Assignments

Homework 4 (expression grammar)

Due: Friday, April 13, 11:59PM

We'll be spending the next few assignments implementing a programming language, which will be called stonyBrookScript. For this assignment, we will be implementing an expression evaluator. This takes an expression as input, evaluates it to a result, and prints the result to standard output.
We will use the parser generator Ply: Python Lex and Yacc
1. Download Python Lex and Yacc ply from: http://www.dabeaz.com/ply/
2. Download and unzip.
3. Install with: python setup.py install
4. Read documentation and try examples.
5. Use mailing list to ask questions.

Our Language:

* Data Types: our language has three data types:
** Numbers: Integer and Real = should be implemented using the equivalent Python integer/real type.
** List = can be implemented using a Python list.
** String = should be implemented using the equivalent Python String type.When you print a string, you must print the single quotations around a string! This is because I want to distinguish between the cases when the output is an integer, like 1, vs. a string, like '1'.

Each type has a literal representation:
- Integer (simplified: only decimals). An integer is represented by one or more digits (0-9). For example, 42 is an integer literal.
- Real (simplified: no exponents). A real is represented by zero or more digits (0-9) followed by the decimal point "." and zero or more digits (0-9). For example, 3.14159 is a real literal.
- String. A string literal starts with a double or single quotes, may be followed by zero or more non-quote characters, and ends with a similar quote. The value of the string literal should not include the starting and ending quotes. An example string literal is "Hello SeaWolf".
- List. A list literal consists of a square bracket, followed by a comma-separated list of zero or more expressions. For example, [307, 130, 100+3] is a list literal.
- Boolean: True or False.

* Operators: the following operators are listed in precedence order, from highest to lowest (all associative operators are left-associative, except ** (power), which is right associative):

Operator	Description
literal	The literals given above.
(expression)	A parenthesized expression.
a [b]	Indexing. B may be any expression.
a * b, a / b	Multiplication and division.
a % b	Modulus (divides left hand operand by right hand operand and returns remainder).
a ** b	Exponent Performs exponential (power) calculation on operators = a to the power b. 2**3**4 = 2**(3**4) = 24178516392292583494123 52
a // b	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.
a + b, a - b	Addition and subtraction.
a in b	Evaluates to true if it finds a variable in the specified sequence and false otherwise.
a < b, a <= b, a == b, a <> b, a > b, a >= b	Comparison (works for numbers and strings like in python).
not a	Boolean NOT.
a and b	Boolean AND.
a or b	Boolean OR.

The operators have the following semantics:
- Indexing: B must be an integer, a must be either a string or a list. If a is a string, returns the b-th character of the string as a string. If a is a list, returns the b-th element of the list (whatever type it has). The index is zero-based, so the first element is at index 0. If the index is out of bounds, it is a semantic error.
- Addition: A and B must be either both numbers, both strings or both lists. If they are integers or reals, then addition (with standard semantics) is performed. If they are both strings, than string concatenation is performed. If they are both lists, then concatenation of the lists is performed.
- Multiplication, Division and Subtraction: A and B must be integers or reals. For division only, B must not be 0. These are performed using standard multiplication semantics.
- Comparison: A and B must be integers, reals or strings. The two values are compared, and the result is True if the comparison is true, and False if the comparison is false.
- Boolean AND, OR, NOT: A and, if present, B must be booleans.

Your program will be called with a single argument. This argument will be a file containing expressions, with **one expression per line.**
You should process each expression and print one of three possible outputs:
- If the line contains a syntax error, you should print out: SYNTAX ERROR.
- A semantic error occurs when the line does not contain a syntax error, but one of the "must" conditions given above is violated when evaluating it. If the line contains a semantic error, you should print out: SEMANTIC ERROR.
Otherwise, you should evaluate the expression, and print the result.

An example input file might look like:
1 - 2 + 3
1 2 3
42 + "Red"
1 - (2 + 3)
"Hello" + " " + "SeaWolf."
[1, 2, 3][0] + 40

The output from this file should look like:
2
SYNTAX ERROR
SEMANTIC ERROR
-4
'Hello SeaWolf.'
41

This will be the first of several projects that uses this code, so it will do you well to make sure your code is reusable.

Your program will be run with a command like:
`python3 stonyBrookScript.py inputFileName.txt`

Your program must take input from the file specified on the command line and print the results in the console: one output line corresponding to every input line.
You should include your name and ID as the first line (commented out) in your python file. Your program must not produce spurious output.

Submit only the python file **`stonyBrookScript.py`** on Blackboard (and nothing else!!). You cannot use multiple files for this assignment.

Grading:
There are a total of 40 points. Your program will be run against several complex use cases using similar expressions to the sample input. If it generates a correct result you will get the points associated for each use case (if the solution is incorrect, you will loose the points for that use case).

Homework 1

Due: Friday, February 2, 2018, 11:59pm

1. (5 Points) Write and test the GCD Program in the following languages: C, Prolog (XSB), SML, Python and Javascript.

- In C: gcd.c

```
#include <stdio.h>

int main() {

int i, j;

scanf("%d",&i);

scanf("%d",&j);

while (i != j) {

if (i > j) i = i - j;

else j = j - i;

}

printf("GCD = %d\n",i);

}
```

- In Prolog: gcd.pl

```
gcd(A,A,A).

gcd(A,B,G) :-

A > B,

C is A-B,
```

Assignments

Homework 5

Due: Friday, April 27, 11:59PM

For this assignment, we add support for variables, assignments, conditionals, loops, and output to our language. For now, our language has only a single scope for variables: a global static scope.

Variables and Assignments

Variable names must begin with an ASCII letter, which may be followed by zero or more ASCII letters, digits and underscore. A regular expression that matches variable names is '[A-Za-z][A-Za-z0-9_]*'.

Our expressions from the previous assignment need to be extended in two ways:

1. we need to add support for assignments to variables (for example, x=1) (including assignment to indexed list variables, for example: l[2] = 5 where l was a variable that contained a list, for example, l was previously assigned a list: l=[1,2,3]; after the assignment l[2] = 5, l contains [1,2,5]).

2. we need to add support for variables used in expressions (for example, if x was assigned 1, then print(x) will print 1). Similarly, we should evaluate indexed variables if they occur in a place where their value is needed (for example, if l was assigned [1,2,5], then print(l[0]+l[1]+l[1+1]) should print 8).

Evaluating a variable for its value should have the following behavior: if the variable has had a value assigned to it, the value should be returned. Otherwise, a "Semantic error" should be generated and the program should stop.

When an indexed list variable is used in an expression, then both the list and the index are evaluated to their value and the indexed list expression is evaluated. If the variable is not a list (or a string), or the index is not a integer number, then a "Semantic error" should be produced. If the index is outside the bounds of the array, a "Semantic error" should be produced.

The new types of statements are:

- Block: a block statement consists of zero-or-more statements enclosed in curly braces {...}. When the block executes, each of the statements in the block is executed in sequential order.

- Assignment: an assignment statement consists of an expression, an equals sign, an expression, and a semicolon. When the assignment statement executes, the left expression is assigned the value evaluated of the right-expression.

- Print: a print statement consists of the "print" keyword, a left-parenthesis, an expression, a right-parenthesis, and a semicolon. When the print statement executes, the expression is evaluated for its value.

- Conditional statements:

o If statements: consists of the keyword "if", a left-parenthesis, an expression, a right-parenthesis, and a body block statement. When the if statement executes, if the expression is **True**, the body block statement is executed.

o If-else statements: consists of the keyword "if", a left-parenthesis, an expression, a right-parenthesis, a body block statement, an “else” keyword and a body block statement . When the if-else statement executes, if the expression is **True**, the first body block statement is executed, **else** the second body is executed. The “else” goes with the closest previous “if” in the same block.

- While: a while statement consists of the keyword "while", a left-parenthesis, an expression, a right-parenthesis, and a body block statement. Executing the while statement begins by evaluating the condition for its value. If that value is **False**, the while statement terminates. Otherwise, the while statement executes its body block statement, and execution repeats.

An input program consists of a single outermost block statement. Executing the program consists of executing this block.

Your interpreter will be called with a single argument. This argument will be a file containing an input program. If the program contains syntax error, your interpreter should print out: "Syntax error" and stop. Otherwise, you should execute the program. If the execution of the program causes a semantic error, then your interpreter should print out "Semantic error" and stop.

Execution of the program may cause print statements to execute and the output from print statements should be sent to standard output.

No other output should be produced!

An example input script might look like this:

```
{
  number = 33;
```

```
  isPrime = 1;
  i = 2;
  while(isPrime==1 and i<number){
    if (number%i==0) {
      isPrime = 0;
    }
    i = i + 1;
  }
  if(isPrime==1){
    print("isPrime is true");
  } else {
    print("isPrime is false");
  }
}
```

The output from this file should be only:

```
isPrime is false
```

To complete this assignment, I suggest you:

- Create a global map/dictionary to store the values of variables.

- Implement the evaluate() method of the Variable class so that it returns the values of variables.

- Implement the execute method of each of the statements.

You should include your name and ID as the first line (commented out) in your python file. The program must run under Python 3.x. The program must be named main.py. Your program will be run with a command like:

```
python3 main.py inputProgram.txt
```

Grading:

There are a total of 50 points. Your program will be run against 5 use cases using similar programs to the sample input. If it generates a correct result you will get 10 points for each use case (if the solution is incorrect, you will lose the points for that use case).

Additional example input program:

```
{
  data = [ [ 100, 42 ], [ 100, 50 ], [ 123, 456 ], [ 300, 9000 ] ];
  result = [ 0, 0, 0, 0 ];
  i = 0;
  while (i < 4){
    a = data[i][0];
    b = data[i][1];
    if (a > 0){
      while (b > 0){
        if (a > b){
          a = a - b;
        } else {
          b = b - a;
        }
       }
    }
    result[i] = a;
```

```
    i = i + 1;

  }

  print(result);

}
```

The program will print:

```
[2, 50, 3, 300]
```

Printing strings should execute as in python (no quotes around single strings printed and single quotes around strings in lists):

{print("a");}

the output should be:

a

{print([1, "a", 2]);}

the output should be:

[1, 'a', 2]

Homework 4 (expression grammar)

Extended deadline: Friday, April 20, 11:59PM
Originally Due: Friday, April 13, 11:59PM

We'll be spending the next few assignments implementing a programming language, which will be called stonyBrookScript. For this assignment, we will be implementing an expression evaluator. This takes an expression as input, evaluates it to a result, and prints the result to standard output.
We will use the parser generator Ply: Python Lex and Yacc
1. Download Python Lex and Yacc ply from: http://www.dabeaz.com/ply/
2. Download and unzip.
3. Install with: python setup.py install
4. Read documentation and try examples.
5. Use mailing list to ask questions.

Our Language:

* Data Types: our language has three data types:
** Numbers: Integer and Real = should be implemented using the equivalent Python integer/real type.
** List = can be implemented using a Python list.
** String = should be implemented using the equivalent Python String type.When you print a string, you must print the single quotations around a string! This is because I want to distinguish between the cases when the output is an integer, like 1, vs. a string, like '1'.

Each type has a literal representation:
- Integer (simplified: only decimals). An integer is represented by one or more digits (0-9). For example, 42 is an integer literal.
- Real (simplified: no exponents). A real is represented by zero or more digits (0-9) followed by the decimal point "." and zero or more digits (0-9). For example, 3.14159 is a real literal.
- String. A string literal starts with a double or single quotes, may be followed by zero or more non-quote characters, and ends with a similar quote. The value of the string literal should not include the starting and ending quotes. An example string literal is "Hello SeaWolf".
- List. A list literal consists of a square bracket, followed by a comma-separated list of zero or more expressions. For example, [307, 130, 100+3] is a list literal.
- Boolean: True or False.

* Operators: the following operators are listed in precedence order, from highest to lowest (all associative operators are left-associative, except ** (power), which is right associative):
Operator Description
literal The literals given above.
(expression) A parenthesized expression.
a [b] Indexing. B may be any expression.
a ** b Exponent Performs exponential (power) calculation on operators = a to the power b. 2**3**4 = 2**(3**4)
= 241785163922925834941235 2
a * b, a / b Multiplication and division.
a % b Modulus (divides left hand operand by right hand operand and returns remainder).
a // b Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.
a + b, a - b Addition and subtraction.
a in b Evaluates to true if it finds a variable in the specified sequence and false otherwise.
a < b, a <= b, a == b, a <> b, a > b, a >= b Comparison (works for numbers and strings like in python).
not a Boolean NOT.
a and b Boolean AND.
a or b Boolean OR.