Fall Term 2014

CS 345H - Programming Languages: Honors Assignment 3

Assigned on: **Oct 06 2014**
Due by: **Oct 20 2014**

1 Extending mySIMPL (60 Points)

The task is to extend the parser and evaluator for mySIMPL (my Simple IMperative Programming Language) from assignment 2. In this assignment, we add support for functions, conditional statement, and loops.

The extended syntax (bold font used to highlight changes to the previous grammar) of mySIMPL is:

prog	::= retStatement ’.’ | **funcDecl ’;’ prog** | statement ’;’ prog
funcDecl	::= ’function’ < *id* > ’(’ < *id* > ’)’ ’{’ prog ’}’
retStatement	::= ’return’ base
statement	::= declaration | assignment | conditional | loop
declaration	::= *var* < *id* >
assignment	::= < *id* > := base
conditional	::= ’if’ ’(’ condition ’)’ ’then’ statementSeq ’else’ statementSeq ’endif’
loop	::= ’while’ ’(’ condition ’)’ ’do’ statementSeq ’done’
statementSeq	::= statement ’.’ | statement ’;’ statementSeq
condition	::= base comp base
base	::= < *id* > | < *number* > | ’(’ expression ’)’ | **funcCall**
funcCall	::= < *id* > ’(’ base ’)’
expression	::= [expression addOp] term
term	::= [term mulOp] factor
factor	::= base
addOp	::= ’+’ | ’−’
mulOp	::= ’*’ | ’/’
comp	::= ’==’ | ’<’ | ’>’ | ’<=’ | ’>=’ | ’! =’

Extend the predicates

`parse(+TokenList, -AST)`

`evaluate(+AST, -Number)`

according to the new grammar, static scoping rules, and call-by-value semantics.

Example programs are:

```
['var', x, ';', x, ':=', 1, ';', 'if', '(', x, '<', 0, ')', 'then', x, ':=', 10, '.', 'else', x, ':=
 20, '.', 'endif', ';', return, x, '.']

[function, f, '(', x, ')', '{', return, x, '.', '}', ';', return, f, '(', '(', 10, +, 1,
')',  ')',  '.']
```

2 Nested Functions (Optional, 10 Bonus Points)

The grammar permits function declarations to be nested. Extend the evaluate predicate to correctly deal with the scoping issues of nested functions.

Additional Information and Hints

- All single-quoted terminals in the grammar should be considered keywords and hence excluded as identifiers. Numbers are excluded from being identifiers.
- Albeit expressed through the same element in the grammar, identifiers for functions and variables should be considered separate namespaces and therefore a function and a variable of the same name should be allowed to coexist within a program.
- One of the key challenges of this assignment is to deal with the scoping and visibility of variables in blocks. E.g., declarations within function bodies should not leak into the enclosing scope.
- It might be necessary to "clone" parts of the environment during evaluation. The Prolog predicates `copy_term/2`, and `duplicate_term/2` can be helpful.
- We will expect function declarations to always precede any corresponding function call even though the grammar does not enforce this. Therefore, it is safe to use a single pass for evaluation.