CS280
Programming Assignment 3
Fall 2019

Using the token definitions from programming assignment 2, we can construct a language with the following grammar rules:

```
Prog := Slist
Slist := SC { Slist } | Stmt SC { Slist }
Stmt := IfStmt | PrintStmt | LetStmt | LoopStmt
IfStmt := IF Expr BEGIN Slist END
LetStmt := LET ID Expr
LoopStmt := LOOP Expr BEGIN Slist END
PrintStmt := PRINT Expr
Expr := Prod { (PLUS | MINUS) Prod }
Prod := Rev { (STAR | SLASH) Rev }
Rev := BANG Rev | PRIMARY
Primary := ID | INT | STR | LPAREN Expr RPAREN
```

This language will be used for the remainder of the semester.

The following items describe the language.

1. The language contains two types: integer and string.
2. The PLUS MINUS STAR and SLASH operators are left associative.
3. The BANG operator is right associative.
4. An IfStmt evaluates the Expr. The Expr must evaluate to an integer. If the integer is nonzero, then the Slist is executed.
5. A LetStmt evaluates Expr and saves its value in memory associated with the ID. If the ID does not exist, it is created. If the ID already exists, its value is replaced.
6. A LoopStmt evaluates the Expr. The Expr must evaluate to an integer. If the integer is nonzero, then the Slist is executed. This process repeats until the Expr evaluation does not result in a nonzero integer.
7. A PrintStmt evaluates the Expr and prints its value.
8. The type of an ID is the type of the value assigned to it.
9. The PLUS and MINUS operators in Expr represent addition and subtraction.
10. The STAR and SLASH operators in Prod represents multiplication and division.
11. The BANG operator represents reversing its operand.
12. It is an error if a variable is used before a value is assigned to it.
13. Addition is defined between two integers (the result being the sum) or two strings (the result being the concatenation).
14. Subtraction is defined between two integers (the result being the difference).

15. Multiplication is defined between two integers (the result being the product) or for an integer and a string (the result being the string repeated integer times).
16. Division is defined between two integers.
17. Reversing is defined for integers and strings. The result of reversing an integer is the operand in reverse order (! 123 is 321). The result of reversing a string is the operand in reverse order (! “hello” is “olleh”).
18. Performing an operation with incorrect types or type combinations is an error.
19. Multiplying a string by a negative integer is an error.
20. An IF or LOOP statement whose Expr is not integer typed is an error.

Note that by the time the semester is over, you will need to handle all of these items that describe the language. However, you will not need to handle most of them in assignment 3.

For Programming Assignment 3, you MUST implement a recursive descent parser. You may use the lexical analyzer you wrote for Assignment 2, OR you may use a solution provided by the professor.

A skeleton for the solution, with some of the rules implemented, is provided as starter code. You may use this to begin the assignment if you like.

You must create a test program for your parser. The program takes zero or one command line argument. If zero command line arguments are specified, the program should take input from the standard input. If one command line argument is specified, the program should use the argument as a file name and take input from that file. If the file cannot be opened, the program should print COULD NOT OPEN followed by the name of the file, and should then stop. If more than one command line argument is specified, the program should print TOO MANY FILENAMES, and should then stop.

The result of an unsuccessful parse is a set of error messages printed by the parse functions. If the parse fails, the program should stop after the parse function returns.

The assignment does not specify the exact error messages that should be printed out by the parse; however the format of the message should be the line number, followed by a colon and a space, followed by some descriptive text. Suggested messages might include “No statements in program”, “Missing semicolon”, “Missing identifier after set”, etc.

The result of a successful parse is a parse tree.

Assignment 3 is meant to test that you can properly detect poorly formed programs, and that you can traverse the parse tree for well formed programs.

If a parse tree has been generated, it should be traversed to perform various tests.

1. Make sure that a variable has a LET statement before it is used. If you find a variable that does not have a LET statement before it is used, print UNDECLARED VARIABLE followed by a space and the variable name
2. Number of bang operators (format is BANG COUNT: N, where N is the count of bang nodes in the tree)
3. Max depth of tree (Format is MAX DEPTH: N, where N is the maximum depth of the tree; the root of the tree is at depth 1)

NOTE that your program might be graded using different input file names and error cases. SOLVE THE GENERAL PROBLEM and DO NOT HARDCODE output for test cases.

PART 1 due 11/13

PART 2 due 11/20

NOTE that your startercode includes partial implementations of some of the files in the assignment. Files with SKEL in the name are partial, skeleton implementations. You can use these as a starting point if you like.

You may also use the professor's solution to program 2 once it is published.