Installation of software, go through flex documentation.

# The structure of LEX programs

A LEX program consists of three sections Declarations, Rules and Auxiliary functions.

DECLARATIONS

%% RULES

%%

AUXILIARY FUNCTIONS

**Declaration**:

The declarations section consists of two parts, *regular definitions* and *auxiliary declarations*. LEX allows the use of short-hands and extensions to regular expressions for the regular definitions. The auxiliary declarations are copied as such by LEX to the output lex.yy.c file.

Example:

%{

#include<stdio.h>

int global\_variable; //Auxiliary declarations

%}

number [0-9]+ //Regular definitions

op [-|+|\*|/|^|=]

%%

/\* Rules \*/

%%

/\* Auxiliary functions \*/

A regular definition in LEX is of the form : D R

where D is the symbol representing the regular expression R. The auxiliary declarations (which are optional) are written in C language and are enclosed within ' %{ ' and ' %} ' . It is generally used to declare functions, include header files, or define global variables and constants.

## Rules

Rules in a LEX program consists of two parts :

* + 1. The pattern to be matched
    2. The corresponding action to be executed

Example:

/\* Declarations\*/

%%

{number} {printf(“ number”);}

{op} {printf(“ operator”);}

%%

/\* Auxiliary functions \*/

The pattern to be matched is specified as a regular expression.

**Sample Input/Output for the above example:**

I/P: 234

O/P: number

I/P: \*

O: operator

I/P: 2+3

O/P: number operator number

LEX obtains the regular expressions of the symbols number and op from the declarations section and generates code into a function yylex() in the lex.yy.c file. This function checks the input stream for the first match to one of the patterns specified and executes code in the action part corresponding to the pattern.

## Auxiliary functions

LEX generates C code for the rules specified in the Rules section and places this code into a single function called yylex(). (To be discussed in detail later). In addition to this LEX generated code, the programmer may wish to add his own code to the lex.yy.c file. The auxiliary functions section allows the programmer to achieve this.

Example:

/\* Declarations \*/

%%

/\* Rules \*/

%%

int main()

{

yylex(); return 1;

}

The C code in the auxiliary section and the declarations in the declaration section are copied as such to the lex.yy.c file.

Q1: Display specified output for a specified input.

Solution:

%{

#include <stdio.h>

%}

%%

"hello world" printf("GOODBYE!");

\n {return 0;}

%%

int yywrap()

{

}

int main()

{

yylex();

return 0;

}

Q2. Write a lex program that count number of spaces in a given string e.g,

**Input[1]: h e l l o**

**Output[1]: spaces = 4**

**Input[2]: hello world**

**Output[2]: spaces = 1**

Solution:

%{

int space\_count = 0;

%}

%%

" " { space\_count++; }

\n { printf("spaces = %d\n", space\_count); space\_count = 0; }

. /\* ignore other characters \*/

%%

int yywrap()

{

}

int main() {

yylex();

return 0;

}

Q3. Count number of words in a given string.

**Input[1]: hello world**

**Output[1]: words = 2**

**Input[2]: hello world, I'm Lex**

**Output[2]: words = 4**

Solution:

%{

int word\_count = 0;

%}

%%

[a-zA-Z]+ { word\_count++; }

\n { printf("words = %d\n", word\_count); word\_count = 0; }

. /\* ignore other characters \*/

%%

int yywrap()

{

}

int main() {

yylex();

return 0;

}

Q4: count the number of vowels and consonents in a given string.

**Input[1]: Hello everyone**

**Output[1]: Number of vowels are: 6**

**Number of consonants are: 7**

**Input[2]: This is GeeksforGeeks**

**Output[2]: Number of vowels are: 7**

**Number of consonants are: 12**

Solution:

%{

int vowel\_count = 0;

int consonant\_count = 0;

%}

%%

[AaEeIiOoUu] { vowel\_count++; }

[BbCcDdFfGgHhJjKkLlMmNnPpQqRrSsTtVvWwXxYyZz] { consonant\_count++; }

\n {

printf("Number of vowels are: %d\n", vowel\_count);

printf("Number of consonants are: %d\n", consonant\_count);

vowel\_count = 0;

consonant\_count = 0;

}

. /\* ignore other characters \*/

%%

int yywrap()

{

}

int main() {

yylex();

return 0;

}

**Task:**

**Q1: Count words that are less than 10 and greater than 5 in a given string.**

**Input[1]: geeksforgeeks hey google test lays**

**Output[1]: words = 1**

**Input[2]: hello world, I'm Lex**

**Output[2]: words = 0**

Solution:

%{

int word\_count = 0;

%}

%%

[a-zA-Z]{6,9} { word\_count++; }

\n { printf("words = %d\n", word\_count); word\_count = 0; }

. /\* ignore other characters \*/

%%

int yywrap()

{

}

int main() {

yylex();

return 0;

}