2.11 LEX TOOL

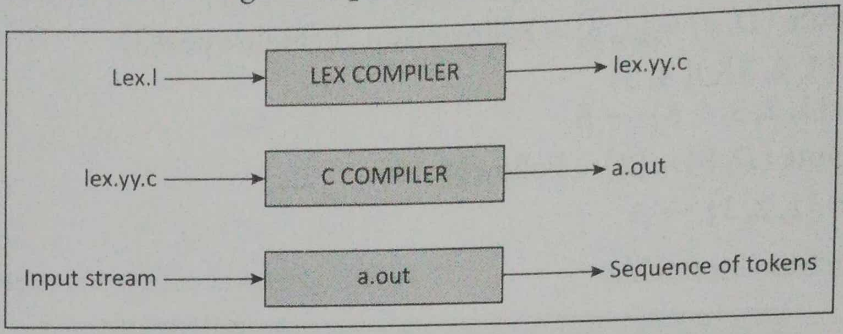
Lex is a program generator designed for lexical processing of character input streams. It accepts a high-level, problem-oriented specification for character string matching, and produces a program in a general-purpose language which recognises regular expressions. The Lex written code recognises these expressions in an input stream and partitions the input stream into strings matching the expressions. Lex is not a complete language, but rather a generator representing a new language feature which can be added to different programming languages, called "host languages."

A Lex program consists of three parts:

a. Declaration part
b. Translation rules part
c. Auxiliary procedures part

2.11.1 Declarations

The declaration section includes declarations of variables, manifest constants (A manifest constant is an identifier that is declared to represent a constant, e.g., # define PIE 3.14), the files to be included and definitions of regular expressions.



2.11.2 Transition Rules

The *translation rules* of a Lex program are statements of the form:

```
R.E → p1     {action 1} → pgm fragment
                          written in C
      p2     {action 2}
      p3     {action 3}
      ...    ...
      ...    ...
```

where each *p* is a regular expression and each action is a program fragment describing what action the lexical analyser should take when a pattern *p* matches a lexeme.

In Lex, the actions are written in C.

2.11.3 Auxiliary Procedures → compiled separately in C.

The third section holds whatever auxiliary procedures are needed by the actions. Alternatively, these procedures can be compiled separately and loaded with the lexical analyser. The auxiliary procedures are written in C language.

Meta character	Matches
.	Any character except newline
\n	New line
*	Zero or more copies of the preceding expression
+	One or more copies of the preceding expression
?	Zero or one copy of the preceding expression

Meta character	Matches
^	Beginning of line
$	End of line
a|b	*a* or *b*
(ab)+	One or more copies of *ab*
"a+b"	Literal *a* + *b*
[ ]	Character class

ptr 1st char

yytext is a variable that is a pointer to the first character of the lexeme.

yyleng is an integer telling how long the lexeme is.

2.11.4 Lex Library

The **Lex** library contains the following subroutines:

scanner starts scanning the src pgm

main()	Invokes the lexical analyser by calling the **yylex** subroutine.
yywrap()	Returns the value 1 when the end of input occurs. (till encounter; 0 when scanner continues scanning)
yymore()	Appends the next matched string to the current value of the **yytext** array rather than replacing the contents of the **yytext** array.
yyless(int *n*)	Retains *n* initial characters in the **yytext** array and returns the remaining characters to the input stream.
yyreject()	Allows the lexical analyser to match multiple rules for the same input string. (The **yyreject** subroutine is called when the special action reject is used.)

called when EOF

Expression	Matches
abc	*abc*
*abc**	*ab, abcc, aaabc, abbbc, abcccc…*
abc+	*abc, aaabc, abbbc, abcccc …*
a(bc)+	*abc, abcbc, abbcbcbc, …*
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*