CMSC 430 Project 1

The first project involves modifying the attached lexical analyzer and the compilation listing generator code. You need to make the following modifications to the lexical analyzer, `scanner.l`:

1. The following reserved words should be added:

   `else, elsif, endfold, endif, fold, if, left, real, right, then`

   Each reserved words should be a separate token. The token name should be the same as the lexeme, but in all upper case.

2. Two additional logical operators should be added. The lexeme for the first should be `|` and its token should be `OROP`. The second logical operator added should be `!` and its token should be `NOTOP`.
3. Five relational operators should be added. They are `=`, `<>`, `>`, `>=` and `<=`. All of the lexemes should be represented by the single token `RELOP`.
4. One additional lexeme should be added for the `ADDOP` token. It is binary operator `-` that is the subtraction operator.
5. One additional lexeme should be added for the `MULOP` token. It is `/` that is the division operator.
6. A new token `REMOP` should be added for the remainder operator. Its lexeme should be `%`.
7. A new token `EXPOP` should be added for the exponentiation operator. Its lexeme should be `^`.
8. A new token `NEGOP` should be added for the unary minus operator. Its lexeme should be `~`.
9. A second type of comment should be added that begins with `--` and ends with the end of line. As with the existing comment, no token should be returned.
10. The definition for the identifiers should be modified so that underscores can be included, however, no more than two consecutive underscores are permitted, but leading and trailing underscores should not be permitted.
11. One additional type of integer literal should be added, which are hexadecimal integers. The begin with the `#` character followed by one of more decimal digits, or the letter A-F, in either upper or lower case.
12. A real literal token should be added. It should begin with a sequence of zero or more digits following by a decimal point followed by one or more additional digits. It may optionally end with an exponent. If present, the exponent should begin with an `e` or `E`, followed by an optional plus or minus sign followed by one or more digits.
13. The definition for the character literals should be modified so that five additional escape characters are also allowed: `'\b'`, `'\t'`, `'\n'`, `'\r'` and `'\f'`.

You must also modify the header file `tokens.h` to include each the new tokens mentioned above.

The compilation listing generator code should be modified as follows:

1. The `lastLine` function should be modified to compute the total number of errors. If any errors occurred the number of lexical, syntactic and semantic errors should be displayed. If no errors occurred, it should display `Compiled Successfully`. It should return the total number of errors.
2. The `appendError` function should be modified to count the number of lexical, syntactic and semantic errors. The error message passed to it should be added to a queue of messages that occurred on that line.
3. The `displayErrors` function should be modified to display all the error messages that have occurred on the previous line and then clear the queue of messages.

An example of the output of a program with no lexical errors is shown below:

```
   1  // Function with Arithmetic Expression
   2
   3  function main returns integer;
   4  begin
   5       7 + 2 * (5  + 4);
   6  end;

Compiled Successfully
```

Here is the required output for a program that contains more than one lexical error on the same line:

```
   1  // Function with Two Lexical Errors
   2
   3  function main returns integer;
   4  begin
   5       7 $ 2 ? (2 + 4);
Lexical Error, Invalid Character $
Lexical Error, Invalid Character ?
   6  end;

Lexical Errors 2
Syntax Errors 0
Semantic Errors 0
```

You are to submit two files.

1. The first is a `.zip` file that contains all the source code for the project. The `.zip` file should contain the flex input file, which should be a `.l` file, all `.cc` and `.h` files and a `makefile` that builds the project.
2. The second is a Word document (PDF or RTF is also acceptable) that contains the documentation for the project, which should include the following:
    a. A discussion of how you approached the project
    b. A test plan that includes test cases that you have created indicating what aspects of the program each one is testing and a screen shot of your compiler run on that test case
    c. A discussion of lessons learned from the project and any improvements that could be made