Flex exercise

The program `flex` has been around a very long time, and works quite well — indeed, you still see it used in various places, including compiling the Linux kernel. While there is some legitimate criticism for its use of global variables (and expectation that the user will also use these), it is fast and efficient.

I would like you to create a simple standalone `flex` program named `step1-exercise.flex`. I have a created a skeleton for you to use in this tar file, which also includes a Makefile: step1.tar.

Currently, all the program `step1-exercise` is count lines in a file:

```
$ ./step1-exercise < input-01.lang
There are 15 lines.
$ ./step1-exercise < input-02.lang
There are 20 lines.
```

We would like our `step1-exercise` to actually recognize the elements in our two test files:

```
$ ./step1-exercise < input-01.lang
line 2: PROGRAM
line 2: ID = 'MYPROG'
line 3: LBRACE
line 4: VARIABLES
line 4: COLON
line 5: VAR
line 5: ID = 'VAR1'
line 5: SEMICOLON
line 7: FUNCTIONS
line 7: COLON
line 8: DEFINE
line 8: ID = 'FUNC1'
line 8: LPARENTHESIS
line 8: VAR
line 8: ID = 'X'
line 8: RPARENTHESIS
line 9: LBRACE
line 10: IF
line 10: LPARENTHESIS
line 10: ID = 'X'
line 10: RPARENTHESIS
line 11: THEN
line 12: ID = 'print'
line 12: LPARENTHESIS
line 12: ID = 'X'
line 12: COMMA
line 12: ID = 'X'
line 12: RPARENTHESIS
line 12: SEMICOLON
line 13: ELSE
line 15: RBRACE
line 16: RBRACE
$ ./step1-exercise < input-02.lang
line 2: PROGRAM
line 2: ID = 'MYPROG2'
line 2: SEMICOLON
line 4: VARIABLES
line 5: LBRACE
```

```
line 6: VAR
line 6: ID = 'VAR1'
line 6: SEMICOLON
line 7: VAR
line 7: ID = 'VAR2'
line 7: SEMICOLON
line 8: RBRACE
line 11: FUNCTIONS
line 12: LBRACE
line 13: DEFINE
line 13: ID = 'FUNC1'
line 13: LPARENTHESIS
line 13: VAR
line 13: ID = 'X'
line 13: RPARENTHESIS
line 13: COLON
line 13: ID = 'int'
line 14: LBRACE
line 15: IF
line 15: LPARENTHESIS
line 15: ID = 'X'
line 15: RPARENTHESIS
line 15: THEN
line 15: ID = 'print'
line 15: LPARENTHESIS
line 15: ID = 'X'
line 15: COMMA
line 15: ID = 'X'
line 15: RPARENTHESIS
line 15: SEMICOLON
line 16: ELSE
line 16: ID = 'print'
line 16: LPARENTHESIS
line 16: ID = 'X'
line 16: RPARENTHESIS
line 16: SEMICOLON
line 17: END
line 18: RBRACE
line 20: RBRACE
```

Here is a comprehensive list of possible tokens to recognize:

```
string       token
======       ======
"program"    PROGRAM
"end"        END
"variables"  VARIABLES
"var"        VAR
"functions"  FUNCTIONS
"define"     DEFINE
"statements" STATEMENTS
"if"         IF
"then"       THEN
"else"       ELSE
"while"      WHILE
","          COMMA
"("          LPARENTHESIS
")"          RPARENTHESIS
"{"          LBRACE
"}"          RBRACE
```

```
":"           COLON
";"           SEMICOLON
[a-zA-Z0-9]+ ID
```

Also, make sure to set the variable `yylval` for your return value so that you can print the actual string found for your tokens identified as ID tokens.

Your rules should not do any output; instead, you will want to write a loop around using "`yylex()`" repeatedly to get the next token; your output should be done in that main loop rather than trying to have your rules do the output.

Submission: Please submit a new `step1.tar` file on Canvas that has the original contents modified to do the correct recognition of the tokens in the test input files. Your submission is due by 11:59pm on Sunday, September 22.