**Q1. Design a LEX Code to count the number of lines, space, tab-meta character, and rest of characters in each Input pattern.**

## Solution:

```
%{
#include<stdio.h>
int line = 0 , space = 0 , tab = 0 , meta = 0 , other = 0;
%}

%%
"\n"                    {line++;}
" "                     {space++;}
"\t"                    {tab++;}
[\^\$\%\*\+\-\/\&\(\)\{\}\[\]\;\"]  {meta++;}
.                       {other++;}
%%

int main(){
   yylex();
   printf("Line:%d\n",line);
   printf("Space:%d\n",space);
   printf("Tab:%d\n",tab);
   printf("Meta:%d\n",meta);
   printf("Other:%d\n",other);

   return 0;
}

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

## Output:

```
int main(){
      cout << "hello";
}
Line:3
Space:3
Tab:1
Meta:7
Other:18
```

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

### Solution:

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

%%
int|float|return|void|if|else       {}
[a-zA-Z_]+[a-zA-Z_0-9]*           {printf("Identifier:%s\n",yytext);}
[0-9]+[a-zA-Z_]*                    {}
.                          {}
%%

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

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

### Output:

```
var_43
Identifier:var_43

num
Identifier:num

temp
Identifier:temp
```

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

**Solution:**

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

%%
[-]?[0-9]+\.[0-9]+          {printf("Float:%s\n",yytext);}
[-]?[0-9]+              {printf("Int:%s\n",yytext);}
[0-9]*[a-zA-z]*[0-9]*       {printf("Nothing");}
.                   {}
%%

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

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

**Output:**

```
1234
Int:1234

123.54
Float:123.54

45_frr
Nothing
```

## Q4. Design a LEX Code for Tokenizing (Identify and print OPERATORS, SEPERATORS, KEYWORDS, IDENTIFERS) in the C-fragment:

## Solution:

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

%}

KEYWORDS "int"|"float"|"return"|"if"|"else"|"for"|"while"
OPERATORS [+\-*/^<>&|!%]
SEPERATORS [;,\(\)\{\}\"]

%%
{KEYWORDS} {printf("Keyword: %s\n", yytext);}
{OPERATORS} {printf("Operators: %s\n", yytext);}
{SEPERATORS} {printf("Seperators: %s\n", yytext);}
[a-zA-Z_]+[a-zA-Z0-9_]* {printf("Id: %s\n", yytext);}
. {}
%%

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

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

## Output:

```
int
Keyword: int

+
Operators: +

;
Seperators: ;

var_23
Id: var_23
```

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

## Solution:

```
%{
   #include<stdio.h>
   int word =0, char_len=0, space = 0;
%}

%%
[a-zA-Z0-9]+ {
      word++;
      char_len += yyleng;}
[ \n\t] {space++;}

%%

int main(){
   FILE *file = fopen("z-5-6.txt", "r");
   if(!file) {printf("no file"); return 1;}

   yyin = file;
   yylex();
   printf("word:%d\n",word);
   printf("char:%d\n",char_len);
   printf("space:%d\n",space);
   fclose(file);
   return 0;
}

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

## Output:

```
word:7
char:46
space:48
```

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

## Solution:

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

%%
[ \t\n]+    { fprintf(out, " "); }   // Replace multiple spaces, tabs, or newlines with a single space
.           { fprintf(out, "%s", yytext); }  // Write other characters as they are
%%

int main() {
   // Open input and output files
   yyin = fopen("z-5-6.txt", "r");
   if (!yyin) {
      perror("Error opening input.txt");
      return 1;
   }

   out = fopen("output.txt", "w");
   if (!out) {
      perror("Error creating output.txt");
      fclose(yyin);
      return 1;
   }
   yylex();

   fclose(yyin);
   fclose(out);

   printf("Whitespace replaced successfully. Check output.txt\n");
   return 0;
}

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

## Output:

```
input : hello            my name is          justin
jhdjhejdjjdejdjehd             ehidhehdhd

output :
hello my name is justin jhdjhejdjjdejdjehd ehidhehdhd
```

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

## Solution:

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

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

int main() {
   char input_file[100];

   printf("Enter the C program filename (e.g., program.c): ");
   scanf("%s", input_file);

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

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

   fclose(yyin);
   fclose(out);

   printf("Comments removed successfully. Check out.c\n");
   return 0;
}

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

## Output:

```
input :
#include <stdio.h> // header file
```

```
int main()
{                  // program start
   printf("Hello"); // print stmt
} // exit
```

output :

```
#include <stdio.h>
int main()
{
   printf("Hello");
}
```

**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.**

**Solution:**

```
%{
#include <stdio.h>
#include <stdlib.h>
FILE *out;
%}
%%
\ !--([^-\n]|(-[^-])|(\n))*-->   {}
\ <[a-zA-Z/!][^>]*>              { fprintf(out, "%s\n", yytext); }
.                         {}
\n                        { fprintf(out, "\n"); }
%%
int main() {
   char input_file[100], output_file[100];

   printf("Enter the HTML file name (e.g., index.html): ");
   scanf("%s", input_file);

   printf("Enter the output file name (e.g., tags.txt): ");
   scanf("%s", output_file);

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

   out = fopen(output_file, "w");
   if (!out) {
      perror("Error opening output file");
      fclose(yyin);
      return 1;
   }
   yylex();
   fclose(yyin);
   fclose(out);
   printf("HTML tags extracted successfully. Check %s\n", output_file);
   return 0;
}
int yywrap(){
   return 1;
}
```

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

**Solution:**

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

int state = 0;
int invalid_input = 0;
%}

%%
a   {
        if (state == 0) state = 1;
    else if (state == 1) state = 0;
    else if (state == 2) state = 3;
    else if (state == 3) state = 2;
}

b   {
        if (state == 0) state = 2;
    else if (state == 1) state = 3;
    else if (state == 2) state = 0;
    else if (state == 3) state = 1;
}

[c-zC-Z0-9]+   {
    invalid_input = 1;
}

\n   {
    if (invalid_input == 1) {
        printf("Rejected (Invalid characters found)\n");
    } else if (state == 0) {
        printf("Accepted\n");
    } else {
        printf("Rejected\n");
    }
    state = 0;
    invalid_input = 0;
}

.   { /* Ignore spaces or special characters */ }
%%

int main() {
    char input[100];

    while (1) {
        printf("Enter a string of a's and b's (end with Enter, type 'exit' to quit): ");
        fgets(input, sizeof(input), stdin);
        if (input[0] == 'e' && input[1] == 'x' && input[2] == 'i' && input[3] == 't') {
            break;
        }
        yylex();
    }

    return 0;
}

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

## Output:

```
Enter a string of a's and b's (end with Enter, type 'exit' to quit):
aabb
Accepted
abab
Accepted
bbaab
Rejected
```

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

**Solution:**

```
%{
#include <stdio.h>
#include <string.h>
char window[4] = "   ";
%}
%%
[a|b] {
   window[0] = window[1];
   window[1] = window[2];
   window[2] = yytext[0];
}
\n {
   if (strlen(window) >= 3 && window[0] == 'a') {
      printf("Accepted\n");
   } else {
      printf("Rejected\n");
   }
   strcpy(window, "   ");
}
.        {}
%%
int main() {
   char input[100];
   while (1) {
      printf("Enter a string of a's and b's (end with Enter, type 'exit' to quit): ");
      fgets(input, sizeof(input), stdin);
      if (strncmp(input, "exit", 4) == 0) {
         break;
      }
      yylex();
   }
   return 0;
}
int yywrap(){
   return 1;
}
```

**Output:**

```
aabb
Accepted
aaabbb
Rejected
aaaaab
Accepted
```

## Q11. Design a DFA in LEX Code to Identify and print Integer &amp; Float Constants and Identifier.

### Solution:

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

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

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

[a-zA-Z_]+[a-zA-Z_0-9]*    {printf("Identifier: %s\n", yytext); }

[\n\t ]+        { }

.               {printf("Unrecognized token: %s\n", yytext);}
%%
int main() {
   printf("Enter your input (type 'exit' to quit):\n");
   yylex();
   return 0;
}

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

### Output:

```
Enter your input (type 'exit' to quit):
1234567
Integer: 1234567
123456.34567
Float: 123456.34567
dfgh
Identifier: dfgh
23456.21
Float: 23456.21
```

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

## Solution:

**Yaac:**

```
%{
   #include <stdio.h>
   #include <stdlib.h>
   void yyerror(const char *msg);
   int yylex();
%}

%token NUMBER
%token PLUS MINUS MUL DIV LPAREN RPAREN EOL

%%

input:
   expression EOL   { printf("Valid Expression\n"); }
   | error EOL      { printf("Invalid Expression\n"); yyerrok; }
   ;

expression:
   expression PLUS term
   | expression MINUS term
   | term
   ;

term:
   term MUL factor
   | term DIV factor
   | factor
   ;

factor:
   NUMBER
   | LPAREN expression RPAREN
   ;

%%

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

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

**Lex:**

```
%{
   #include "12.tab.h"
   #include <ctype.h>
%}

%%

[0-9]+           { yylval = atoi(yytext); return NUMBER; }
[ \t]            ;    // Ignore spaces and tabs
"+"              { return PLUS; }
"-"              { return MINUS; }
"*"              { return MUL; }
"/"              { return DIV; }
"("              { return LPAREN; }
")"              { return RPAREN; }
\n               { return EOL; }
.                { return yytext[0]; } // Catch all for any other character

%%

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

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

**Solution:**

**Yaac:**

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

int yylex();
void yyerror(const char *s) {
   printf("Error: %s\n", s);
}
%}

%token NUMBER
%token PLUS MINUS MULTIPLY DIVIDE

%left PLUS MINUS
%left MULTIPLY DIVIDE

%%

expression:
       expression PLUS expression      { $$ = $1 + $3; }
   |  expression MINUS expression     { $$ = $1 - $3; }
   |  expression MULTIPLY expression { $$ = $1 * $3; }
   |  expression DIVIDE expression    { $$ = $1 / $3; }
   |  '(' expression ')'            { $$ = $2; }
   |  NUMBER                      { $$ = $1; }
   ;

%%

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

**Lex:**

```
%{
#include "13.tab.h"
%}

%%

[0-9]+              { yylval = atoi(yytext); return NUMBER; }
[\t\n ]             { /* Ignore whitespace */ }
"+"                 { return PLUS; }
"-"                 { return MINUS; }
"*"                 { return MULTIPLY; }
"/"                 { return DIVIDE; }
.                   { return yytext[0]; }

%%

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