**Experiment No. 4 Date: 26/10/2020**

**LEX**

**Aim:** To study LEX and write LEX programs to:

1. Compute average of a given set of numbers.

2. Implement Caesar cipher: it replaces every input character with the one, three

letters after it in alphabetical order wrapping around at ‘z’.

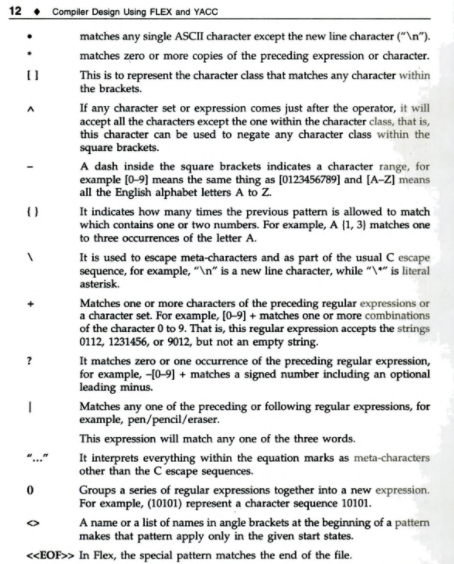
3. Validate the tokens: Person Name, Office Address, Bank Account Number

4. Validate identifier, numeric constant, keywords, operator, parenthesis, header.

5. Find the longest word (defined as a contiguous string of upper-case and lower-case letters) in the given input

**Theory:**

The characters that form regular expressions are:



**Reference:** Vinu V. Das ; Compiler design with FLEX and YACC; PHI publication, ISBN:978- 81-203-3251-5

yywrap() – when the scanner receives an end-of-file indication it checks the yywrap() function.

If yywrap() returns false then the function has gone ahead and set yyin to another input file and scanning continues

If yywrap() returns true then the scanner terminates, returning 0 to its caller (default)

Can provide own version of yywrap() or use % option noyywrap (scanner behaves as though yywrap() returned 1.

CONTEXT SENSITIVITY.  LEX provides some support for contextual grammatical rules.

* If ^ is the first character in an expression, then this expression will only be matched at the beginning of a line.
* If $ is the last character in an expression, then this expression will only be matched at the end of a line.
* If *r* and *s* are two LEX regular expressions then *r*/*s* is another LEX regular expression.
  1. It matches *r* if and only if it is followed by an *s*.
  2. It is called a *trailing context*.
  3. After use in this context, *s* is then returned to the input before the action is executed. So the action only sees the text matched by *r*
  4. *Left context* is handled by means of *start conditions.*

Ref: <https://www.csd.uwo.ca/~mmorenom/CS447/Lectures/Lexical.html/node11.html>

**PROGRAMS:**

1. **Compute average of a given set of numbers.**

root@kali:~# vi expt4prgm1

%{

#include<stdio.h>

int count=0, sum=0, num=0;

float avg=0;

%}

%%

[0-9]+ {

num= atoi(yytext);

sum+=num;

count++;

}

\n {

printf("Sum = %d, Average = %f\n",sum,(double)sum/count);

return 0;

}

%%

int main()

{ yylex();

return 0;

}

int yywrap(){}

root@kali:~# lex expt4prgm1

root@kali:~# gcc lex.yy.c

root@kali:~# vi expt4prgm1

root@kali:~# gcc lex.yy.c

**OUTPUT:**

root@kali:~# ./a.out

1 2 3 4 5 6 7 8 9

Sum = 45, Average = 5.000000

root@kali:~#

1. **Implement Caesar cipher: it replaces every input character with the one, three letters after it in alphabetical order wrapping around at ‘z’.**

root@kali:~# vi expt4pgm2

%{

//program ceasar cipher

%}

%%

[a-z] {

char ch=yytext[0];

ch+=3;

if(ch>'z')

ch='z'+1-'a';

printf("%c",ch);

}

[A-Z] {

char ch = yytext[0];

ch += 3;

if (ch > 'Z')

ch-=('Z'+1-'A');

printf ("%c", ch);

}

%%

int main()

{

yylex();

return 0;

}

int yywrap(){}

root@kali:~# lex expt4pgm2

root@kali:~# gcc lex.yy.c

**OUTPUT:**

root@kali:~# ./a.out

prajwal

sudmzdo

1. **Validate the tokens: Person Name, Office Address, Bank Account Number**

root@kali:~# vi expt4pgm3

%{

//to validate tokens

#include<string.h>

%}

%%

[0-9]{10} {printf("\n%s is a Bank Account Number\n",yytext);}

[a-zA-Z]+([][a-zA-Z]\*){2} {printf("\n%s is Person's Name\n",yytext);}

[a-zA-Z0-9\-,]+([][a-zA-Z0-9\-,]\*)\* {printf("\n%s is a Office Address\n",yytext);}

.|\n

%%

int main()

{

yylex();

return 0;

}

int yywrap(){}

root@kali:~# lex expt4pgm3

root@kali:~# gcc lex.yy.c

**OUTPUT:**

root@kali:~# ./a.out

1234567890

1234567890 is a Bank Account Number

prajwal

prajwal is Person's Name

Goa40

Goa40 is a Office Address

1. **Validate identifier, numeric constant, keywords, operator, parenthesis, header.**

root@kali:~# vi expt4pgm4

%{

//lex program to validate identifiers.

#include<stdio.h>

%}

%%

if|else|for|while|do|int|main|return {printf("\n<%s, keyword>",yytext);}

#include<.\*> {printf("\n<%s, preprocessing-directive>",yytext);}

[a-zA-Z\_][a-zA-Z0-9\_]\* {printf("\n<%s, identifier>",yytext);}

-?[0-9]+ {printf("\n<%s, Numeric constant>",yytext);}

[+|-|\*|/|%] {printf("\n<%s, operator>",yytext);}

[{|}|(|)] {printf("\n<%s. parenthesis>",yytext);}

.\* {printf("\n<%s, other>",yytext);}

%%

int main()

{

yylex();

return 0;

}

int yywrap(){}

root@kali:~# lex expt4pgm4

root@kali:~# gcc lex.yy.c

**OUTPUT:**

root@kali:~# ./a.out

if

<if, keyword>

#include<stdio.h>

<#include<stdio.h>, preprocessing-directive>

var

<var, identifier>

123

<123, Numeric constant>

+

<+, operator>

{

<{. parenthesis>

123abc

<123abc, other>

1. **Find the longest word (defined as a contiguous string of upper-case and lower-case letters) in the given input**

root@kali:~# vi expt4pgm5

%{

//lex program to find the longest word(defined as contigious string of upper and lower case letters)//in the given input.

#include<string.h>

int longest=0;

char longestWord[100];

%}

%%

[a-zA-Z]+ {

if(yyleng>longest)

{

longest =yyleng;

strcpy(longestWord,yytext);

}

}

[ | |\n|^|,]\*;

%%

int main()

{

extern FILE \*yyin;

yyin=fopen("prajwal4.txt","r");

yylex();

printf("Longest word is \"%s\" of length %d.\n",longestWord,longest);

return 0;

}

int yywrap(){}

root@kali:~# lex expt4pgm5

root@kali:~# gcc lex.yy.c

root@kali:~# cat prajwal4.txt

root@kali:~# cat> prajwal4.txt

prajwal cassandramaam

^Z

[9]+ Stopped cat > prajwal4.txt

**OUTPUT**

root@kali:~# ./a.out

Longest word is "cassandramaam" of length 13.

**Conclusion**: Lex Programs were Successfully Implemented