**✅ Title: Parser for Arithmetic Grammar using LEX & YACC**

**🎯 Aim:**

Write a program using **LEX and YACC** to create a parser for arithmetic grammar and implement a simple **calculator**.

**📌 Objectives:**

1. Understand the **YACC tool** for parser generation.
2. Learn how to use **YACC and LEX together** for parsing.
3. Understand the **compilation and execution process** of .y and .l files.

**📖 Theory Summary**

**📘 1. Grammar - Context Free Grammar (CFG):**

* Grammar defines the structure of valid arithmetic expressions.
* For example:
* E → E + E | E - E | E \* E | E / E | ( E ) | NUM

**🔄 2. Co-routine Working of Scanner & Parser:**

* **LEX** acts as **scanner**, breaks input into **tokens**.
* **YACC** is the **parser**, applies grammar rules using those tokens.
* They work together using yylex() and yyparse().

**⚙️ 3. Syntax and Semantic Actions in .y File:**

* Each grammar rule is followed by a block {} of **semantic actions** (e.g., calculations).
* YACC uses **$$, $1, $2** to represent values of grammar symbols.

**🧪 Input:**

Arithmetic expression like:

3 + 5 \* (2 - 1)

**📤 Output:**

Evaluated result of the arithmetic expression.

**✅ Conclusion:**

Parser and evaluator for arithmetic expressions was successfully implemented using **LEX and YACC**.

**💻 Platform:**

Linux (LEX & YACC)

**🎤 Viva Questions & One-Line Answers**

**🔍 Conceptual**

1. **What is a top-down parser?**  
   → It starts from the start symbol and expands using grammar rules.
2. **What is a bottom-up parser?**  
   → It builds the parse tree from leaves (tokens) upward to the start symbol.
3. **What is shift-reduce conflict?**  
   → A conflict when the parser can't decide between shifting the token or reducing it.
4. **What is reduce-reduce conflict?**  
   → A conflict where the parser is unsure which rule to reduce with.
5. **How does YACC resolve grammar ambiguities?**  
   → By default associativity and precedence rules or by using %left, %right, %nonassoc.

**⚙️ YACC & LEX**

1. **What does yyparse() do?**  
   → It starts parsing the input based on defined grammar.
2. **What is yylval?**  
   → Used to pass token values from LEX to YACC.
3. **What are %left, %right in YACC?**  
   → Used to define associativity and precedence of operators.

**🧑‍💻 Code Implementation – Arithmetic Calculator**

**📄 calc.l – LEX File**

%{

#include "y.tab.h"

%}

%%

[0-9]+ { yylval = atoi(yytext); return NUM; }

[\n] { return '\n'; }

[+\-\*/()] { return yytext[0]; }

[ \t] ;

. { printf("Invalid character: %s\n", yytext); }

%%

int yywrap() {

return 1;

}

**📄 calc.y – YACC File**

%{

#include <stdio.h>

#include <stdlib.h>

%}

%token NUM

%left '+' '-'

%left '\*' '/'

%%

input: /\* empty \*/

| input line

;

line: expr '\n' { printf("Result = %d\n", $1); }

;

expr: expr '+' expr { $$ = $1 + $3; }

| expr '-' expr { $$ = $1 - $3; }

| expr '\*' expr { $$ = $1 \* $3; }

| expr '/' expr {

if ($3 == 0) { printf("Divide by zero error!\n"); exit(1); }

else $$ = $1 / $3;

}

| '(' expr ')' { $$ = $2; }

| NUM { $$ = $1; }

;

%%

int main() {

printf("Enter an arithmetic expression:\n");

return yyparse();

}

void yyerror(const char \*s) {

fprintf(stderr, "Error: %s\n", s);

}

**🧪 Compile and Run**

yacc -d calc.y

lex calc.l

gcc y.tab.c lex.yy.c -o calc -ll

./calc

**🔢 Sample Input:**

3 + 5 \* (2 - 1)

**✅ Output:**

Result = 8