For the remainder of the semester we will be building a small program that does arithmetic on polynomials.

Remember that a polynomial generally expressed in the form

$a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_0$

In our system will represent a polynomial by a curly brace enclosed list of coefficients:

$\{ a_n, a_{n-1}, a_{n-2}, \ldots , a_0 \}$

The coefficients of the polynomials may be integers or floating point numbers. The system will support evaluating polynomials, printing them out, and performing various mathematical operations on them.

The remainder of the semester will be broken into three pieces:

1. Lexical analyzer
2. Parser
3. Interpreter

For the lexical analyzer, you will be provided with a description of the lexical syntax of the language. You will produce a lexical analysis function and a program to test it.

The lexical rules of the language are as follows:

1. The language has identifiers, which are defined to be a letter followed by zero or more letters or numbers. This will be the token ID
2. The language has integer constants, which are defined to be an optional leading dash (for a negative number), followed by one or more digits. This will be the token ICONST
3. The language has real constants, which are defined to be an optional leading dash (for a negative number), followed by one or more digits, followed by a dot, followed by one or more digits. This will be the token FCONST
4. The language has quoted strings, which are letters enclosed inside of double quotes, all on the same line. This will be the token STRING
5. The language has reserved the keywords print and set. They will be the tokens PRINT and SET
6. The language has several single-character tokens. They are + - * , { } [ ] ( ) ; which will be the tokens PLUS MINUS STAR COMMA LBR RBR LSQ RSQ LPAREN RPAREN SC
7. A comment is all characters from a # to the end of the line; it is ignored and is not returned as a token. NOTE that a # in the middle of a STRING is NOT a comment!
8. An error will be denoted by the ERR token, and EOF by the DONE token

Note that any error detected by the lexical analyzer will result in the ERR token, with the lexeme value equal to the string recognized when the error was detected.

The program for Assignment 2 implements a lexical analyzer. The calling sequence for the lexical analyzer, a definition for all of the tokens, and a class definition for Token, is included in the startercode for the assignment.

The assignment is to write the lexical analyzer function and some test code around it. The test code takes several command line arguments:

-v (optional) if present, every token is printed when it is seen
-stats (optional) if present, statistics are printed
-sum (optional) if present, summary information is printed
filename (optional) if present, read from the filename; otherwise read from standard in

Note that no other flags (arguments that begin with a dash) are permitted. If an unrecognized flag is present, the program should print “Invalid argument {arg}”, where {arg} is whatever flag was given, and it should stop running.

At most one filename can be provided, and it must be the last command line argument. If more than one filename is provided, the program should print “Too many file names” and it should stop running.

If the program cannot open a filename that is given, the program should print “Could not open {arg}”, where {arg} is the filename given, and it should stop running.

The program should repeatedly call the lexical analyzer function until it returns DONE or ERR. If it returns DONE, the program proceeds to handling the -stats and -sum options, if any, and then exits. If it returns ERR, the program should print “Error on line N ({lexeme})”, where N is the line number in the token and lexeme is the lexeme from the token, and it should stop running.

If the -v option is present, the program should print each token as it is read and recognized, one token per line. The output format for the token is the token name in all capital letters (for example, the token LPAREN should be printed out as the string LPAREN. In the case of token ID, ICONST, FCONST and STRING, the token name should be followed by a space and the lexeme in parens. For example, if the identifier “hello” is recognized, the -v output for it would be ID (hello)

If the -stats option is present, the program should, after seeing the DONE token, print out the following report:

Total IDs: N
List of IDs: X

Where N is the number of times the ID token was seen and X is a comma separated list of identifier lexemes. If N is 0, then List of IDs is not printed.

If the -sum option is present the program should, after seeing the DONE token and processing the -stats option, print out the following report:

Total lines: L
Total tokens: N
Most frequently used tokens: X

Where L is the number of input lines, N is the number of tokens (not counting DONE), and X is a comma separated list of the tokens that appear most frequently. There may be only one item in the list, or there may be many. The list should be in alphabetical order. If N is 0, then the Most frequently used tokens line is not printed.

TEST CASES AND DUE DATES

PART 1: Due Mon Feb 6 by 11:55 PM (6 points)
(10% penalty per late day, 5 days)

1. Compiles
2. Recognizes invalid command line options (items beginning with -)
3. Recognizes a file that cannot be opened
4. Recognizes case with more than one file name
5. Prints correct number of lines for empty file
6. Prints correct number of lines for file containing nothing but comments

PART 2: Due TBD by 11:55 PM (10 points)
(10% penalty per late day, 5 days)

1. All PART 1 cases (6 points)
2. Recognizes string with a newline in it as an error
3. Recognizes string with a # in it as a string, not a comment
4. Recognizes all token types correctly
5. Printout with -v option is correct

Part 3: Due TBD by 11:55 PM (24 points)
(10% penalty per late day, 5 days)

1. All PART1 & 2 cases (10 points)
2. Proper -stats output when there are no ID
3. Proper -stats output, one ID
4. Proper -stats output, multiple IDs
5. Recognizes error with badly formed floating point number
6. Proper -sum output, large file
7. Proper -sum output, many tokens with same frequently used count
8. Proper -stats and -sum combined output
9. Proper -stats and -sum and -v combined output
10. Case 6 from standard input
11. Case 7 from standard input
12. Case 9 from standard input
13. Proper output with no flag arguments
14. Very large file -stats and -sum
15. Very large file -stats and -sum, standard input