

Compiler Design

Assignment – 4

K. Sai Vignesh

AP21110010958 (CSE-O)

1. ****Regular Expressions:****

- ``digit [0-9]*``: This defines a regular expression for matching zero or more digits.
- ``letter [a-zA-Z]``: This defines a regular expression for matching a single letter.
- ``id {letter}({letter}|{digit})*``: This defines a regular expression for matching identifiers. An identifier starts with a letter and can be followed by zero or more letters or digits.
- ``int_num {digit}+`, `uint_num 0|({int_num})``: These regular expressions define signed and unsigned integers. An unsigned integer can be either 0 or a sequence of one or more digits.

2. ****Floating-Point Numbers:****

- ``float_num ({int_num}\.{digit}+)|({int_num}\.)(\.{digit}+)``: This regular expression matches different forms of floating-point numbers. It can be an integer part followed by a decimal point and one or more digits (``{int_num}\.{digit}+``), an integer part followed by just a decimal point (``{int_num}\.``), or just a decimal point followed by one or more digits (``\.{digit}+``).

3. ****Exponential Notation:****

- ``exp_num ({int_num}|{float_num})[eE][+-]?{int_num}``: This regular expression matches numbers in exponential notation. It can be an integer or float part followed by 'e' or 'E', an optional '+' or '-', and then one or more digits.

4. ****Tokens and Actions:****

- The section after the ``%%`` delimiter contains rules for recognizing various tokens.
- For example, ``"/`` is a pattern to match a double forward slash, and the action ``{scom=1;}`` sets the single-line comment flag to 1.
- Keywords, relational operators, assignment operator, etc. are recognized based on the provided patterns.

5. ****Ignoring Comments:****

- ``/* ... */`` style comments are ignored using the rules for ``"/*`` and ``*/``.
- Single-line comments are ignored using the rule for ``"/``.

6. ****Printing and Storing:****

- When a token is recognized, it prints a message to the output file ('yyout') indicating the type of the token.
- Identifiers are also stored in the symbol table ('st') if they haven't been encountered before.

7. ****Main Function:****

- 'main()' opens the input and output files, calls 'yylex()' to start the lexical analysis, and then prints the contents of the symbol table.

8. ****`look_up` Function:****

- This function checks if a given identifier ('id') is already in the symbol table.

9. ****`yywrap` Function:****

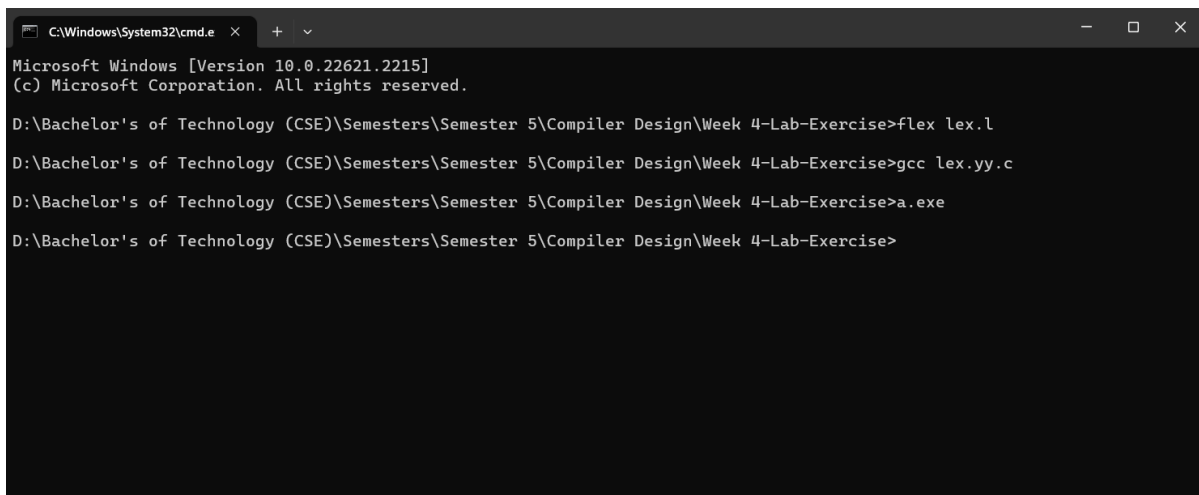
- This function is used to indicate the end of input.

10. ****File Handling:****

- The program reads from a file named 'x.txt' and writes to a file named 'y.txt'.

This Lex program will tokenize the input based on the specified rules and print the results to 'y.txt'. The program also maintains a symbol table and handles different types of numeric constants and identifiers as per the provided regular expressions.

Commands to run the program :



```
C:\Windows\System32\cmd.e  x  +  v
Microsoft Windows [Version 10.0.22621.2215]
(c) Microsoft Corporation. All rights reserved.

D:\Bachelor's of Technology (CSE)\Semesters\Semester 5\Compiler Design\Week 4-Lab-Exercise>flex lex.l
D:\Bachelor's of Technology (CSE)\Semesters\Semester 5\Compiler Design\Week 4-Lab-Exercise>gcc lex.yy.c
D:\Bachelor's of Technology (CSE)\Semesters\Semester 5\Compiler Design\Week 4-Lab-Exercise>a.exe
D:\Bachelor's of Technology (CSE)\Semesters\Semester 5\Compiler Design\Week 4-Lab-Exercise>
```