CS280 Programming Assignment 2
Fall 2017
Part 1 due Nov 6 before midnight, no later than Nov 11
Part 2 due Nov 13 before midnight, no later than Nov 18
Part 3 due Nov 20 before midnight, no later than Nov 20

We define an EBNF grammar for a simple language. This grammar uses the tokens defined in Programming Assignment 2. It defines nine different nonterminal symbols. The start symbol for the grammar is "Prog". For this assignment, you will write a recursive descent parser for this grammar. Your goal is to build a parse tree, perform some semantic checks on the tree, and generate some test output.

You are expected to use your getToken() from Assignment 2 to read tokens, and to implement nine functions for this parser. A skeleton implementation will be provided for you. You should use the provided parser.h and lexer.h files. You should not change lexer.h. You may add any other files, or change any other files that are given to you.

For this assignment, you must create a main routine that parses input, either from a file or, if no filename is specified, from the standard input. An optional "-t" argument may be provided.

If the program is unable to open a file, it must print the filename followed by FILE NOT FOUND, and should then stop.

In all situations that require the generation of an error message, if the error message is specified to include a filename, and if there is no filename (i.e. the standard in case), then the message should NOT include the "filename:" portion of the message.

The nine grammar rules are below

```
Prog ::= StmtList
StmtList ::= { Stmt T_SC } { StmtList }
Stmt ::= Decl | Set | Print
Decl ::= T_INT T_ID | T_STRING T_ID
Set ::= T_SET T_ID Expr
Print ::= T_PRINT Expr | T_PRINTLN Expr
Expr ::= Term { (T_PLUS|T_MINUS) Expr }
Term ::= Primary { (T_STAR|T_SLASH) Term }
Primary ::= T_ICONST | T_SCONST | T_ID | T_LPAREN Expr T_RPAREN
```

The following are the semantic rules associated with this language:

1. There are only two types: integer and string.
2. Variables must be declared before they are used. They are declared with Decl.
3. There is one single scope for variables.
4. Variable names must be unique; once a name has been declared, it may not be declared again.
5. An integer constant is of type integer.
6. A string constant is of type string.
7. The type of an identifier is the type assigned when it is declared.
8. The type of an Expr or a Term is the type of the result of the operation.
9. Adding two integers results in an integer representing the sum
10. Subtracting two integers results in an integer representing the difference
11. Multiplying two integers results in an integer representing the product
12. Dividing two integers results in an integer representing the quotient
13. Adding two strings results in a string representing the concatenation
14. Multiplying an integer by a string results in a string which is the string operand repeated integer times (i.e. 3 * “hi” is “hihihi”)
15. Dividing two strings results in an a string where the first instance of the denominator that is found in the numerator is removed.
16. ALL OTHER COMBINATIONS OF OPERATIONS ARE UNDEFINED and are an error
17. A variable is assigned a value using a Set. The type of the variable being set must match the type of the expression; if it does not this is an error.
18. An expression can be printed using either “print” or “println”. In both cases, the value of the Expr is printed to the standard output. In the case of “println”, a newline is printed to standard out at the end of the execution of the program.

If the program finds a syntax error during the parse, it should print a message in the following format:

```
filename:linenumber:Syntax error {some message describing the error}
```

If the program finds any errors during the parse phase, it should terminate after the parse.

If the “-t” flag argument is specified, then the program should produce a “trace” of the tree.

A trace is performed by doing a postorder traversal of the parse tree and printing output during the traversal, as follows:

- Before following the left child in the tree, print the letter L
- After following the left child in the tree, print the letter u
- Before following the right child in the tree, print the letter R
- After following the right child in the tree, print the letter U
- To visit the node, print the letter N

All of these characters should be printed on a single line with a newline printed at the end.

Next, the program should perform tests to ensure compliance with semantic rules #2 and #4. If the input does not comply with those rules then, after printing any error messages, the program should terminate.

If the semantic checks are all successful, then it should traverse the tree, gather statistics, and print a report on the statistics in the format below:

```
Total number of identifiers: X
Total number of set: X
Total number of +: X
Total number of *: X
```

Where X is the total count for, respectively, identifiers, sets, + operators, and * operators. Note that you should NOT gather these statistics by counting tokens, but rather by traversing the

If semantic check #2 fails, the program should print a message in the following format:

```
Filename:linenumber:variable XX is used before being declared
```

If semantic check #4 fails, the program should print a message in the following format:

```
Filename:linenumber:variable XX was already declared
```