# Assignment No 1

Aim:
Implement a lexical analyzer for a sample language using LEX.

Objective:

1. To understand basic syntax of LEX specifications,built-in functions and variables.
2. To design lexical analyser for sample language

Software:

1. Linux Operating System
2. GCCcompiler

MATHEMATICALMODEL:

Consider a set S consisting of all the elements related to a program.The mathematical model is given as below,
S=s,e,X,Y,Fme,DD,NDD,MemsharedWhere,
s=Initial State
e=End State
X=sample language
Y=token,symboltable
Fme=Algorithm/Functionusedinprogram.foreg.yylexDD=Deterministic Data
NDD=Nondeterministic Data
Memshared=Memory shared by processor.

THEORY
Introduction:
THEORY:
Introduction:

Lexical tool for building lexical analyzer so lexers. A lexer takes an arbitrary input stream and tokenizes it, i.e. ,divides it up in to lexical tokens. This tokenized output can then be processed further, usually by yacc, or it can be the endproduct.

Lex Specifications:
A Lex program has the following form:

%
<C global variables, prototypes, comments>
%

[DEFINITION SECTION]
%%

%%

<Cauxiliary subroutines>
%%
[RULESSECTION]

<pattern><action to take when matched> <pattern><action to take when matched> %%

LEX Regular Expressions:

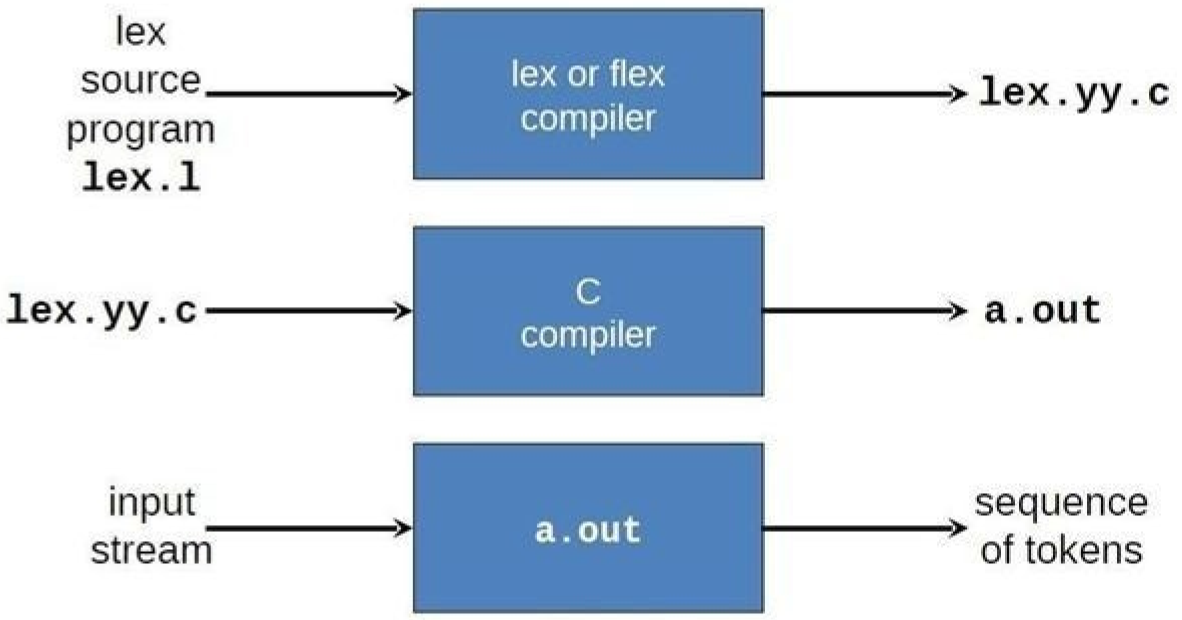
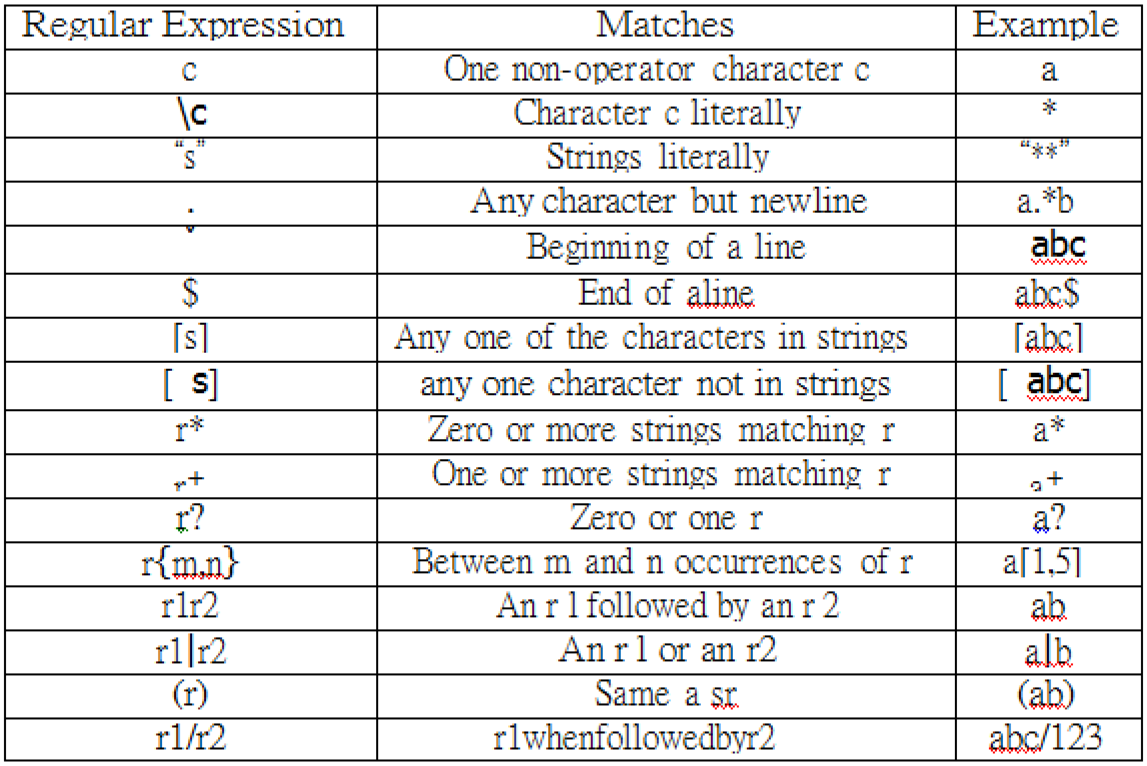


Fig 1 : Creating a Lexical Analyser with LEX



| Regular Expression | Matches | Example |
|---|---|---|
| c | One non-operator character c | a |
| \c | Character c literally | * |
| "s" | Strings literally | "**" |
| . | Any character but newline | a.*b |
|  | Beginning of a line | abc |
| $ | End of aline | abc$ |
| [s] | Any one of the characters in strings | [abc] |
| [ s] | any one character not in strings | [ abc] |
| r* | Zero or more strings matching r | a* |
| r+ | One or more strings matching r | a+ |
| r? | Zero or one r | a? |
| r{m,n} | Between m and n occurrences of r | a[1,5] |
| r1r2 | An r 1 followed by an r 2 | ab |
| r1\|r2 | An r 1 or an r2 | a\|b |
| (r) | Same a sr | (ab) |
| r1/r2 | r1whenfollowedbyr2 | abc/123 |

Ambiguous Source Rules:

Lex can handle ambiguous specifications. When more than one expression can match the current input, Lex choose s as follows:

1. The long est match his preferred.
2. Am on grules which matched the same number of characters, the rule given first is preferred.

Built in Functions and variables:

1. Int yy lex(void):
   Performs lexical analysis on the in put;t his is the primary function generated by the lex utility. The function shall return zero when the

end of input is reached; otherwise, it shall return non-zero values (tokens) determined by the actions that are selected.

2. Int yy more(void):
   When called, indicates that when the next input string is recog-nized, it is to be appended to the current value of yy text rather than replacing it; the value in yy length all be adjusted accordingly.

3. Int yy less(int n):
   Retains n initial characters in yytext, NUL-terminated, and treats

The remaining characters as if they had not been read; the value in yyleng shall be adjusted accordingly.

Int input(void):
Returns the next character from the input, or zero on end-of-file. It shall obtain input from the stream pointer yyin, although possible via an intermediate buffer. Thus, onces can ning has begun, the effect of altering the value of yyin is udefined. The charac-ter reads hall be removed from the input stream of the scanner without any processing by the scanner.

Int unput(int c):
Returns the charactercto the input; yytext and yylen gareun-defined until then ext expression is matched. The result of using unput() for more characters than have been input is unspecified.

Int yywrap(void):
Called by yylex() atend-of-file; the default yywrap() shall always return 1.If the application requires yylex() to continue processing with another source of input, then the application can include a function yywrap(), which associates another file with the external variable FILE*yyin and shall return a value of zero.

int main(int argc,char*argv[]):
Calls yylex() to perform lexical analysis, thenexits. The user code can contain main() to perform application-specific opera-tions, calling yylex() as applicable.

Yyin and yyout:
These are FILE* s to the input and output file respectively. Both are accessible and can be assigned to a value other than stdin and stdout.

yytext:
Array of or pointer to char where lex places the current tokens lexeme. The string is automatically null-terminated. It can be modified but not lengthened.

yyleng:
Integer that hold sstrlen(yytext).

yylineno:
Integer that keeps count of lines read. It is maintained automati-c all y by yylex() for each — nen countered.

Example of a lex program:

%
/* C code to be copied verbatim */
#include<stdio.h>
%

/*This tells flex to read only one input file*/
%option no yywrap

%%
/***Rules section***/

/*$[0-9]+matchesastringofoneormoredigits*/$[0-9]+

/*yytext is a string containing the matched text. */printf(Sawaninteger:%s,yytext);

/*Ignore all other characters.*/

%%
/***C Code section***/

Int main(void)

```
/*Call the lexer, then quit.*/yylex();
return0;
```

Usage:

There are two steps in compiling a LEX source program. First, the LEX source must be turned into a generated program in the host general purpose language. Then this program must be compiled and loaded, usually with a library of LEX subroutines. The generated program is on a file named lex.yy.c. The I/O library is defined in terms of the C standard library.

Command:

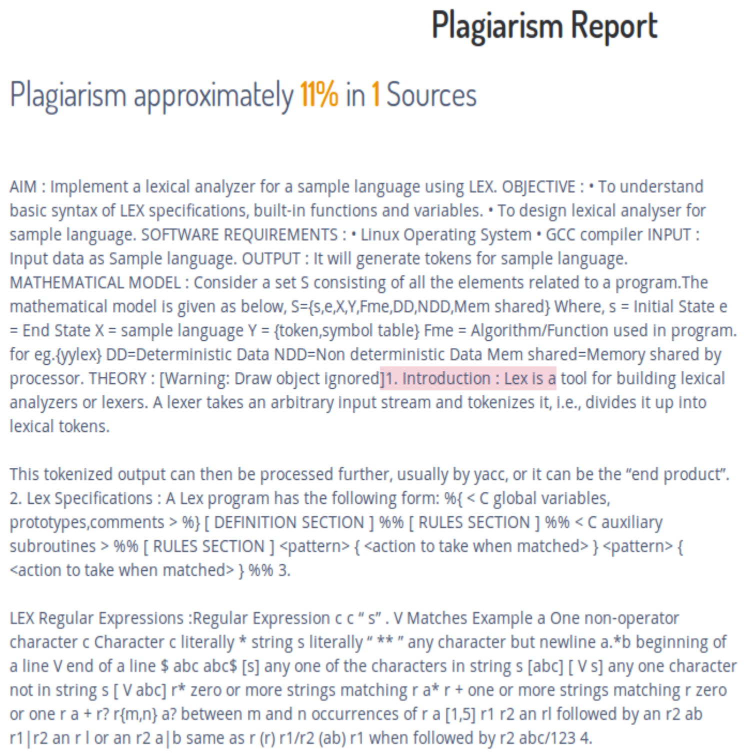
```
$lex<programname>.l $gcclex.yy.cll $./a.out
```

CONCLUSION :

Thus, we have studied Lexical Analyzer for sample language.

| *Roll No* | *Name of Student* | *Date of performance* | *Date of Checking* | *Signature of* |
|---|---|---|---|---|
| *BECOC*357 | Sunny Shah | 23 / 06 / 2017 | 30 / 06 / 2017 | |

## 1 PLAGARISM REPORT :


Plagiarism Report

Plagiarism approximately 11% in 1 Sources

AIM : Implement a lexical analyzer for a sample language using LEX. OBJECTIVE : • To understand basic syntax of LEX specifications, built-in functions and variables. • To design lexical analyser for sample language. SOFTWARE REQUIREMENTS : • Linux Operating System • GCC compiler INPUT : Input data as Sample language. OUTPUT : It will generate tokens for sample language. MATHEMATICAL MODEL : Consider a set S consisting of all the elements related to a program.The mathematical model is given as below, S={s,e,X,Y,Fme,DD,NDD,Mem shared} Where, s = Initial State e = End State X = sample language Y = {token,symbol table} Fme = Algorithm/Function used in program. for eg.{yylex} DD=Deterministic Data NDD=Non deterministic Data Mem shared=Memory shared by processor. THEORY : [Warning: Draw object ignored]1. Introduction : Lex is a tool for building lexical analyzers or lexers. A lexer takes an arbitrary input stream and tokenizes it, i.e., divides it up into lexical tokens.

This tokenized output can then be processed further, usually by yacc, or it can be the "end product". 2. Lex Specifications : A Lex program has the following form: %{ < C global variables, prototypes,comments > %} [ DEFINITION SECTION ] %% [ RULES SECTION ] %% < C auxiliary subroutines > %% [ RULES SECTION ] <pattern> { <action to take when matched> } <pattern> { <action to take when matched> } %% 3.

LEX Regular Expressions :Regular Expression c c " s" . V Matches Example a One non-operator character c Character c literally * string s literally " ** " any character but newline a.*b beginning of a line V end of a line $ abc abc$ [s] any one of the characters in string s [abc] [ V s] any one character not in string s [ V abc] r* zero or more strings matching r a* r + one or more strings matching r zero or one r a + r? r{m,n} a? between m and n occurrences of r a [1,5] r1 r2 an rl followed by an r2 ab r1|r2 an r l or an r2 a|b same as r (r) r1/r2 (ab) r1 when followed by r2 abc/123 4.

Figure 1: Plagarism Checker: www.smallseotools.com/plagarism-checker