Assignment no 3

Problem Statement: Write a program to recognize infix expression using LEX and YAAC.

Objectives:

1. To understand LEX and YACC Concepts
2. To implement LEX & YACC Program

Software Requirement: Flex, bison and Devc++

Operating System recommended :- 64-bit Open source Linux or its derivative

Programming tools recommended: -LEX tool / Eclipse IDE

Hardware Requirement:I3 and I5 machines

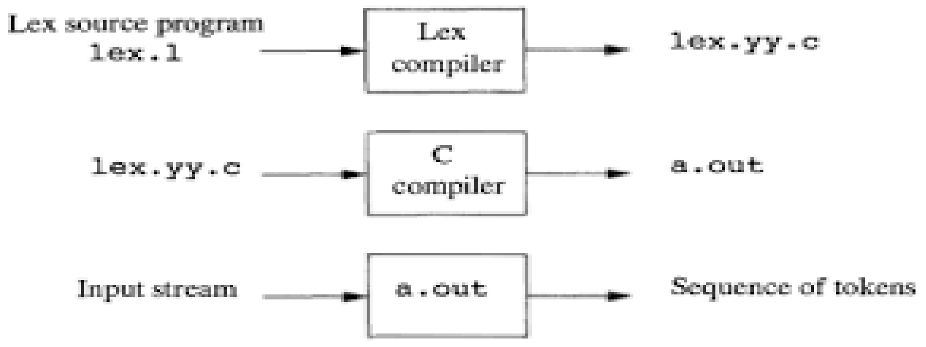
Theory:

Lex:

Lex stands for Lexical Analyzer. Lex is a tool for generating Scanners. Scanners areprograms that recognize lexical patterns in text. These lexical patterns (or regular Expressions) are defined in a particular syntax. A matched regular expression may have anassociated action. This action may also include returning a token. When Lex receives inputin the form of a file or text, it takes input one character at a time and continues until a patternis matched, then lex performs the associated action (Which may include returning a token).If, on the other hand, no regular expressioncan be matched, further processing stops and Lexdisplays an error message.

Lex and C are tightly coupled. A .lex file (Files in lex have the extension .lex) is passed through the lex utility, and produces output files in C. These file(s) are coupled to produce an executable version of the lexical analyzer.

Lex turns the user‟s expressions and actions into the host general –purpose language; the generatedprogram is named yylex. The yylex program will recognize expressions in a stream (calledinput in this memo) and perform the specified actions for each expression as it is detected.



Overview of Lex Tool

- **Programming in Lex:-**

  Programming in Lex can be divided into three steps:

  1. Specify the pattern-associated actions in a form that Lex can understand.
  2. Run Lex over this file to generate C code for the scanner.
  3. Compile and link the C code to produce the executable scanner.

... definitions ...

%%

.

.. rules ...

%%

... subroutines ...

YACC :

Yacc (**Yet Another Compiler-Compiler**) is a computer program for the Unix operating system developed by Stephen C. Johnson. It is a Look Ahead Left-to-Right (LALR) parser generator, generating a parser, the part of a compiler that tries to make syntactic sense of the source code, specifically a LALR parser, based on an analytic grammar written in a notation similar to Backus–Naur Form (BNF). Yacc is supplied as a standard utility on BSD and AT&T Unix. GNU-based Linux distributions include Bison, a forward-compatible Yacc replacement.

Yacc is one of the automatic tools for generating the parser program. Basically Yacc is a LALR parser generator. The Yacc can report conflicts or ambiguities (if at all) in the form of error messages. LEX and Yacc work together to analyse the program syntactically.

Yacc is officially known as a “parser”. Its job is to analyze the structure of the input stream, and operate of the “big picture”. In the course of it’s normal work, the parser also verifies that the input is syntactically sound.

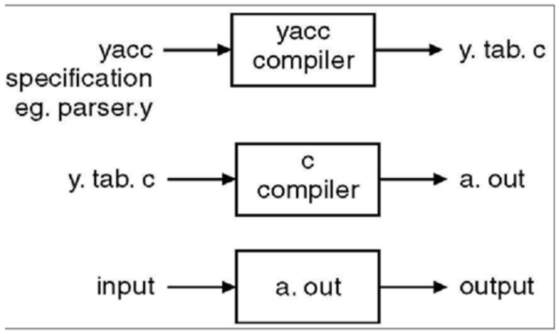


Fig:-YACC: Parser Generator Model

Structure of a yacc file:

A yacc file looks much like a lex file:

...definitions..
%%
...rules...
%%
...code...

Definitions As with lex, all code between %{ and %} is copied to the beginning of the resulting C file. Rules As with lex, a number of combinations of pattern and action. The patterns are now those of a context-free grammar, rather than of a regular grammar as was the 3 case with lex code. This can be very Elaborate, but the main ingredient is the call to yyparse, the grammatical parse.

Input to yacc is divided into three sections. The definitions section consists of token declarations andC code bracketed by "**%{**" and "**%}**". The BNF grammar is placed in the rules section and user subroutines are added in the subroutines section.

Infix Expression:

Expressions are usually represented in what is known as **Infix notation**, in which each operator is written between two operands (i.e., A + B). With this notation, we must distinguish between (A + B)*C and A + (B * C) by using either parentheses or some operator-precedence convention. Thus, the order of operators and operands in an arithmetic expression does not uniquely determine the order in which the operations are to be performed.

1. **Polish notation (prefix notation) –**
   It refers to the notation in which the operator is placed before its two operands. Here no parentheses are required, i.e., +AB
2. **Reverse Polish notation(postfix notation) –**
   It refers to the analogous notation in which the operator is placed after its two operands. Again,no parentheses is required in Reverse Polish notation, i.e., AB+

There are 3 levels of precedence for 5 binary operators as given below:

```
Highest: Exponentiation (^)
Next highest: Multiplication (*) and division (/)Lowest:
Addition (+) and Subtraction (-)
```

For example –

```
Infix notation: (A-B)*[C/(D+E)+F]
Post-fix notation: AB- CDE +/F +*
```

Conclusion:

Learn about Lex and yacc tools. Programming with lex and yacc and infix expression concepts.