CS4318: Compiler Constructions
Project Phase 1: Scanner for mC
100 Points

Objectives

Write a scanner for mC

Section 1: Description

For this class you will be writing a compiler for the `mC` language. `mC` is a `C` like programming language, but please do not assume it will support everything that `C` does. Please pay attention to the specification of this 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 or Keywords

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. Note that the quotation marks are not part of the string literal. A string literal can contain the following escape sequences:

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

Your scanner should be able to handle these escape sequences and throw the following error if it finds any other escape sequence. Escape sequences start with a `‘\’` .
e.g., if a substring `\b` occurs in a string literal, your program should generate:

```
“<ERROR, Unrecognized escape character in String> : (line number, column number)”.
```

- 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.

- Whitespace

All whitespace should be ignored. Your scanner should take no action 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
(4) Unrecognized escape character in String
(5) String spans multiple lines
```

For each error message the scanner should 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.

Section 2: Sample Input and Output

Provided is a set of sample input and output to test your program. Your program will be tested with these provided test cases and several other hidden test cases to ensure that your scanner is able to recognize legal and illegal tokens correctly.

```
Test case 0: Input:
-------------------
int int
int
int
int int int

Test case 0: Output:
--------------------
<KEYWORD, int> : (1:1)
<KEYWORD, int> : (1:5)
<KEYWORD, int> : (2:1)
<KEYWORD, int> : (3:1)
<KEYWORD, int> : (4:1)
<KEYWORD, int> : (4:5)
<KEYWORD, int> : (4:9)

Test case 1: Input:
------------------
int int
Intmain

Test case 1: Output:
------------------
<KEYWORD, int> : (1:1)
<KEYWORD, int> : (1:5)
<ID, intmain> : (2:1)
```

```
Test case 2: Input:
------------------
" This \" is\\ an \t ok \n string"
printf("Hello world! \t \n \\ \b ");

Test case 2: Output:
-------------------
<STRING, " This " is\ an    ok
string"> : (1:1)
<ID, printf> : (2:1)
<PUNCTUATION, (> : (2:7)
<ERROR, Unrecognized escape character in String> : (2:32)
<PUNCTUATION, )> : (2:35)
<PUNCTUATION, ;> : (2:36)
```

Section 3: Submission Instructions

You can do this project individually or may form a group of two. Please create a zipped folder named "`Project1_<netId1>_<netId2>`" and submit it on canvas.
Please make sure that the folder name is exactly as we have provided. The grading rubric is given below:

Category	Points
Recognizing tokens	45
Detecting illegal comments	10
Detecting illegal strings	10
Escape sequences	10
Correct Error message	10
Documentation	5
Compile and run	5
Write up	5

Submission Deadline: 16th February, Sunday. 11:59 PM
Late Submission Policies: See Syllabus

Section 4: 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. In case you decide to implement the scanner yourself, please update the provided `makefile` so that we can compile using the command “`make`”.
- A header file called “`tokendef.h`” is provided.
- A driver program that displays the token types and values to stdout has been provided for you (`driver.c`). You should test your scanner with this driver before submitting.

Section 5: Writeup

You need to write a short document named `writeup.txt`.
In it, you should include the following (numbered) sections.

1. Your name, course id.
2. If a group project, then who contributed in which part.
3. Provide two examples of how you will test the correctness of your code.
These examples should be different from the ones provided in this document.
4. Extra grading instructions if you choose not to use makefile.

Section 6: Compilation, Testing, and Submission

- Your program should be compiled using the commands
`make clean; make.`
- You are allowed to do otherwise but make sure to add proper instructions in the `writeup.txt` file.
- The makefile for building your scanner is included.
- We have also provided the following skeleton files – `driver.c` and `scanner.l.` We have also provided a supporting header file, `tokendef.h,` that defines all token types.
- During the final submission, please make sure that the latest versions of all of the following files are present in the zipped folder – `driver.c, scanner.l, tokendef.h, makefile,` and `writeup.txt` (described above).