CS280
Programming Assignment 3
Spring 2020

Part 1 - Due 4/8
Part 2 - Due 4/15
Part 3 - Due 4/22

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

```
Prog := Sl
Sl := SC { Sl } | Stmt SC { Sl }
Stmt := PrintStmt | PrintlnStmt | RepeatStmt | Expr
PrintStmt := PRINT Expr
PrintlnStmt := PRINTLN Expr
RepeatStmt:= Repeat Expr BEGIN Stmt END
Expr := Sum { EQ Sum }
Sum := Prod { (PLUS | MINUS) Prod }
Prod := Primary { (STAR | SLASH) Primary }
Primary := IDENT | ICONST | SCONST | LPAREN Expr RPAREN
```

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

The following items describe the language.

PLEASE NOTE that while this description of the language is provided here, it DOES NOT MEAN that you have to implement all of this for assignment 3! The description is provided to help you understand the language and how it will be used in Assignment 4.

1. The language contains two types: integer and string.
2. The PLUS MINUS STAR and SLASH operators are left associative.
3. The EQ operator is right associative.
4. A Repeat statement evaluates the Expr. The Expr must evaluate to an integer. If the integer is nonzero, then the Stmt is executed, otherwise it is not.
5. A PrintStmt evaluates the Expr and prints its value.
6. A PrintlnStmt evaluates the Expr and prints its value, followed by a newline character.
7. The type of an IDENT is the type of the value assigned to it.
8. The EQ operator assigns values to variables. It evaluates the Expr on the right hand side and saves its value in memory associated with the left hand side (an IDENT). If the IDENT does not exist, it is created. If the IDENT already exists, its value is replaced.
9. The PLUS and MINUS operators in Expr represent addition and subtraction.
10. The STAR and SLASH operators in Prod represent multiplication and division.

11. It is an error if a variable is used before a value is assigned to it.
12. Addition is defined between two integers (the result being the sum) or two strings (the result being the concatenation).
13. Subtraction is defined between two integers (the result being the difference) or two strings (the result being removal of the first instance of the second string from the first string, if any).
14. 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).
15. Division is defined between two integers.
16. Performing an operation with incorrect types or type combinations is an error.
17. Multiplying a string by a negative integer is an error.
18. A Repeat 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.

Note that while the -v option from Program 2 is not required for program 3, it is often helpful for debugging; therefore, although it will not be used or tested or graded, you might want to leave it in your program.

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 END”, 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 been assigned to (using EQ) before it is used. If you find a variable that was not assigned to, print UNDECLARED VARIABLE followed by a space and the variable name
2. Number of + operators (output format is PLUS COUNT: N, where N is the count of + nodes in the tree)
3. Number of = operators (output format is EQ COUNT: N, where N is the count of = nodes in the tree)
4. 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 4/8

PART 2 due 4/15

PART 3 due 4/22