**Exp 5: Write a Lex and Yacc Program to simulate Predictive Parser.**

**Lex code:**

%{

#include "y.tab.h"

%}

%%

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

[+\-] { return \*yytext; }

[\*\/] { return \*yytext; }

\( { return '('; }

\) { return ')'; }

\n { return 0; }

[ \t] { /\* Ignore spaces \*/ }

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

%%

**Yacc Code:**

%{

#include <stdio.h>

#include <stdlib.h>

void yyerror(const char \*s);

int yylex();

%}

%token NUMBER

%%

E : T Ep { printf("Reduced: E -> T Ep\n"); } ;

Ep : '+' T Ep { printf("Reduced: Ep -> + T Ep\n"); }

| '-' T Ep { printf("Reduced: Ep -> - T Ep\n"); }

| /\* epsilon \*/ { printf("Reduced: Ep -> ε\n"); } ;

T : F Tp { printf("Reduced: T -> F Tp\n"); } ;

Tp : '\*' F Tp { printf("Reduced: Tp -> \* F Tp\n"); }

| '/' F Tp { printf("Reduced: Tp -> / F Tp\n"); }

| /\* epsilon \*/ { printf("Reduced: Tp -> ε\n"); } ;

F : '(' E ')' { printf("Reduced: F -> (E)\n"); }

| NUMBER { printf("Reduced: F -> NUMBER\n"); } ;

%%

void yyerror(const char \*s) {

printf("Syntax Error: %s\n", s);

}

int main() {

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

if (yyparse() == 0)

printf("Parsing successful!\n");

else

printf("Parsing failed!\n");

return 0;

}

**Output:**

3 + 5 - 8

Reduced: F -> NUMBER

Reduced: Tp -> ε

Reduced: T -> F Tp

Reduced: F -> NUMBER

Reduced: Tp -> ε

Reduced: T -> F Tp

Reduced: F -> NUMBER

Reduced: Tp -> ε

Reduced: T -> F Tp

Reduced: Ep -> ε

Reduced: Ep -> - T Ep

Reduced: Ep -> + T Ep

Reduced: E -> T Ep

Parsing successful!

 yytext:

* This is a global character array that holds the current matched text from the input.

 atoi(yytext):

* Converts the matched text (which is assumed to be a string representation of a number) into an integer.

 yylval:

* This is a global variable (often a union in yacc or bison parser) used to store semantic values for tokens.

 return NUMBER;:

* Returns the token NUMBER (which should be defined elsewhere, usually in a yacc or bison file).