Regular Expressions, Lex and Yacc

Problem One : Using Lex

decomment

The C preprocessor is an important part of the C programming system. Given a C source code file, the C preprocessor performs three jobs:

· Merge *physical* lines of source code into *logical* lines. That is, when the preprocessor detects a line that ends with the backslash character, it merges that physical line with the next physical line to form one logical line. More precisely, if the preprocessor detects a backslash character immediately followed by a newline character, then it simply removes both characters.

· Remove comments from ("de-comment") the source code.

· Handle preprocessor directives (`#define`, `#include`, etc.) that reside in the source code.

The second of those jobs — the de-comment job — is more substantial than you might think. For example, when de-commenting a program the C preprocessor must be sensitive to:

· The fact that a comment is a token delimiter. After removing a comment, the C preprocessor must make sure that a white space character is in its place.

· Line numbers. After removing a comment, the C preprocessor sometimes must insert blank lines in its place to preserve the original line numbering.

· String and character literal boundaries. The preprocessor must not consider the character sequence (`/*...*/`) to be a comment if it occurs inside a string literal (`"..."`) or character literal (`'...'`).

Your task is to compose a C program named decomment that performs a subset of the de-comment job of the C preprocessor, as defined below. The operative part of the code should be done using lex and the driver code (decomment) should invoke yylex() as discussed in class.

Your program must be a Linux *filter*. A filter is a program that reads characters from the standard input stream, and writes characters to the standard output stream and possibly to the standard error stream. Specifically, your program must (1) read text, presumably a C program, from the standard input stream, (2) write that same text to the standard output stream with each comment removed, as prescribed below, and (3) write error and warning messages as appropriate to the standard error stream. A typical execution of your program from the shell might look like this:

./decomment < *somefile*.c > *somefileWithoutComments*.c 2> *errorsAndWarnings*

The following bullet points represent the specs of your program.

· Your program must replace each single-line comment with a space.

· Your program must define "comment" as in the C90 standard. In particular, your program must consider text of the form (/*...*/) to be a comment. It must *not* consider text of the form (//...) to be a comment.

· Your program must allow a comment to span multiple lines. That is, your program must allow a comment to contain newline characters. Your program must replace each multi-line comment with a space, followed by newline characters as necessary to preserve the original line numbering.

· Your program must not recognize nested comments.

· Your program must handle C string literals. In particular, your program must not consider text of the form (/*...*/) that occurs within a string literal ("...") to be a comment.

· Similarly, your program must handle C character literals. In particular, your program must not consider text of the form (/*...*/) that occurs within a character literal ('...') to be a comment.

· Your program must handle escaped characters within string literals. That is, when your program reads a backslash (\) while processing a string literal, your program must consider the next character to be an ordinary character that is devoid of any special meaning. In particular, your program must consider text of the form ("...\"...") to be a valid string literal which happens to contain the double quote character.

· Similarly, your program must handle escaped characters within character literals. That is, when your program reads a backslash (\) while processing a character literal, your program must consider the next character to be an ordinary character that is devoid of any special meaning. In particular, your program must consider text of the form ('...\'...') to be a valid character literal which happens to contain the quote character.

· Your program must handle newline characters in C string literals without generating errors or warnings.

· Your program must handle unterminated string and character literals without generating errors or warnings.

· Your program must detect an unterminated comment. If your program detects end-of-file before a comment is terminated, it must write the message "Error: line X: unterminated comment" to the standard error stream. "X" must be the number of the line on which the unterminated comment begins.

· Your program (more precisely, its main function) must return EXIT_FAILURE if it is unsuccessful, that is, if it detects an unterminated comment and so is unable to remove comments properly. Otherwise it must return EXIT_SUCCESS or, equivalently, 0.

· Your program must work for standard input lines of any length.

Examples of each of the bullet points are listed in this link.

In the .l file document please annotate the specific rule that takes care of each of the above bullet points.

Please understand that you have to use lex to complete this assignment.

Problem Two Using Yacc

Postfix to Prefix Conversion

Postfix: An expression is called the postfix expression if the operator appears in the expression after the operands. Simply of the form (operand1 operand2 operator).

Example : AB+CD-* (Infix : (A+B) * (C-D))

Prefix : An expression is called the prefix expression if the operator appears in the expression before the operands. Simply of the form (operator operand1 operand2).

Example : *+AB-CD (Infix : (A+B) * (C-D))

This link explains the three different notations (post/pre/in fix)

Task: Given a Postfix expression, convert it into a Prefix expression.

Conversion of Postfix expression directly to Prefix without going through the process of converting them first to Infix and then to Prefix is much better in terms of computation and better understanding the expression (Computers evaluate using Postfix expression). The task is to implement a parser to read the postfix expression from the "postfix.txt" file & convert it to the equivalent prefix expression. In the process also evaluate the postfix expression. The output (prefix expression as well as result) needs to be stored in a result file. The exercise has to be done using lex and yacc.

Example:

Input : Postfix : ABC/-AK/L-*

Output : Prefix : *-A/BC-/AKL