CSL3040: Programming Assignment 1

Submission deadline: Sep 14, 2022

Instructions:

1. Plagiarism is prohibited. It would lead to zero marks for this assignment.
2. This is a group assignment. You have to form groups of two and any one member has to submit. Clearly write the name of the group members in the README file.
3. You should submit a single ZIP file with the following naming convention *B20CSxxx_B20CSxxx_PA1.zip*

Design a basic lexical analyzer in C/C++ to accept the language mentioned below. The analyzer should remove whitespaces from the input code, ignore the text within comments (anything enclosed within /* */) and recognize the tokens. The description of the lexeme patterns are as follows:

- Identifier: A string starting with an underscore or a letter and followed by any number of underscores, letters and digits. Identifiers with two leading underscores (__) are disallowed.
- Keywords: **short int float double bool char signed unsigned for while do return void switch break case continue goto long static union default**
- Signed and unsigned Integer constants such as 45, 10, -1234, +5432, 0, -32, etc.
- Signed and unsigned Floating-point constants such as 1.4, -3.57, 0.72, etc.
- Arithmetic operators: +, -, *, /, %, ++, --
- Assignment operators: =, +=, -=, *=, /=
- Relational operators: <, >, <=, >=, ==
- Special symbols: ( ) ,(comma) [ ] { } ;

In this assignment, which is the first in the series of assignments, you have to submit the following module.

FSM - Hard code the finite state machine description functions for each of the above specified lexeme patterns. Write one function for each pattern which will recognize the corresponding lexeme. Same machine should recognize both the keyword and identifier. The keywords and identifiers are to be entered in a symbol table and stored in the file "**symbol_table_1.out**". The symbol table must contain two fields which are used to store (a) the lexeme itself and (b) the marker (keyword or identifier).

Each of the functions, implementing FSM, reads the next token and does the following:

i. **Return the token ID -** Token ID is a unique number assigned to each token and is specified in the header file **def.h** attached with the assignment. Remember to include this header file as is, in your code.
ii. **Fill up the attributes of the token -** The definition of the attribute structure is as mentioned in the header file. This contains the exact lexeme of the token, in case the token is an Integer or Floating-point constant or an identifier (string).
iii. **Update the symbol table –** Once recognized, an identifier needs to be inserted in the symbol table (mentioned above) if it is not already present.

Make sure that you are able to distinguish between keywords and identifiers in the symbol table. The **“symbol_table_1.out”** file should be updated after each such insert.

A sample input file “**input.in**” is attached with the assignment. Your code should take it as input (redirect input from command line). It is preferred that you read the input into a buffer string and carry out the FSM simulation using this buffer (using Lexeme_Begin and Forward pointers). Simulate each of these FSMs on this input. If all of these FSMs enter a halt state (without accept), then return the error code. Otherwise, return the token which has the longest prefix match with the input.

The output tokens should be written to a file “**pa_1.out**” in the following format:

- Each token’s description is written in a new line in the same order as in the input file. On every line, there is a token ID (as mentioned in the header file), followed by a space, and then the attribute’s value.
  Examples:
  ```
  300 value     //identifier with attribute “value”
  301 234      // integer constant with attribute 234
  302 23.4     // floating constant with attribute 23.4
  303          //% symbol
  ```

  The format in which the symbol table is stored in the file “**symbol_table_1.out**” is as follows:
- Each symbol’s description is written in a new line. On every line, there is a symbol name followed by space and then a number - 0 or 1. The number is 0 if it is a keyword and 1 if it is an identifier.
  Examples:
  ```
  value 1
  switch 0
  ```

Note that “input.in” is just a sample file. Your code should work for any input program. Submit the source code to generate the executable named FSM. The following test procedure will be used to evaluate the assignment.

input.in: Source code on which lexical analysis is to be done.

```
./FSM < input.in
```
a. Initialize symbol table
b. Should read the symbol table from symbol_table_1.out
c. Should output the tokens in pa_1.out
d. Should output the updated symbol table in symbol_table_1.out

Readme file:
Also, submit a Readme file containing instructions to run your programs and to create the executables mentioned above. If you make any assumptions, explain that in details. Remember to stick to the executable and output file names and format mentioned here.