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

Code: -

```
%{
#include<stdio.h>
int lineCount=0,spaceCount=0,tabCount=0,restCount=0;
%}

%%
[\n] {++lineCount;}
[" "] {++spaceCount;}
[\t] {++tabCount;}
[^\t" "\n] {++restCount;}
END {return 0;}
%%

int yywrap(){
        return 1;
}

int main(){

        printf("Enter the Sentence : \n");

        yylex();
        printf("Number of Lines are : %d\n",lineCount);
        printf("Number of Spaces are : %d\n",spaceCount);
        printf("Number of Tabs Characters are : %d\n",tabCount);
        printf("Number of Rest Characters are : %d\n",restCount);
        return 0;
}
```

Output: -

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> flex prog1.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> gcc lex.yy.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> ./a.exe
Enter the Sentence :
Wake Up to Reality
Nothing ever goes as planned in this accursed   World
The longer you live, the more you realize that
                        by Ghost
^Z
Number of Lines are : 4
Number of Spaces are : 19
Number of Tabs Characters are : 4
Number of Rest Characters are : 103
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design>
```

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

Code: -

```
%{
#include<stdio.h>
%}
%%
[a-zA-Z_][a-zA-Z0-9]*     { printf("Identifier\n"); }
.*                        { printf("Not an Identifier\n"); }
%%
int yywrap()
{
return 1;
}
int main(void)
{
while(yylex());
return 0;
}
```

Output: -

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> ./a.exe
count
Identifier

12e
Not an Identifier

12
Not an Identifier
```

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

Code: -

```
%{
#include <stdio.h>
%}
%%
[0-9]+              { printf("This is an Integer Number\n"); }
[0-9]+[.][0-9]+     { printf("This is a Floating Point Number\n"); }
.+                  { printf("This is not a Valid Number\n"); }
%%
int main() {
   yylex();
   return 0;
}
int yywrap(){ return 0;
}
```

Output: -

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> flex prog3.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> gcc lex.yy.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> ./a.exe
12
This is an Integer Number

1.2
This is a Floating Point Number

1.21.
This is not a Valid Number

PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design>
```

4. **Design a LEX Code for Tokenizing (Identify and print OPERATORS, SEPERATORS, KEYWORDS, and IDENTIFIERS) in the given input:**

Code: -

```
%{
#include <stdio.h>
%}
%%
"if"|"else"|"int"|"return"|"for"   { printf("%s is a keyword.\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]*  { printf("%s is an identifier.\n", yytext); }
[0-9]+     { printf("%s is a constant.\n", yytext); }
"=="|"!="|"<="|">="|"+"|"-"|"*"|"/"  { printf("%s is an operator.\n", yytext); }
"="|"<"|">"                    { printf("%s is an operator.\n", yytext); }
"{"|"}"|"("|")"|"["|"]"|";"|","        { printf("%s is a separator.\n", yytext); }
[ \t\n]      ;
%%
int main() {
   printf("Enter C code for tokenization:\n");
   yylex();
   return 0;
}
int yywrap() { return 0; }
```

Output: -

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> ./a.exe
Enter C code for tokenization:
int a=10,b=15;
int is a keyword.
a is an identifier.
= is an operator.
10 is a constant.
, is a separator.
b is an identifier.
= is an operator.
15 is a constant.
; is a separator.
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design>
```

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

```
%{
#include<stdio.h>
int totChars=0,whiteSpace=0,wordCount=0;
%}
%%
[ \n\t] {++whiteSpace;}
[^ \n\t]+ {++wordCount; totChars=totChars+yyleng;}
%%
int yywrap(){
   return 1;
}
int main(){
   extern FILE *yyin;
   yyin = fopen("input.txt","r");
   yylex();
   printf("White Spaces : %d\nWords : %d\nTotal Characters :
%d\n",whiteSpace,wordCount,whiteSpace+totChars);
   return 0;
}
```

   Output: -

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> flex prog5.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> gcc lex.yy.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> ./a.exe
White Spaces : 8
Words : 8
Total Characters : 45
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design>
```

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

Code: -

```
%{
#include<stdio.h>
%}
%%
[ \t]+ fprintf(yyout," ");
. fprintf(yyout,"%s",yytext);
%%
int yywrap(){
   return 1;
}
int main(){
   extern FILE *yyin,*yyout;
   yyin=fopen("input.txt","r");
   yyout=fopen("output.txt","w");
   yylex();
   return 0;
}
```

Output: -

<u>Input File: -</u>

```
Hello       There,
i'm Arjun
ans i'm     from Ramnagar
```

<u>Output File: -</u>

```
Hello There,
i'm Arjun
ans i'm from Ramnagar
```

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

Code: -

```
%{
#include<stdio.h>
%}
%%
"//".* ;
"/*"([^*]|[*]+[^/])*[*]+"/" ;
. fprintf(yyout,"%s",yytext);
%%
int yywrap(){
   return 1;
}
int main(){
   extern FILE *yyin,*yyout;
   yyin=fopen("input.txt","r");
   yyout=fopen("output.txt","w");
   yylex();
   return 0;
}
```

Output: -

<u>Input File: -</u>

```
#include<bits/stdc++.h> //header file
using namespace std; // std namespace
int main() { //main function
    cout<<"Hello World<<endl; //prints Hello world on screen
}
/*
This is a multi-line comment in C++.
It starts with /* and ends with */
```

Output File: -

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    cout<<"Hello World<<endl;
}
```

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: -

```
%{
#include<stdio.h>
%}
%%
"<"[^>]*">" {fprintf(yyout,"%s",yytext);}
. ;
%%
int yywrap(){
   return 1;
}
int main(){
   extern FILE *yyin,*yyout;
   yyin=fopen("index.html","r");
   yyout=fopen("output.txt","w");
   yylex();
   return 0;
}
```

Html file :-

```
<!DOCTYPE html>
<html>

<head>
    <title>I don't know</title>
</head>

<body>
    <h1>Wassup Everyone</h1>
    <p>My name is <strong>Arjun</strong>...</p>
</body>

</html>
```

Output file: -

```
<!DOCTYPE html>
<html>

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

<body>
<h1></h1>
<p><strong></strong></p>
</body>

</html>
```

9. **Design a LEX Code to recognize and print the following tokens: a) string b) keywords c) constants d) identifiers e) literals.**

Code: -

```
%{
#include <stdio.h>
%}
%%
"int"|"float"|"char"|"if"|"else"|"return"|"string"|"printf"  { printf("KEYWORD: %s\n", yytext); }

\"[^\"]*\"      { printf("STRING: %s\n", yytext); }

'[^']*'        { printf("LITERAL: %s\n", yytext); }

[0-9]+         { printf("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 yywrap() {
   return 1;
}
int main() {
   yylex();
   return 0;
}
```

Output :-

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> ./a.exe
int a = 10;
KEYWORD: int
IDENTIFIER: a
UNKNOWN: =
CONSTANT: 10
UNKNOWN: ;
string str = "Hello World!"
KEYWORD: string
IDENTIFIER: str
UNKNOWN: =
STRING: "Hello World!"
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design>
```

10. Design a LEX Code to take check whether the given number is even or odd.

Code :-

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

%%
[0-9]+          { long num = atol(yytext);
                if (num % 2 == 0) {
                   printf("%s is an even number.\n", yytext);
                } else {
                   printf("%s is an odd number.\n", yytext);
                }
              }
.+              { printf("This is not a valid number.\n"); }
%%
int main() {
   yylex();
   return 0;
}
int yywrap(){ return 0;}
```

Output :-

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> flex prog10.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> gcc lex.yy.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> ./a.exe
12
12 is an even number.

13
13 is an odd number.

13a
This is not a valid number.
```

11. Design a LEX Code to count number of vowels and consonants in a given pattern.

Code: -

```
%{
#include <stdio.h>
int vowels = 0;
int consonants = 0;
%}

%%
[aeiouAEIOU]   { vowels++; }
[bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ] { consonants++; }
.|\n        ;
%%

int main() {
   yylex();
   printf("Total number of Vowels are: %d\n", vowels);
   printf("Total number of Consonants are: %d\n", consonants);
   return 0;
}
int yywrap(){
   return 1;
}
```

Output :-

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> flex prog11.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> gcc lex.yy.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design> ./a.exe
This is the LEX code.
^Z
Total number of Vowels are: 6
Total number of Consonants are: 10
```

12. Design a LEX Code to check for a valid E-mail Id.

Code: -

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

%%
^[a-zA-Z0-9._%+-]+@gmail\.com$ {
   printf("Valid Email\n");
}
.* {
   printf("Not a valid email\n");
}
%%

int yywrap(){
   return 1;
}
int main(){
   printf("Enter your email: ");
   yylex();
   return 0;
}
```

Output: -

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-12> flex valid_email.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-12> gcc lex.yy.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-12> ./a.exe
Enter your email: arjunkandari15@gmail.com
Valid Email

arjun.com
Not a valid email
```

13. Design a DFA in LEX Code which accepts all possible set of string containing even number of 'a' and even number of 'b' over input alphabet Σ = {a, b}.

Code: -

```
%{
  #include <stdio.h>
%}
%option noyywrap
%s A B C DEAD

%%
<INITIAL>a          BEGIN A;
<INITIAL>b          BEGIN B;
<INITIAL>\n         { printf("String Accepted\n"); BEGIN INITIAL; }
<INITIAL>[^ab\n]      BEGIN DEAD;

<A>a              BEGIN INITIAL;
<A>b              BEGIN C;
<A>\n             { printf("String Unaccepted\n"); BEGIN INITIAL; }
<A>[^ab\n]          BEGIN DEAD;

<B>a              BEGIN C;
<B>b              BEGIN INITIAL;
<B>\n             { printf("String Unaccepted\n"); BEGIN INITIAL; }
<B>[^ab\n]          BEGIN DEAD;

<C>a              BEGIN B;
<C>b              BEGIN A;
<C>\n             { printf("String Unaccepted\n"); BEGIN INITIAL; }
<C>[^ab\n]          BEGIN DEAD;

<DEAD>\n            { printf("String Unaccepted\n"); BEGIN INITIAL; }
<DEAD>[^\n]         BEGIN DEAD;
```

```
%%


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

Output: -

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-13> flex even_a_and_even_b.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-13> gcc lex.yy.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-13> ./a.exe
ab
String Unaccepted
aabb
String Accepted
ababa
String Unaccepted
abababab
String Accepted
```

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

Code: -

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

%option noyywrap
%s INT A FLOAT ID DEAD

%%

<INITIAL>[0-9]        BEGIN INT;
<INITIAL>[a-zA-Z_]      BEGIN ID;
<INITIAL>[^a-zA-Z0-9_\n] BEGIN DEAD;
<INITIAL>\n           { printf("Invalid String\n"); BEGIN INITIAL; }

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

<A>[0-9]              BEGIN FLOAT;
<A>[^0-9\n]           BEGIN DEAD;
<A>\n                 { printf("Invalid String\n"); BEGIN INITIAL; }

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

<ID>[a-zA-Z0-9_]       BEGIN ID;
<ID>[^a-zA-Z0-9_\n]     BEGIN DEAD;
```

```
<ID>\n              { printf("Identifier\n"); BEGIN INITIAL; }


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


%%


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

Output: -

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\program-14> flex constants_and_identifiers.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\program-14> gcc lex.yy.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\program-14> ./a.exe
int
Identifier
12
Integer
12.5
Floating Number
12.
Invalid String
in3
Identifier
3a
Invalid String
```

15. Design a DFA in LEX Code over Σ = {a, b} which contains set of all possible strings where every string starts with a and ends with b.

Code: -

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

%option noyywrap
%s A B DEAD

%%

<INITIAL>a          BEGIN A;
<INITIAL>[^a\n]       BEGIN DEAD;
<INITIAL>\n         { printf("String not accepted\n"); BEGIN INITIAL; }

<A>a              BEGIN A;
<A>b              BEGIN B;
<A>[^ab\n]          BEGIN DEAD;
<A>\n             { printf("String not accepted\n"); BEGIN INITIAL; }

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

<DEAD>[^\n]         BEGIN DEAD;
<DEAD>\n            { printf("String not accepted\n"); BEGIN INITIAL; }

%%

int main() {
```

```
    yylex();
    return 0;
}
```

Output: -

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-15> flex DFA_StartA_EndB.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-15> gcc lex.yy.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-15> ./a.exe
ab
String Accepted
aab
String Accepted
aba
String not accepted
abc
String not accepted
aaaabbbbab
String Accepted
```

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

Code: -
Arithmetic_Expression_Validator.l

```
%{
#include "parser.tab.h"
%}
%%
[0-9]+ {yylval = atoi(yytext); return NUMBER;}
"+" {return PLUS;}
"-" {return MINUS;}
"*" {return MUL;}
"/" {return DIV;}
"(" {return LP;}
")" {return RP;}
[ \t] {} /* Skip spaces and tabs */
[\n] {return 0;} /* Return 0 to signal end of input on newline */
. {printf("Invalid character: %s\n", yytext); return 0;}
%%
int yywrap(){
   return 1;
}
```

Parser.y

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

%token NUMBER
%token PLUS MINUS MUL DIV LP RP
%left PLUS MINUS
%left MUL DIV
%%

input: /* empty */
    | expr   {printf("Valid Arithmetic Expression\n");}
    ;

expr: expr PLUS expr
   | expr MINUS expr
   | expr DIV expr
   | expr MUL expr
   | LP expr RP
```

```
  | NUMBER
  ;

%%

void yyerror(const char *s){
  printf("Not a Valid Arithmetic Expression\n");
}

int main(){
  printf("Enter an Arithmetic Expression: ");
  int x = yyparse();
  return 0;
}
```

Output file: -

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-16> bison -d parser.y
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-16> flex Arithmetic_Expression_Validator.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-16> gcc lex.yy.c parser.tab.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-16> ./a.exe
Enter an Arithmetic Expression: 2+3
Valid Arithmetic Expression
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-16> bison -d parser.y
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-16> flex Arithmetic_Expression_Validator.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-16> gcc lex.yy.c parser.tab.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-16> ./a.exe
Enter an Arithmetic Expression: 2+3
Valid Arithmetic Expression
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-16> ./a.exe
Enter an Arithmetic Expression: 2+
Not a Valid Arithmetic Expression
```

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

i. No Precedence

lexer_no_precedence.l

```
%{
#include "parser_no_precedence.tab.h"
%}
%%
[0-9]+     { yylval = atoi(yytext); return NUMBER; }
"+"        { return PLUS; }
"-"        { return MINUS; }
"*"        { return MUL; }
"/"        { return DIV; }
"("        { return LP; }
")"        { return RP; }
[ \t]      { /* Skip spaces and tabs */ }
[\n]       { return 0; /* End of input */ }
.          { printf("Invalid character: %s\n", yytext); return 0; }
%%
int yywrap() {
   return 1;
}
```

parser_no_precedence.y

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

%token NUMBER
```

```
%token PLUS MINUS MUL DIV LP RP



%%

input: /* empty */
    | expr    { printf("Result = %d\n", $1); }
    ;

expr: term    { $$ = $1; }
   | expr PLUS term    { $$ = $1 + $3; }
   | expr MINUS term   { $$ = $1 - $3; }
   ;

term: factor    { $$ = $1; }
   | term MUL factor   { $$ = $1 * $3; }
   | term DIV factor   {
      if ($3 == 0) {
         yyerror("Division by zero");
         $$ = 0;
      } else {
         $$ = $1 / $3;
      }
    }
   ;

factor: NUMBER    { $$ = $1; }
     | LP expr RP    { $$ = $2; }
     | MINUS factor  { $$ = -$2; } /* Unary minus */
     ;
```

```
%%

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

int main() {
   printf("Enter an arithmetic expression (without operator precedence grammar): ");
   yyparse();
   return 0;
}
```

Output :-

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-17\No_Precedence> bison -d parser.y
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-17\No_Precedence> flex lexer.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-17\No_Precedence> gcc lex.yy.c parser.tab.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-17\No_Precedence> ./a.exe
Enter an arithmetic expression (without operator precedence grammar): 2+3*4
Result = 14
```

ii. With precedence

Lexer.y

```
%{
#include "parser.tab.h"
%}
%%
[0-9]+     { yylval = atoi(yytext); return NUMBER; }
"+"        { return PLUS; }
"-"        { return MINUS; }
"*"        { return MUL; }
"/"        { return DIV; }
"("        { return LP; }
")"        { return RP; }
[ \t]      { /* Skip spaces and tabs */ }
[\n]       { return 0; /* End of input */ }
.          { printf("Invalid character: %s\n", yytext); return 0; }
%%
int yywrap() {
   return 1;
}
```

Parser.y

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

%token NUMBER
%token PLUS MINUS MUL DIV LP RP

/* Operator precedence declarations */
%left PLUS MINUS  /* Lower precedence */
%left MUL DIV     /* Higher precedence */
%right UMINUS     /* Highest precedence (for unary minus) */

%%

input: /* empty */
    | expr    { printf("Result = %d\n", $1); }
    ;

expr: expr PLUS expr    { $$ = $1 + $3; }
   | expr MINUS expr   { $$ = $1 - $3; }
   | expr MUL expr     { $$ = $1 * $3; }
   | expr DIV expr     {
      if ($3 == 0) {
         yyerror("Division by zero");
         $$ = 0;
      } else {
         $$ = $1 / $3;
      }
    }
   | LP expr RP       { $$ = $2; }
   | MINUS expr %prec UMINUS  { $$ = -$2; } /* Unary minus */
   | NUMBER          { $$ = $1; }
   ;

%%

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

int main() {
   printf("Enter an arithmetic expression (with operator precedence grammar): ");
```

```
    yyparse();
    return 0;
}
```

Output:

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-17> cd With_Precedence
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-17\With_Precedence> bison -d parser.y
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-17\With_Precedence> flex lexer.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-17\With_Precedence> gcc lex.yy.c parser.tab.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-17\With_Precedence> ./a.exe
Enter an arithmetic expression (with operator precedence grammar): 2+3*4
Result = 14
```

18. Design YACC / LEX code that translates INFIX Expression to POSTFIX Expression.

Code :-

Lexer.l

```
%{
#include "parser.tab.h"
extern int yylval;
%}

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

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

Parser.y

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

%token NUM
%left '+' '-'
%left '*' '/'
%right NEGATIVE

%%
S: E {printf("\n"); return 0;}
  ;
E: E '+' E  {printf(" + ");}
  | E '*' E {printf(" * ");}
  | E '-' E {printf(" - ");}
  | E '/' E {printf(" / ");}
  | '(' E ')'
  | '-' E %prec NEGATIVE {printf(" -");}
  | NUM   {printf("%d", $1);}
  ;
%%

int main()
```

```
{
   printf("Input infix expression: ");
   yyparse();
   return 0;
}

void yyerror(const char *s)
{
   printf("\nError: %s\n", s);
}
```

Output :-

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-18> bison -d parser.y
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-18> flex Infix_To_Postfix_Translator.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-18> gcc lex.yy.c parser.tab.c
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-18> ./a.exe
Input infix expression: 5+6*9+2
569 *  + 2 +
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-18> ./a.exe
Input infix expression: 3-9/5
395 /  -
```

19. Design a Desk Calculator using YACC / LEX code.

Code: -

Lexer.y

```
%{
#include <stdlib.h>
#include <stdio.h>
int yylval;
#include "parser.tab.h"
%}
%%
[a-z]    { yylval = *yytext - 'a'; return id; }
[0-9]+   { yylval = atoi(yytext); return digit; }
[-+()=/*\n] { return *yytext; }
[ \t]    ; /* ignore whitespace */
.       { yyerror("invalid character"); }
%%
int yywrap(void)
{
   return 1;
}
```

Parser.y

```
%{
#include <stdio.h>
void yyerror(char *);
int yylex(void);
int sym[26];
%}
%token id digit
%left '+' '-'
%left '*' '/'
%%
P: P S '\n'
  | /* empty */
  ;
S: E     { printf("Output: %d\n", $1); }
  | id '=' E { sym[$1] = $3; }
  ;
E: digit  { $$ = $1; }
  | id   { $$ = sym[$1]; }
  | E '+' E { $$ = $1 + $3; }
  | E '-' E { $$ = $1 - $3; }
  | E '*' E { $$ = $1 * $3; }
  | E '/' E {
     if ($3 == 0) {
```

```
        yyerror("division by zero");
        $$ = 0;
      } else {
        $$ = $1 / $3;
      }
    }
  | '(' E ')' { $$ = $2; }
  ;
%%
void yyerror(char *s)
{
   fprintf(stderr, "Error: %s\n", s);
}
int main(void)
{
   printf("Enter Expression to Evaluate:\n");
   yyparse();
   return 0;
}
```

Output :-

```
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-19> bison -d parser.y
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-19> flex Desk_Calculator.l
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-19> gcc lex.yy.c parser.tab.c
Desk_Calculator.l:5:19: fatal error: y.tab.h: No such file or directory
 #include "y.tab.h"
                   ^
compilation terminated.
Desk_Calculator.l:12:3: warning: implicit declaration of function 'yyerror' [-Wimplicit-function-declaration]
 .         { yyerror("invalid character"); }
   ^~~~~~~
PS C:\Users\arjun\OneDrive\Desktop\Compiler-Design\Program-19> ./a.exe
Enter Expression to Evaluate:
7+6*9-15/3
Output: 56
a=5
b=5
c=3
d=a+b
e=a*b*c*d
e
Output: 750
2@13-14
Error: invalid character
Output: 2
Error: syntax error
```