1. Implement LALR parser using LEX and YACC for the following Grammar:

E → E+T |T
E'→ T*F | F
F → (E) | d

LEX

```
%{
#include "parser.tab.h"
%}
%%
[0-9]+ {yylval=atoi(yytext);
return NUMBER;
}
[\t] ;
\n return 0;
. return yytext[0];
%%
```

YACC implementation:

```
%{
#include<stdio.h>
%}
%token NUMBER
%%
S: E { printf("The result is =%d\n",$1);}
;
E: E'+'T { $$ = $1 + $3; }
| T { $$ = $1;}
;
T: T'*'F { $$ = $1 * $3; }
| F { $$ = $1;}
;
```

```
F: '('E')' { $$ = $2;}
| NUMBER { $$ = $1;}
;
%%
int main(){
yyparse();
}
int yywrap(){
return 1;
}
void yyerror(char *s){
printf("Error %s",s);
}
```

Precedence and associativity in YACC

LEX:

```
%{
#include "parser2.tab.h"
%}
%%
[0-9]+ {yylval=atoi(yytext);
return DIGIT;
}
[\t] ;
```

```
\n return 0;
. return yytext[0];
%%


YACC:
%{
#include<stdio.h>
%}
%token DIGIT
%left '-' '+'
%left '*' '/'
%nonassoc UMINUS
%%
S: E { printf("The result is =%d\n",$1);}
;
E: E'+'E { $$ = $1 + $3; }
| E'-'E { $$ = $1 - $3; }
| E'*'E { $$ = $1 * $3; }
| E'/'E { if($3 == 0)

yyerror("Divide by zero");
else
$$ = $1 / $3; }
| '-'E %prec UMINUS { $$ = -$2; }
| '(' E ')' { $$ = $2; }
| DIGIT {$$ = $1;}
;
%%
int main()
{
yyparse();
}
int yywrap(){
return 1;
}
void yyerror(char *s){
printf("Error %s",s);exit(0);
}
```

```
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. All rights reserved.

C:\Users\chandugeetu>cd Desktop

C:\Users\chandugeetu\Desktop>flex parser2.l

C:\Users\chandugeetu\Desktop>yacc -d parser2.y

C:\Users\chandugeetu\Desktop>gcc lex.yy.c parser2.tab.c -w

C:\Users\chandugeetu\Desktop>a
4+3*4
The result is =16

C:\Users\chandugeetu\Desktop>a
3+-4*5
The result is =-17

C:\Users\chandugeetu\Desktop>a
4*/3
Error syntax error
C:\Users\chandugeetu\Desktop>a
5/(3-3)
Error Divide by zero
C:\Users\chandugeetu\Desktop>_
```