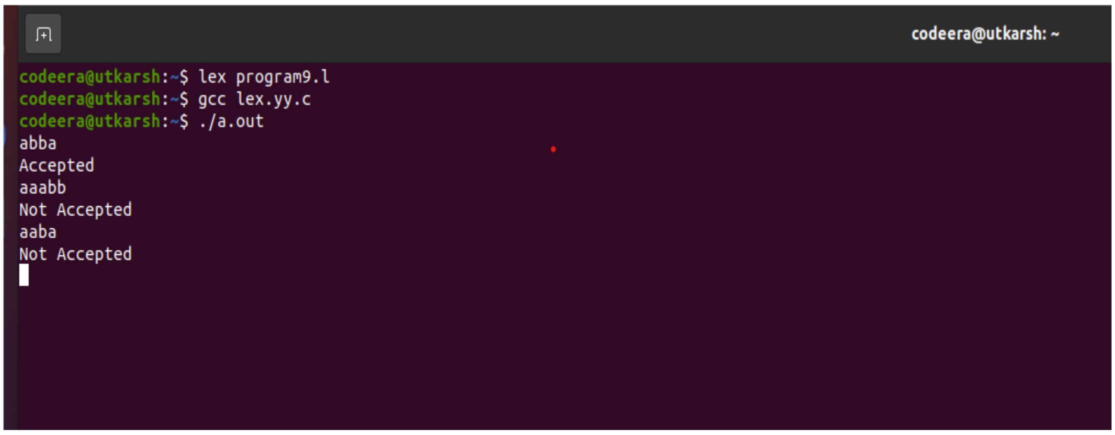
LAB PROGRAM 9

Design a DFA inn LEX Code which accepts the strings containing even number of a's and even number of b's over input alphabet {a,b}

```
%{
%}

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

%%
{reg}  {printf("Accepted");}
.*       {printf("Not Accepted");}
%%

int yywrap(){}

int main()
{
        yylex();
        return 0;
}
```

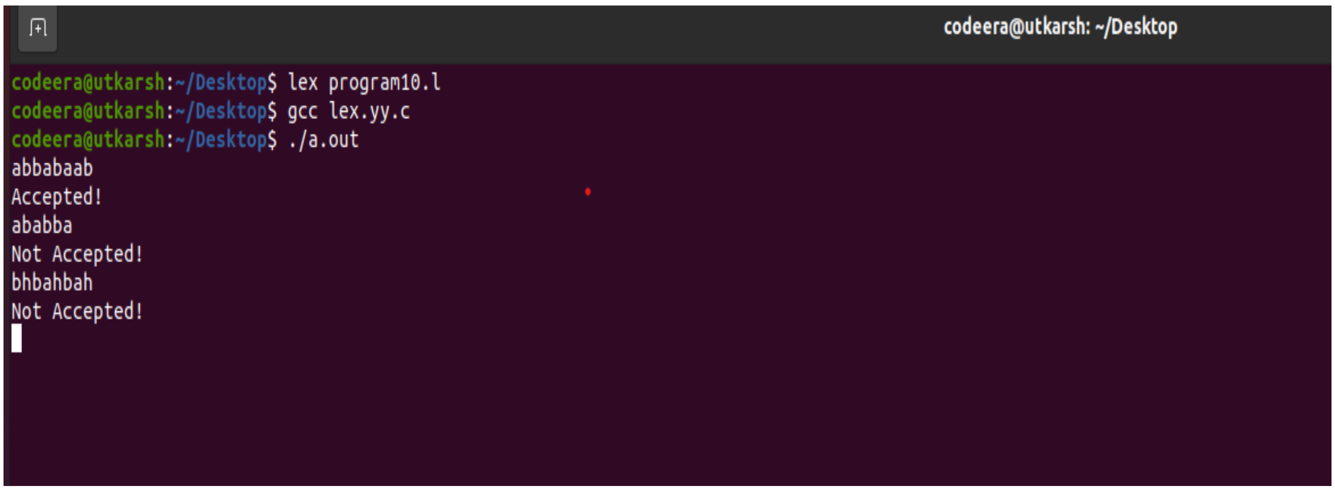


```
codeera@utkarsh:~$ lex program9.l
codeera@utkarsh:~$ gcc lex.yy.c
codeera@utkarsh:~$ ./a.out
abba
Accepted
aaabb
Not Accepted
aaba
Not Accepted
```

LAB PROGRAM 10:

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

```
%{
%}

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

%%
{reg} {printf("Accepted!");}
.*      {printf("Not Accepted!");}
%%

int yywrap(){}

int main(){
yylex();
return 0;
}
```



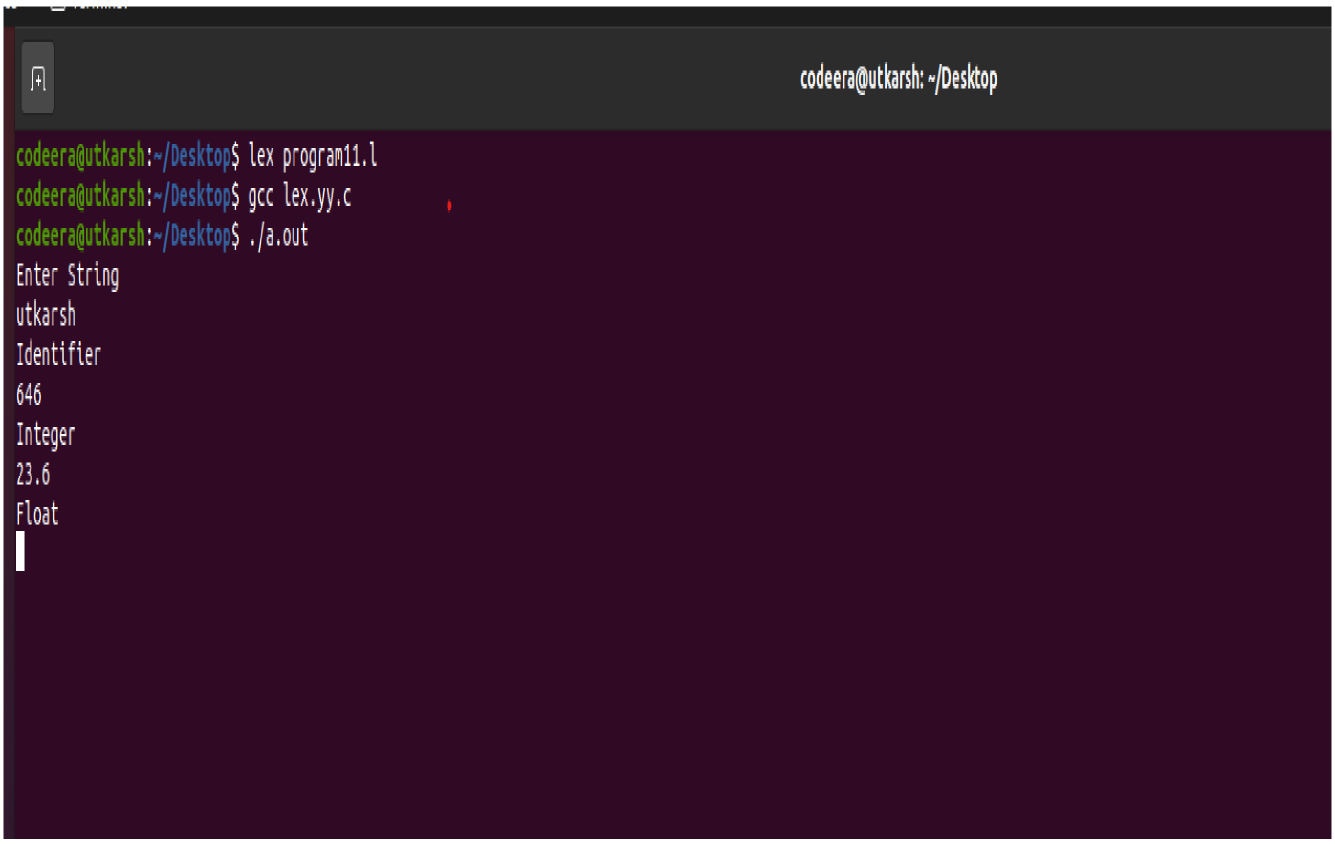
```
codeera@utkarsh: ~/Desktop
codeera@utkarsh:~/Desktop$ lex program10.l
codeera@utkarsh:~/Desktop$ gcc lex.yy.c
codeera@utkarsh:~/Desktop$ ./a.out
abbabaab
Accepted!
ababba
Not Accepted!
bhbahbah
Not Accepted!
```

LAB PROGRAM 11

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

```
%{
%}


%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;
}
```



codeera@utkarsh: ~/Desktop

```
codeera@utkarsh:~/Desktop$ lex program11.l
codeera@utkarsh:~/Desktop$ gcc lex.yy.c
codeera@utkarsh:~/Desktop$ ./a.out
Enter String
utkarsh
Identifier
646
Integer
23.6
Float
```

LAB PROGRAM 12

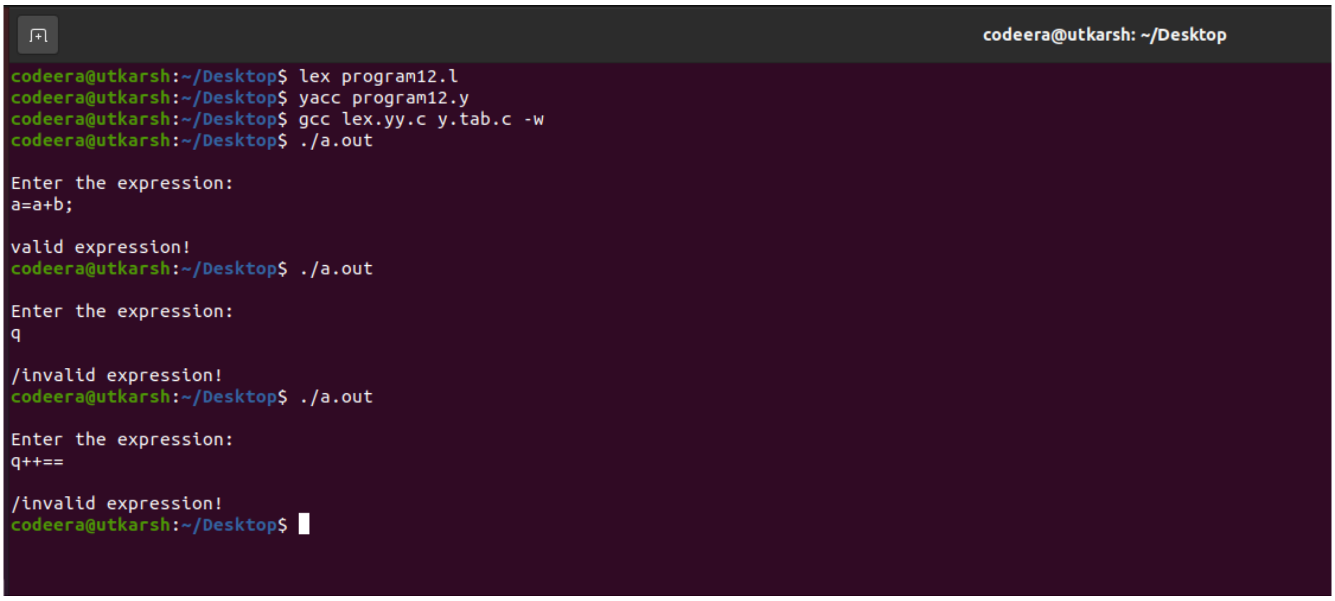
Design YACC/LEX code to recognize valid arithmetic expression with operators +, -, * and /

LEX Code:

```
%{
#include "y.tab.h"
%}
%%
[a-zA-Z_][a-zA-Z_0-9]* return id;
[0-9]+(\.[0-9]*)? return num;
[+/*] return op;
. return yytext[0];
\n return 0;
%%
int yywrap()
{
return 1;
}
```

YACC Code:

```
%{
#include<stdio.h>
int valid=1;
%}
%token num id op
%%
start : id '=' s ';'
s : id x
| num x
| '-' num x
| '(' s ')' x
;
x : op s
| '-' s
|
;
%%
int yyerror()
{
valid=0;
printf("\n/invalid expression!\n");
return 0;
}
int main()
{
printf("\nEnter the expression:\n");
yyparse();
if(valid)
{
printf("\nvalid expression!\n");
}
}
```



```
codeera@utkarsh:~/Desktop$ lex program12.l
codeera@utkarsh:~/Desktop$ yacc program12.y
codeera@utkarsh:~/Desktop$ gcc lex.yy.c y.tab.c -w
codeera@utkarsh:~/Desktop$ ./a.out

Enter the expression:
a=a+b;

valid expression!
codeera@utkarsh:~/Desktop$ ./a.out

Enter the expression:
q

/invalid expression!
codeera@utkarsh:~/Desktop$ ./a.out

Enter the expression:
q++==

/invalid expression!
codeera@utkarsh:~/Desktop$
```

LAB PROGRAM 13:

Design YACC/LEX code to evaluate arithmetic expression involving operators +, -, * and / without operator precedence grammar & with operator precedence grammar.

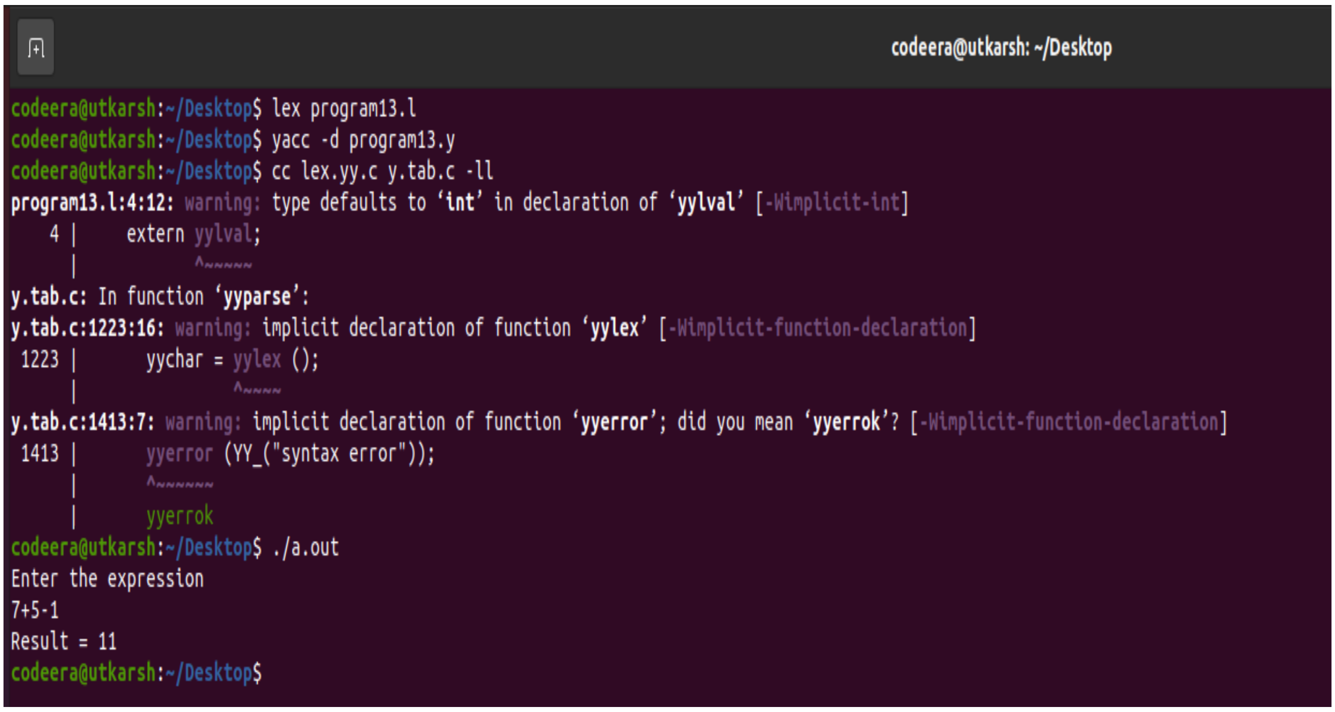
LEX Code

```
%{
   /* Definition section*/
   #include "y.tab.h"
   extern yylval;
%}

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

[a-zA-Z]+    { return ID; }
[ \t]+        ;  /For skipping whitespaces/

\n          { return 0; }
.         { return yytext[0]; }

%%
```

<u>YACC Code</u>

```
%{
   /* Definition section */
  #include <stdio.h>
%}

%token NUMBER ID
// setting the precedence
// and associativity of operators
%left '+' '-'
%left '*' '/'

/* Rule Section */
%%
E : T       {
              printf("Result = %d\n", $$);
              return 0;
            }

T : T '+' T { $$ = $1 + $3; } | T '-' T { $$ = $1 - $3; }| T '*' T { $$ = $1 * $3; } | T '/' T
{ $$ = $1 / $3; }| '-' NUMBER { $$ = -$2; } | '-' ID { $$ = -$2; } | '(' T ')' { $$ = $2; } |
NUMBER { $$ = $1; }| ID { $$ = $1; };

%%
```

```
int main() {
    printf("Enter the expression\n");
    yyparse();
}


/* For printing error messages */
int yyerror(char* s) {
    printf("\nExpression is invalid\n");
}
```



```
codeera@utkarsh: ~/Desktop
codeera@utkarsh:~/Desktop$ lex program13.l
codeera@utkarsh:~/Desktop$ yacc -d program13.y
codeera@utkarsh:~/Desktop$ cc lex.yy.c y.tab.c -ll
program13.l:4:12: warning: type defaults to ‘int’ in declaration of ‘yylval’ [-Wimplicit-int]
    4 |     extern yylval;
      |            ^~~~~~
y.tab.c: In function ‘yyparse’:
y.tab.c:1223:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-function-declaration]
 1223 |       yychar = yylex ();
      |                ^~~~~
y.tab.c:1413:7: warning: implicit declaration of function ‘yyerror’; did you mean ‘yyerrok’? [-Wimplicit-function-declaration]
 1413 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
      |       yyerrok
codeera@utkarsh:~/Desktop$ ./a.out
Enter the expression
7+5-1
Result = 11
codeera@utkarsh:~/Desktop$
```

PROGRAM 14:

Design YACC/LEX code that translates infix expression to postfixexpression.

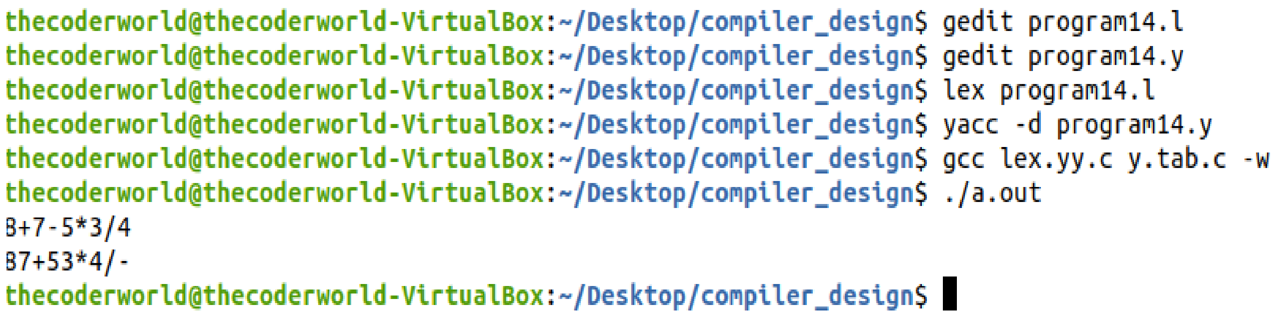
LEX CODE:

```
%{
#include"y.tab.h"
extern int yylval;
%}
%%
[0-9]+ {yylval=atoi(yytext); return NUM;}
\n return 0;
. return *yytext;
%%
int yywrap(){ return 1;}
```

YACC CODE:

```
%{
#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);
```



```
thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$ gedit program14.l
thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$ gedit program14.y
thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$ lex program14.l
thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$ yacc -d program14.y
thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$ gcc lex.yy.c y.tab.c -w
thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$ ./a.out
3+7-5*3/4
37+53*4/-
thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$ 
```

Program 15
YACC/LEX program to implement simple desk calculator.

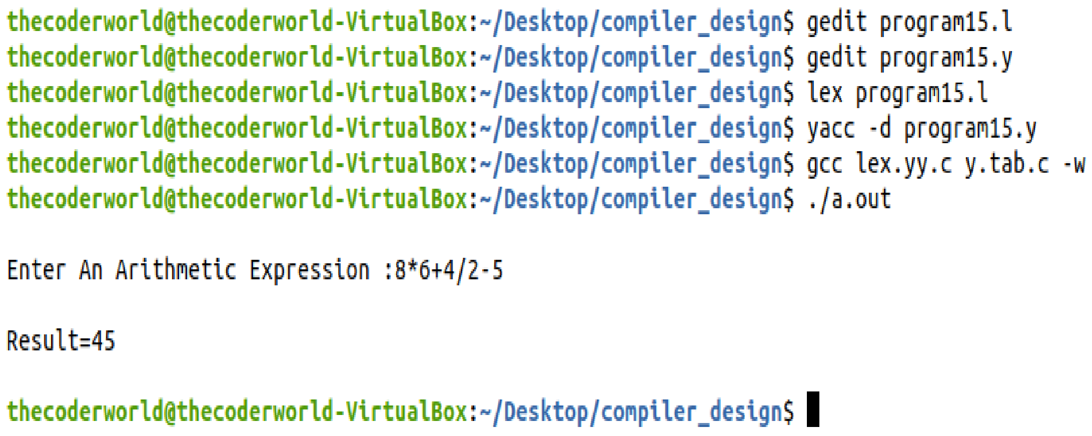
LEX CODE:

```
%{
#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 CODE:

```
%{
/* Definition section */
#include<stdio.h>
int flag=0;
%}
%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
/* Rule Section */
%%
ArithmeticExpression: E{printf("\nResult=%d\n\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 An Arithmetic Expression :");yyparse();
}
void yyerror()
{
printf("\nEntered arithmetic expression is Invalid\n\n");flag=1;
}
```



```
thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$ gedit program15.l
thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$ gedit program15.y
thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$ lex program15.l
thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$ yacc -d program15.y
thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$ gcc lex.yy.c y.tab.c -w
thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$ ./a.out

Enter An Arithmetic Expression :8*6+4/2-5

Result=45

thecoderworld@thecoderworld-VirtualBox:~/Desktop/compiler_design$
```