CSCI3136
Assignment 7

Instructor: Alex Brodsky

Due: 9:00am, Friday, July 12, 2019

Consider the grammar for the Splat programming language in Figure 1, where the terminal `int` denotes an integer, the terminal `symbol`, and the terminal `string` denotes a quoted string.

$$
\begin{array}{rcl}
S & \rightarrow & \epsilon \\
 & \rightarrow & E\_LIST \\
E\_LIST & \rightarrow & EXPR\ E\_TAIL \\
E\_TAIL & \rightarrow & \epsilon \\
 & \rightarrow & E\_LIST \\
EXPR & \rightarrow & S\_EXPR \\
S\_EXPR & \rightarrow & ANDOP\ S\_TAIL \\
S\_TAIL & \rightarrow & \epsilon \\
 & \rightarrow & '|'\ S\_EXPR \\
ANDOP & \rightarrow & RELOP\ A\_TAIL \\
A\_TAIL & \rightarrow & \epsilon \\
 & \rightarrow & '\&'\ ANDOP \\
RELOP & \rightarrow & TERM\ R\_TAIL \\
R\_TAIL & \rightarrow & \epsilon \\
 & \rightarrow & '<'\ RELOP \\
 & \rightarrow & '>'\ RELOP \\
 & \rightarrow & '='\ RELOP \\
 & \rightarrow & '\#'\ RELOP \\
TERM & \rightarrow & FACT\ T\_TAIL \\
T\_TAIL & \rightarrow & \epsilon \\
 & \rightarrow & '+'\ TERM \\
 & \rightarrow & '-'\ TERM
\end{array}
\qquad
\begin{array}{rcl}
FACT & \rightarrow & VALUE\ F\_TAIL \\
F\_TAIL & \rightarrow & \epsilon \\
 & \rightarrow & '*'\ FACT \\
 & \rightarrow & '/'\ FACT \\
VALUE & \rightarrow & LIST \\
 & \rightarrow & UNARY \\
 & \rightarrow & LITERAL \\
 & \rightarrow & '('\ EXPR\ ')' \\
 & \rightarrow & SYMBOL \\
LIST & \rightarrow & '['\ ARGS\ ']' \\
UNARY & \rightarrow & '-'\ VALUE \\
 & \rightarrow & '!'\ VALUE \\
ARGS & \rightarrow & \epsilon \\
 & \rightarrow & EXPR\ A\_LIST \\
A\_LIST & \rightarrow & \epsilon \\
 & \rightarrow & ','\ EXPR\ A\_LIST \\
SYMBOL & \rightarrow & symbol \\
LITERAL & \rightarrow & integer \\
 & \rightarrow & string \\
 & \rightarrow & 'true' \\
 & \rightarrow & 'false' \\
 & \rightarrow & 'nil'
\end{array}
$$

Figure 1: A grammar for the Splat language.

Splat, is a functional language where all programs consist of one or more expressions. For now, an expression can be one of:

arithmetic expressions such as `1 + 2 * 3`
boolean expressions such as `1 < 2 & 3 > 2`
string expressions such as `"Hello " + "world!"`
list expressions such as `["a", "b", "c"]` and `[3, 1, 2] + [4, 5, 6]`

Expressions in Splat are evaluated in the standard way. For example,

`1 + 2 * 3` evaluates to 7
`1 < 2 & 3 > 2` evaluates to *true*
`"Hello " + "world!"` evaluates to `"Hello world"`
`[3, 1, 2] + [4, 5, 6]` evaluates to `[3, 1, 2, 4, 5, 6]`

Formally, an expression is either a literal

integer such as 42,
boolean such as *true*
string such as *"Hello"*
list of constants such as `[1,2,3]`

or a composition of subexpressions and operators such as above. The expression is evaluated by applying the operators just like in most programming languages. The operators in Splat are:

Operator	Description
`+`	add two integers concatenate two strings concatenate two lists
`-`	subtract one integer expression from another integer expression, e.g., `3 - 2` negate an integer expression, e.g., `-42`
`*, /`	multiply and divide integer expressions
`<, >, =, #`	compare two expressions: less than, greater than, equals, not equals
`&, |, !`	combine boolean expressions with *and*, *or*, and *not*

Table 1: Operators in Splat.

All expressions are evaluated in the same way as in most other languages such as Java or Python. The *list* expression is evaluated by evaluating each of the elements of the list. E.g., the evaluation of `[1+2, 3+4, "Hello"]` is `[3, 7 "Hello"]`.

[50 marks] A recursive descent parser, called `splat.1.py`, (written in Python) that parses Splat programs using the context-free grammar specified in Assignment 5 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 an evaluator that evaluates parsed expressions.

Note: The provided solution automatically builds an abstract syntax tree (AST) as it parses the input. Each node in the AST contains a field called *atoms* which is the list of its children. The AST differs from the parse tree in that all the right-recursive parts of the parse tree are flattened. For example, The parse tree, rooted at *TERM* for the expression $1 + 2 + 3$ is shown in Figure 2, where as the corresponding AST is shown in Figure 3.

To evaluate the expressions, your evaluator will need either a parse tree, or an abstract syntax tree. Note: it is impossible to evaluate the expressions without a tree representation. The output of your evaluator should evaluate the expressions and print out the results.

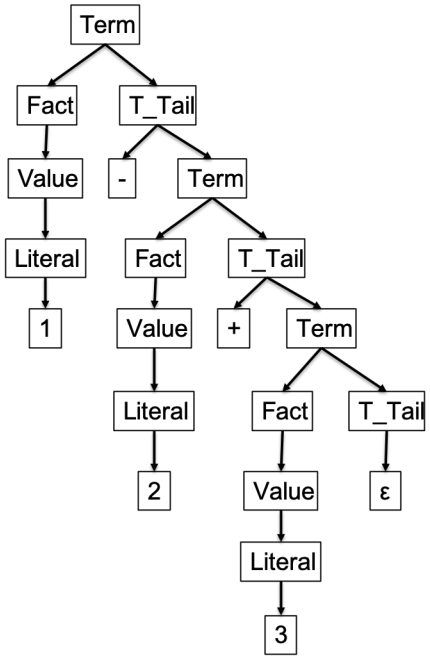

Figure 2: Parse tree for $1 - 2 + 3$

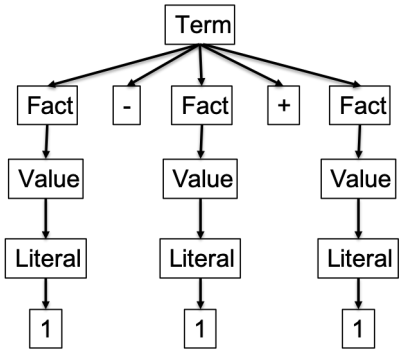

Figure 3: Abstract syntax tree for $1 - 2 + 3$

The following evaluation rules apply:

- The evaluation of a literal or a symbol is just the literal or symbol.
- The evaluation of a list is a list with each element evaluated.
- The evaluation of an expression is the application of the operator to the subexpressions, e.g., $1 - 2 + 3$ should result in 2 being the output.
- If an operator cannot be applied to a subexpression, the message `Evaluation Error` is outputted and the evaluator should exit.

All operators and their precedence in the grammar in Figure 1 should be implemented.

The evaluator should print out the result of each of the the top-level evaluations (one per line), in the same format as the input. See the provided tests for examples.

You may implement your evaluator in any language that you wish, but it must be runnable on the `bluenose` server. The evaluator should take input from `stdin` and output to `stdout`.

A set of test cases is provided for you to test your evaluator. A test script `test.sh` is 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 evaluator. For example the script for a Python solution looks like:

```
#!/bin/sh
# Command to run your program
python3 splat.2.py
```

To submit this part of the assignment please use Brightspace. Remember, you need to include a script called `runme.sh` that will let the marker run your code.

CSCI3136: Assignment 7

Summer 2019

Student Name	Login ID	Student Number	Student Signature

	Mark
Functionality	/25
Structure	/25
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.