CSE 307 – Principles of Programming Languages
Fall, 2019
Homework Assignment 03
Building a Programming Language: Expressions

Assigned: 10/17/2019
Due: 10/31/2019, at 11:55 PM
Total Points: 50

Over the course of the next three programming assignments (HW03, 04, and 05) we will be implementing a programming language. We will call this language, SBML.

For this assignment we will be constructing an expression evaluator. The evaluator will take an expression as input, evaluate the expression, and then print the result to standard output.

The code written for this assignment will be reused in future assignments. Please keep that in mind when working. Keep your code clean, neat, and modular so that it will be easy to update and extend in future assignments.

We will use the parser generator PLY – Python Lex and Yacc – to specify and implement the SBML language.

Installation Instructions for PLY:

1. Using PIP:

   > pip3 install ply --user

   or

   > python3 -m pip install ply --user

2. Manual Installation Using setuptools

   -Download the ply archive; https://www.dabeaz.com/ply/

   -Unzip the archive.

   -From command-line, navigate to the ply directory.
   -Run the command: python3 setup.py install --user

SBML description:

SBML Datatypes:

Numbers: Integers and Reals – implement as Python integers and floats.

Booleans: True and False – implement as Python Booleans.

Strings: Sequences of characters enclosed within matching single or double quotes in a single line. Strings should be implemented using the equivalent Python String type.

List: Finite, ordered sequence of elements separated by commas and enclosed within matching square brackets. Elements of the list need not be of the same type. Implement as Python list.

Tuple: Finite, ordered sequence of elements separated by commas and enclosed within matching parentheses. Elements of the tuple need not be of the same type.

SBML Literal Representation of Data Types:

Integer: Positive (no sign) or negative (unary -) whole numbers in base-10 representation (decimal representation). An integer literal is one or more digits, 0-9.
Examples: 57, -18, 235

Real: A real value is represented by 0 or more digits (0-9), followed by a decimal point, ".", followed by 0 or more digits (0-9), except that a decimal point by itself with no leading or trailing digit is not a real.
Examples: 3.14159, 0.7, .892, 32787.

A real can also contain exponents as in scientific notation. In this case, a real value, as defined above, is followed by an "e" character and then a positive or negative integer, as defined above.
Examples: 6.02e-23, 17.0e4

Boolean: True, False (just as in Python)

String: A string literal begins with a single or double quote, followed by zero or more non-quote characters, and ends with a matching quote. The value of the string literal does not include the starting and ending quotes.
Examples: "Hello World!", "867-5309"

List: A list literal is composed by a left square bracket, followed by a comma-separated sequence of zero or more expressions, followed by a right square bracket.
Examples: ["a", "b"], [1, 2], [307, "307", 304+3]

SBML Operators:

Operators are listed in order of precedence from highest to lowest. The exponentiation (**) operator and the cons (::) operator are both right-associative. All other associative operators are left-associative.

01. (expression) – A parenthesized expression

02. (expression1, expression2, …) – Tuple constructor

03. #i(tuple) – returns the argument at index *i* in the tuple. Indices start at 1 as in SML.

04. a[b] – Indexing Operation. *b* can be any expression.

05. a ** b – Exponentiation. base *a* raised to the power *b*. right associative: 2**3**4 == 2**(3**4)

06. a * b – Multiplication. Overloaded for integers and reals.

07. a / b – Division. Overloaded for integers and reals, but result is always a real value.

08. a div b – Integer Division. Returns just the quotient.

09. a mod b – Modulus. Divides *a* by *b* and returns just the remainder.

10. a + b – Addition. Overloaded for integers, reals, strings, and lists.

11. a – b – Subtraction. Overloaded for integers and reals.

12. a in b – Membership. Evaluates to True if it finds the value of *a* inside the string or list represented by *b*.

13. a::b – Cons. Adds operand *a* to the front of the list referred to by operand *b*.

14. not a – Boolean negation.

15. a andalso b – Boolean Conjunction (AND)

16. a orelse b – Boolean Disjunction (OR)

17. a < b – Less than. Comparison.

18. a <= b – Less than or equal to. Comparison.

19. a == b – Equal to. Comparison.

20. a <> b – Not equal to. Comparison.

21. a >= b – Greater than or equal to. Comparison.

22. a > b – Greater than. Comparison.

SMBL Operator Semantics:

Indexing: Operand *a* must be either a string or a list. Operand *b* must be an integer. If *a* is a string, then return the *b*-th character as a string. If *a* is a list, then return the *b*-th element as an instance of whatever type it is. The index is 0-based. If the index is out of bounds, then this is a semantic error.

Addition: Operands must either both be numbers, or both be strings, or both be lists. If they are integers or reals, then addition with standard (Python) semantics is performed. If *a* and *b* are both strings, then string concatenation is performed. If *a* and *b* are both lists, then list concatenation is performed.

Subtraction: Operands must both be integers or reals. Performed using standard subtraction semantics.

Multiplication: Operands must both be integers or reals. Performed using standard multiplication semantics.

Division: Operands must both be integers or reals. Operand *b* cannot be 0. Performed using standard division semantics.

Booleans: Operands for Boolean operations (not, andalso, orelse) must be Boolean values.

Comparisons: Operands must either both be numbers or both be strings. Comparison of numbers (integers and strings) should follow standard semantics. Comparison of strings should follow the Python semantics. Returns True if comparison is true, and False if comparison is False.

Program Behavior:

-Your program will be called with a single command-line argument. This argument will name an input file. The input file will contain a list of expressions, one per line. Each expression will be terminated with a semi-colon.

Like So: python3 sbml.py <input_file_name.txt>

-Your program should process each expression one-by-one, and produce one of the following three outputs, printed to STDOUT:

1. If the line contains a syntax error, then print: "SYNTAX ERROR".

2. If the line contains a semantic error, then print: "SEMANTIC ERROR".

3. Otherwise, evaluate the expression and print the result.

Example:

Input File:	Output:
1 - 2 + 3;	2
1 2;	SYNTAX ERROR
42 + "Red";	SEMANTIC ERROR
1 - (2 + 3);	-4
"Hello" + " " + "SBML.";	Hello SBML.
[[1], 2, 3][0][0] + 40;	41

Submission Instructions:

1. Please name your program sbml.py
2. Please include your name and student id as comments at the top of your program file.
3. Please collect and submit your program file as a compressed zip file (I know this seems silly for one file, but it helps the graders).
4. The title of the compressed file should be:
   cse307_hw03_LastNameFirstName.zip
5. This is an individual assignment. Any collaboration on writing your programs will be treated as a violation of academic integrity.