**CSB 353: Compiler Design**

**LAB 10-11**

**Submitted By:**

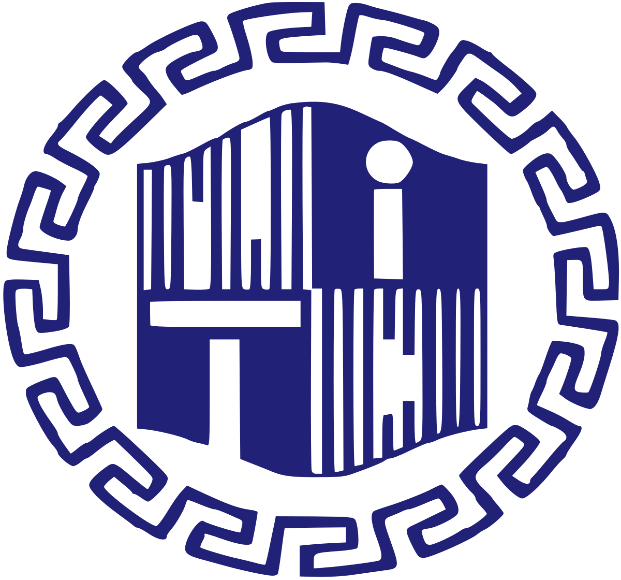
**Name: PREM KUMAR**

**Roll No: 191210037**

**Branch: CSE**

**Semester: 6 th**

**Submitted To: Dr. Shelly Sachdeva**

Department of Computer Science and Engineering

**NATIONAL INSTITUTE OF TECHNOLOGY DELHI**

2019-2023

Ques 1. Consider the example of simple desk calculator that performs simple

operations on integer expressions with the grammar:

exp -> exp addop term | term addop -> + | -

term -> term mulop factor | factor mulop -> \*

factor -> (exp) | number

number -> number digit | digit

digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

You are required to write YACC specifications for this grammar so that the

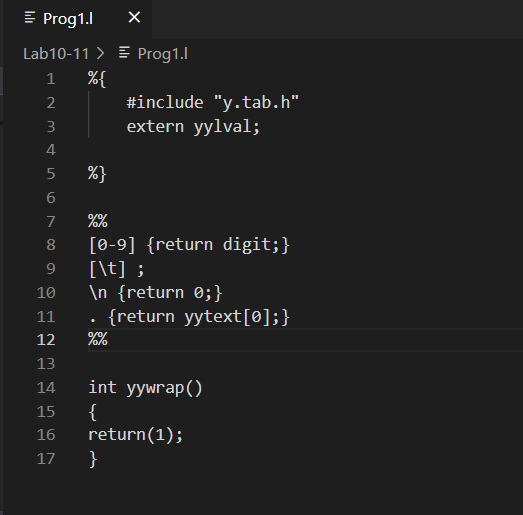
parser evaluates any arithmetic expressions and the output shows each grammar

rule as it is applied in the parsing process. Show your parsing sequence for the

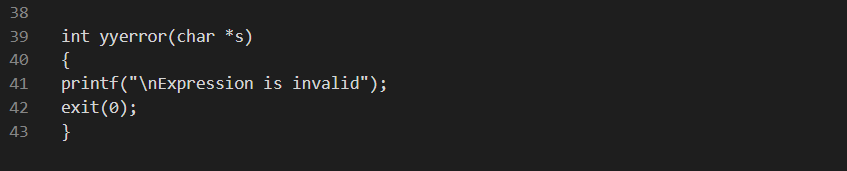
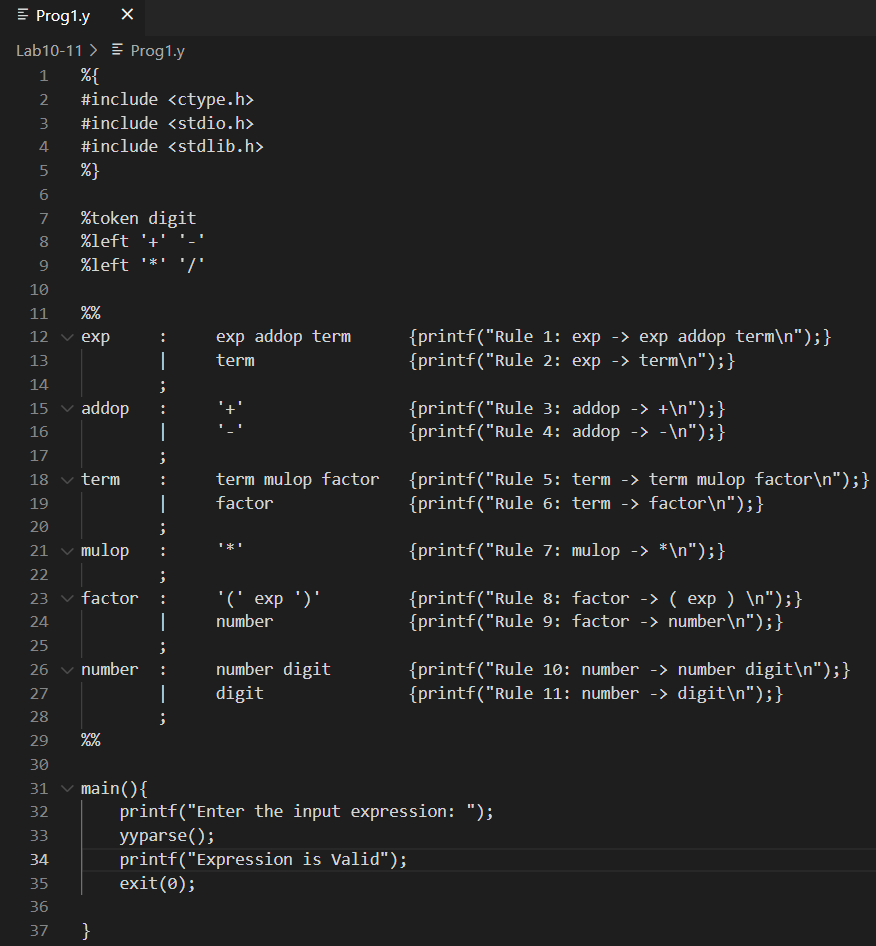
input string: (2+(3\*4)).

Code:

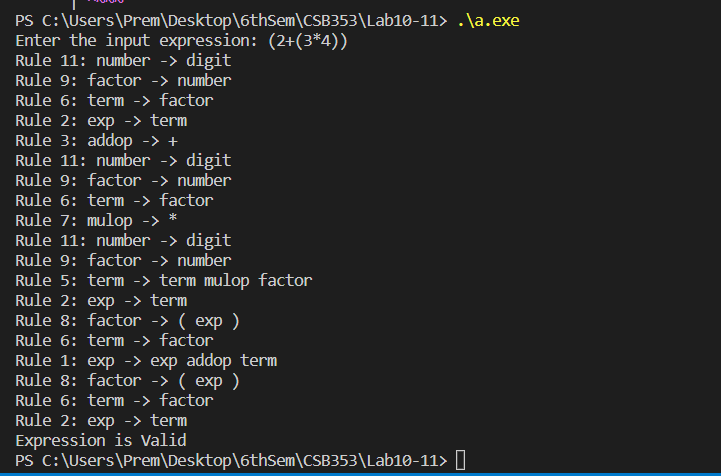
* prog1.l (Lex File)



* prog1.y (Yacc File)



Output:



Ques 2. The following grammar describes a boolean expression (exp) consisting of

operators "&", "|", "!", "==", "! =", brackets "(" and ")", and identifiers.

exp : exp\_2 | exp '&' exp\_2 ;

exp\_2 : exp\_3 | exp\_3 '|' exp\_2 ;

exp\_3 : exp\_4 | exp\_4 '==' exp\_4 | exp\_4 '!=' exp\_4 ;

exp\_4 : exp\_5 | '!' exp\_5 ;

exp\_5 : identifier | '(' exp ')' ;

Write Yacc code to recognize a series of expressions, each on a new line. Each

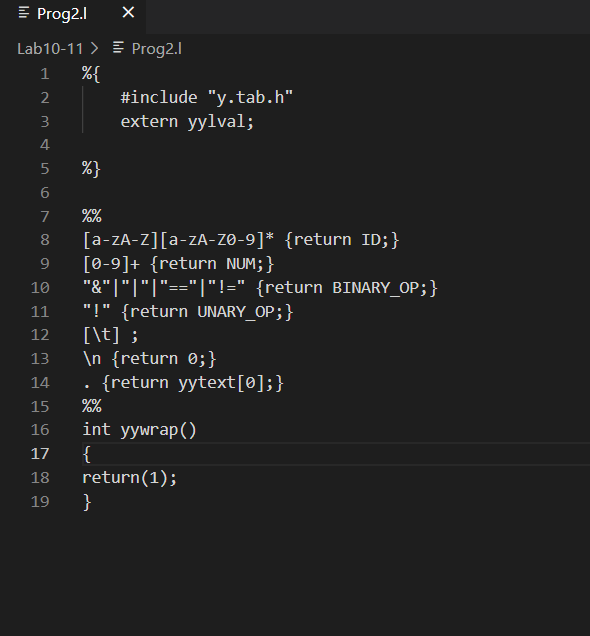
expression should unambiguously demonstrate precedence & associativity of

the operators, by showing the orders in which operators would be evaluated.

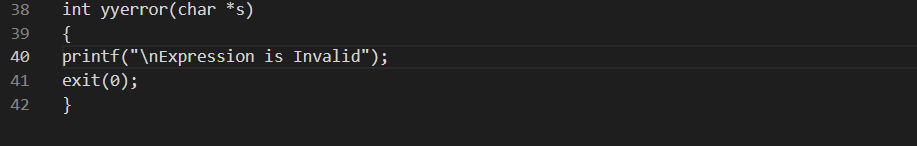
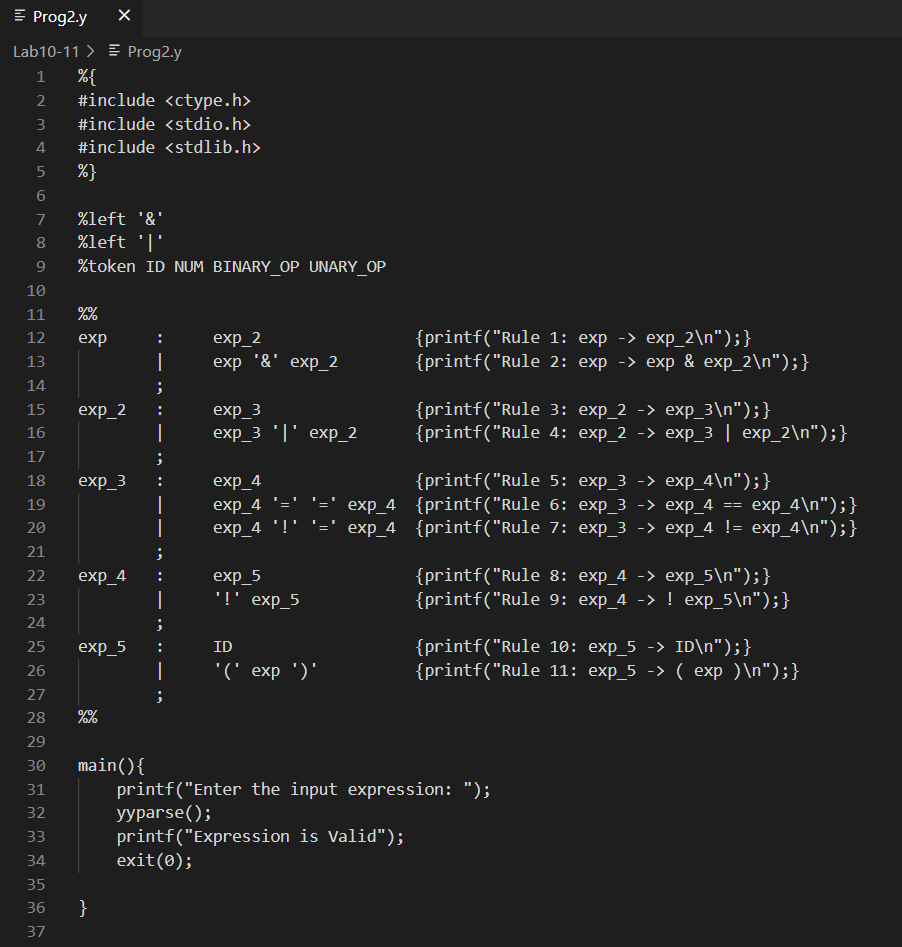
Also, give a parse sequence for each of the expressions.

Code:

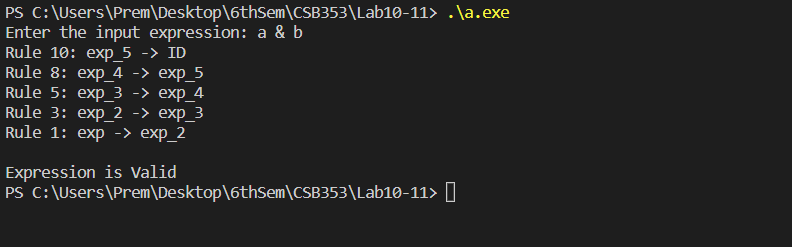
* prog2.l (Lex File)



* prog2.y (Yacc File)



Output:



Ques 3. Using YACC, write semantic actions that translate arithmetic expressions

(generated from the given grammar) from infix into postfix notation

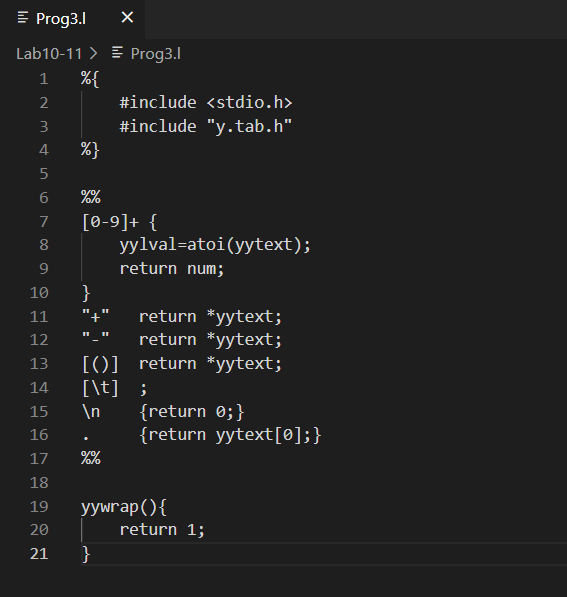
E -> E + T E -> T

T-> T \* F T -> F

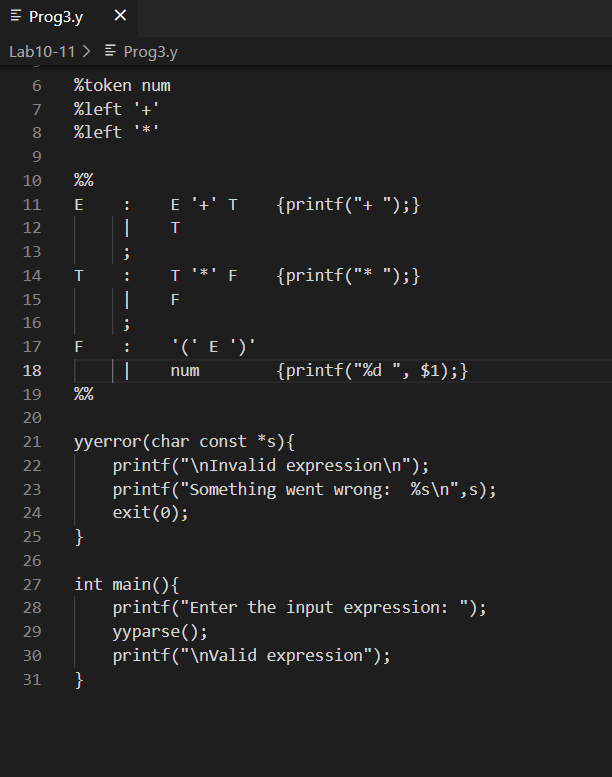
F -> ( E ) F -> num.

Code:

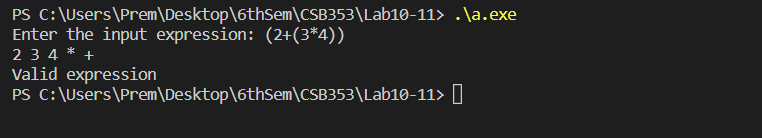
* prog3.l (Lex File)



* prog3.y (Yacc File)



Output:



Ques 4. For the grammar below, write a YACC program to compute the decimal value

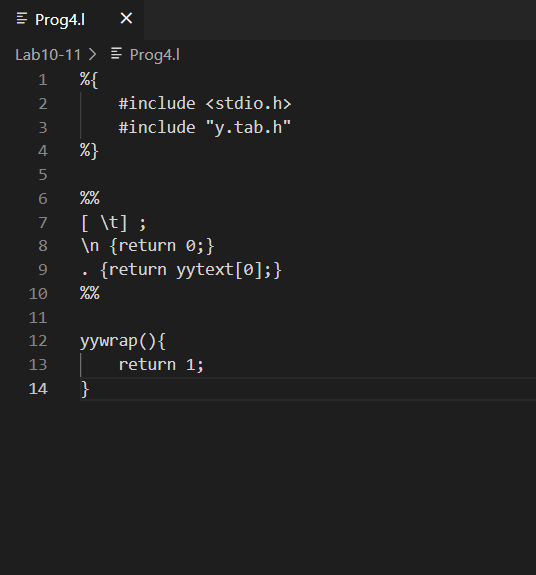
of an input string in binary. For example the translation of the string 101.101

should be the decimal number 5.625.

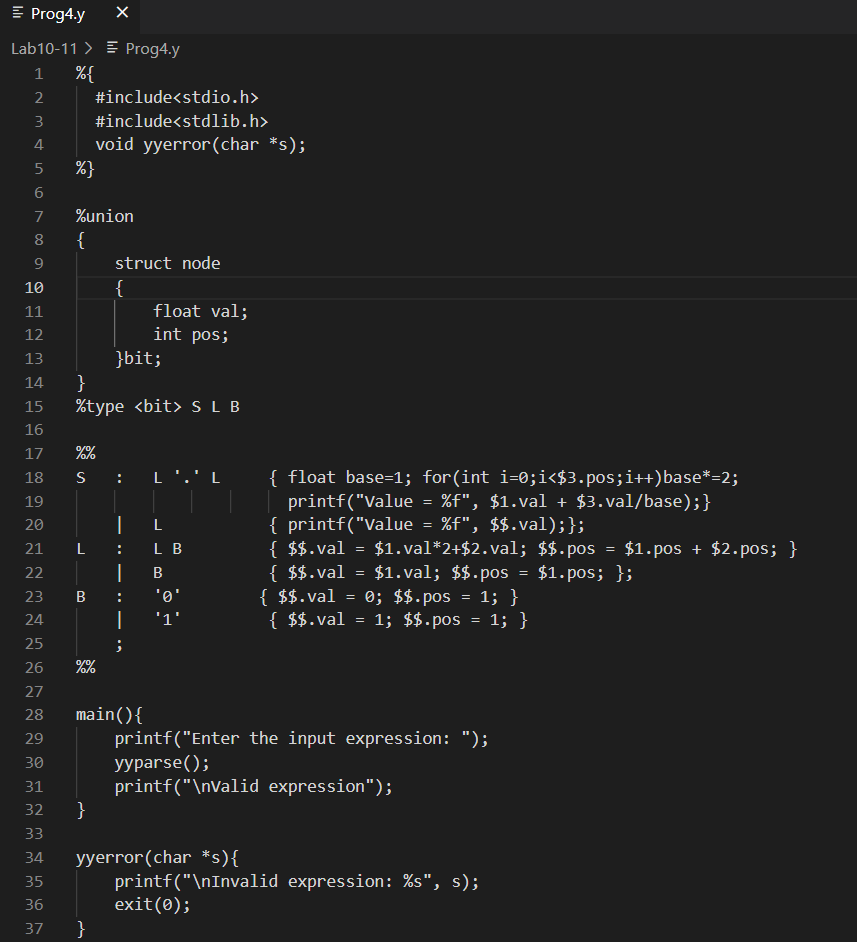
S→L.L|L L→LB|B B→0|1.

Code:

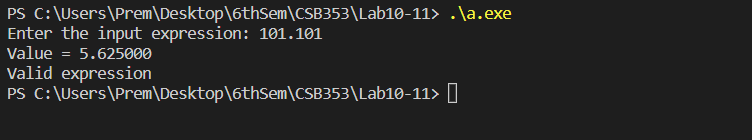
* prog4.l (Lex File)



* prog4.y (Yacc File)



Output:



Ques 5. Write a YACC program that accepts valid strings as per the grammar given

below and produces as output a linear representation of the same list

Grammar:

S (L)| char

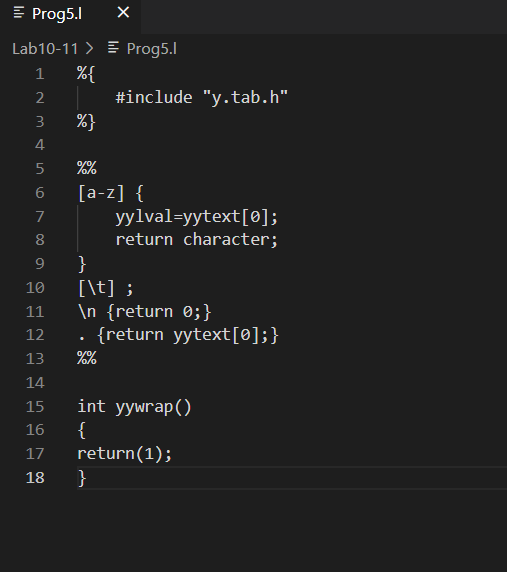
L L,S|S

and char is any character between a-z

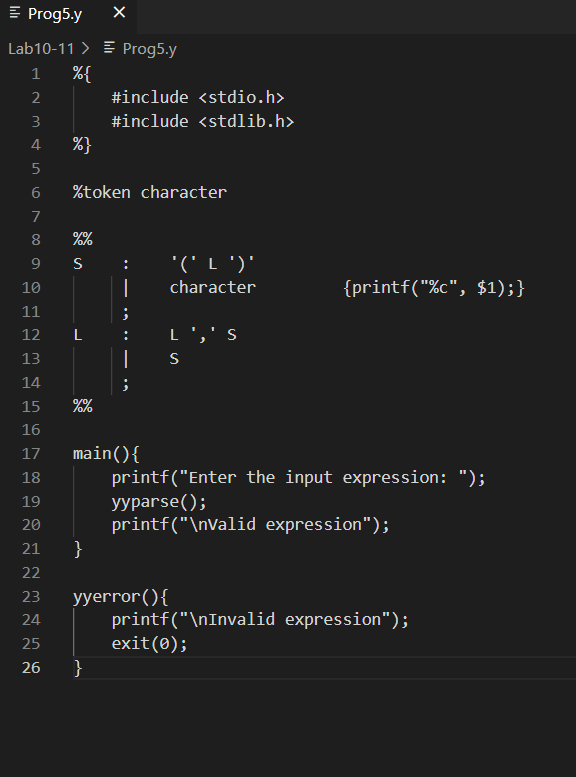
e.g. ((a,c),d,(h)) Produces acdh but (a,c),d,(h)) is an invalid input.

Code:

* prog5.l (Lex File)



* prog5.y (Yacc File)



Output:

