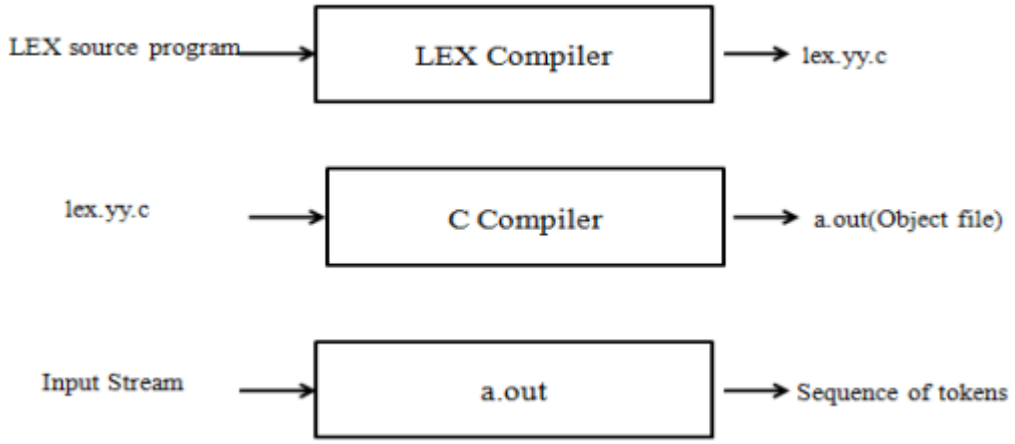
LEX

- A LEX source program is a requirement of a lexical analyzer, consisting of a set of regular expressions together with an action for each regular expression.
- The action is a part of code which is to be executed when a token specified by the equivalent regular expression is recognized.
- LEX is a tool generally used to specify lexical analyzers for a variety of languages.
- LEX tool require a Lex compiler, specification for the LEX tool is the Lex language.

Using a Scanner Generator: Lex

- Lex is a lexical analyzer generator developed by Lesk and Schmidt of AT&T Bell Lab, written in C, running under UNIX.
- Lex produces an entire scanner module that can be compiled and linked with other compiler modules.
- Lex associates regular expressions with arbitrary code fragments. When an expression is matched, the code segment is executed.
- A typical lex program contains three sections separated by %% delimiters.

Role of LEX:



Creating a lexical analyzer with Lex (or) Specification of LEX

A Lex program (the lex.l file) consists of three parts:

%{
auxiliary declarations
%}
regular definitions
%%
translation rules
%%
auxiliary procedures

Declarations section:

It 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.

Transition rules section:

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

p1	{action 1}
p2	{action 2}
p3	{action 3}
..	...

- Where each P1is a regular expression called a pattern, over the alphabet consisting of Σ and the auxiliary definition names. The patterns describe the form of the tokens.
- Each action 1is a program fragment describing what action the lexical analyzer should take when token P's found. The actions are written in a conventional programming language, rather than any particular language, we use pseudo language.
- To create the lexical analyzer L, each of the actions must be compiled into machine code.

Auxiliary procedures:

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

Auxiliary Definition

Letter= A|B|...|Z

Digit= 0|1|...|9

How does this Lexical analyzer work?

- The lexical analyzer created by Lex behaves in concert with a parser in the following manner. When activated by the parser, the lexical analyzer begins reading its remaining input, one character at a time, until it has found the longest prefix of the input that is matched by one of the regular expressions p.
- Then it executes the corresponding action. Typically, the action will return control to the parser. However, if it does not, then the lexical analyzer proceeds to find more lexemes, until an action causes control to return to the parser. The repeated search for lexemes until an explicit return allows the lexical analyzer to process white space and comments conveniently.
- The lexical analyzer returns a single quantity, the token, to the parser. To pass an attribute value with information about the lexeme, we can set the global variable yylval.

LEX Actions

- yytext() is a variable that is a pointer to the first character of the lexeme.
- yywrap() yywrap is called when lexical analyzer reach end of file. It yywrap returns a then lexical analyzer continue scanning. When yywrap return 1 means end of file is encountered.
- yyleng() is an integer telling how long the lexeme is.
- yyin() –It is used to read the source program from file and then stored in yyin.

Sample LEX Program

1. Program to Find the Capital letters in some string using LEX

```
%{
%}

%%
[A-Z]   {printf("%s",yytext);}
. ;
%%

main()
{printf("Enter Some string");
yylex();
}
int yywrap()
{return 1;
}
```

Declarations section

Transition rules section

Auxillary procedures

Input:
Enter Some string
Rohini College of Engineering and Technology
Output:
RECT