Experiment No - 3

Aim: Implement following Programs Using Lex
a. Convert Roman to Decimal
b. Check weather given statement is compound or simple
c. Extract html tags from .html file

Date:

Competency and Practical Skills: Understanding of Lex tool and its usage in compiler design, understanding of regular expressions and data structures, improving programming skill to develop programs using lex tool

Relevant CO: CO1

Objectives:
1. To introduce students to Lex tool and its usage in compiler design
2. To provide practical knowledge of regular expressions and their use in pattern matching
3. To enhance students' understanding of data structures such as arrays, lists, and trees
4. To develop students' problem-solving skills in developing and implementing programs using Lex tool
5. To develop students' debugging skills to identify and resolve program errors and issues

Software/Equipment: Computer system, Text editor, Lex tool, C compiler, Terminal or Command prompt.

Theory:

❖ **LEX:**
- Lex is a program that generates lexical analyzers. It is used with a YACC parser generator.
- The lexical analyzer is a program that transforms an input stream into a sequence of tokens.
- It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.
- During the first phase the compiler reads the input and converts strings in the source to tokens.
- With regular expressions we can specify patterns to lex so it can generate code that will allow it to scan and match strings in the input. Each pattern specified in the input to lex has an associated action.
- Typically an action returns a token that represents the matched string for subsequent use by the parser. Initially we will simply print the matched string rather than return a token value.

O Function of LEX:
- Firstly lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and produces a C program lex.yy.c.
- Finally C compiler runs the lex.yy.c program and produces an object program a.out.
- a.out is a lexical analyzer that transforms an input stream into a sequence of tokens.

O LEX File Format:
- A Lex program is separated into three sections by %% delimiters. The formal of Lex source is as follows: %{ definitions %}
  %%
  { rules }
  %%
  { user subroutines }
- Definitions include declarations of constant, variable and regular definitions.

- Rules define the statement of form p1 {action1} p2 {action2}. pn {action}.
- Where pi describes the regular expression and action1 describes the actions the lexical analyzer should take when pattern pi matches a lexeme.
- User subroutines are auxiliary procedures needed by the actions. The subroutine can be loaded with the lexical analyzer and compiled separately.

❖ **FLEX: Fast Lexical Analyzer Generator**

- FLEX is a tool/computer program for generating lexical analyzers (scanners or lexers) written by Vern Paxson in C around 1987. It is used together with Berkeley Yacc parser generator or GNU Bison parser generator. Flex and Bison both are more flexible than Lex and Yacc and produce faster code.
- Bison produces parsers from the input file provided by the user. The function yylex() is automatically generated by the flex when it is provided with a .l file and this yylex() function is expected by parser to call to retrieve tokens from current/this token stream.

STEPS:

- Step 1 : An input file describes the lexical analyzer to be generated named lex.l is written in lex language. The lex compiler transforms lex.l to C program, in a file that is always named lex.yy.c.
- Step 2 : The C compiler compile lex.yy.c file into an executable file called a.out.
- Step 3 : The output file a.out take a stream of input characters and produce a stream of tokens.
- Program Structure:

In the input file, there are 3 sections:
Definition Section: The definition section contains the declaration of variables, regular definitions, and manifest constants. In the definition section, text is enclosed in "%{ %}" brackets. Anything written in this brackets is copied directly to the file lex.yy.c Syntax:

```
%{
// Definitions
%}
```

Rules Section: The rules section contains a series of rules in the form: pattern action and pattern must be unintended and action begin on the same line in {} brackets. The rule section is enclosed in "%% %%". Syntax:

```
%%
pattern action
%%
```

User Code Section: This section contains C statements and additional functions. We can also compile these functions separately and load with the lexical analyzer.

How to run the program:
To run the program, it should be first saved with the extension .l or .lex. Run the below commands on terminal in order to run the program file.

- Step 1: lex filename.l or lex filename.lex depending on the extension file is saved with name.l extension.
- Step 2: gcc lex.yy.c
- Step 3: ./a.out
- Step 4: Provide the input to program in case it is required

Suggested Reference:

1. Aho, A.V., Sethi, R., & Ullman, J.D. (1986). Compilers: Principles, Techniques, and Tools. Addison-Wesley.
2. Levine, J.R., Mason, T., & Brown, D. (2009). lex & yacc. O'Reilly Media, Inc.
3. Lex - A Lexical Analyzer Generator. Retrieved from https://www.gnu.org/software/flex/manual/
4. Lexical Analysis with Flex. Retrieved from https://www.geeksforgeeks.org/flex-fast-lexicalanalyzer-generator/
5. The Flex Manual. Retrieved from https://westes.github.io/flex/manual/

Rubric wise marks obtained:

Rubrics	Understanding of Lex tool (2)		Problem Recognition (2)		Logic Building (2)		Completeness and accuracy (2)		Ethics (2)		Total
	Good (2)	Avg. (1)	Good (2)	Avg. (1)	Good (2)	Avg. (1)	Good (2)	Avg. (1)	Good (2)	Avg. (1)	
Marks											