CS280 Programming Assignment 2
Fall 2017
Part 1 due Oct 16 before midnight, no later than Oct 21
Part 2 due Oct 23 before midnight, no later than Oct 28

For this assignment we are going to create a function that reads an input stream and classifies it into “tokens” from a language that we define here.

In this language we make the following rules:

- An identifier is a letter followed by zero or more letters or numbers
- An integer constant is one or more digits
- A string constant is characters enclosed in double quotes, all on the same line
- The following keywords are defined:
    - int
    - string
    - set
    - print
    - println
- There are four operators
    - +
    - -
    - *
    - /
- The left and right parenthesis are valid in the language
- A semicolon is valid in the language
- *A comment begins with two slashes //. The two slashes and all characters up to a newline are discarded*
- *Any amount of whitespace can be used to delimit tokens. The whitespace is ignored.*[1]

Anything that is not a recognized token is an error.

The file lexer.h is provided and must be used, unchanged, for this assignment.

All of the tokens that this assignment will recognize are provided in lexer.h as a member of the TokenType enum.

For the assignment, you must write the function getToken. The function is defined in lexer.h as follows:

```
extern Token getToken(istream* br);
```

[1] Updated Oct 4

The Token that is returned is a class that is defined in lexer.h. The getToken() function should return the “next token”... or T_ERROR if it cannot recognize a token… or T_DONE when there is no more input.

You should implement the getToken function in one source file that #include’s lexer.h

You must also implement a separate main function in a separate file. The main function is meant to serve as a test engine for the getToken function.

The main program is controlled by command line flags, as follows

- -v ; optional; if provided, be “verbose” and print every valid token that is recognized
- -s ; optional; if provided, print all string constants defined in the input
- -i ; optional; if provided, print all the identifiers defined in the input

In addition to the (optional) flags, the program takes 0 or 1 file name. If no filename is provided, the program should read the standard input. If one filename is provided, the program should take input from that file. If more than one filename is provided, the program should print TOO MANY FILES, and should stop.

If an unrecognized flag is provided (an arg beginning with a - and not matching one of the three listed flags above), the program should print the flag followed by INVALID FLAG, and should stop.

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

If the test program finds that getToken returns an error, it should print “Lexical error” followed by a printout of the token, and should stop.

If the -v option is provided, every token should be printed out, as specified below.

After reaching end of input (the T_DONE token), the output of the program should be as follows:

Token count: N
Identifier count: N
String count: N
Strings: {strings in alphabetical order} (only if -s is specified)
Identifiers: {identifiers in alphabetical order} (only if -i is specified)

When printing out a token, the program should print the TokenType and, if required, the lexeme. The printed representation for the TokenType is a string equal to the symbolic name of the token. For example, the string representation for a T_ID is the string “T_ID”.

In the case of T_ID, T_ICONST, T_SCONST and T_ERROR, the printed representation of the symbolic name is followed by the lexeme in parenthesis. So for example, if getToken recognizes “foo” as an identifier, the output for the token will be T_ID(foo). Cosmic hint: implement operator<< for Token to make printing out tokens easy.