Q1. Design a LEX Code to count number of lines, spaces, tab-meta character, and rest of characters in each input pattern.

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

int lines = 0;
int spaces = 0;
int tabs = 0;
int others = 0;
%}

%%
\n          { lines++; }
[ \r]       { spaces++; }
\t          { tabs++; }
.           { others++; }
%%

int main() {
    printf("Enter input (Ctrl+D to end on Linux/macOS, Ctrl+Z on Windows):\n");
    yylex();
    printf("\n--- Statistics ---\n");
    printf("Lines: %d\n", lines);
    printf("Spaces: %d\n", spaces);
    printf("Tabs: %d\n", tabs);
    printf("Other characters: %d\n", others);
    return 0;
}

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

Q2. Design a LEX Code to identify and print valid Identifier of C/C++ in given Input pattern.

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

%%
auto|double|int|struct|break|else|long|switch|case|
enum|register|typedef|char|extern|return|union|continue
|for|signed|void|do|if|static|while|default|goto|sizeof
|volatile|const|float|short|unsigned;

([a-zA-Z][0-9])+|[a-zA-Z]* {printf("Identifier\n");}

^[0-9]+ {printf("Not a Identifier\n");}

.|\n;
%%
int yywrap()
{
return 1;
}
int main(void)
{
yylex();
return 0;
}
```

Q3. Design a LEX Code to identify and print integer and float value in given input pattern.

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

%%
[0-9]+"."[0-9]+ {ECHO; printf("\nDecimal Number\n");}
[0-9]+ {ECHO; printf("\nInteger Number\n");}

%%
/*call the yywrap function*/
int yywrap()
{return 1;}

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

Q4. Design a LEX Code to for tokenizing (Identify and print OPERATORS, SEPERATORS, KEYWORDS, IDENTIFIERS) in the C-fragment.

```
%{
int n = 0;
%}

%%

"while"|"if"|"else" {n++; printf("\t keyword : %s", yytext);}
"int"|"float" {n++; printf("\t keyword : %s", yytext);}

[a-zA-Z_][a-zA-Z0-9_]* {n++; printf("\t identifier : %s", yytext);}

"<="|"=="|"="|"++"|"-"|"*"|"+" {n++; printf("\t operator : %s", yytext);}

[(){}|,;] {n++; printf("\t separator : %s", yytext);}

[0-9]*"."[0-9]+ {n++; printf("\t float : %s", yytext);}

[0-9]+ {n++; printf("\t integer : %s", yytext);}

. ;
%%
int main() {
   yylex();
   printf("\n total no. of token = %d\n", n);
}
```

Q5. Design a LEX Code to count and print the number of total characters, words, white spaces in given "Input.txt" file.

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

int char_count = 0;
int word_count = 0;
int space_count = 0;
%}

%%
[ \t\n]+      {
               space_count += yyleng;
               word_count++;
               char_count += yyleng;
              }

[A-Za-z0-9]+  {
               word_count++;
               char_count += yyleng;
              }

.             { char_count++; }
%%

int main(int argc, char **argv) {
    FILE *file = fopen("Input.txt", "r");
    if (!file) {
        printf("Failed to open Input.txt\n");
        return 1;
    }
```

```
    yyin = file;
    yylex();

    if (word_count > 0) word_count /= 2;

    printf("Total Characters: %d\n", char_count);
    printf("Total Words: %d\n", word_count);
    printf("Total White Spaces: %d\n", space_count);

    fclose(file);
    return 0;
}
```

Q6. Design a LEX Code to replace white spaces of "Input.txt" file by a single blank character into "Output.txt" file.

```
%{
#include <stdio.h>
FILE *out;
%}

%%
[ \t\n]+      { fputc(' ', out); }
.             { fputc(yytext[0], out); }
%%

int main() {
   FILE *in = fopen("Input.txt", "r");
   out = fopen("Output.txt", "w");
   if (!in || !out) {
      printf("Error: Cannot open Input.txt or Output.txt\n");
      return 1;
   }
   yyin = in;
   yylex();
   fclose(in);
   fclose(out);
   printf("Whitespace replaced successfully in Output.txt\n");
   return 0;
}
```

Q7. Design a LEX Code to remove the comments from any C-Program given at run-time and store into "out.c" file.

```
%{
#include <stdio.h>
FILE *out;
%}
%x COMMENT

%%
"//".*                  { /* Single-line comment: skip it */ }
"/*"                    { BEGIN(COMMENT); }
<COMMENT>[^*]*          { /* Skip comment content */ }
<COMMENT>"*"+[^*/]*     { /* Still inside comment */ }
<COMMENT>"*/"           { BEGIN(INITIAL); }
<COMMENT><<EOF>>        { BEGIN(INITIAL); }

[^\n]+                  { fputs(yytext, out); }
\n                      { fputc('\n', out); }
.                       { fputc(yytext[0], out); }
%%

int main(int argc, char **argv) {
   if (argc < 2) {
      printf("Usage: %s <C-source-file>\n", argv[0]);
      return 1;
   }

   FILE *in = fopen(argv[1], "r");
   if (!in) {
      printf("Cannot open input file: %s\n", argv[1]);
      return 1;
   }
```

```
    out = fopen("out.c", "w");
    if (!out) {
        printf("Cannot open output file: out.c\n");
        fclose(in);
        return 1;
    }

    yyin = in;
    yylex();

    fclose(in);
    fclose(out);
    printf("Comments removed successfully. Output saved to 'out.c'\n");

    return 0;
}
```

Q8. Design a LEX Code to extract all html tags in the given HTML file at run time and store into Text file given at run time.

```
%{
#include <stdio.h>
FILE *out;
%}
%%
<[^>]+>     { fprintf(out, "%s\n", yytext); }
[^<]+       { /* Skip text between tags */ }
.           { /* Ignore anything else */ }
%%
int main(int argc, char **argv) {
   if (argc < 3) {
      printf("Usage: %s <input.html> <output.txt>\n", argv[0]);
      return 1;    }
   FILE *in = fopen(argv[1], "r");
   if (!in) {
      printf("Error: Cannot open input file %s\n", argv[1]);
      return 1;
   }
   out = fopen(argv[2], "w");
   if (!out) {
      printf("Error: Cannot open output file %s\n", argv[2]);
      fclose(in);
      return 1;
   }
   yyin = in;
   yylex();
   fclose(in);
   fclose(out);
   printf("HTML tags extracted to %s\n", argv[2]);
   return 0;}
```

Q9. Design a DFA in LEX Code which accepts string containing even numbers of "a" and even number of "b" over input alphabet {a,b}.

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

enum State { S0, S1, S2, S3 };
enum State current_state = S0;

void transition(char c) {
    switch (current_state) {
        case S0:
            if (c == 'a') current_state = S1;
            else if (c == 'b') current_state = S2;
            break;
        case S1:
            if (c == 'a') current_state = S0;
            else if (c == 'b') current_state = S3;
            break;
        case S2:
            if (c == 'a') current_state = S3;
            else if (c == 'b') current_state = S0;
            break;
        case S3:
            if (c == 'a') current_state = S2;
            else if (c == 'b') current_state = S1;
            break;
    }
}
%}

%%
[a|b]     { transition(yytext[0]); }
```

```
\n      {
          if (current_state == S0)
              printf("Accepted: Even number of a's and b's\n");
          else
              printf("Rejected: Not even number of a's and/or b's\n");
          current_state = S0;
      }
[^ab\n]   { printf("Invalid character: %s\n", yytext); }
%%

int main() {
   printf("Enter strings over {a, b} (Ctrl+D to end):\n");
   yylex();
   return 0;
}
```

Q10. Design a DFA in LEX Code which accepts string containing third last element "a" over input alphabet {a,b}.

```
%{
#include <stdio.h>
#include <string.h>
char last3[4] = "";
int len = 0;
%}
%%
[a|b] {
   if (len < 3) {
      last3[len++] = yytext[0];
   } else {
      last3[0] = last3[1];
      last3[1] = last3[2];
      last3[2] = yytext[0];
   }
}
\n {
   if (len >= 3 && last3[0] == 'a')
      printf("Accepted: Third last character is 'a'\n");
   else
      printf("Rejected: Third last character is not 'a' or input too short\n");
   len = 0;
   last3[0] = last3[1] = last3[2] = '\0';
}
[^ab\n] {          printf("Invalid character: %s\n", yytext);          }
%%
int main() {
   printf("Enter strings of a and b (press Enter to check, Ctrl+D to exit):\n");
   yylex();
   return 0;}
```

Q11. Design a DFA in LEX Code to identify and print Integer & Float Constants and Identifier.

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

%%

[0-9]+\.[0-9]+          { printf("Float Constant: %s\n", yytext); }
[0-9]+                  { printf("Integer Constant: %s\n", yytext); }
[_a-zA-Z][_a-zA-Z0-9]*   { printf("Identifier: %s\n", yytext); }
[ \t\n]+                { /* Ignore whitespace */ }
.                       { printf("Unknown: %s\n", yytext); }

%%

int main() {
   printf("Enter input containing integers, floats, and identifiers (Ctrl+D to end):\n");
   yylex();
   return 0;
}
```

Q12. Design YACC/LEX code to recognize valid arithmetic expression with operators +, -, * and /.

LEX Code :

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

%%
[0-9]+          { yylval = atoi(yytext); return NUMBER; }
[a-zA-Z_][a-zA-Z0-9_]*  { return ID; }

[+]             { return PLUS; }
[-]             { return MINUS; }
[*]             { return MUL; }
[/]             { return DIV; }
[(]             { return LPAREN; }
[)]             { return RPAREN; }

[ \t\n]+        ;
.               { return yytext[0]; }
%%
```

YACC Code :

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

void yyerror(const char *s);
int yylex();
%}

%token NUMBER ID
```

```
%token PLUS MINUS MUL DIV
%token LPAREN RPAREN

%left PLUS MINUS
%left MUL DIV

%%

stmt:
    expr        { printf("Valid expression\n"); }
;

expr:
      expr PLUS expr
    | expr MINUS expr
    | expr MUL expr
    | expr DIV expr
    | LPAREN expr RPAREN
    | NUMBER
    | ID
;

%%
void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

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

Q13. Design YACC/LEX code to evaluate arithmetic expression involving operators +, -, * and / without operator precedence grammar & with operator precedence grammar.

LEX Code :

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

%%

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

%%
```

YACC Code :

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

int yylex();
void yyerror(const char *s);
int use_precedence = 1;
%}

%token NUMBER
%left '+' '-'
%left '*' '/'
%%
stmt: expr { printf("Result = %d\n", $1); return 0; }
;
```

```
expr:
    // WITH precedence rules
      expr '+' expr    { $$ = use_precedence ? $1 + $3 : fallback_add($1, $3); }
    | expr '-' expr    { $$ = use_precedence ? $1 - $3 : fallback_sub($1, $3); }
    | expr '*' expr    { $$ = use_precedence ? $1 * $3 : fallback_mul($1, $3); }
    | expr '/' expr    { $$ = use_precedence ? $1 / $3 : fallback_div($1, $3); }
    | '(' expr ')'     { $$ = $2; }
    | NUMBER
;
%%
int fallback_add(int a, int b) {
    return a + b;
}
int fallback_sub(int a, int b) {
    return a - b;
}
int fallback_mul(int a, int b) {
    return a * b;
}
int fallback_div(int a, int b) {
    return a / b;
}
void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}
int main() {
    printf("Use operator precedence? (1 = yes, 0 = no): ");
    scanf("%d", &use_precedence);
    printf("Enter expression:\n");
    yyparse();
    return 0;
}
```