Lex & Yacc

By Dr. M M Math, Professor , 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 programs.

I. Introduction

- **What is LEX ?**
  - LEX is a programming tool on UNIX platform that takes Lex input file specification (which are set of descriptions ***Tokens*** in the form **Regular expression**) that generates a C-routine (lex.yy.c) which is termed as ***lexical analyzer or Lexer or scanner.***
- **Tokens :** Are nothing but the primitives of any programming language that includes ***Identifiers, constants, keywords, operators and punctuation symbols***,
- **REGULAR Expression :** 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
- **Lexical analyzer :** **I**s a initial program for a typical compiler that reads the input streams character by Character and groups them into a meaningful sequence called as a tokens.
- **LEX Tool** is Written and developed by ***Eric Schmidt*** and ***Mike Lesk.***

- **Applications of LEX :**

  1. Mainly used by compiler writers to write compiler and interpreter.

  2. 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, separated by a line with just %% in it:

```
%{
    SECTION -I      { Definition section }
%}
%%
    SCETION –II     { RULES Sections  }
%%
SECTION – III  { User subroutine section }
```

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 generated C-code.
- The C program code is surrounded with two special delimiters %{ and %}
- It also contains declarations of simple name definitions 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.

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 of a vertical bar ('|') means "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 ); }
```

III. 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 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.

IV. Prerequisite for Writing LEX program

- Study of Regular expression :

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

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

  Alphabets { a,b, c…z,A,B,C….Z }

  digits {0,1,2…9}

  operators and Special symbols :

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

Lex- Pattern Matching Primitives

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

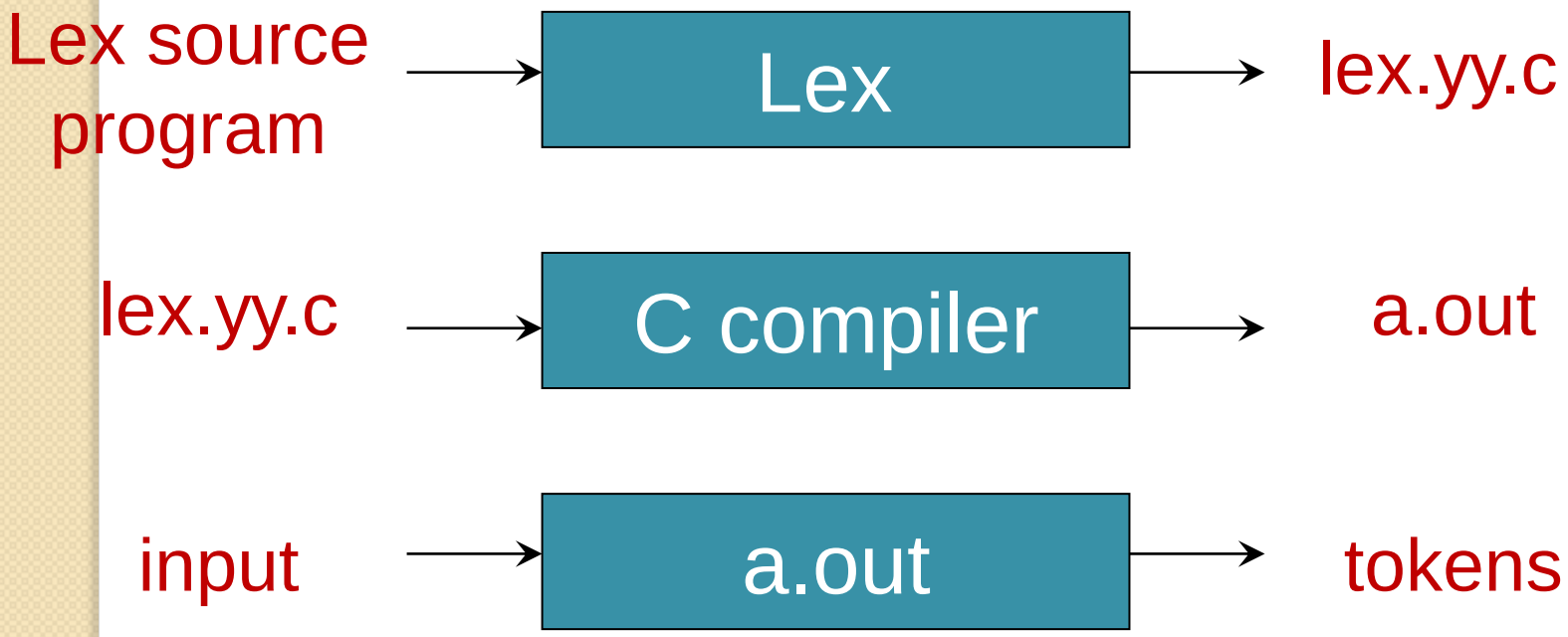
Expression	Matches
abc	abc
abc*	ab abc abcc abccc ...
abc+	abc abcc abccc ...
a(bc)+	abc abcbc abcbcbc ...
a(bc)?	a abc
[abc]	one of: a, b, c
[a-z]	any letter, a-z
[a\-z]	one of: a, -, z
[-az]	one of: -, a, z
[A-Za-z0-9]+	one or more alphanumeric characters
[\t\n]+	whitespace
[^ab]	anything except: a, 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
`int yylex(void)`	call to invoke lexer, returns token
`char *yytext`	pointer to matched string
`yyleng`	length of matched string
`yylval`	value associated with token
`int yywrap(void)`	wrapup, return 1 if done, 0 if not done
`FILE *yyout`	output file
`FILE *yyin`	input file
`INITIAL`	initial start condition
`BEGIN`	condition switch start condition
`ECHO`	write matched string

Running a Lex program




```
$lex xyz.l
$ cc lex.yy.c –ll
$\.a.out
```

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
vi filename	Creates a new file if it already does not exist, otherwise opens existing file
vi -R filename	Opens an existing file in read only mode.
view filename	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 save the file

  :wq – Save and exit

  : q! – exit without saving

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
4. Write a LEX program to check the entered string is an Identifier or keyword of C-language.