CS280
Programming Assignment 4
Spring 2020

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

Part 1 - due 4/29
Part 2 - due 5/6

To build an interpreter, we must add three things:

- A class to represent values
- Functions to do runtime checks, perform evaluations, and return any calculated 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 := 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
```

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 EQ operator is right associative.

4. A Repeat statement evaluates the Expr. The Expr must evaluate to an integer **[RT]**. If the integer is nonzero, then the Stmt is executed, otherwise it is not. The value of the Expr is decremented by 1 each repetition. In other words “repeat 5 begin println “hello” end” will print “hello” 5 times.
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 **[RT]**.
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 **[RT]**.
17. Multiplying a string by a negative integer is an error **[RT]**.
18. A Repeat 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.

For runtime errors, your program should throw an exception indicating a RUNTIME ERROR. The exception should be caught and printed, and the program should stop.

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.

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 4/29

PART 2 due 5/6

You may type “runcase” to list the names of the test cases in each part.