CS280
Programming Assignment 4
Fall 2019

Using the token definitions from programming assignment 2, and the parser from programming assignment 3, we can now construct an interpreter.

To build an interpreter, we must add three things:

- A class to represent values
- Functions to perform evaluations and return values
- A symbol table mapping identifiers to values

You will be given a partial implementation of a “Val” class to store values. The Val class will contain prototypes for operator functions that you should implement. The operator functions shall perform type checking.

To perform evaluations, we must create a virtual function called Eval() and implement it for every class. Eval should take a reference to the symbol table and should return the Val that results from the evaluation. You will be given some notes about Eval in your startercode.

As a reminder, the language has 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
```

The following items describe the language. In places where a runtime type check is required, I have added **[RT]** as a reminder. Failing a runtime type check halts the interpreter.

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 **[RT]**. 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 **[RT]**. 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) **[RT]**.
14. Subtraction is defined between two integers (the result being the difference) **[RT]**.
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) **[RT]**.
16. Division is defined between two integers **[RT]**. **Division by zero is an error [RT].**
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 **[RT]**.
19. Multiplying a string by a negative integer is an error **[RT]**.
20. An IF or LOOP statement whose Expr is not integer typed is an error **[RT]**.

For Programming Assignment 4, you may use your parser from assignment 3, or you may use a solution provided by the professor. 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 main program for your interpreter. 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 result of a successful parse is a parse tree.

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, then stop.

If the parse succeeds, and if there are no errors on declared variables, Eval the parse tree. There may be runtime errors during the evaluation. When an error is encountered, the program should generate an error message in the format RUNTIME ERROR at N: description, where N is the line number where the error occurred (stored in the parse tree node!) and description is some descriptive text about the error. HINT: easiest way to do this is to build a string, throw it as an exception, and catch it in main.

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 12/4

PART 2 due 12/11

PART 2 will contain more complex error checking and runtime errors.