Question 1

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

Code:

```
%{
    // c code
    #include <stdio.h>

    int noLines = 0;
    int noSpace = 0;
    int noTabs = 0;
    int noCharacter = 0;
%}

%%
\n noLines++;
\t noTabs++;
[ ] noSpace++;
. noCharacter++;
%%

int main(){
    yylex();
    printf("noLines: %d\n", noLines);
    printf("noTabs: %d\n", noTabs);
    printf("noSpaces: %d\n", noSpace);
    printf("noWords: %d\n", noCharacter);
    return 0;
}
```

Output:

```
totoro on catbus ./lab-cd/1 on 🌱 mast
❯❯❯❯ lex prog.l

totoro on catbus ./lab-cd/1 on 🌱 mast
❯❯❯❯ gcc lex.yy.c -lfl

totoro on catbus ./lab-cd/1 on 🌱 mast
❯❯❯❯ ./a.out
hi this is _ my         plaace
i live here :
noLines: 2
noTabs: 1
noSpaces: 8
noWords: 27
```

Question 2

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

Code:

```
%{
    // c code
    #include <stdio.h>
%}

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

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

Output:

```
totoro on catbus ./lab-cd/2 on 🌱 mast [ 🤖 ]
❯❯❯❯ lex prog.l

totoro on catbus ./lab-cd/2 on 🌱 mast [ 📝 🤖 ]
❯❯❯❯ gcc lex.yy.c -lfl

totoro on catbus ./lab-cd/2 on 🌱 mast [ 📝 🤖 ]
❯❯❯❯ ./a.out
2hi
Invalid Identifier: 2hi

shi
Valid Identifier: shi

_shi
Valid Identifier: _shi

_23shi
Valid Identifier: _23shi

2
Invalid Identifier: 2
```

Question 3

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

Code:

```
%{
    // c code
    #include <stdio.h>
%}

%%
[0-9]*"."[0-9]* printf("Float value: %s\n", yytext);
[0-9]+ printf("Integer Value: %s\n", yytext);
.|\n {/* Ignore all other characters. */}
%%

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

Output:

```
totoro on catbus ./lab-cd/3 on 🌱 mast [ 📝 🤖 ]
❯❯❯❯ lex prog.l

totoro on catbus ./lab-cd/3 on 🌱 mast [ 📝 🤖 ]
❯❯❯❯ gcc lex.yy.c -lfl

totoro on catbus ./lab-cd/3 on 🌱 mast [ 📝 🤖 ]
❯❯❯❯ ./a.out
231
Integer Value: 231
123.32
Float value: 123.32
.32
Float value: .32
123.
Float value: 123.
231.1
Float value: 231.1
2
Integer Value: 2
```

Question 4

Design a LEX Code for Tokenizing (Identify and print OPERATORS, SEPARATORS, KEYWORDS, IDENTIFIERS) from 'in.c' file.

Code:

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

%%
int|float|if|else|while|main|return { fprintf(yyout, "Keyword: %s\n", yytext);
    }
== |
\<= |
\>= |
-- |
\+\+ |
= |
\/ |
\* |
\+ |
- |
% |
\< |
\> |
! |
~ { fprintf(yyout, "Operator: %s\n", yytext); }
[,;(){}] { fprintf(yyout, "Separator: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]* { fprintf(yyout, "Identifier: %s\n", yytext); }
.|\n {/* Ignore all other characters. */}
%%

int main(){
    extern FILE *yyin, *yyout;
    yyin = fopen("in.c","r");
    yyout = fopen("out.txt","w");
    yylex();
    return 0;
}
```

Output:

Input File:

```
int p=1,d=0,r=4;
float m=0.0, n=200.0
while (p <= 3)
        { if(d==0)
            { m= m+n*r+4.5; d++;  }
        else
            { r++; m=m+r+1000.0;  }
        p++;  }
```

Output File:

```
Keyword: int
Identifier: p
Operator: =
Separator: ,
Identifier: d
Operator: =
Separator: ,
Identifier: r
Operator: =
Separator: ;
Keyword: float
Identifier: m
Operator: =
Separator: ,
Identifier: n
Operator: =
Keyword: while
Separator: (
Identifier: p
Operator: <=
Separator: )
Separator: {
Keyword: if
Separator: (
Identifier: d
Operator: ==
Separator: )
Separator: {
Identifier: m
Operator: =
Identifier: m
Operator: +
Identifier: n
Operator: *
Identifier: r
Operator: +
Separator: ;
Identifier: d
Operator: ++
Separator: ;
Separator: }
Keyword: else
Separator: {
```

```
Identifier: r
Operator: ++
Separator: ;
Identifier: m
Operator: =
Identifier: m
Operator: +
Identifier: r
Operator: +
Separator: ;
Separator: }
Identifier: p
Operator: ++
Separator: ;
Separator: }
```

Question 5

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

Code:

```
%{
    // c code
    #include <stdio.h>

    int noCharacters = 0;
    int noWords = 0;
    int noSpace = 0;
%}

%%
[ ] {noSpace++;}
[^ \n\t]+ {noWords++, noCharacters=noCharacters+yyleng;}
\n {noCharacters++;}
%%

int main(){
    extern FILE *yyin;
    yyin = fopen("input.txt", "r");
    FILE *fp = fopen("output.txt", "w");
    yylex();
    fprintf(fp, "noWords: %d\n", noWords);
    fprintf(fp, "noSpaces: %d\n", noSpace);
    fprintf(fp, "noCharacters: %d\n", noCharacters);
    fclose(fp);
    return 0;
}
```

Output:

Input File:

```
hii total no of
words in this file are
10
```

Output:

```
noWords: 10
noSpaces: 7
noCharacters: 34
```

Question 6

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

Code:

```
%{
    // c code
    #include <stdio.h>
%}

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

int main(){
    extern FILE *yyin, *yyout;
    yyin = fopen("input.txt", "r");
    yyout = fopen("output.txt", "w");
    yylex();
    return 0;
}
```

Output:

Input File:

```
hi my   name is ram
and i am in ds section
```

Output File:

```
hi my name is ram and i am in ds section
```

Question 7

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

Code:

```
%{
    // c code
    #include <stdio.h>
%}

%%
\/\/(.*) {};
\/\*(.*\n)*.*\*\/ {};
%%

int main(){
    extern FILE *yyin, *yyout;
    yyin = fopen("input.c", "r");
    yyout = fopen("out.c", "w");
    yylex();
    return 0;
}
```

Output:

Input File:

```
#include <stdio.h>

// main function
int main()
{
    /* code */
    printf("hello world!");
    return 0;
}
```

Output File:

```
#include <stdio.h>


int main()
{

    printf("hello world!");
    return 0;
}
```

Question 8

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.

Code:

```
%{
    // c code
    #include <stdio.h>
%}

%%
\<[^>]*\> fprintf(yyout, "%s\n", yytext);
.|\n {};
%%

int main(){
    extern FILE *yyin, *yyout;
    char in[100], out[100];
    printf("Enter the input filename: ");
    scanf("%s", in);
    printf("Enter the ouput filename: ");
    scanf("%s", out);
    yyin = fopen(in, "r");
    yyout = fopen(out, "w");
    yylex();
    return 0;
}
```

Output:

Input File:

```
<!DOCTYPE html>
<html>

<head>
    <title>Document</title>
</head>

<body>
    <p>
        This is a html file.
    </p>
</body>

</html>
```

Output File:

```
<!DOCTYPE html>
<html>
<head>
<title>
</title>
</head>
<body>
<p>
</p>
</body>
</html>
```

Question 9

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

Code:

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

%s A B C DEAD

%%
<INITIAL>a BEGIN A;
<INITIAL>b BEGIN B;
<INITIAL>[^\n] BEGIN DEAD ;
<INITIAL>\n BEGIN INITIAL; {printf("Accepted!\n");}
<A>a BEGIN INITIAL;
<A>b BEGIN C;
<A>[^\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; {printf("Not Accepted!\n");}

<B>a BEGIN C;
<B>b BEGIN INITIAL;
<B>[^\n] BEGIN DEAD;
<B>\n BEGIN INITIAL; {printf("Not Accepted!\n");}

<C>a BEGIN B;
<C>b BEGIN A;
<C>[^\n] BEGIN DEAD;
<C>\n BEGIN INITIAL; {printf("Not Accepted!\n");}

<DEAD>[^\n] BEGIN DEAD;
<DEAD>\n BEGIN INITIAL; {printf("Invalid!\n");}
%%

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

Output:

```
totoro on catbus ./lab-cd/9 on 🌱 mast [ 📝 ]
❯❯❯❯ lex prog.l

totoro on catbus ./lab-cd/9 on 🌱 mast [ 📝 ]
❯❯❯❯ gcc lex.yy.c -lfl

totoro on catbus ./lab-cd/9 on 🌱 mast [ 📝 ]
❯❯❯❯ ./a.out
aabbaa
Accepted!
aba
Not Accepted!
abaab
Not Accepted!
abab
Accepted!
aa12
Invalid!
^C
```

Question 10

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

Code:

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

%s A B C D E F G DEAD

%%
<INITIAL>a BEGIN A;
<INITIAL>b BEGIN INITIAL;
<INITIAL>[^\n] BEGIN DEAD;
<INITIAL>\n BEGIN INITIAL; { printf("Not Accepted\n");}

<A>a BEGIN B;
<A>b BEGIN F;
<A>[^\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; { printf("Not Accepted\n");}

<B>a BEGIN C;
<B>b BEGIN D;
<B>[^\n] BEGIN DEAD;
<B>\n BEGIN INITIAL; { printf("Not Accepted\n");}

<C>a BEGIN C;
<C>b BEGIN D;
<C>[^\n] BEGIN DEAD;
<C>\n BEGIN INITIAL; { printf("Accepted\n");}

<D>a BEGIN E;
<D>b BEGIN G;
<D>[^\n] BEGIN DEAD;
<D>\n BEGIN INITIAL; { printf("Accepted\n");}

<E>a BEGIN B;
<E>b BEGIN F;
<E>[^\n] BEGIN DEAD;
<E>\n BEGIN INITIAL; { printf("Accepted\n");}

<F>a BEGIN E;
<F>b BEGIN G;
<F>[^\n] BEGIN DEAD;
<F>\n BEGIN INITIAL; { printf("Not Accepted\n");}

<G>a BEGIN A;
<G>b BEGIN INITIAL;
<G>[^\n] BEGIN DEAD;
<G>\n BEGIN INITIAL; { printf("Accepted!\n");}

<DEAD>[^\n] BEGIN DEAD;
<DEAD>\n BEGIN INITIAL; { printf("Invalid!\n");}
%%
```

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

Output:

```
totoro on catbus ./lab-cd/10 on 🌱 mast [ 📝 🤖 ]
❯❯❯❯ lex prog.l

totoro on catbus ./lab-cd/10 on 🌱 mast [ 📝 🤖 ]
❯❯❯❯ gcc lex.yy.c -lfl

totoro on catbus ./lab-cd/10 on 🌱 mast [ 📝 🤖 ]
❯❯❯❯ ./a.out
bbabb
Accepted!
bbbbaaa
Accepted
aaa
Accepted
a
Not Accepted
aa
Not Accepted
aba
Accepted
^C
```

Question 11

Design a DFA in LEX Code to Identify and print Integer and Float Constants and Identifier.

Code:

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

%s A B C DEAD

%%
<INITIAL>[_A-Za-z] BEGIN A;
<INITIAL>[0-9] BEGIN B;
<INITIAL>[.] BEGIN C;
<INITIAL>[^\n] BEGIN DEAD;
<INITIAL>\n BEGIN INITIAL; {printf("Input something\n");}

<A>[_A-Za-z0-9] BEGIN A;
<A>[^\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; {printf("Identifier\n");}

<B>[0-9] BEGIN B;
<B>[.] BEGIN C;
<B>[^\n] BEGIN DEAD;
<B>\n BEGIN INITIAL; { printf("Integer\n"); }

<C>[0-9] BEGIN C;
<C>[^\n] BEGIN DEAD;
<C>\n BEGIN INITIAL; { printf("Float\n"); }

<DEAD>[^\n] BEGIN DEAD;
<DEAD>\n BEGIN INITIAL; { printf("Invalid!\n");}
%%

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

Output:

```
totoro on catbus ./lab-cd/11 on mast [ ]
❯❯❯❯ lex prog.l

totoro on catbus ./lab-cd/11 on mast [ ]
❯❯❯❯ gcc lex.yy.c -lfl

totoro on catbus ./lab-cd/11 on mast [ ]
❯❯❯❯ ./a.out
shiv
Identifier
_ran
Identifier
234234
Integer
2342.423
Float
.1234
Float
asfd243
Identifier
fafsd##2
Invalid!
^C
```

Question 12

Design a DFA which accepts strings ending with 00.

Code:

```
%{
    // design a DFA which accepts strings ending with 00.
    #include <stdio.h>
    #include <stdlib.h>
%}

%s A B DEAD

%%
<INITIAL>0 BEGIN A;
<INITIAL>1 BEGIN INITIAL;
<INITIAL>[^\n] BEGIN DEAD;
<INITIAL>\n BEGIN INITIAL; { printf("Not Accepted\n"); }

<A>0 BEGIN B;
<A>1 BEGIN INITIAL;
<A>[^\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; { printf("Not Accepted\n"); }

<B>0 BEGIN B;
<B>1 BEGIN INITIAL;
<B>[^\n] BEGIN DEAD;
<B>\n BEGIN INITIAL; { printf("Accepted\n"); }

<DEAD>[^\n] BEGIN DEAD;
<DEAD>\n BEGIN INITIAL; { printf("Invalid\n"); }
%%

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

Output:

```
totoro on catbus ./lab-cd/12 on 🌱 mast [ 🤖 ]
❯❯❯❯ lex prog.l

totoro on catbus ./lab-cd/12 on 🌱 mast [ 🤖 ]
❯❯❯❯ gcc lex.yy.c -lfl

totoro on catbus ./lab-cd/12 on 🌱 mast [ 🤖 ]
❯❯❯❯ ./a.out
123100
Invalid
1100
Accepted
11010000
Accepted
101
Not Accepted
^C
```

Question 13

Design a DFA which accepts even number of 1s.

Code:

```
%{
    // design a DFA which accepts even number of 1s.
    #include <stdio.h>
    #include <stdlib.h>
%}

%s A DEAD

%%
<INITIAL>0 BEGIN INITIAL;
<INITIAL>1 BEGIN A;
<INITIAL>[^\n] BEGIN DEAD;
<INITIAL>\n BEGIN INITIAL; { printf("Accepted\n"); }

<A>0 BEGIN A;
<A>1 BEGIN INITIAL;
<A>[^\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; { printf("Not Accepted\n"); }

<DEAD>[^\n] BEGIN DEAD;
<DEAD>\n BEGIN INITIAL; { printf("Invalid\n"); }
%%

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

Output:

```
totoro on catbus ./lab-cd/13 on 🌱 mast [ 🏚 ] took 6s
✗  lex prog.l

totoro on catbus ./lab-cd/13 on 🌱 mast [ 🏚 ]
❯❯❯❯ gcc lex.yy.c -lfl

totoro on catbus ./lab-cd/13 on 🌱 mast [ 🏚 ]
❯❯❯❯ ./a.out
11
Accepted
101
Accepted
1011
Not Accepted
111
Not Accepted
^C
```

Question 14

Design a DFA which accepts strings ending with 01.

Code:

```
%{
    // design a DFA which accepts strings ending with 01.
    #include <stdio.h>
    #include <stdlib.h>
%}

%s A B DEAD

%%
<INITIAL>0 BEGIN A;
<INITIAL>1 BEGIN INITIAL;
<INITIAL>[^\n] BEGIN DEAD;
<INITIAL>\n BEGIN INITIAL; { printf("Not Accepted\n"); }

<A>1 BEGIN B;
<A>0 BEGIN A;
<A>[^\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; { printf("Not Accepted\n"); }

<B>0 BEGIN A;
<B>1 BEGIN INITIAL;
<B>[^\n] BEGIN DEAD;
<B>\n BEGIN INITIAL; { printf("Accepted\n"); }

<DEAD>[^\n] BEGIN DEAD;
<DEAD>\n BEGIN INITIAL; { printf("Invalid\n"); }
%%

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

Output:

```
totoro on catbus ./lab-cd/14 on 🌱 mast [ 🤷 ] took 14s
X  lex prog.l

totoro on catbus ./lab-cd/14 on 🌱 mast [ 🤷 ]
❯❯❯❯ gcc lex.yy.c -lfl

totoro on catbus ./lab-cd/14 on 🌱 mast [ 🤷 ]
❯❯❯❯ ./a.out
0010
Not Accepted
0001
Accepted
01
Accepted
11101
Accepted
^C
```

Question 11

Design a DFA which accepts strings starting with 11.

Code:

```
%{
    // design a DFA which accepts strings starting with 11.
    #include <stdio.h>
    #include <stdlib.h>
%}

%s A B C DEAD

%%
<INITIAL>1 BEGIN A;
<INITIAL>0 BEGIN C;
<INITIAL>[^\n] BEGIN DEAD;
<INITIAL>\n BEGIN INITIAL; { printf("Not Accepted\n"); }

<A>1 BEGIN B;
<A>0 BEGIN C;
<A>[^\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; { printf("Not Accepted\n"); }

<B>0 BEGIN B;
<B>1 BEGIN B;
<B>[^\n] BEGIN DEAD;
<B>\n BEGIN INITIAL; { printf("Accepted\n"); }

<C>0 BEGIN C;
<C>1 BEGIN C;
<C>[^\n] BEGIN DEAD;
<C>\n BEGIN INITIAL; { printf("Not Accepted\n"); }

<DEAD>[^\n] BEGIN DEAD;
<DEAD>\n BEGIN INITIAL; { printf("Invalid\n"); }
%%

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

Output:

```
❯❯❯❯ lex prog.l

totoro on catbus ./lab-cd/15 on 🌱 mast [ 🤖 ]
❯❯❯❯ gcc lex.yy.c -lfl

totoro on catbus ./lab-cd/15 on 🌱 mast [ 🤖 ]
❯❯❯❯ ./a.out
11
Accepted
1100
Accepted
101
Not Accepted
001
Not Accepted
010
Not Accepted
110000000
Accepted
110a
Invalid
^C
```

Question 16

Design a DFA which accepts strings with odd 1s and even 0s.

Code:

```
%{
    // design a DFA which accepts strings with odd 1s and even 0s.
    #include <stdio.h>
    #include <stdlib.h>
%}

%s A B C DEAD

%%
<INITIAL>1 BEGIN A;
<INITIAL>0 BEGIN C;
<INITIAL>[^\n] BEGIN DEAD;
<INITIAL>\n BEGIN INITIAL; { printf("Not Accepted\n"); }

<A>0 BEGIN B;
<A>1 BEGIN INITIAL;
<A>[^\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; { printf("Accepted\n"); }

<B>1 BEGIN C;
<B>0 BEGIN A;
<B>[^\n] BEGIN DEAD;
<B>\n BEGIN INITIAL; { printf("Not Accepted\n"); }

<C>0 BEGIN INITIAL;
<C>1 BEGIN B;
<C>[^\n] BEGIN DEAD;
<C>\n BEGIN INITIAL; { printf("Not Accepted\n"); }

<DEAD>[^\n] BEGIN DEAD;
<DEAD>\n BEGIN INITIAL; { printf("Invalid\n"); }
%%

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

Output:

```
totoro on catbus ./lab-cd/16 on 🌱 mast [ 📝 🤖 ]
❯❯❯❯ lex prog.l

totoro on catbus ./lab-cd/16 on 🌱 mast [ 📝 🤖 ]
❯❯❯❯ gcc lex.yy.c -lfl

totoro on catbus ./lab-cd/16 on 🌱 mast [ 📝 🤖 ]
❯❯❯❯ ./a.out
0011
Not Accepted
00111
Accepted
011
Not Accepted
01
Not Accepted
001
Accepted
^C
```