**Practical No. 2**

**Aim:** Write YACC specification

A) to check syntax of an arithmetic  expression involving various operations such as addition, multiplication, subtraction, division. Also, convert this expression to postfix form.

B) To validate syntax of programming language constructs.

**Theory:**

**YACC**

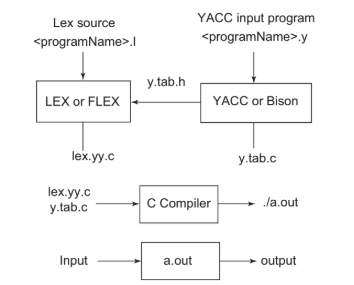
* Yacc generates C code for syntax analyzer, or parser.
* Yacc uses grammar rules that allow it to analyze tokens from Lex and create a syntax tree.

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.

The input to Yacc is a grammar with snippets of C code (called "actions") attached to its rules. Its output is a shift-reduce parser in C that executes the C snippets associated with each rule as soon as the rule is recognized. Typical actions involve the construction of parse trees. Using an example from Johnson, if the call node (label, left, right) constructs a binary parse tree node with the specified label and children, then the rule. recognizes summation expressions and constructs nodes for them. The special identifiers $$, $1 and $3 refer to items on the parser's stack.

Yacc produces only a parser (phrase analyzer); for full syntactic analysis this requires an external lexical analyzer to perform the first tokenization stage (word analysis), which is then followed by the parsing stage proper.

**Yacc Functioning**

****

**Format for Yacc file**

A full specification file looks like

declarations

%%

rules

%%

programs

The declaration section may be empty. Moreover, if the programs section is omitted, the second %% mark may be omitted also; thus, the smallest legal Yacc specification is

%%

Rules

The rules section is made up of one or more grammar rules. A grammar rule has the form:

A : BODY ;

A represents a nonterminal name, and BODY represents a sequence of zero or more names and literals. The colon and the semicolon are Yacc punctuation.

If there are several grammar rules with the same left hand side, the vertical bar ``|'' can be used to avoid rewriting the left hand side. In addition, the semicolon at the end of a rule can be dropped before a vertical bar. Thus the grammar rules

A : B C D ;

A : E F ;

A : G ;

can be given to Yacc as

A : B C D

| E F

| G

;

**Inclusions in yacc declaration**

%start Specify the grammar's start symbol

%token Declare a terminal symbol (token type name) with no precedence or

associativity specified

%type Declare the type of semantic values for a nonterminal symbol

%right Declare a terminal symbol (token type name) that is right-associative

%left Declare a terminal symbol (token type name) that is left-associative

%nonassoc Declare a terminal symbol (token type name) that is nonassociative (using it

in a way that would be associative is a syntax error, ex: x *op*. y *op*. z is syntax

error)

**Pseudovariables**

* To facilitate easy communication between the actions and the parser, the action statements are altered slightly. ‘$’ is used
* To return a value, the action normally sets the pseudovariable ‘$$’ to some value.
* For example, {$$ = 1;}
* To obtain the values returned by previous actions and the lexical analyser, the action may use the pseudovariables $1, $2, . . .
* For example,   
  A : B C D ;

$1 has the value returned by B,

$2 has the value returned by C

$3 the value returned by D.

**Steps to execute YACC programs in Linux**

flex *pgmname*.l

bison -d *pgm name*.y

gcc -c lex.yy.c *pgm name*.tab.c

gcc -o a.out lex.yy.o *pgm name*.tab.o -lfl

./a.out

**Additional TASK: Write the answers to following Qs:**

1. What is the role of tab.h file and when is it created?

Before writing the LEX program, there must be some way by which the YACC program can tell the LEX program that DIGIT is a valid token that has been declared in the YACC program. This communication is facilitated by the file "y.tab.h" which contains the declarations of all the tokens in the YACC program. The "y.tab.h" is automatically generated by YACC when the 'yacc' command is executed with the -d flag.

1. What is the role of tab.s file and when is it created?
2. Where is main written?

Main is written in y file after void yyerror(char \* msg) function.

1. How is the program run by calling yylex internally from main

YACC generates the definition for yyparse() in y.tab.c and LEX generates the definition for yylex() in lex.yy.c. yyparse() repetitively calls yylex() to read tokens from the input stream.

The [main() function](https://silcnitc.github.io/ywl.html#navya) in y.tab.c begins execution. It calls yyparse() which inturn calls yylex() for tokens.

1. Use of option –lfl and option –d

-lfl: flex generates as output a C source file, `lex.yy.c', which defines a routine `yylex()'. This file is compiled and linked with the `-lfl' library to produce an executable.

-d: Generates the file y.tab.h, which contains the constant definition statements for token names. This lets other modules of a multimodule program access these symbolic names. This is the same as -D, except that the user does not specify the header file name.

1. Which parsing technique does YACC use internally?

YACC uses LALR parsing technique internally.