PRACTICAL -2

**NAME:-** Shreyas Namjoshi

**Roll No :-** 53

**BRANCH:-** CSEB

### **Topic:** Lexical Analysis + Syntax Analysis

**Platform:** Windows or Linux

**Tool to be used:** FLEX and Bison

# Aim:

**Instructor Led Practical:**

**I1:** *Write YACC specification to check syntax of a simple expression involving operators +, -, \* and / and evaluate the expression.*

**CODE :-**

***LEX FILE –***

%{

#include "y.tab.h"

%}

%%

[0-9]+ {

}

[a-zA-Z] {

}

yylval=atoi(yytext); return NUMBER;

return ID;

\n {

}

return NL;

. {

return yytext[0];

}

%%

***YACC FILE –***

**%{**

**#include<stdio.h> #include<stdlib.h> int res=0;**

**%}**

**%token NUMBER ID NL**

**%left '+' '-'**

**%left '\*' '/'**

**%%**

**stmt: exp NL {**

**|**

**exp1 NL {**

**printf("Answer: %d.\n", $1); exit(0);**

**}**

**printf("Valid expression, no computation possible.\n"); exit(0);**

**}**

**;**

**exp: exp '+' exp {$$=$1+$3;}**

**| exp: exp '-' exp {$$=$1-$3;}**

**| exp: exp '\*' exp {$$=$1\*$3;}**

**| exp: exp '/' exp {$$=$1/$3;}**

**| '(' exp ')' {$$=$1;}**

**| NUMBER**

**;**

**exp1: exp1 '+' exp1**

**| exp1: exp1 '-' exp1**

**| exp1: exp1 '\*' exp1**

**| exp1: exp1 '/' exp1**

**| '(' exp1 ')'**

**| ID**

**;**

**%%**

**int yyerror(char \*msg)**

**{**

**printf("Invalid expression.\n"); exit(0);**

**}**

**main()**

**{**

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

**}**

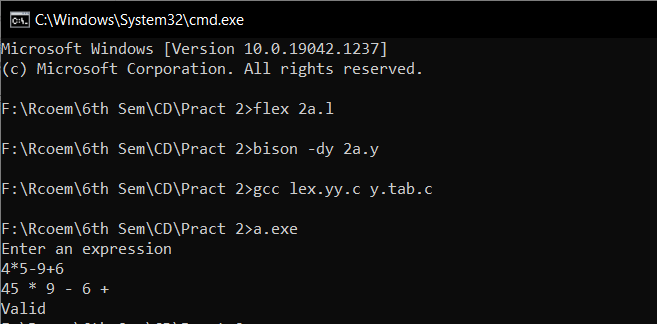
**int yywrap()**

**{**

**return 1;**

**}**

# INPUT & OUTPUT:-



**Evaluation Practical’s:**

**E1:** *Write YACC specification to check syntax of a simple expression involving operators +, -, \* and /. Also convert the arithmetic expression to postfix.*

**CODE :-**

***LEX FILE –***

%{

#include<stdio.h> #include "y.tab.h"

%}

letter [a-zA-Z] digit [0-9]

%%

{letter}(({letter}\*)|({digit}\*)) {yylval=yytext; return ID;} ({digit})+ {yylval = atoi(yytext); return NUMBER;}

\n {return 0;}

. {return yytext[0];}

%%

int yywrap(){ return 1;

}

***YACC FILE –***

%{

#include<stdio.h> int flag = 0;

%}

%token ID NUMBER

%left '+' '-'

%left '\*' '/'

%%

E: E '+' T {printf("+");}

| E '-' T {printf("-");}

| T

;

T: T '\*' F {printf("\*");}

| T '/' F {printf("/");}

| F

;

F: '(' E ')'

| ID {printf("%s",yylval);}

| NUMBER {printf("%d",yylval);}

;

%%

int main()

{

yyparse();

if(flag == 0) printf("\nValid expression");

}

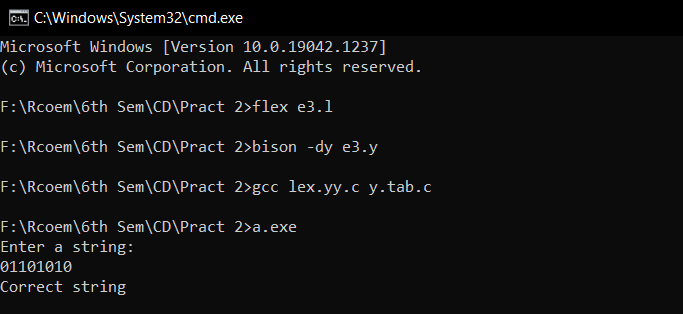
int yyerror (char \*msg)

{

printf ("\nEntered expression is invalid"); flag = 1;

}

# INPUT & OUTPUT:-



**E2:** *Write a YACC specification to accept strings that starts and ends with 0 or 1*.

**CODE :-**

***LEX FILE –***

%{

#include "y.tab.h"

%}

%%

0 {yylval = atoi(yytext); return ZERO;} 1 {yylval = atoi(yytext); return ONE;}

.|\n {yylval = 2; return 0;}

%%

***YACC FILE –***

%{

#include<stdio.h> #include <stdlib.h>

%}

%token ZERO ONE

%%

E : S {printf("Expression is Accepted\n");}

;

S : N

| ZERO A

| ONE B

;

A : N A

| ZERO

;

B : N B

| ONE

;

N : ZERO

| ONE

;

%%

int yyerror(char \*msg)

{

printf("Invalid Expression\n"); exit(0);

}

main()

{

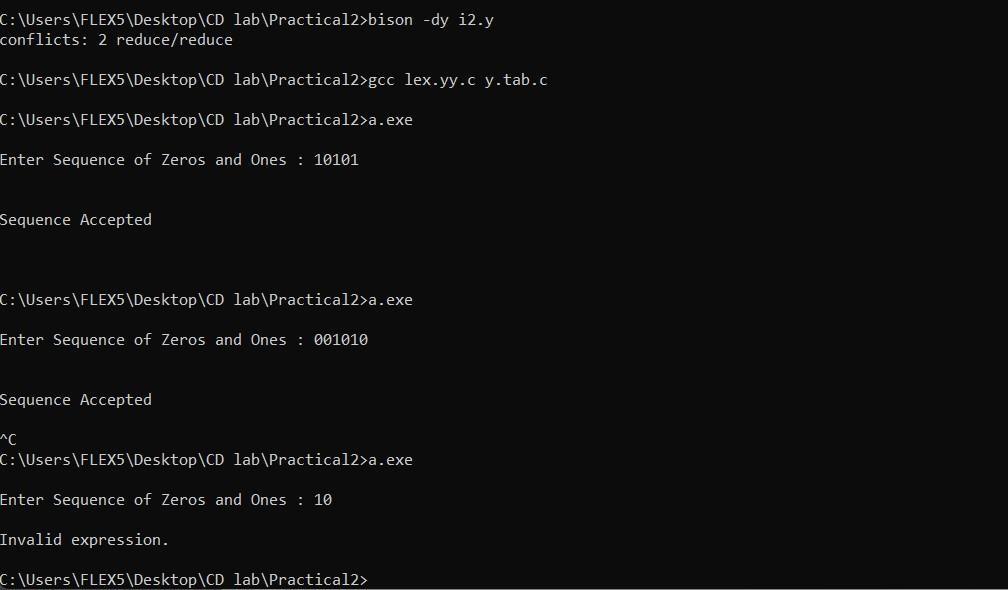
printf("\nEnter Expression : "); yyparse();

}

int yywrap(){ return 1;

}

# INPUT & OUTPUT:-



**E3:** *Write a YACC specification to validate the string having general form as below. Construct a proper grammar for the same and also write the corresponding LEX.:*

1. *Any alphabet(s) @ any alphabet + any digit – any digit.*

**CODE :-**

***LEX FILE –***

%{

#include "y.tab.h"

%}

%%

[a-zA-Z]+ {return ALPHA;}

"@" {return AT;}

"+" {return PLUS;}

"-" {return MINUS;}

[0-9]+ {return NUM;}

\n {return NL;}

. {return yytext[0];}

%%

***YACC FILE –***

%{

#include<stdio.h> #include<stdlib.h>

%}

%token ALPHA NUM AT PLUS MINUS NL

%%

r : s NL { printf("\nValid"); exit(0);

};

s : a AT a PLUS d MINUS d

;

a : ALPHA a

| ALPHA

;

d : NUM d

| NUM

;

%%

int yyerror(char \*msg)

{

printf("\nInvalid"); exit(0);

}

int main()

{

printf("Enter : "); yyparse(); printf("\n"); return 0;

}

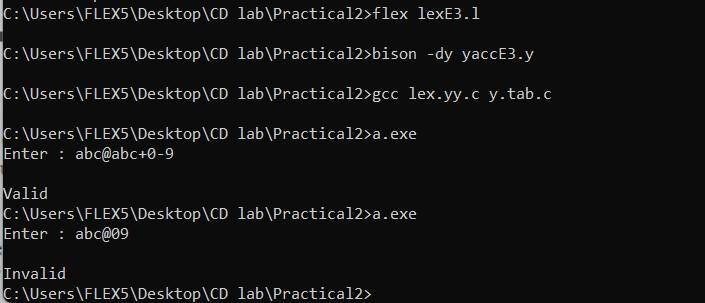
int yywrap()

{

return 1;

}

# INPUT & OUTPUT:-



1. *Date* **CODE :- *LEX FILE –***

%{

#include "y.tab.h"

%}

%%

[4-9] {yylval=atoi(yytext); return NUM;} 0 {yylval = 0; return ZERO;}

1. {yylval = 1; return ONE;}
2. {yylval = 2; return TWO;}
3. {yylval = 3; return THREE;} "/" {return SLASH;}

"-" {return DASH;}

\n { return NL;

}

%%

## YACC FILE –

%{

#include<stdio.h> #include<stdlib.h>

%}

%token NUM ZERO ONE TWO THREE SLASH DASH NL

%%

r : s NL { printf("Valid"); exit(0);

};

s : d SLASH d SLASH y

| d DASH d DASH y

;

d : a NUM

| NUM

| a a

| a THREE

| THREE ZERO

| THREE ONE

;

a : ONE

| TWO

| ZERO

y : b b

| b b b b

;

b : a

| NUM

| THREE

%%

int yyerror(char \*msg){ printf("Invalid Date"); exit(0);

}

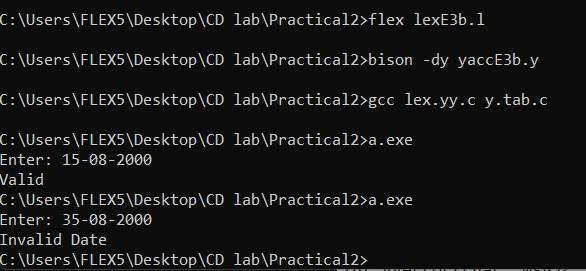
int main(){ printf("Enter: "); yyparse(); return 0;

}

int yywrap(){ return 1;

}

# INPUT & OUTPUT:-



1. *Expression of the form a=b\*c*

**CODE :-**

***LEX FILE –***

%{

#include "y.tab.h"

%}

%%

[a-zA-Z0-9]+ {return ALPHANUM;}

"=" {return EQUAL;} "\*" {return MUL;}

\n {return NL;}

. {return yytext[0];}

%%

***YACC FILE –***

%{

#include<stdio.h> #include<stdlib.h>

%}

%token ALPHANUM MUL EQUAL NL

%%

r : s NL { printf("Valid"); exit(0);

};

s : a EQUAL a MUL a

;

a : ALPHANUM

;

%%

int yyerror(char \*msg)

{

printf("Invalid"); exit(0);

}

int main()

{

printf("Enter : "); yyparse(); return 0;

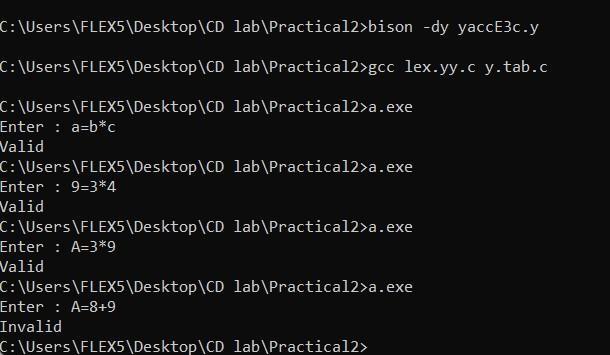
}

int yywrap()

{

return 1;

# INPUT & OUTPUT:-



**E4:** *To validate syntax of following programming language construct:*

1. *if else statement*

**CODE :-**

***LEX FILE –***

%{

#include "y.tab.h"

%}

%%

"if" { return IF;

}

"(" {

return OB;

}

")" {

return CB;

"{" {

return COB;

}

"}" {

return CCB;

}

"printf" { return PRINTF;

}

"\"" {

return SLASH;

}

"else" { return ELSE;

}

";" {

return SC;

}

"<"|">"|"<="|">="|"!=" {

return RO;

}

"&&"|"||" { return LO;

}

[0-9]+ {

return NUM;

}

[a-zA-Z]+ {

return ID;

}

\n { return NL;

%%

***YACC FILE –***

%{

#include<stdio.h> #include<stdlib.h>

%}

%token IF OB CB COB CCB PRINTF ELSE SC RO LO NUM ID NL SLASH

%%

r : s NL { printf("Valid"); exit(0);

};

s : IF OB a RO a CB COB PRINTF OB SLASH b SLASH CB SC CCB ELSE COB PRINTF OB SLASH b SLASH CB SC CCB

;

a : NUM

| ID

;

b : ID b

| NUM b

| NUM

| ID

;

%%

int yyerror(char \*msg) { printf("Invalid");

exit(0);

}

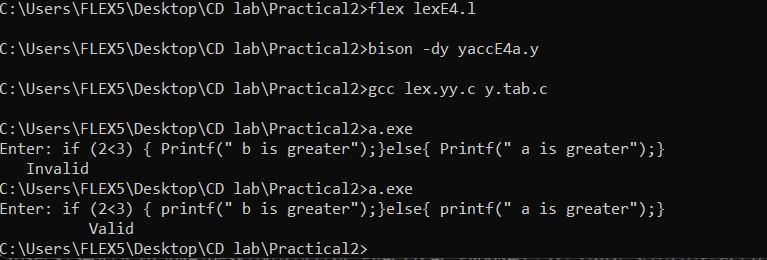
int main(){ printf("Enter: "); yyparse(); return 0;

}

int yywrap(){ return 1;

}

# INPUT & OUTPUT:-



1. *for loop* **CODE :- *LEX FILE –***

%{

#include "y.tab.h"

%}

%%

"for" { return FOR;

}

"(" {

return OB;

}

")" {

return CB;

}

"int"|"float"|"double" {

return DT;

}

[a-zA-Z]+ {

return ID;

}

"=" {

return EQUAL;

}

[0-9]+ {

return NUM;

}

";" {

return SC;

}

"<"|">"|"<="|">="|"=="|"!=" {

return RO;

}

"&&"|"||" { return LO;

}

"++"|"--" {

return INDE;

}

"{" {

return OCB;

}

"}" {

return CCB;

}

\n { return NL;

}

%%

## YACC FILE –

%{

#include<stdio.h> #include<stdlib.h>

%}

%token FOR OB CB DT ID EQUAL NUM SC RO LO INDE OCB CCB NL

%%

r : s NL { printf("Valid"); exit(0);

};

s : FOR OB DT ID EQUAL a SC ID RO a SC ID INDE CB OCB CCB

;

a : ID

| NUM

;

%%

int yyerror(char \*msg){ printf("Invalid"); exit(0);

}

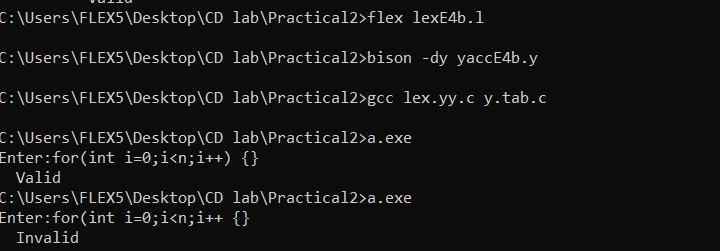
int main(){ printf("Enter:"); yyparse(); return 0;

}

int yywrap(){ return 1;

}

# INPUT & OUTPUT:-



**E5:** *Write YACC specification to recognize strings that can be accepted by grammar of the form: an bn c, n>=1.*

**CODE :-**

***LEX FILE –***

%{

#include "y.tab.h"

%}

%%

1. {return A;}
2. {return B;}
3. {return C;}

\n {return NL;}

. {return yytext[0];}

%%

***YACC FILE –***

%{

#include<stdio.h> #include <stdlib.h>

%}

%token A B C NL

%%

S: A r B C NL {printf("Valid string\n"); exit(0);}

;

r: A r B

|

;

%%

int yyerror(char \*msg)

{

printf("Invalid Expression\n"); exit(0);

}

main()

{

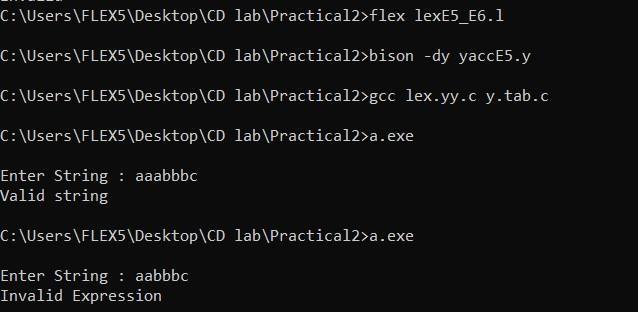
printf("\nEnter String : "); yyparse();

}

int yywrap(){ return 1;

}

# INPUT & OUTPUT:-



**E6:** *Write YACC specification to recognize strings that can be accepted by grammar of the form: {L= an b2n c, n>=1 } .*

**CODE :-**

***LEX FILE –***

%{

#include "y.tab.h"

%}

%%

1. {return A;}
2. {return B;}
3. {return C;}

\n {return NL;}

. {return yytext[0];}

%%

***YACC FILE –***

%{

#include<stdio.h> #include <stdlib.h>

%}

%token A B C NL

%%

S: A r B B C NL {printf("Valid string\n"); exit(0);}

;

r: A r B B

|

;

%%

int yyerror(char \*msg)

{

printf("Invalid Expression\n"); exit(0);

}

main()

{

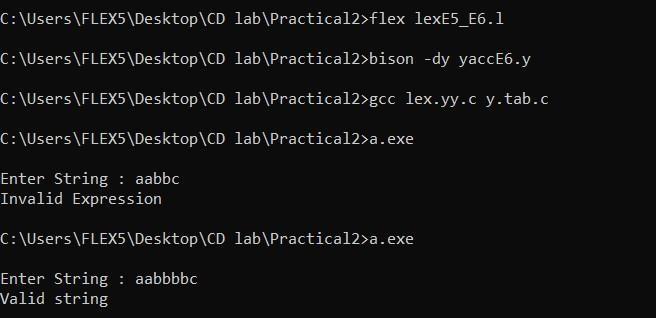
printf("\nEnter String : "); yyparse();

}

int yywrap(){ return 1;

}

# INPUT & OUTPUT:-



**Sample Input & Output:**

### Output: The Entered expression is valid