

# **UNIT-5. LEX and YACC.**

## **Part- A -Lex Programming**

**By Dr. M M Math,  
Professor and Head CC  
Department of CSE,  
GIT, Belagavi**

# **A LEX Tool**

- I. Introduction.**
- II. Structure of LEX program Specification**
- III. Prerequisite for Writing LEX program**
- IV. Running LEX program.**
- v. USE of VI editor.**
- VI. Sample LEX programming exercises .**

# I. Introduction

## ● What is LEX ?

- LEX is a programming tool on UNIX platform that takes Lex input file specification ( which are set of descriptions of **Tokens** in the form **Regular expression-Pattern**) as an input and generates a **C-routine (lex.yy.c)** which is termed as *lexical analyzer or Lexer or scanner*.

## ● What are Tokens ?

- Are nothing but the primitives of any programming language that includes **Identifiers, constants, keywords, operators** and **punctuation symbols**,

## ● What are REGULAR Expressions ?

- It is a mechanism to describe the tokens that consists of combinations of symbols from the **alphabet, digits, operators and special symbols called meta-characters**

## ● What is Lexical analyzer ?

- **It is** an initial program for a typical compiler that reads the input streams of character by character and groups them into a meaningful sequence called as tokens.

## ● Who developed LEX Tool ?

- It is Written and developed by **Eric Schmidt and Mike Lesk.**

## ● What are applications of LEX ? :

- Mainly used by compiler writers to write compiler and interpreter.
- Any application that looks for patterns in its inputs

## II. Structure of LEX program Specification

- The Structure of Lex program specification consists of three sections, as depicted in the following figure.

```
%{  
    SECTION -1    { Definition section }  
}%  
<Name> <pattern> Definition of long pattern  
-optional  
%%  
    SCETION –II    { RULES Sections }  
%%  
SECTION – III { User subroutine section }
```

# A-Definition section :

- It introduces any initial C program code like header files, declaration and initialization of variables used in the program. This is simply copied to the C-code being generated.
- The C program code is surrounded with two special delimiters **%{** and **%}**
- It also contains declarations of simple definitions for patterns to simplify the scanner specification which is of the following form

**name definition → Optional**

- Example:

**DIGIT [0-9]+**

**LETTER [a-zA-Z]**

**NOTE : It is optional it can be used whenever long or complex patterns are to be simplified.**

# B. Rules section

- The **Rules section** of the Lex program specifications contains a series of rules where each rule is made up of two parts separated by whitespaces and is as follows :

- *<pattern> <action>*

*<Pattern>* → is a description of tokens in the form of Regular expressions written in UNIX style or regular name definition specified in the Definition section.

**Example:**

```
{DIGIT}+      {printf(" It is number :  
                %s\n",yytext ); }  
[0-9]+        {printf(" iT is number :  
                %s\n",yytext ); }
```

- **<action>** is made of action routines written using C- language and are enclosed between curly braces { and }.
- It also consists of call statements to c routines written in user subroutine section of LEX specification in section III.
- An **<action>** consisting only a vertical bar ('|') means the action is same as the action for the next rule."

**Example :**

```
{DIGIT} { printf(" IT is number :  
                                %s\n",yytext ); }  
[0-9]+   { printf(" IT is number :  
                                %s\n",yytext ); }
```



## C.User Subroutine Section

- This is a final section which consists of **any legal C-code** that is copied to the end of the Code generated.
- It also consists of **user defined c-functions** which may be called in the action part of the rule section.
- Finally it includes the **main() function** consisting of a call statement to **yylex()**, where **yylex()** is **c-routine generated by LEXER**.
- Unless the action contains explicit return statement, **yylex()** would not return until it has processed the entire input.

### III. Prerequisite for Writing LEX program

- **Study of Regular expression :**

As a part of rule section the pattern is made up of regular expressions written in UNIX style.

The **regular expression** is a **string** made up of **symbols from alphabets, digits, operators** and **special symbols** called **meta-characters**.

- Alphabets  $\rightarrow \{ a,b,c....z,A,B,C,....Z \}$
- Digits  $\rightarrow \{ 0,1,2,...9 \}$
- Operators and Special symbols  $\rightarrow$

" \ [ ] ^ - ? . \* + | ( ) \$ / { } %

# Lex- Pattern Matching Primitives or Meta Characters

Metacharacter	Matches
.	any character except newline
\n	newline
*	zero or more copies of the preceding expression
+	one or more copies of the preceding expression
?	zero or one copy of the preceding expression
^	beginning of line
\$	end of line
a b	a or b
(ab) +	one or more copies of ab (grouping)
"a+b"	literal "a+b" (C escapes still work)
[]	character class

# Example on pattern matching

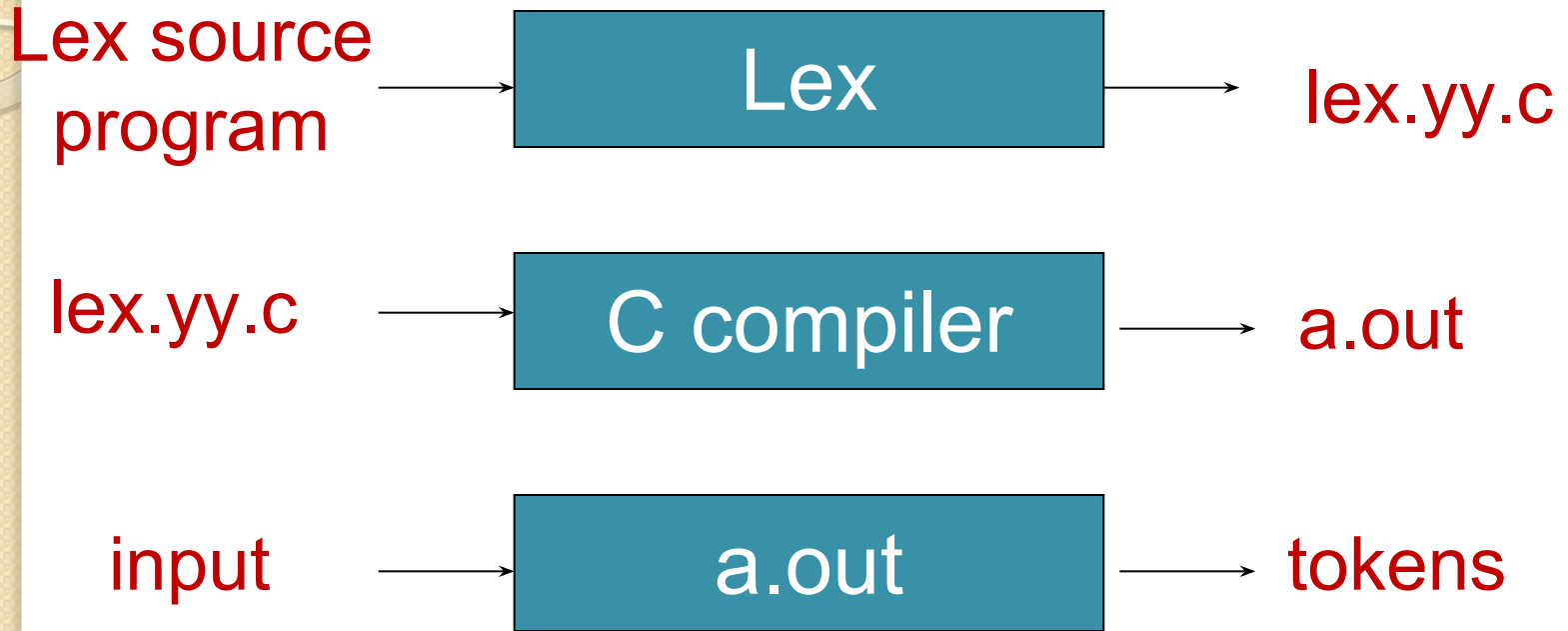
Expression	Matches
abc	abc
abc*	ab abc abcc abccc ...
abc+	abc abcc abccc ...
a(bc) +	abc abcbcb abcbcbcb ...
a(bc) ?	a abc
[abc]	one of: a, b, c
[a-z]	any letter, a-z
[a\ -z]	one of: a, -, z
[-az]	one of: -, a, z
[A-Za-z0-9] +	one or more alphanumeric characters
[ \t\n] +	whitespace
[^ab]	anything except: a, b
[a^b]	one of: a, ^, b
[a b]	one of: a,  , b
a b	one of: a, b

- Pattern Matching examples.

# Lex predefined variables.

Name	Function
<code>int yylex(void)</code>	call to invoke lexer, returns token
<code>char *yytext</code>	pointer to matched string
<code>yylen</code>	length of matched string
<code>yyval</code>	value associated with token
<code>int yywrap(void)</code>	wrapup, return 1 if done, 0 if not done
<code>FILE *yyout</code>	output file
<code>FILE *yyin</code>	input file
<code>INITIAL</code>	initial start condition
<code>BEGIN</code>	condition switch start condition
<code>ECHO</code>	write matched string

## IV. Running a Lex program



**\$lex xyz.l**

→ **Command for LEX Complilation**

**\$ cc lex.yy.c -ll**

→ **Command for C - Compiler**

**\$/a.out**

→ **Command for Execution**

# V. USE of VI editor.

- VI editor is screen-oriented text editor that enables you to edit lines in context with other lines in the file.
- It is considered to be standard editor for unix programming
- To start using vi editor – we can execute the following commands

Command	Description
<b>vi filename</b>	Creates a new file if it already does not exist, otherwise opens existing file
<b>vi -R filename</b>	Opens an existing file in read only mode.
<b>view filename</b>	Opens an existing file in read only mode..



# Operation Modes

- While working with vi editor you would come across following two modes –
- **Command mode** – This mode enables you to perform administrative tasks such as saving files, executing commands, moving the cursor, cutting (yanking) and pasting lines or words, and finding and replacing. In this mode, whatever you type is interpreted as a command.
- **Insert mode** – This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and finally it is put in the file .
- **NOTE :** The vi always starts in command mode. To enter text, you must be in insert mode. To come to in insert mode you simply type **i**. To get out of insert mode, press the **Esc** key, which will put you back into command mode.




# Getting Out of vi-editor

- The command to quit out of vi is  
**ESC:w** → **This to save the file**  
**:wq** → **This to Save and exit**  
**:q!** → **This to Exit without saving**

## VI. Sample LEX programs

1. **Write a LEX program to recognize the verbs of an English language sentence**
2. **Write a LEX program to count the number vowels and consonants in the given English language sentence.**
3. **Write a LEX program to count the number positive and negative integer numbers in a given list of numbers.**
4. **Write a LEX program recognize a simple or compound sentence of an english language.**
5. **Write a LEX program to count number of identifiers, constants and operators present in an arithmetic expression.**

- 
- 6. Write LEX program to count number of integers and float point numbers in a given list of numbers.**
  - 7. Write a LEX program to count number of comments in a C-program**
  - 8. Write a LEX program to check the entered string is an Identifier or keyword of C-language.**
  - 9. Write a LEX program to Count the number of words, lines and Characters in a given English Text.**

**10. Write a LEX program to identify Strings associated with the following Regular language on the alphabet -  $\Sigma = \{a, b\}$ .**

**A.  $a(a+b)^*$ .**

**B.  $b^* a b^*$**

**C.  $ab(a+b)^*$**

**D.  $(a+b)^* abb$**

**E.  $L = \{|w| \bmod 3 = 0, \text{ where } w \in (a+b)^*\}$**

## Lex program for Example -1

```
%{
#include <stdio.h>
%}
%%
[ \t]+  {;}
is |
was |
am |
are |
were |
being |
do |
did |
does { printf("\n %s is verb ",yytext);}
[a-zA-z]+ { printf("\n %s is not a verb",yytext);}
. |
\n {;}
%%
main()
{
    printf("\n Enter Text : ");
    yylex();
}
```

## Lex program for Example -2

```
%{
#include <stdio.h>
int vowel=0;
int consonent=0;
%}
%%
[ \t]+    {;}
[aeiouAEIOU] { vowel= vowel+1;}
[a-zA-z] { consonent= consonent+1;}
. |
\n    {;}
%%
main()
{
    printf("\n Enter Text : ");
    yylex();
    printf(" Number of Vowels : %d \n",vowel);
    printf(" Number of consonent : %d \n",consonent);
    return 0;
}
```

### Lex program for Example -3

```
%{  
#include <stdio.h>  
int p=0, n=0;  
%}  
%%  
[ \t]+ {;}  
\+?[0-9]+ { p++;}  
\-[0-9]+ { n++;}  
\n {;}  
%%  
main()  
{  
    printf("Enter numbers : ");  
    yylex();  
    printf("Positive Numbers : %d \n Negative Numbers : %d ",p,n);  
}
```

## Lex program for Example -4

```
%{
#include <stdio.h>
int comp=0;
}%
%%
[ \t]+ {;}
and |
but |
or |
neither |
either {comp=1;}
[a-zA-Z]+ {;}
. |
\n {;}
%%
main()
{
    printf("Enter a Sentence : ");
    yylex();
    if(comp)
        printf("\n Sentence is Compound statement\n",yytext);
    else
        printf("\n Sentence is not a Compound statement\n",yytext);
}
```



## Lex program for Example -5

```
%{
#include <stdio.h>
int id=0, con=0, op=0;
}%
%%
[ \t]+      ;
[+ \- * /] { op++;}
[0-9]+      { con++;}
[a-zA-z][a-zA-z0-9]* { id++;}
. | \n      ;
%%
main()
{
    printf("Enter expression : ");
    yylex();
    printf("Total Number of operators   : %d \n",op);
    printf("Total Number of Identifiers : %d \n",id);
    printf("Total Number of constants  : %d \n",con);
}
```

## Lex program for Example -6

```
%{
#include <stdio.h>
int i=0, d=0;
}%
%%
\+?[0-9]+    { printf("\n %s is an integer",yytext);i++;}
\+?[0-9]+\.[0-9]+ { printf("\n %s is a Decimal ",yytext);d++;}
%%
main()
{
    printf("Enter numbers : ");
    yylex();
    printf("Total Int Numbers : %d \n",i);
    printf("Total Decimal Numbers : %d ",d);
}
```

## Lex program for Example -7

```
%{  
#include <stdio.h>  
int comment=0;  
%}  
%%  
"/*.*/" {comment++;}  
\n      {;}  
%%  
main()  
{  
    printf(" Read C-program file :");  
    yylex();  
    printf("\nnumber of comments : %d \n",comment);  
    return 0;  
}
```

## Lex program for Example -8 is an exercise

## Lex program for Example -9

```
%{
#include <stdio.h>
int wc=0, lc=0, ch=0;
}%
%%
[^\t\n]+ {wc++; ch+=yyleng;}
\n      {lc++;}
.       {ch++;}
%%
main()
{
    printf("\n Enter an English text : " );
    yylex();
    printf("\nNumber of words %d :",wc);
    printf("\nNumber of lines %d :",lc);
    printf("\nNumber of words %d :",ch);
}
```

## Lex program for Example - 10 - A

```
%{
#include <stdio.h>
int flag=0;
}%
%%
[ \t]+    {;}
[ab]*[a][ab]*  { flag =1;}
[ab]*      { flag =0;}
. |
\n        {;}
%%
main()
{
    printf("Enter string :");
    yylex();
    if (flag)
        printf(" \nvalid \n");
    else
        printf(" \ninvalid ----- \n");
    return 0;
}
```

## Lex program for Example - 10 - B and C are for Exercise

### Lex program for Example - 10 - D

```
%{
#include <stdio.h>
int flag=0;
%}
%%
[ \t]+      {;}
[ab]*abb    { flag =1;}
[ab]*       { flag =0;}
. |
\n          {;}
%%
main()
{
    printf("Enter string :");
    yylex();
    if (flag)
        printf(" \ninvalid \n");
    else
        printf(" \nninvalid ----- \n");
    return 0;
}
```

## Lex program for Example - 10 - E

```
%{
#include <stdio.h>
int flag=0;
}%
%%
[ \t]+      {;}
([ab][ab][ab])*  { flag =1;}
[ab]*       { flag =0;}
. |
\n          {;}
%%
main()
{
    printf("Enter string :");
    yylex();
    if (flag)
        printf(" \nvalid \n");
    else
        printf(" \ninvalid ----- \n");
    return 0;
}
```



**END of LEX Programming.**