1. Design a LEX code to count the number of lines space tab meta character and rest of the characters in each input pattern in C/C++.

```
%{
   #include <stdio.h>
   int lines = 0, spaces = 0, tabs = 0, meta = 0, others = 0;
%}

%%
\n               { lines++; }

" "              { spaces++; }

\t               { tabs++; }

[!@#$%^&*()_+=<>?/\\|] { meta++; }

.                { others++; }
%%

int main() {
   yylex();
   printf("Lines: %d\nSpaces: %d\nTabs: %d\nMeta Characters: %d\nOther
Characters: %d \n", lines, spaces, tabs, meta, others);
   return 0;
}

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

2. Design a LEX code to Identify and print valid identifiers of C/C++ in given input pattern.

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

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

.|\n                      { /* Ignore other characters */ }
%%

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

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

3. Design a LEX code to Identify and print integer and float values in given input pattern.

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

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

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

.|\n           { /* Ignore other characters */ }
%%

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

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

4. Design a LEX code for Tokenizing identify and print operators, seprators. keywords, identifiers) in the C fragment.

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

%%
"+" | "-" | "*" | "/"             { printf("Operator: %s\n", yytext); }

"(" | ")" | "{" | "}" | ";" | "," { printf("Separator: %s\n", yytext); }

"int" | "float" | "if" | "else" | "for" | "while" | "return" { printf("Keyword: %s\n",
yytext); }

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

.                       { printf("Unknown Token: %s\n", yytext); }
%%

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

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

5. Design a LEX code to Count and print the number of total characters, words, white spaces in given input.txt file.

```
%{
   #include <stdio.h>
   extern FILE *yyin;
   int chars = 0, words = 0, spaces = 0;
%}

%%
[a-zA-Z0-9]+   { words++; chars += yyleng; }  // Count word characters

[ \t\n]        { spaces++; chars += yyleng; } // Count spaces and add to character
count

.              { chars++; }          // Count all other single characters
%%

int main() {
   yyin = fopen("input.txt", "r");
   if (!yyin) {
      perror("Error opening file");
      return 1;
   }

   yylex();
   printf("Total Characters (including spaces): %d\n Words: %d\n White Spaces:
%d\n", chars, words, spaces);

   fclose(yyin);
   return 0;
}

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

6. 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 *fp;
extern FILE *yyin;
%}

%%

[ \t\n]+ { fprintf(fp, " "); }
.    { fprintf(fp, "%s", yytext); }

%%

int main() {
  yyin = fopen("input.txt", "r");
  if (!yyin) {
    perror("Error opening input.txt");
    return 1;
  }

  fp = fopen("output.txt", "w");
  if (!fp) {
    perror("Error opening output.txt");
    return 1;
  }

  yylex();
```

```
    fclose(yyin);
    fclose(fp);
    return 0;
}

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

7. Design a LEX code to Remove the comments from any C-program given at runtime and store into out.c file.

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

FILE *fp;
extern FILE *yyin;
%}

%%

"/*"[^*]*"*/"
"//".*      { fprintf(fp, "\n"); }
[ \t]+      { fprintf(fp, "%s", yytext); }
.|\n        { fprintf(fp, "%s", yytext); }

%%

int main() {
  yyin = fopen("input.c", "r");
  if (!yyin) {
    perror("Error opening input.c");
    return 1;
  }

  fp = fopen("out.c", "w");
  if (!fp) {
    perror("Error opening out.c");
    return 1;
  }
```

```
    yylex();

    fclose(yyin);
    fclose(fp);
    return 0;
}

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

8. Design a LEX code to Extract all html tags in the given html file at runtime and store it into text file given at runtime.

```
%{
#include <stdio.h>
FILE *fp;
extern FILE *yyin;

%}

%%

"<"[^>]+">" { fprintf(fp, "%s\n", yytext); }

%%

int main() {
  yyin = fopen("tags.html", "r");
  if (!yyin) {
    perror("Error opening input.c");
    return 1;
  }

  fp = fopen("tags.txt", "w");
  yylex();
  fclose(fp);
  return 0;
}

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

9. Design a DFA in LEX code Which accepts the string containing even number of 'a' and even number of 'b' over input alphabet {a,b}.

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

int count_a = 0, count_b = 0;
%}

%%

a  { count_a++; }
b  { count_b++; }
[^ab\n]  { printf("Rejected: %s\n", yytext); }
\n {
   if (count_a % 2 == 0 CC count_b % 2 == 0)
      printf("Accepted\n");
   else
      printf("Rejected\n");
   count_a = 0; count_b = 0;
}

%%

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

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

10. Design a DFA in LEX code Which accepts string containing third last element 'a' over input alphabet{a,b}.

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

%%

[a-b]*a[a-b][a-b]$ { printf("Accepted: %s\n", yytext); }
. { printf("Rejected: %s\n", yytext); }

%%

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

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

11. Design a DFA in LEX code To identify and print integers samp; float constants and Identifiers.

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

%%

[0-9]+\.[0-9]+([eE][-+]?[0-9]+)? { printf("Float: %s\n", yytext); }
[0-9]+                      { printf("Integer: %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() {
   yylex();
   return 0;
}

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

.#q.. 2

12. Design a YACC/LEX code to recognise valid arithmetic expression with operator +, -, * and /.

YACC Code (to generate y.tab.c s y.tab.h):

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

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

%%

expr: expr '+' expr  { printf("Recognized: +\n"); }
   | expr '-' expr   { printf("Recognized: -\n"); }
   | expr '*' expr  { printf("Recognized: *\n"); }
   | expr '/' expr   { printf("Recognized: /\n"); }
   | NUMBER          { printf("Recognized: Number\n"); }
   ;

%%

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

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

LEX Code:

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

%%
[0-9]+     { yylval = atoi(yytext); return NUMBER; }
[+\-*/]    { return yytext[0]; }
[ \t\n]    { /* Ignore whitespace */ }
.        { printf("Invalid character: %s\n", yytext); }
%%

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

13. Design a YACC/LEX code to Evaluate arithmetic expression involving operators+, -, * and / Without operator precedence grammar samp; with operator precedence grammar.

```
/* LEX Code */
%{
#include "y.tab.h"
%}

%%
[0-9]+     { yylval = atoi(yytext); return NUMBER; }
[+\-*/]    { return yytext[0]; }
"("        { return '('; }
")"        { return ')'; }
[ \t\n]    { /* Ignore whitespace */ }
.        { printf("Invalid character: %s\n", yytext); }
%%

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

YACC Code Without Operator Precedence:

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

%token NUMBER

%%

expr: expr '+' expr   { printf("Result: %d\n", $1 + $3); }
    | expr '-' expr   { printf("Result: %d\n", $1 - $3); }
```

```
   | expr '*' expr  { printf("Result: %d\n", $1 * $3); }
   | expr '/' expr  { if ($3 == 0) { printf("Error: Division by zero\n"); } else { printf("Result: %d\n", $1 / $3); } }
   | '(' expr ')'  { $$ = $2; }
   | NUMBER        { $$ = $1; }
   ;

%%

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

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

YACC Code With Operator Precedence:

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

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

%%
```

```
expr: expr '+' expr  { $$ = $1 + $3; }
   | expr '-' expr  { $$ = $1 - $3; }
   | expr '*' expr   { $$ = $1 * $3; }
   | expr '/' expr   { if ($3 == 0) { printf("Error: Division by zero\n"); } else { $$ = $1 / $3; } }
   | '(' expr ')'   { $$ = $2; }
   | NUMBER         { $$ = $1; }
   ;

%%

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

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