1. Write a lex program to count the number of lines and characters in the input file

```
%{
   #include<stdio.h>
   int lc = 0,cc= 0;
%}

%%
[a-zA-Z0-9] {cc++;}
\n {lc++;}
%%
int yywrap(){};
int main(){
   printf("Enter the pragraph with enters as \n ");
   yylex();

   printf("The Number of lines : %d\nThe Number of chars : %d",lc,cc);

   return 0;

}
```

2. write a Lex program that implement the Caesar cipher it replaces every letter with the 13 letters after in the alphabet order wrapping around Z example a is replaced by b by e and so on Z by c

```
%{
   #include<stdio.h>
   #include<string.h>
   char a[50];
```

```
   int ptr = 0;
%}
%%
[a-zA-Z] {a[ptr++] = yytext[0]+3;}
%%
int yywrap(){}
int main(){

   printf("ENter the alphabetical line to convert \n");
   yylex();

   printf("%s\n",a);
   return 0;

}
```

3. write a Lex program that finds the longest word defined as a contagious string of upper and lowercase

```
%{
   #include<string.h>
   #include<stdio.h>
   char ar[30];
   int ptr=0;
%}
%%
[a-zA-Z]* { if(yyleng > ptr){
            strcpy(ar,yytext);
            ptr = yyleng;
          }}
```

```
%%
int yywrap(){}

int main(){
   printf("Enter the pragraph : \n");
   yylex();

   printf("The longest word is : %s\n",ar);

   return 0;

}
```

4. write a Lex program that distinguishes keywords integers floats identifiers operators and comments in any simple programming language.

```
%{
   #include<stdio.h>
   #include<string.h>
   int kw = 0 , integ = 0 ,flot = 0 , iden = 0 , oper = 0 ,comm = 0;
%}
%%
"//" {comm++;}
[if|else|while|for|do|int|float|double] { kw++;}
^[-+]?[0-9]*[.][0-9]+$ flot++;
[0-9]+ {integ++;}
[+|-|*|/] {oper++;}
[a-zA-Z][0-9a-zA-Z]+ {iden++;}
. {}
```

```
%%


int yywrap(){}


int main(){
   printf("Enter the code fragment here at terminal : ");
   yylex();


   printf("No of Integers :%d\n Keywords :%d\n Float :%d \n Identifier : %d\n Comments : %d\n
Operators : %d\n",integ,kw,flot,iden,comm,oper);
   return 0;
}
```

5. write a Lex program to count the number of identifiers in a c file

```
%{
   #include<stdio.h>
   int varcnt=0;
%}
%%
[int|float|double|include|stdio.h|printf|main {}
[a-z,A-Z,_][a-z,A-Z,0-9,_]* {varcnt++;}]
. {}
%%
int yywrap(){}


int main(){
   yyin= fopen("program.c","r");
```

```
    yylex();

    fclose(yyin);

    printf("The Number of Variables : %d\n",varcnt);

    return 0;
}
```

6. write a Lex program to count the number of words characters blank spaces and lines in a c file

```
%{
    #include<stdio.h>
    int wcnt=0 , ccnt = 0 , bspace=0 ,lin =0;
%}
%%
[\n] {lin++;ccnt += yyleng;}
[ \t] {bspace++;ccnt += yyleng;}
[^\t\n ]+ {wcnt++; ccnt += yyleng;}

%%
int yywrap(){}

int main(){
    yyin= fopen("program.c","r");


    yylex();
```

```
    fclose(yyin);
    printf("The Number of Words : %d\nNumer of Chars :%d\nNumber of Blank Spaces :%d\nNUmber of Lines :%d ",wcnt,ccnt,bspace,lin);
    return 0;
}
```

7. write a leg specification program that generates a C program which takes a string a b c d and prints the following input

abcd

abc

ab

a

```
%{
#include <stdio.h>
char i , j;
%}
%%
[a-z]* {
          for( i = 'd' ; i>='a' ;i--){
             for( j = 'a' ; j<=i ;j++){
                printf("%c",j);
             }
             printf("\n");
          }
       }
%%
int yywrap(){}
```

```
int main() {
   yylex();

   return 0;
}
```

8. a program in lex to recognise a valid arithmetic expression

```
%{
   #include<stdio.h>
   #include<string.h>
   #include<stdbool.h>

   int top = -1 ,i =0 ,j =0 ,var =0 , oper =0;
   bool valid = true;
   char stk[100] , vari[10][10] , opera[10][10] ;
%}

%%
"("|"{"|"[" {top++ ; stk[top] = yytext[0];}
")"|"}"|"]" {
         if(yytext[0] == ')'){
            if(stk[top] != '(' || var-oper != 1 ) {
               valid=false;
            }
            top--;
            var=1;
            oper=0;
         }
```

```
        else if(yytext[0] == '}'){
           if(stk[top] != '{' || var-oper != 1){
           valid = false;}

           top--;
           var=1;
           oper=0;
        }

        else if(yytext[0] == ']'){
           if(stk[top] != '[' || var-oper != 1){
           valid = false;}

           top--;
           var=1;
           oper=0;
        }

     }
[0-9]+|[a-zA-Z][a-zA-Z0-9_]* { var++;
                        strcpy(vari[i],yytext);
                        i++;
                     }
"+"|"-"|"*"|"/" {oper++;
        strcpy(opera[j],yytext);
        j++;
        }
%%
```

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


int main(){
   printf("Enter the Arithmetic Expression");
   yylex();


   if(valid == true && top == -1 && var-oper == 1){
      printf("EXpression Valid!\n");
   }


   else{
      printf("Expression Invalid\n");
   }


   return 0;
}
```

9. write a Yacc program to find the validity of a given expression (for operators + - * and /)

YACC:

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


%token ID NUMBER
%left '+' '-'
%left '*' '/'


%%
```

```
stmt:exp;
exp:exp'+'exp
|exp'-'exp
|exp'*'exp
|exp'/'exp
|NUMBER
|ID
;
%%
int main(){
printf("Enter the expression: ");
yyparse();
printf("Valid Expression\n");
exit(0);}
int yyerror(){
printf("Invalid Expression\n");
exit(0);}
```

LEX:

```
%{
#include "y.tab.h"
%}
%%
[a-zA-Z]+ {return ID;}
[0-9]+ {return NUMBER;}
[ \t] {;}
[\n] {return 0;}
. {return yytext[0];}
%%
```

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

10. a program in Yacc which recognises a valid variable which starts with letter followed by a digit the letter in lower case only.

YACC:

```
%{
#include <stdio.h>
%}
%token VARIABLE NUMBER
%%
stmt:exp;
A:VARIABLE;
B:NUMBER;
exp:A B;

%%
int yyerror(){
printf("Invalid Variable\n");
exit(0);
}
int main(){
printf("Enter the variable: ");
yyparse();
printf("Valid Variable\n    ");
exit(0);}
```

LEX:

```
%{
```

```
#include <stdio.h>
#include "y.tab.h"
%}
%%
[a-z] {return VARIABLE;}
[0-9] {return NUMBER;}
\n {return 0;}
. {return yytext[0];}
%%
int yywrap(){return 1;}
```

11. a program in Yacc to evaluate an expression simple calculator program for addition and subtraction, multiplication, division

YACC:

```
%{
/* Definition section */
#include<stdio.h>
int flag=0;
%}

%token NUMBER

%left '+' '-'

%left '*' '/' '%'

%left '(' ')'

/* Rule Section */
```

```
%%

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;}

;

%%

//driver code
void main()
```

```
{
printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Division, 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;
}
```

LEX:

```
%{
#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;}
```

12. program in Yacc to recognise the strings ab, aabb, aaabbb, … of the language a^nb^n, n>=1.

YACC:

```
%{
%}
%token A B NL
%%
stmt: S NL {printf("Valid word for the language\n");
      exit(0);};
S:A S B| A B;
%%
int yyerror(){printf("Invalid Word for the language");
exit(0);}
int main(){
printf("Enter the word: ");
yyparse();}
```

LEX:

```
%{
#include <stdio.h>
#include "y.tab.h"
%}
%%
[a] {return A;}
[b] {return B;}
\n {return NL;}
. {return yytext[0];}
%%
```

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

13. program in Yacc to recognise the language (a^nb, n>=10). (output to say input is valid or not).

YACC:

```
%{
%}
%token A B NL
%%
stmt : A A A A A A A A A A s B NL
 {
 printf("Valid"); exit(0);
 }
 ;
s : s A
 |
 ;
%%
int yyerror(char *msg)
{
 printf("Invalid String\n");
 exit(0);
}
int main ()
{
 printf("Enter the String\n");
 yyparse();
}
```

LEX:

```
%{
#include "y.tab.h"
%}
%%
[a] {return A;}
[b] {return B;}
\n {return NL;}
. {return yytext[0];}
%%
int yywrap(){return 1;}
```