CS4318/CS5331
Assignment 1: Scanner for `mC`
100 Points
Due: Feb 13, Friday 11:59 PM

Objectives

Write a scanner for `mC` (minimal C) using `lex`

Description

For this class we will be writing a compiler for the `mC` language. The first step in this process is writing a scanner that recognizes tokens in `mC`. The types of tokens that need to be recognized are listed below:

- Identifiers

  Identifiers must begin with an uppercase or lowercase letter. The first letter can be followed by any number of letters or digits. No predetermined restriction should be set on the length of identifiers.

- Reserved words

  Following is a list of reserved words in `mC`. A different token should be returned for each reserved word.

  ```
  if
  else
  while
  int
  string
  char
  return
  void
  ```

- Integer Constants

  An integer constant is an unsigned sequence of digits representing a base-10 number. Leading zeros are not allowed.

- Character Constants

  A character constant is a single character enclosed in single quotes (e.g., `‘a’` is the character constant `a`).

- String Constants

  A string constant is a sequence of characters enclosed in double quotes (e.g., “Hello World”). It is illegal for a string constant to cross line boundaries and your scanner should generate an appropriate error message when it does so. Your scanner also needs to handle escape sequences within string constants. The four escape sequences that need to be handled are :

`\n` for newline
`\t` for tab
`\"` for double quotes
`\\` for the backslash itself

- Operators and Special symbols

  Following is a list of operators and special symbols in `mC`. A separate token needs to be returned for each of these symbols.

  ```
  +
  -
  *
  /
  <
  >
  <=
  >=
  ==
  !=
  =
  [
  ]
  {
  }
  (
  )
  ,
  ;
  ```

- Comments

  Comments are any sequence of symbols enclosed within a pair of symbols /* and */. No token should be generated for comments. But your scanner should generate error messages for unterminated comments (see below).

- Whitespace

  All whitespace should be ignored. Your scanner should do *nothing* when it encounters a newline, a tab or a space.

For each token identified, your scanner should return an integer value aliased to a symbolic token name (the symbolic names are given for you in file tokendef.h). For identifiers, integers, character and string constants your scanner needs to pass additional information on to the next phase. For identifiers, the identifier *name* should be passed and for string constants the string literal should be passed. For integer constants the numeric value of the integer should be passed. For character constants the ASCII value of the character should be passed.

Your scanner needs to generate the following error messages

(1) illegal token
(2) unterminated comment
(3) unterminated string constant

For each error message the scanner should attempt to report the line and column number where the error occurred. For (2) and (3), the line and column number should correspond to the *starting position* of the unterminated comment and string, respectively.

Implementation Instructions

- You may use either `lex` or `flex` to implement your scanner. Alternatively, you can also build a hand-coded scanner. Make sure you provide sufficient documentation so we are able to understand how your hand-coded scanner works
- A driver program that displays the token *types* and *values* to stdout will be provided for you. You should test your scanner with this driver before submitting.
- You should create several test files in `mC` to test each of your flex rules. At this point you may assume the syntax of `mC` is identical to C.
- To avoid duplication, you can construct a global string table that stores a single copy of each unique identifier name and string constant. The attribute for identifiers and string constants that is passed on to the next phase would then be just an integer that represents an index to the string table.
- To pass additional information with identifiers, strings etc., consider using the `yylval` variable in flex
- Create a makefile to build your scanner and other targets and submit this file with your assignment

Submission

Create a README.txt that contains a listing of files required to build your scanner. The README should also contain instructions on building your scanner. Any special comments and known bug information should also go into the README.

Create a tar archive called assg1.yourlastname.tar.gz with the README.txt and all other files required to build your scanner (.l, .h, makefile etc. Do NOT tar the lex library). Submit the tar archive using the assignment submission link on the course web page.

Due dates and times are firm. An assignment submitted one minute past the due date is considered late.