WEEK 6

Design a suitable grammar for evaluation of
arithmetic expression having + and – operators.
+ has least priority and it is left associative
- has higher priority and is right associative

lex

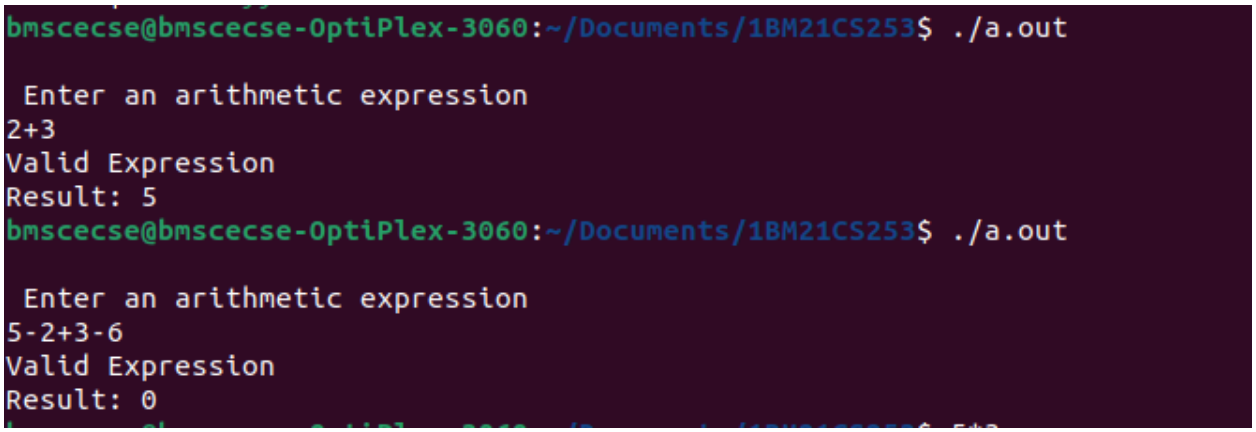
```
%{
#include "y.tab.h"
%}

%%
[0-9]+ {yylval=atoi(yytext); return NUM;}
[\t]       ;
\n         return 0;
.          return yytext[0];
%%

int yywrap()
{
}
```

yacc

```
%{
#include<stdio.h>
%}


%token NUM
%left '+'
%right '-'


%%
expr:e {printf("Valid Expression\n"); printf ("Result: %d\n",$$); return 0;}
e:e'+'e {$$=$1+$3;}
| e'-'e   {$$=$1-$3;}
| NUM            {$$=$1;}
;
%%


int main()
```

```
{
printf("\n Enter an arithmetic expression\n");
        yyparse();
        return 0;
}


int yyerror()
{
        printf("\nInvalid expression\n");
        return 0;
}
```

OUTPUT



2.Design a suitable grammar for evaluation of arithmetic expression having + , – , * , / , %, ^ operators.

^ having highest priority and right associative
% having second highest priority and left associative
* , / have third highest priority and left associative
+ , - having least priority and left associative

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

```
[\t]    ;
\n      return 0;
.       return yytext[0];
%%
int yywrap()
{
}
```

```
%{
#include<stdio.h>
%}

%token NUM
%left '+' '-'
%left '*' '/' '%'
%right '^'

%%

expr: e { printf("Valid expression\n"); printf("Result: %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 '^' e     {
                int result = 1;
                for (int i = 0; i < $3; i++) {
                    result *= $1;
                }
```

```
            $$ = result;
        }
 | NUM       {$$ = $1;}
 ;

%%

int main()
{
   printf("\nEnter an arithmetic expression:\n");
   yyparse();
   return 0;
}

int yyerror()
{
   printf("\nInvalid expression\n");
   return 0;
}
```

OUTPUT