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 implements the Caesar cipher: it replaces every letter with the one three letters after in alphabetical order, wrapping around at Z. e.g. a is replaced by d, 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 contiguous string of upper and lower-case letters) in the input.

```
%{
    #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 Lex specification program that generates a C program which takes a string "abcd" and prints the following output. 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 recognize 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 /)**

```
%{
#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);}
```

10. A Program in YACC which recognizes a valid variable which starts with letter followed by a digit. The letter should be in lowercase only.

```
/* yacc.y */
%{
  #include <stdio.h>
  #include <stdlib.h>
  extern int yylex();
  void yyerror(char *);
%}

%token A B

%%
S : E '\n' { printf("VALID STRING\n"); exit(0); }
  ;
E : A E B
  | A B
  ;
%%
int main()
{
   yyparse();
   return 0;
}

void yyerror(char *msg) {
  fprintf(stderr, "INVALID STRING\n");
  exit(1);
}

/* lex.l */
%{
  #include <stdio.h>
  #include <stdlib.h>

  #if __has_include("y.tab.h")
    #include "y.tab.h"
  #endif
%}

%option noyywrap
%%
```

```
[a] { return A; }
[b] { return B; }
[ |\n|\t] { return yytext[0]; }
. { return yytext[0]; }
%%
```

11. A Program in YACC to evaluate an expression (simple calculator program for addition and subtraction, multiplication, division).

```
/* yacc.y */
%{
  #include <stdio.h>
  #include <stdlib.h>
  extern int yylex();
  void yyerror(char *);
%}

%union { float f; }
%token <f> NUM
%type <f> E T F
%%
S : E { printf("%f\n", $1); }
  ;
E : E '+' T { $$ = $1 + $3; }
  | E '-' T { $$ = $1 - $3; }
  | T
  ;
T : T '*' F { $$ = $1 * $3; }
  | T '/' F { $$ = $1 / $3; }
  | F
  ;
F : '(' E ')' { $$ = $2; }
  | '-' F { $$  = -$2; }
  | NUM
  ;
%%
int main()
{
   yyparse();
   return 0;
}

void yyerror(char *msg) {
  fprintf(stderr, "%s\n", msg);
  exit(1);
}

/* lex.l */
%{
  #include <stdio.h>
  #include <stdlib.h>

  #if __has_include("y.tab.h")
    #include "y.tab.h"
  #endif
%}

%option noyywrap
%%
[0-9]+(\.[0-9]+)? { yylval.f = atof(yytext); return NUM; }
[\-+()*/] { return yytext[0]; }
[ \t\n]+ { ; }
%%
```

12. Program in YACC to recognize the strings "ab", "aabb", "aaabbb",… of the language (a^n b^n, n>=1).

```
/* yacc.y */
%{
  #include <stdio.h>
  #include <stdlib.h>
  extern int yylex();
  void yyerror(char *);
%}

%token A B

%%
S : E '\n' { printf("VALID STRING\n"); exit(0); }
  ;
E : A E B
  | A B
  ;
%%
int main()
{
   yyparse();
   return 0;
}

void yyerror(char *msg) {
  fprintf(stderr, "INVALID STRING\n");
  exit(1);
}

/* lex.l */
%{
  #include <stdio.h>
  #include <stdlib.h>

  #if __has_include("y.tab.h")
    #include "y.tab.h"
  #endif
%}

%option noyywrap
%%
[a] { return A; }
[b] { return B; }
[ |\n|\t] { return yytext[0]; }
```

```
. { return yytext[0]; }
%%
```

13. Program in YACC to recognize the language (*anb* , n>=10). (Output to say input is valid or not

```
/* yacc.y */
%{
  #include <stdio.h>
  #include <stdlib.h>
  extern int yylex();
  void yyerror(char *);
%}

%token A B

%%
S : E '\n' { printf("VALID STRING\n"); exit(0); }
  ;
E : A E B
  | A B
  ;
%%
int main()
{
   yyparse();
   return 0;
}

void yyerror(char *msg) {
  fprintf(stderr, "INVALID STRING\n");
  exit(1);
}

/* lex.l */
%{
  #include <stdio.h>
  #include <stdlib.h>

  #if __has_include("y.tab.h")
    #include "y.tab.h"
  #endif
%}

%option noyywrap
%%
[a] { return A; }
[b] { return B; }
[ |\n|\t] { return yytext[0]; }
. { return yytext[0]; }
%%
```