CSC 533: Programming Languages
Spring 2025

HW1: Syntax and EBNF Grammars

In the first half of the semester, you will be developing an interpreter for a simple scripting language named SILLY (Simple, Interpreted, Limited Language for You). The EBNF grammar rules for SILLY v. 24 are as follows:

```
<statement>  --> <assign> | <print> | <if> | <while> | <repeat> |
                 <compound> | <func> | <return>

<assign>     --> <id> '=' <expr>

<print>      --> 'print' <expr>

<if>         --> 'if' <expr> <compound> 'else' <compound>

<while>      --> 'while' <expr> <compound>

<repeat>     --> 'repeat' <expr> <compound>

<compound>   --> '{' { <statement> } '}'

<func>       --> 'func' <id> '(' { <id> } ] ')' <compound>

<return>     --> 'return' <expr>

<expr>       --> <number> | <boolean> | <char> | <string> | <list> | <id> |
                 '(' ( <id> | <op>) { <expr> } ')'

<id>         --> <letter> { (<letter> | <digit>) }
<number>     --> [ '-' ] <digit> { <digit> } [ '.' { <digit> } ] [ ('E' | 'e') { <digit> } ]
<string>     --> '"' { <nwq> } '"'
<char>       --> '\'' <nwq> '\''
<boolean>    --> 'true' | 'false'
<list>       --> '[' { <expr> } ']'

<letter>     --> 'a' | 'b' | 'c' | ... | 'z' | 'A' | 'B' | 'C' | ... | 'Z'
<digit>      --> '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
<nwq>        --> any non-whitespace character other than '"' or '\''

<op>         --> 'and' | 'or'  | 'not' | '+'   | '*'   | '/'   |
                 '=='  | '!='  | '>'   | '>='  | '<'   | '<='  |
                 'len' | 'get' | 'cat' | 'str'
```

Recall that `|` denotes alternatives within a rule, `[ ]` denotes an optional (0 or 1) element, and `{ }` denotes a repetitive (0 or arbitrarily many) element. There is no whitespace (i.e., spaces, tabs or new lines) between the characters in an identifier, number, character or string. Delimiters, i.e. '(', ')', '{', '}', '[', and ']', do not require spaces to separate them from other tokens, but all others do.

Answer the following questions about the syntax of SILLY based on the above grammar rules.

1. Which of the following are valid identifiers? For each valid identifier, provide a parse tree (with the `id`

abstraction at the root). If invalid, briefly describe what aspect violates the rules.

```
Q          4b          a1BC        X_1
```

2. Which of the following are valid numbers? For each valid number, provide a parse tree (with the `number` abstraction at the root). If invalid, briefly describe what aspect violates the rules.

```
44         -44         0.5         .5          5.

-.2        --9         E4          -3e2        -3.1e2.5
```

3. Which of the following are valid expressions? For each valid expression, provide a parse tree (with the `expr` abstraction at the root). If invalid, briefly describe what aspect violates the rules.

```
x          (x)         (-x)        (x + 1)     (+ x 1)        (- x 1)
```

4. Which of the following are valid if statements? If valid, provide a parse tree (with the `if` abstraction at the root). If invalid, briefly describe what aspect violates the rules.

```
if (== x y) {
  print "eq"
}
```

```
if true { }
else { }
```

```
if (> m n) {
  print 0
} else {
  print 1
}
```

```
if (avg >= 80) {
  if (avg >= 90)
    print "A"
  else
    print "B"
} else {
  print "F"
}
```

5. Is it valid to mix the types of values in a list, e.g., ["foo" 3]? If so provide a parse tree for this list expression. If not, explain why not.

6. Is it valid to nest a list inside another list, e.g., [1 [2 3]]? If so provide a parse tree for this list expression. If not, explain why not.

7. Give an example of an assignment statement (`assign`) that is as short as possible, in terms of number of tokens. Note that keywords (such as `print`) and operators (such as ==) are indivisible and thus count as a single token. You do not need to provide a parse tree.

8. Give an example of a while statement (`while`) that is as short as possible, in terms of number of tokens. You do not need to provide a parse tree.

9. Give an example of a function declaration (`func`) that is as short as possible, in terms of number of tokens. You do not need to provide a parse tree.