CS280
Programming Assignment 3
CS280 Spring 2018

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

```
Prog := Slist
Slist := Stmt SC { Slist }
Stmt := VarStmt | SetStmt | PrintStmt | RepeatStmt
VarStmt := VAR IDENT Expr
SetStmt := SET IDENT Expr
PrintStmt := PRINT Expr
RepeatStmt := REPEAT Expr Stmt
Expr := Term { (+|-) Term }
Term := Factor { * Factor }
Factor := Primary { [ Expr : Expr ] }
Primary := IDENT | ICONST | SCONST | 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 language contains 4 operators: +, -, *, and []. All operators are left associative.
3. A VarStmt is a variable declaration and initialization in one statement. The type of the IDENT is determined by the type of the Expr.
4. A SetStmt sets the value of IDENT to the value of Expr.
5. A PrintStmt prints the value of the Expr followed by a newline.
6. A RepeatStmt repeats Stmt Expr times.
7. The + and - operators represent addition and subtraction.
8. The * operator represents multiplication.
9. The [ ] operator represents slicing a string.
10. It is an error if a variable is used in an Expr before it is declared in a VarStmt.
11. It is an error if the type of a variable does not match the type of Expr in a SetStmt.
12. It is an error if the Expr in a RepeatStmt is not a non-negative integer.
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) or two strings (the result being the removal of the first occurrence of the second string from the first string, or the first string if the second string is not present in the first string).
15. Multiplication is defined for integers (the result being the product) or for an integer and a string (the result being the string repeated integer times).

16. Slicing is defined as creating a substring of a Primary. The first Expr must be a nonnegative integer. The second Expr must be an integer greater than the first Expr. Slicing returns a new string beginning at the character position of the first Expr and ending before the character position of the second Expr. Character positions begin at 0.
17. Performing an operation with incorrect types or type combinations is an error.
18. Multiplying a string by a negative integer is an error.
19. Slicing an integer is an error.
20. Specifying slicing arguments that are outside the range of the string being sliced 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.

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. Once a parse tree has been generated, it can be traversed to perform various tests.

The following information must be printed out for the parse tree:

1. Number of leaves (Format is LEAF COUNT: N, where N is the number of leaves)
2. Number of identifiers (Format is IDENT COUNT: N, where N is the number of identifiers)
3. Count of unique identifiers (Format is UNIQUE IDENT COUNT: N)
4. Comma separated list of unique identifiers

If there are no identifiers, steps 3 and 4 can be skipped

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

Part 1 of Assignment 3 tests against badly formed programs and incorrect arguments.

Part 2 of Assignment 3 tests against well formed programs.

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