# PROGRAM 9: Design a DFA in LEX Code which accepts string containing even number of ‘a’ and even number of ‘b’ over input alphabet {a, b}.

**Code:**

%{

#include<stdio.h>

%}

reg (aa|bb)\*((ab|ba)(aa|bb)\*(ab|ba)(aa|bb)\*)\*

%%

{reg} printf(%s is accepted",yytext);

.\* printf("%s is not accepted",yytext);

%%

int yywrap(){}

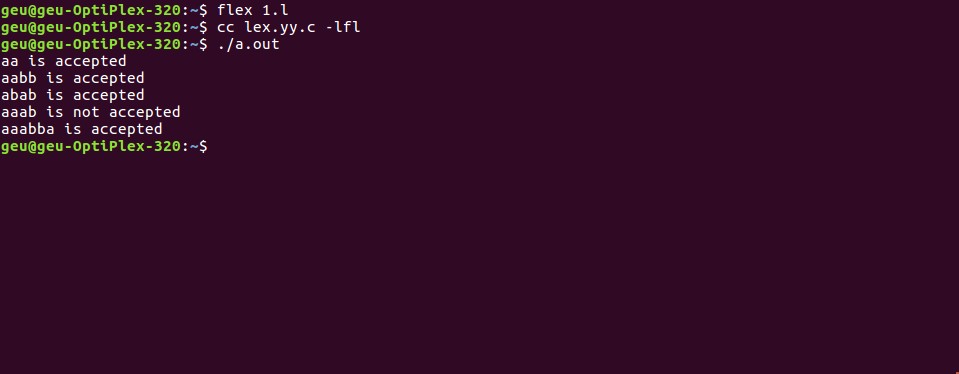
int main(int argc, char \*argv[]){ extern FILE \*yyin;

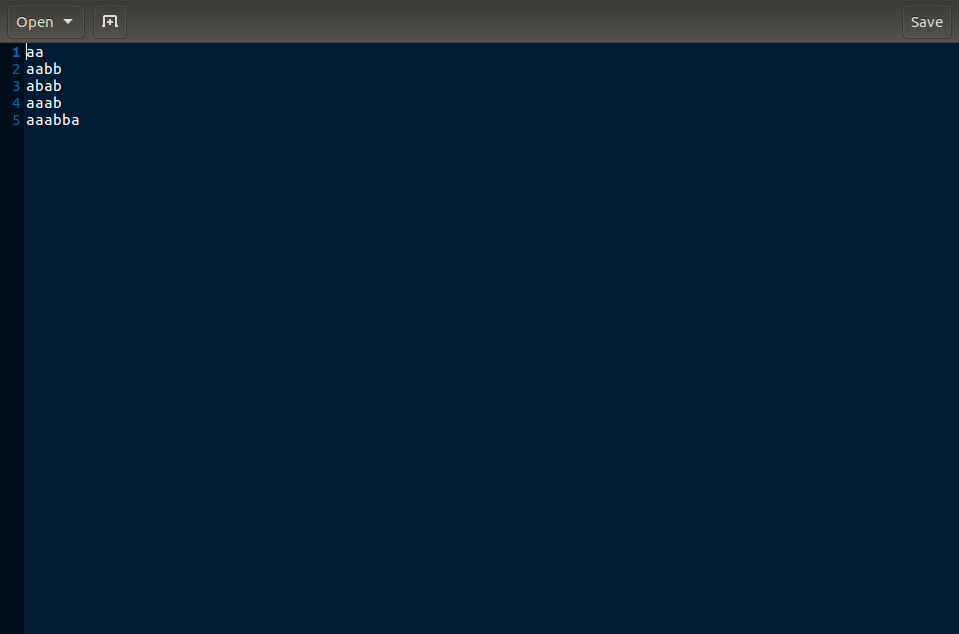
yyin= fopen("Input.txt","r"); yylex();

return 0;

}

# OUTPUT:





**PROGRAM 10: Design a DFA in LEX Code which accepts string containing third last element ‘a’ over input alphabet {a, b}.**

**Code:**

%{

#include<stdio.h>

%}

reg (a\*b\*)\*a(aa|bb|ab|ba)

%%

{reg} printf("%s is accepted", yytext);

.\* printf("%s is not accepted", yytext);

%%

int yywrap(){return 1;} int main()

{

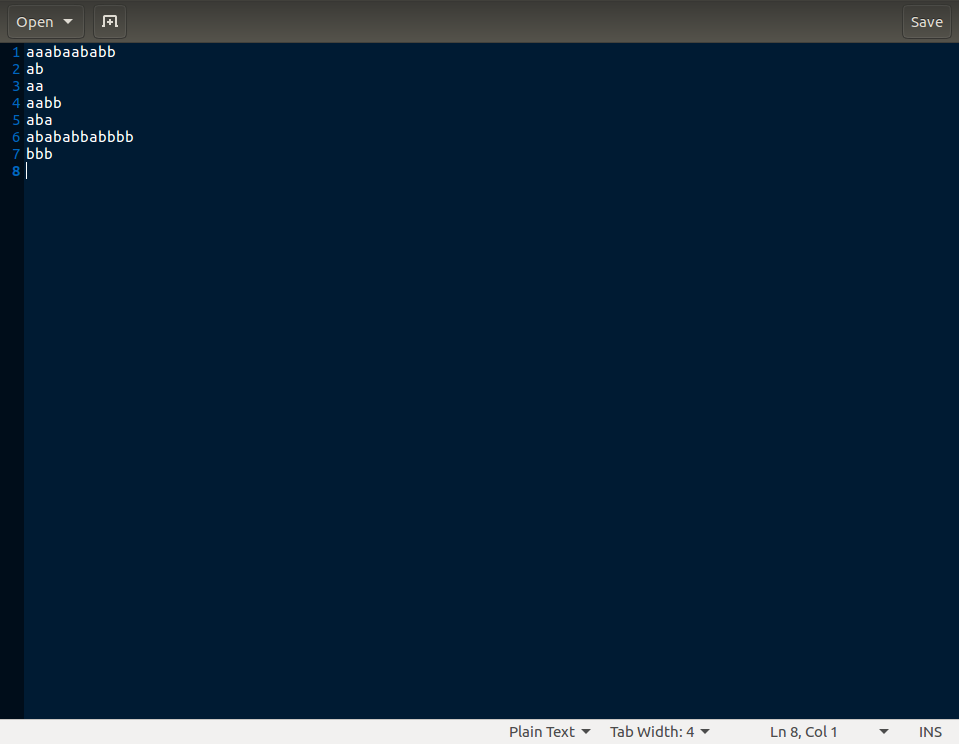
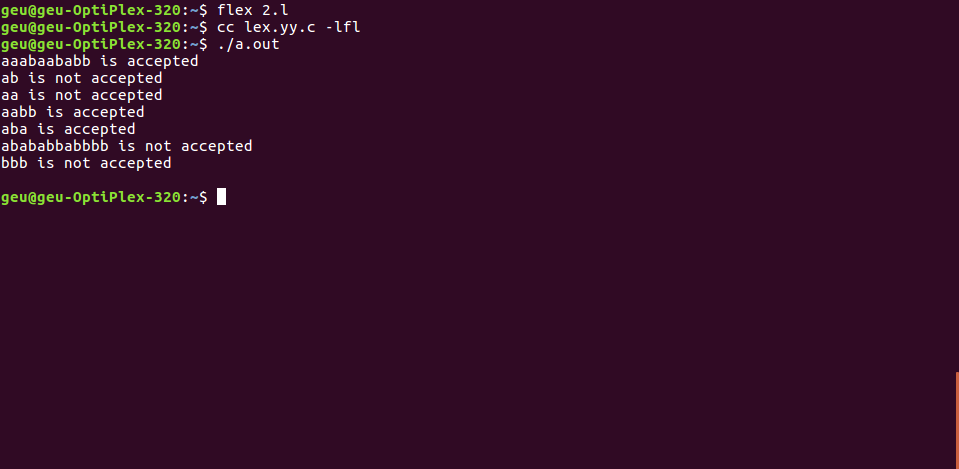
extern FILE \*yyin;

yyin = fopen("Input2.txt","r"); yylex();

return 0;

}

# OUTPUT:



**PROGRAM 11: Design a DFA in LEX Code to Identify and print Integer & Float Constants and Identifier.**

**Code:**

%{

#include<stdio.h>

%}

%%

[+-]?[0-9]"."[0-9]+ printf("%s is Float Constants",yytext); [+-]?[0-9]+ printf("%s is Integer Constants",yytext);

[a-zA-Z]+|[a-zA-Z]+[0-9]+ printf("%s is Identifiers",yytext);

.\* printf("%s is Neither Ineger, Float constants nor identifiers",yytext);

%%

int yywrap(){} int main()

{

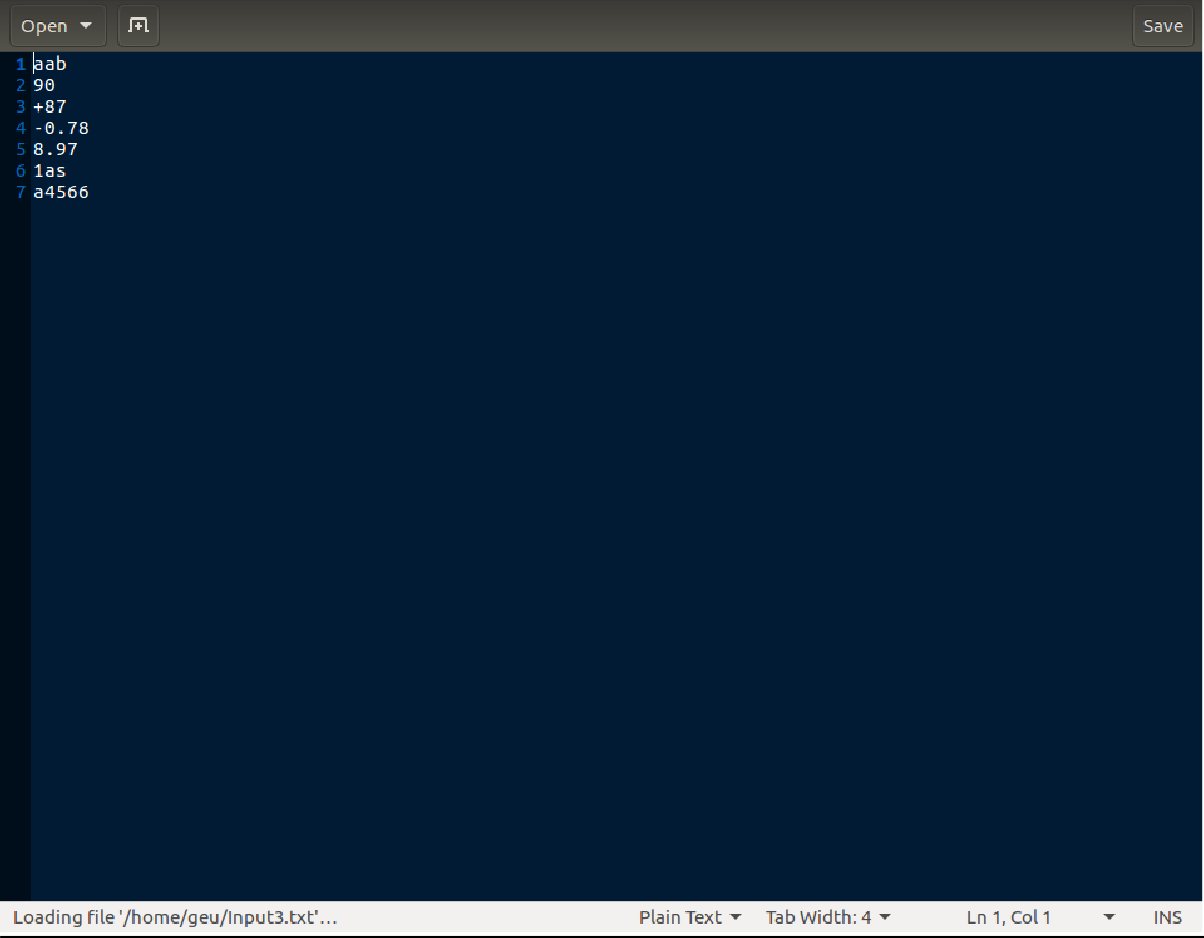
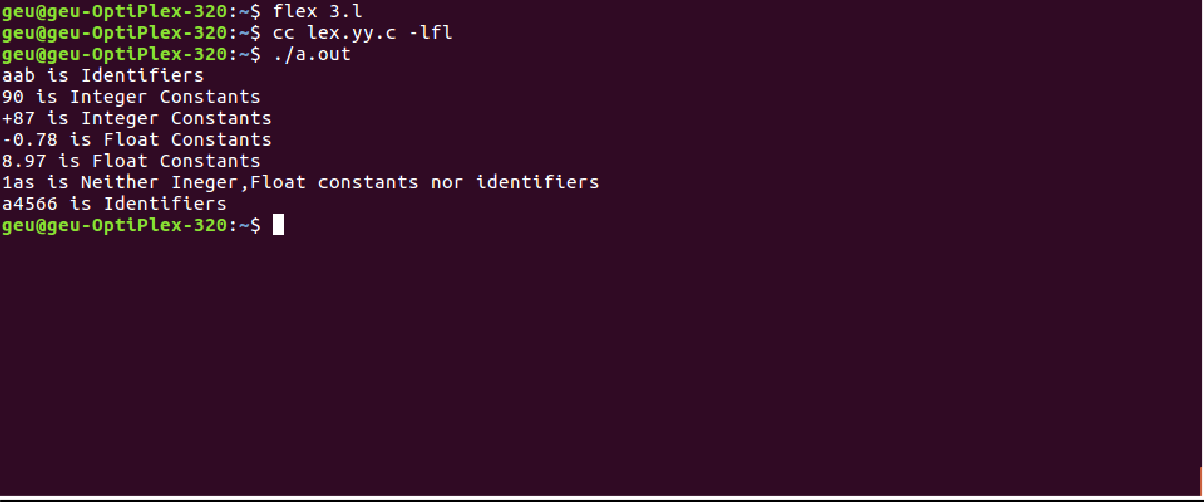
extern FILE \*yyin;

yyin = fopen("Input3.txt","r"); yylex();

return 0;

}

# OUTPUT:



**PROGRAM 12: Design YACC/LEX code to recognize valid arithmetic expression with operators**

**+, -, \* and /.**

**Code:**

%{

#include <stdio.h> #include <string.h>

int operators\_count = 0, operands\_count = 0, valid = 1, top = -1, l = 0, j = 0; char operands[10][10], operators[10][10], stack[100];

%}

%%

"(" {top++; stack[top] = '(';}

"{" {top++; stack[top] = '{';}

"[" {top++; stack[top] = '[';}

")" {if (stack[top] != '('){ valid = 0;

}else if(operands\_count>0 && (operands\_count-operators\_count)!=1){ valid=0;

}else{

top--; operands\_count=1; operators\_count=0;

}

}

"}" {

if(stack[top] != '{') { valid = 0;

}else if(operands\_count>0 && (operands\_count-operators\_count)!=1){ valid=0;

}else{

top--; operands\_count=1; operators\_count=0;

}

}

"]" {if (stack[top] != '[') { valid = 0;

}else if(operands\_count>0 && (operands\_count-operators\_count)!=1){ valid=0;

}else{

top--; operands\_count=1; operators\_count=0;

}

}

"+"|"-"|"\*"|"/" { operators\_count++; strcpy(operators[l], yytext); l++;

}

[0-9]+|[a-zA-Z][a-zA-Z0-9\_]\* {

operands\_count++; strcpy(operands[j], yytext); j++;

}

%%

int yywrap(){return 1;} int main()

{

int k;

printf("Enter the arithmetic expression: "); yylex();

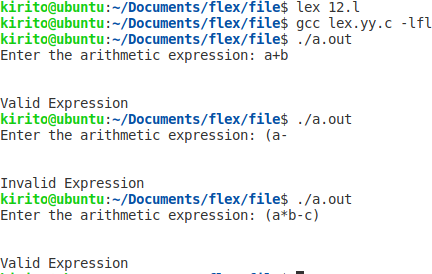
if (valid == 1 && top == -1) { printf("\nValid Expression\n");

}else

printf("\nInvalid Expression\n"); return 0;

}

# Output:



**PROGRAM 13: Design YACC/LEX code to evaluate arithmetic expression involving operators +,**

**-, \* and / without operator precedence grammar. Lex Program:**

%{

#include<stdio.h> #include "y.tab.h" extern int yylval;

%}

%%

[0-9]+ {

yylval=atoi(yytext); return NUMBER;

}

[\t] ;

[\n] return 0;

. return yytext[0];

%%

int yywrap()

{

return 1;

}

# Yacc Program:

%{

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

%}

%token NUMBER

%left '+' '-'

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

%left '(' ')'

%%

ArithmeticExpression: E{ printf("\nResult=%d\n",$$); return 0;

};

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

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

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

|E'/'E {$$=$1/$3;}

|E'%'E {$$=$1%$3;}

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

| NUMBER {$$=$1;}

;

%%

void main()

{

printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");

yyparse(); if(flag==0)

printf("\nEntered arithmetic expression is Valid\n\n");

}

void yyerror()

{

printf("\nEntered arithmetic expression is Invalid\n\n"); flag=1;

}

# Output:

**2. Design YACC/LEX code to evaluate arithmetic expression involving operators +, -, \* and / with operator precedence grammar.**

**Lex Program:**

%{

#include<stdio.h> #include "y.tab.h" extern int yylval;

%}

%%

[0-9]+ {

yylval=atoi(yytext); return NUMBER;

}

[\t] ;

[\n] return 0;

. return yytext[0];

%%

int yywrap()

{

return 1;

}

# Yacc Program:

%{

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

%token NUMBER

%left '+' '-'

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

%left '(' ')'

%%

ArithmeticExpression: E{ printf("\nResult=%d\n",$$); return 0;

}

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

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

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

|E'/'E {$$=$1/$3;}

|E'%'E {$$=$1%$3;}

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

| NUMBER {$$=$1;}

;

%%

void main()

{

printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");

yyparse();if(flag==0)

printf("\nEntered arithmetic expression is Valid\n\n");

}

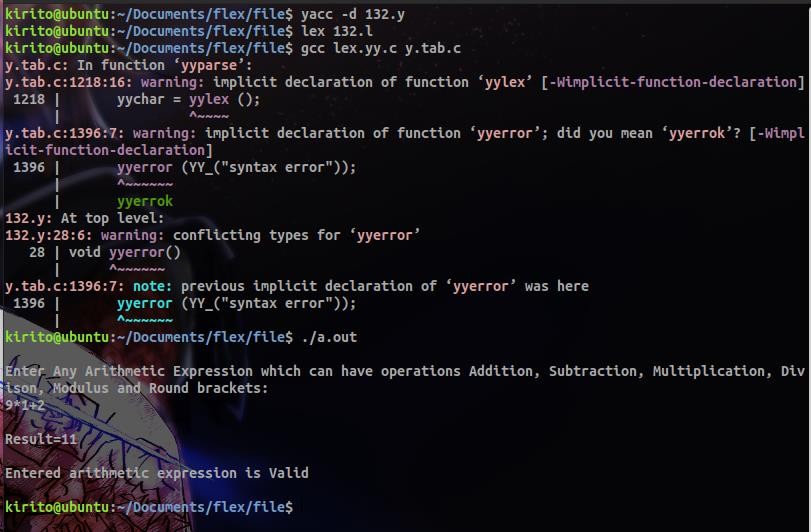
void yyerror()

{

printf("\nEntered arithmetic expression is Invalid\n\n"); flag=1;

}

# Output:



**PROGRAM 14: Design YACC/LEX code to evaluate arithmetic expression involving operators**

**+, -, \* and / with operator precedence grammar. Lex Program:**

%{

#include"y.tab.h" extern int yylval;

%}

%%

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

\n return 0;

. return \*yytext;

%%

int yywrap(){ return 1;

}

# Yacc Program:

%{

#include<stdio.h>

%}

%token NUM

%left '+' '-'

%left '\*' '/'

%right NEGATIVE

%%

S: E {printf("\n");} ;E: E '+' E {printf("+");}

| E '\*' E {printf("\*");}

| E '-' E {printf("-");}

| E '/' E {printf("/");}

| '(' E ')'

| '-' E %prec NEGATIVE {printf("-");}

| NUM

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

;

%%

int main(){ yyparse();

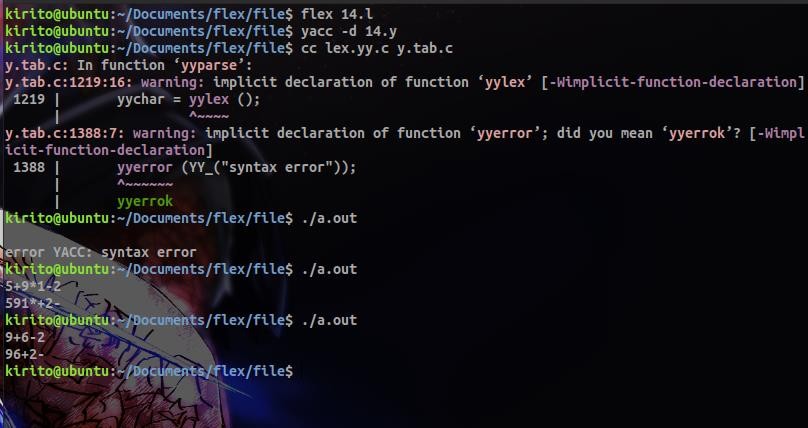
}

int yyerror (char \*msg) {

return printf ("error YACC: %s\n", msg);

}

# Output:



**PROGRAM-15: Design Desk Calculator using YACC/LEX code. Lex Program:**

%{

#include<stdio.h> #include "y.tab.h" extern int yylval;

%}

%%

[0-9]+ {

yylval=atoi(yytext); return NUMBER;

}

[\t] ;

[\n] return 0;

. return yytext[0];

%%

int yywrap()

{

return 1;

}

# Yacc Program:

%{

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

%}

%token NUMBER

%left '+' '-'

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

%left '(' ')'

%%

ArithmeticExpression: E{ printf("\nResult=%d\n",$$); return 0;

};

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

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

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

|E'/'E {$$=$1/$3;}

|E'%'E {$$=$1%$3;}

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

| NUMBER {$$=$1;}

;

%%

void main()

{

printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");

yyparse(); if(flag==0)

printf("\nEntered arithmetic expression is Valid\n\n");

}

void yyerror()

{

printf("\nEntered arithmetic expression is Invalid\n\n"); flag=1;

}

# Output:

