}\texttt{set}\text{'}\ \mathit{SYMBOL}\ \text{'='}\ \mathit{EXPR} \\
\mathit{LAMBDA} &\rightarrow \text{'}\texttt{lambda}\text{'}\ \text{'('}\ \mathit{PARAMS}\ \text{')'}\ \mathit{BODY} \\
\mathit{PARAMS} &\rightarrow \epsilon \\
&\rightarrow \mathit{SYMBOL}\ \mathit{P\_LIST} \\
\mathit{P\_LIST} &\rightarrow \epsilon \\
&\rightarrow \text{','}\ \mathit{SYMBOL}\ \mathit{P\_LIST} \\
\mathit{CALL} &\rightarrow \epsilon \\
&\rightarrow \text{'('}\ \mathit{ARGS}\ \text{')'}
\end{aligned}
$$

and replace production

$$\mathit{VALUE} \rightarrow \mathit{SYMBOL}$$

with

$$\mathit{VALUE} \rightarrow \mathit{SYMBOL}\ \mathit{CALL}$$

2. [**30 marks**] Implement the evaluation of the `set`, `lambda`, and function call expressions in your evaluator. **Your implementation should perform lexical scoping and deep binding.**

   Suggestions:

   - To implement `set`, you will need to add functionality to your reference environment to update bindings. The provided solution already implements this.
   - Do not evaluate the body of closure until it is actually called.
   - When a closure is called, link in a new frame into the reference environment to hold the parameter/argument bindings. Once the call completes, this frame is no longer needed.

3. [**10 marks**] Add three built-in functions into your Splat evaluator: `prepend()`, `head()` and `tail()`. The `prepend()` function takes a value and a list and prepends the value to the list, and returns the new list. The latter two functions take a single argument, which is a list. The `head()` function returns the first item in the list. The `tail()` function returns removes the head and returns the remainder of the list.

Input	Output
`def nums = [1, 2, 3, 4]`	`[1, 2, 3, 4]`
`head(nums)`	`1`
`tail(nums)`	`[2, 3, 4]`
`prepend(0, nums)`	`[0, 2, 3, 4]`

Figure 2: Examples of `prepend()`, `head()` and `tail()`.

4. [**Bonus 5 marks**] For bonus marks we want to add the `if` expression to our evaluator. The `if` expression has the form:

   `if` *condition body*
   `elseif` *condition body*
   ...
   `else` *body*

   which consists of one required and two optional parts. The required part consists of an `if` followed by a condition expression and a body. There are zero or more `elseif` parts which consists of an `elseif` followed by a condition expression and a body. Lastly, there is an optional `else` part, which consists of an `else` followed by a body. Add the following productions to your Splat evaluator.

$$
\begin{aligned}
\mathit{EXPR} &\rightarrow \mathit{IF} \\
\mathit{IF} &\rightarrow \text{'}\texttt{if}\text{'}\ \mathit{EXPR}\ \mathit{BODY}\ \mathit{ELSEIF}\ \mathit{ELSE} \\
\mathit{ELSEIF} &\rightarrow \epsilon \\
&\rightarrow \text{'}\texttt{elseif}\text{'}\ \mathit{EXPR}\ \mathit{BODY}\ \mathit{ELSEIF} \\
\mathit{ELSE} &\rightarrow \epsilon \\
&\rightarrow \text{'}\texttt{else}\text{'}\ \mathit{BODY}
\end{aligned}
$$

5. [**Bonus 20 marks**] For bonus marks implement the evaluation of `if` expressions.

   The `if` expression is evaluated by first evaluating the condition after the `if`. If this condition evaluates to `true`, the result of the body that follows the condition is evaluated and yields the overall evaluation. Otherwise, if an `elseif` follows, if the condition of the `elseif` evaluates to `true`, the body of the `elseif` is evaluated and yields the result. Otherwise, the next `elseif` is evaluated, and so on. If all conditions evaluate to `false` and there is an `else` part, the body of the `else` is evaluated and yields the result. If there is no `else` and all conditions evaluate to `false` the result of the whole `if` expression is `false`.

Input	Output

`def a = 37`
`def b = 42`
 	
`if a < b {`
`  b - a`
`}`
 	
`if a > b {`
`  a - b`
`}`
 	
`if a > b {`
`  a - b`
`} elseif b > a {`
`  b - a`
`}`
 	
`if a > b {`
`  a - b`
`} elseif b = a {`
`  0`
`}`
 	
`if a > b {`
`  a - b`
`} elseif b = a {`
`  0`
`} else {`
`  b - a`
`}`
	`37`
	`42`
	`5`
	`false`
	`5`
	`false`
	`5`

Figure 3: Examples of `if` expressions.

A set of test cases is provided for you to test your evaluator. Two test scripts `test.sh` and `bonus.sh` are provided to run the tests. Since the choice of language is up to you, you must provide a standard script called `runme.sh` to run your interpreter, just like in the previous assignment.

To submit this part of the assignment please use Brightspace.

CSCI3136: Assignment 9

Summer 2019

Student Name	Login ID	Student Number	Student Signature

	Mark	
Question 1		/10
Question 1		/30
Functionality	/15	
Structure	/15	
Question 3		/10
Question 4 Bonus		/5
Question 5 Bonus		/20
Functionality	/10	
Structure	/10	
Total		**/50**

Comments:

Assignments are due by 9:00am on the due date. Assignments *must* be submitted electronically via Brightspace. 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 University's regulations regarding academic integrity.