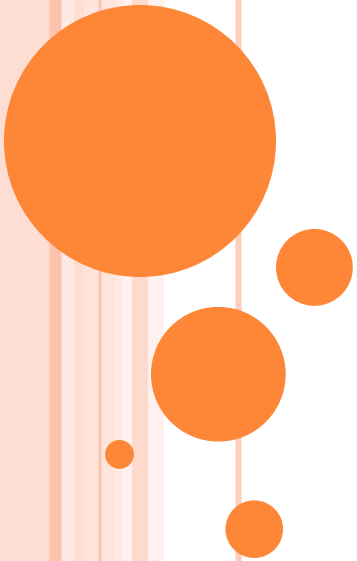
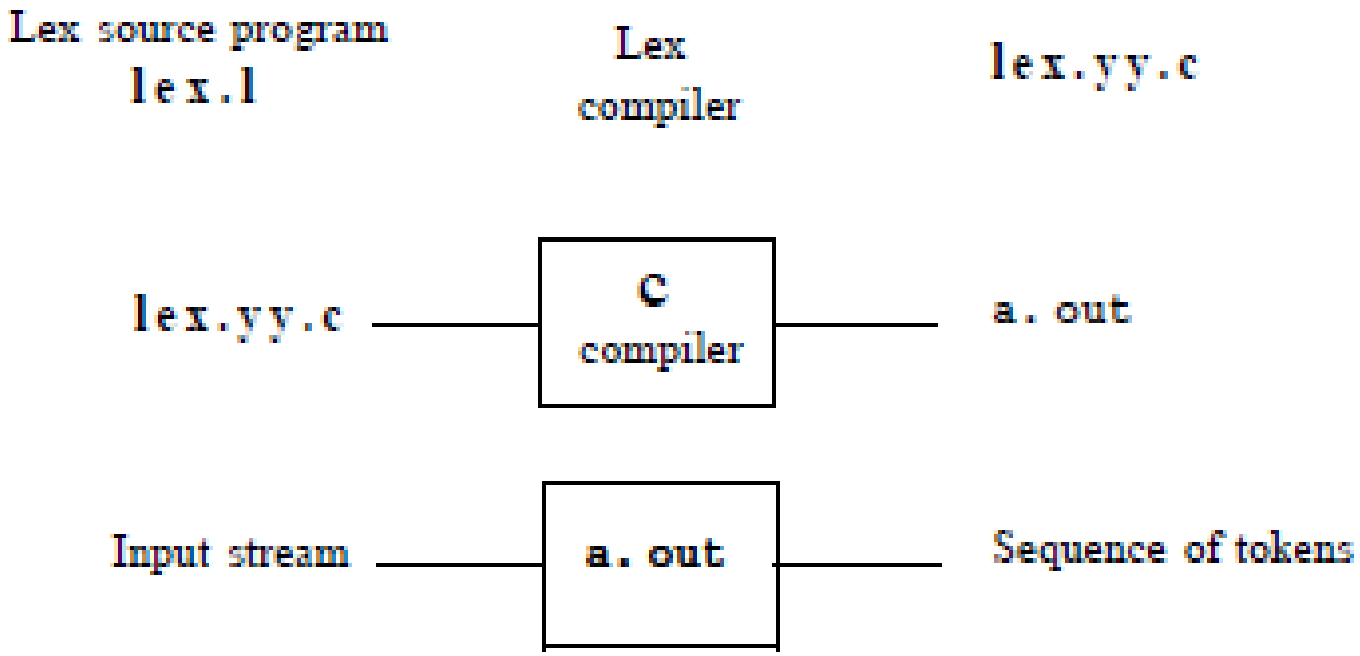


LEX

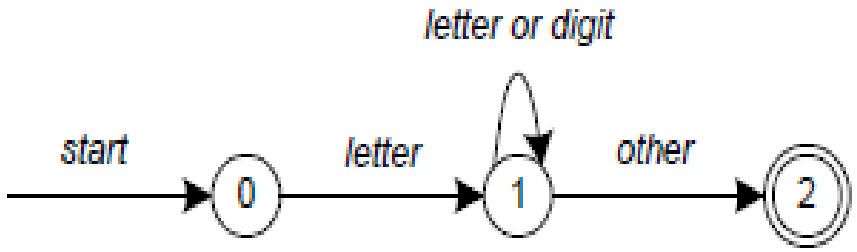
Introduction

- Lex is a tool for generating a lexical analyzer for a compiler.
- In Unix, it is available in a package called Flex.
- It processes regular expressions and produces a table-driven lexical analyzer/scanner/lexer which is saved in a file called lex.yy.c. The scanner can also be output to any c file.
- The scanner is then used to scan the input and generate the tokens.


Lex source program
lex.l
Lex
compiler
lex.yy.c
lex.yy.c
C
compiler
a. out
Input stream
a. out
Sequence of tokens

LEX AND DFA

Exampe: For identifiers: letter(letter | digit)



Sample code for the above dfa

```
start:   goto state0

state0:  read c
         if c = letter goto state1
         goto state0

state1:  read c
         if c = letter goto state1
         if c = digit goto state1
         goto state2

state2:  accept string
```

Special characters

Pattern	Matches
.	any character except newline
\.	literal .
\n	newline
\t	tab
^	beginning of line
$	end of line

OPERATORS

Pattern	Matches
?	zero or one copy of the preceding expression
*	zero or more copies of the preceding expression
+	one or more copies of the preceding expression
a|b	a or b
(ab)+	one or more copies of ab (grouping)
"a+b"	literal "a+b" (C escapes still work)
abc	abc
abc*	ab abc abcc abccc ...
"abc*"	literal abc*
abc+	abc abcc abccc ...
a(bc)+	abc abcbc abcbcbc ...
a(bc)?	a abc

More characters and their uses

Characters	Description
-	Used for range of charcaters. Eg. A-Z,a-z
[ ]	Matches any character from the class of characters within [ ]
[^]	Does not match any character from the class of characters within [ ]
|	Or operation
/	Look ahead character. Eg: exp1/exp2 means pattern exp1 is matched only if it is followed by exp2.
( )	Group series of expression to a new expression
{s}	Matches a string already defined for s.

Structure of lex program

- %{
  - //Header files, variable declarations etc. This section //is optional.
- %}
- %option
- %%
- Regular expressions {action(s)}//
- %%
- Code by user.

Lex Variables

- yyin: It is of type FILE*. This points to the current file being parsed by the lexer.
- yyout: It is of type FILE*. This points to the location where the output of the lexer will be written. By default, both yyin and yyout point to standard input and output.
- yytext: The text of the matched pattern is stored in this variable (char*).
- yyleng: gives the length of the matched pattern.
- yylineno :It provides current line number information

DIRECTIVES(SIMILAR TO C KEYWORDS)

- **ECHO:-** This directive copies the value of yytext to the scanner's output.
- **BEGIN:-** This directive followed by the name of the start symbol, places the scanner in the corresponding rules. Lex activates the rules using the directive BEGIN and a start condition.
- **REJECT:-** It directs the scanner to proceed on to the "scanned best" rule which matched the input (or a prefix of the input)..

Some Lex functions

- yylex(): Entry point for the generating the scanner.
- int yywrap(): this function indicates the end of input. It returns 1 if there is no more input. If it returns 0 then yylex will assume that another file is opened for processing and hence continues.

References

- "*LEX & YACC TUTORIAL*" by Tom Niemann.
- "*Compiler Design*" by K Muneeswaran.
- "*Compilers: Principles, Techniques and Tools*" by Alfred V Aho, Ravi Sethi and Jeffrey D. Ullman.