**PROGRAM-11**

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

**Program:**

%{

%}

%s A B C DEAD

%%

<INITIAL>[0-9]+ BEGIN A;

<INITIAL>[0-9]+[.][0-9]+ BEGIN B;

<INITIAL>[A-Za-z\_][A-Za-z0-9\_]\* BEGIN C;

<INITIAL>[^\n] BEGIN DEAD;

<INITIAL>\n BEGIN INITIAL; {printf("Not Accepted\n");}

<A>[^\n] BEGIN DEAD;

<A>\n BEGIN INITIAL; {printf("Integer\n");}

<B>[^\n] BEGIN DEAD;

<B>\n BEGIN INITIAL; {printf("Float\n");}

<C>[^\n] BEGIN DEAD;

<C>\n BEGIN INITIAL; {printf("Identifier\n");}

<DEAD>[^\n] BEGIN DEAD;

<DEAD>\n BEGIN INITIAL; {printf("Invalid\n");}

%%

int yywrap() {return 1;}

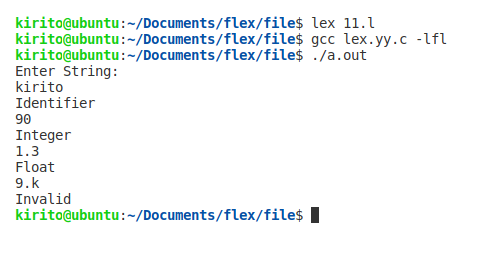
int main()

{

printf("Enter String:\n");

yylex();

return 0;

}

**Output:**

**PROGRAM-12**

**Design YACC/LEX code to recognize valid arithmetic expression with operators +, -, \* and /.**

**Program:**

%{

#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");

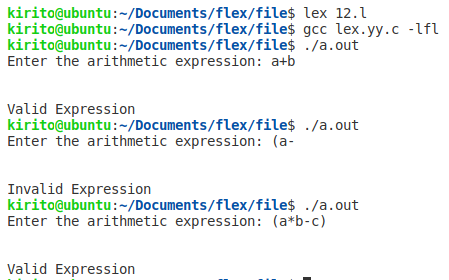
}else

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()

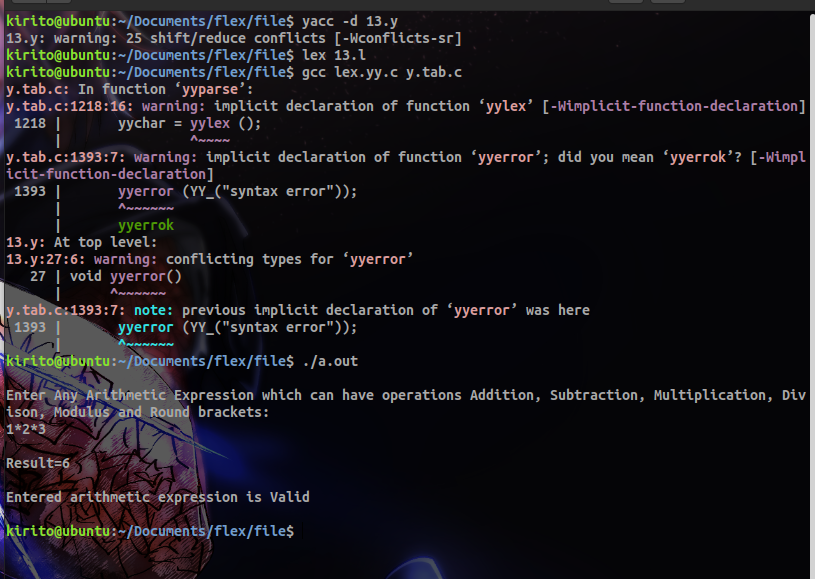
{

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()

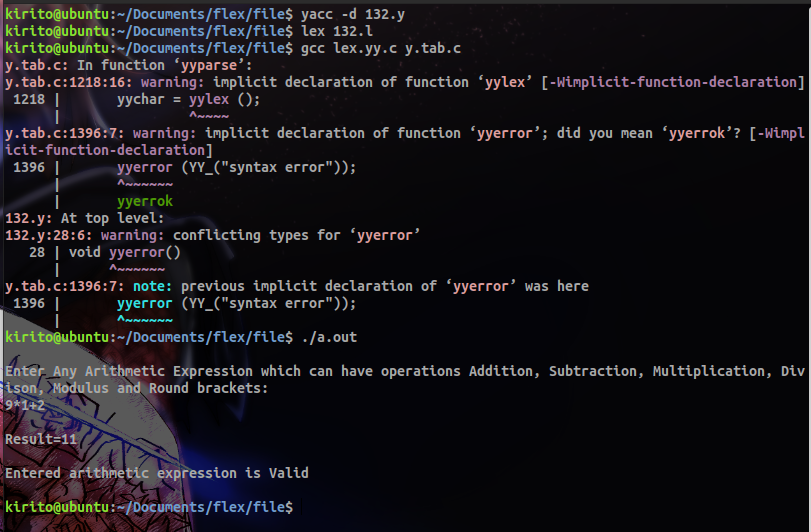
{

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();

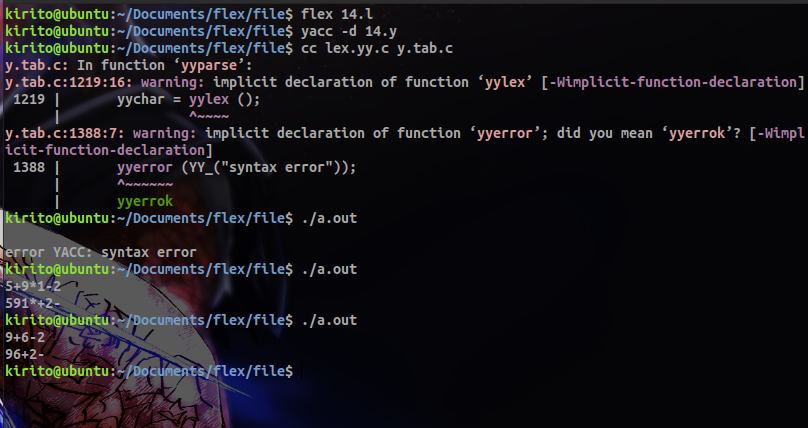
}

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()

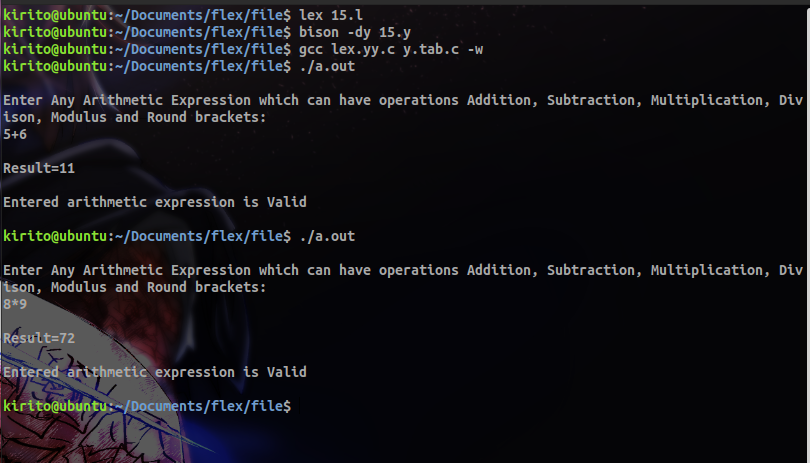
{

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

flag=1;

}

**Output:**

****