PRACTICAL-1

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

Source Code :

```
%{
#include<stdio.h>
int l=0,s=0,t=0,c=0,o=0;
%}

%%
\n {l++;}
" " {s++;}
"\t" {t++;}
[A-Za-z0-9] {c++;}
. {o++;}
%%

int main()
{
   printf("Enter input:\n");
   yylex();
   printf("\n lines:%d\n spaces:%d\n tabs:%d\n characters:%d\n others:%d\n ",l,s,t,c,o);
   return 0;
}

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

OUTPUT

```
ubuntu $ cd Himanshu
ubuntu $ lex q1.l
ubuntu $ gcc lex.yy.c
ubuntu $ ./a.out
Enter input:
Hello! Welcome in this tutorial, where we will learn our first linux command - ls.

 lines:1
 spaces:14
 tabs:0
 characters:64
 others:4
```

PRACTICAL-2

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

Source Code :

```
%{
#include<stdio.h>
%}
%%
^[_]*[a-zA-Z][a-zA-Z0-9_]* { printf("Valid identifier\n");}
.* {printf("Invalid identifier\n");}
%%
int yywrap()
{
    return 1;
}
int main()
{
    printf("Enter I/P:\n");
    yylex();
    return 0;
}
```

OUTPUT

```
ubuntu $ cd Himanshu
ubuntu $ lex q2.l
ubuntu $ gcc lex.yy.c
ubuntu $ ./a.out
Enter I/P:
_123
Invalid identifier

_Arr
Valid identifier

Arr12
Valid identifier
```

PRACTICAL-3

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

Source Code :

```
%{
#include<stdio.h>
%}
%%
-?[0-9]+ {printf("Integer\n");}
-?[0-9]+[.][0-9]+ {printf("Float\n");}
.* {printf("NotANumber\n");}
%%
int yywrap()
{
    return 1;
}
int main()
{
    printf("Enter I/P:\n");
    yylex();
    return 0;
}
```

OUTPUT

```
ubuntu $ cd Himanshu
ubuntu $ lex q3.l
ubuntu $ gcc lex.yy.c
ubuntu $ ./a.out
Enter I/P:
1234
Integer

123.123
Float

3123.1231.23123.
NotANumber
```

PRACTICAL-4

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

Source Code :

```
%{
#include <stdio.h>
%}
%array
%{
char identifiers[100][256], keywords[100][256], operators[100][256], separators[100][256];
int counts[4] = {0};
%}
%%
"int"|"float"|"while"|"if"|"else"|"for" { strcpy(keywords[counts[1]++], yytext); }
[_a-zA-Z][_a-zA-Z0-9]* { strcpy(identifiers[counts[0]++], yytext); }
"="|"<="|"=="|"+"|"*"|"++" { strcpy(operators[counts[2]++], yytext); }
"{"|"}"|"("|")"|";"|"," { strcpy(separators[counts[3]++], yytext); }
.|\n {}
%%
int yywrap() { return 1; }
int main() {
   printf("Enter Input:\n");
   yylex();
   char *labels[4] = {"Identifiers", "Keywords", "Operators", "Separators"};
   for (int i = 0; i < 4; ++i) {
      printf("\n%s:\n", labels[i]);
      for (int j = 0; j < counts[i]; ++j)
         printf("%s\t", i == 0 ? identifiers[j] : i == 1 ? keywords[j] : i == 2 ? operators[j] :
separators[j]);
   }
   return 0;
}
```

OUTPUT

```
ubuntu $ cd Himanshu
ubuntu $ lex q4.l
ubuntu $ gcc lex.yy.c
ubuntu $ ./a.out
Enter Input:
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++,  }

Identifiers:
p       d       r       m       n       p       d       m       m       n       r       d       r       m       m       r       p
Keywords:
int     float   while   if      else
Operators:
=       =       =       =       =       <=      ==      =       +       *       +       ++      ++      =       +       +       ++
Separators:
,       ,_      ,       ,       ,       (       )       {       (       )       {       ,       ,       }       {       ,       ,       }       ,       }
```

PRACTICAL-5

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

Source Code :

```
%{

   #include<stdio.h>

   int charCount = 0;

   int wordCount = 0;

   int spaceCount = 0;

%}

%%

[^\t\n" "]+   { charCount += yyleng; wordCount++; }

" "     { spaceCount++; }

%%

int yywrap() {

   return 1;

}

int main() {

   yyin=fopen("input.txt","r");

   yylex();

   printf("Number of characters: %d\n", charCount);

   printf("Number of words: %d\n", wordCount);

   printf("Number of white spaces: %d\n", spaceCount);

   return 0;

}
```

OUTPUT

Input.txt

Himanshu > input.txt

```
Hello! Welcome in this tutorial, where we will learn our first linux command - ls.
Enjoy the course and be prepared for the quiz on the end!
Good luck!
```

Code Output

```
ubuntu $ cd Himanshu
ubuntu $ lex q5.l
ubuntu $ gcc lex.yy.c
ubuntu $ ./a.out



Number of characters: 123
Number of words: 29
Number of white spaces: 26
```

PRACTICAL-6

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

Source Code :

```
%{

#include<stdio.h>

%}

%%

[" "]+ {fprintf(yyout," ");}

.|\n {fprintf(yyout,"%s",yytext);}

%%

int yywrap()

{

    return 1;

}

int main()

{

    yyin=fopen("input.txt","r");

    yyout=fopen("output.txt","w");

    yylex();

    return 0;

}
```

OUTPUT

Input.txt

Himanshu > input.txt

```
Enjoy        the course      and be
             prepared  for the      quiz on the      end!
```

Output.txt

Himanshu > output.txt

```
Enjoy the course and be
 prepared for the quiz on the end!
```

PRACTICAL-7

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

Source Code:

```
%{

#include<stdio.h>

%}

%%

"//".*\n { ; }

"/*"([^*]|[*]+[^/])*"*/" { ; }

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

%%


int yywrap()

{

    return 1;

}


int main()

{

    yyin=fopen("input.c","r");

    yyout=fopen("output.c","w");

    yylex();

    return 0;

}
```

OUTPUT

Input.c

Himanshu > input.c

```
// Online C++ compiler to run C++ program online
#include <iostream>
/*
Hello! Welcome in this tutorial,
where we will learn our first linux command - ls.
Enjoy the course and be prepared for the quiz on the end!
Good luck!
*/
int main() {
    // Write C++ code here
    std::cout << "Try programiz.pro";

    return 0;
}
```

Output.c

Himanshu > output.c

```
#include <iostream>

int main() {
        std::cout << "Try programiz.pro";

    return 0;
}
```

PRACTICAL-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.

Source Code :

```
%{

#include<stdio.h>

%}



%%

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

.|\n {;}

%%



int yywrap()

{

    return 1;

}



int main()

{

    yyin=fopen(“input.html”,”r”);

    yyout=fopen("out.txt","w");

    yylex();

    return 0;

}
```

OUTPUT

Input.html

Himanshu > input.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <p>Hello Everyone</p>
</body>
</html>
```

Out.txt

Himanshu > out.txt

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>
</title>
</head>
<body>
<p></p>
</body>
</html>
```

PRACTICAL-9

Implementation of DFA accepting even number of a and b over input {a, b} with dead state

Source Code :

```
%{
#include<stdio.h>
int a=0,b=0,c=0,o=0;
void q0(){printf("%s q0 state \n",yytext);}
void q1(){printf("%s q1 state \n",yytext);}
void q2(){printf("%s q2 state \n",yytext);}
void q3(){printf("%s q3 state \n",yytext);}
void check(){
   if(o)printf("%s dead State\n",yytext);
   else if(!a && !b)q0();
   else if(a && !b)q1();
   else if(a && b)q2();
   else q3();
}
void answer(){
   if(!a && !b && !c)printf("\nString Accepted\n");
   else printf("\nString Not Accepted\n");
}
%}
%%
"a" {a^=1;check();}
"b" {b^=1;check();}
"\n" {answer();a=0;b=0;c=0;}
. {o=1;check();}
%%
int yywrap(){return 1;}
int main(){
   printf("Enter Inputs \n");
   yylex();return 0;}
```

OUTPUT

```
ubuntu $ cd Himanshu
ubuntu $ lex q9.l
ubuntu $ gcc lex.yy.c
ubuntu $ ./a.out
Enter Inputs
aabba
a q1 state
a q0 state
b q3 state
b q0 state
a q1 state

String Not Accepted
aabb
a q1 state
a q0 state
b q3 state
b q0 state

String Accepted
aabbaabb
a q1 state
a q0 state
b q3 state
b q0 state
a q1 state
a q0 state
b q3 state
b q0 state

String Accepted
```

PRACTICAL-10

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

Source Code :

```
%{
#include<stdio.h>
%}
%s A B C D E F G H
%%
<INITIAL>a BEGIN A;
<INITIAL>b BEGIN INITIAL;
<A>a BEGIN D;
<A>b BEGIN B;
<B>a BEGIN E;
<B>b BEGIN C;
<C>a BEGIN A;
<C>b BEGIN INITIAL;
<D>a BEGIN G;
<D>b BEGIN F;
<E>a BEGIN A;
<E>b BEGIN B;
<F>a BEGIN E;
<F>b BEGIN C;
<G>a BEGIN G;
<G>b BEGIN F;
<INITIAL>\n BEGIN INITIAL;printf("Not Accepted\n\n");
<A>\n BEGIN INITIAL; printf("\nNot Accepted\n\n");
<B>\n BEGIN INITIAL; printf("\nNot Accepted\n\n");
<C>\n BEGIN INITIAL; printf("\nAccepted\n\n");
<D>\n BEGIN INITIAL; printf("\nNot Accepted\n\n");
<E>\n BEGIN INITIAL; printf("\nAccepted\n\n");
<F>\n BEGIN INITIAL; printf("\nAccepted\n\n");
```

```
<G>\n BEGIN INITIAL; printf("\nAccepted\n\n");
<INITIAL>[^ab\n] BEGIN H;
<A>[^ab\n] BEGIN H;
<B>[^ab\n] BEGIN H;
<C>[^ab\n] BEGIN H;
<D>[^ab\n] BEGIN H;
<E>[^ab\n] BEGIN H;
<F>[^ab\n] BEGIN H;
<G>[^ab\n] BEGIN H;
<H>[^\n] BEGIN H;
<H>[\n] BEGIN INITIAL; printf("\nInvalid Input\n\n");
<H>EOF BEGIN INITIAL; printf("\nInvalid Input\n\n");
%%
int yywrap(){return 1;}

int main()
{
   printf("Enter the String of a and b only : \n");
   yylex();
}
```

OUTPUT

```
ubuntu $ cd Himanshu
ubuntu $ lex q10.l
ubuntu $ gcc lex.yy.c
ubuntu $ ./a.out
Enter the String of a and b only :
abababa

Accepted

aabaaabba

Not Accepted
```

PRACTICAL-11

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

Source Code :

```
%{
#include<stdio.h>
%}
%s A B C D Y Z
%%
<INITIAL>[A-Za-z_] BEGIN B;
<INITIAL>[0-9] BEGIN A;
<INITIAL>[.] BEGIN Y;
<INITIAL>[^A-Za-z0-9_.\n] BEGIN Z;
<INITIAL>\n BEGIN INITIAL;printf("  Not accepted\n ");
<A>[.] BEGIN C;
<A>[0-9] BEGIN A;
<A>[A-Za-z_] BEGIN Y;
<A>[^A-Za-z0-9_.\n] BEGIN Z;
<A>\n BEGIN INITIAL; printf( "Integer\n" );
<B>[A-Za-z_] BEGIN B;
<B>[0-9] BEGIN B;
<B>[.] BEGIN Y;
<B>[^A-Za-z0-9_.\n] BEGIN Z;
<B>\n BEGIN INITIAL; printf( "Identifier\n" );
<C>[0-9] BEGIN D;
<C>[.] BEGIN Y;
<C>[A-Za-z_] BEGIN Y;
<C>[^A-Za-z0-9_.\n] BEGIN Z;
<C>\n BEGIN INITIAL; printf( "  Not Accepted\n" );
<D>[0-9] BEGIN D;
<D>[.] BEGIN Y;
<D>[A-Za-z_] BEGIN Y;
<D>[^A-Za-z0-9_.\n] BEGIN Z;
```

```
<D>\n BEGIN INITIAL; printf( "Float\n" );
<Y>[A-Za-z0-9_.] BEGIN Y;
<Y>[^A-Za-z0-9_.\n] BEGIN Z;
<Y>[\n] BEGIN INITIAL; printf("  Not Accepted\n");
<Z>[^\n] BEGIN Z;
<Z>[\n] BEGIN INITIAL; printf("  Invalid Input\n");
%%
int yywrap(){return 1;}
int main()
{
   printf("Enter the char [A-Za-z0-9_.] only : \n ");
   yylex();
}
```

OUTPUT

```
ubuntu $ cd Himanshu
ubuntu $ lex q11.l
ubuntu $ gcc lex.yy.c
ubuntu $ ./a.out
Enter the char [A-Za-z0-9_.] only :
 _arra
Identifier
ashd_123
Identifier
12329
Integer
21341.123123
Float
@31
_ Invalid Input
```

PRACTICAL-12

Design YACC-LEX code for +, -, * and div of integers with precedence specification explicitly.

Source Code :

q12.l

```
%{
#include<stdio.h>
#include<stdlib.h>
#include"y.tab.h"
extern int yylval;
%}
%%
[0-9]+ {yylval = atoi(yytext);return digit;}
[-+*/\n] return *yytext;
. ;
%%
int yywrap(void){return 1;}
```

q12.y

```
%{
#include<stdio.h>
int yylex(void);
void yyerror(char *);
%}
%token digit
%left '+' '-'
%left '*' '/'
%%
S:S E'\n' {$$=$2; printf("\n\nOutput = %d\n\n",$$);}
 | ;
E:E '+' E {$$=$1+$3;}
 |E '-' E {$$=$1-$3;}
```

```
|E '*' E {$$=$1*$3;}
|E '/' E {$$=$1/$3;}
|digit   {$$ = $1;}
;
%%
int main()
{
   printf("Enter Arithmetic Expression : \n\n");
   yyparse();
   return 0;
}
void yyerror(char *msg)
{
   printf("\n\n%s", msg);
   printf("\n\nArithmetic Expression is Invalid\n\n");
}
```

OUTPUT

```
ubuntu $ cd Himanshu
ubuntu $ lex q12.l
ubuntu $ yacc -d q12.y
ubuntu $ gcc lex.yy.c y.tab.c -o a.out -ll
ubuntu $ ./a.out
Enter Arithmetic Expression :

1+53-12


Output = 42

1--3


syntax error

Arithmetic Expression is Invalid
```

PRACTICAL-13

Program 13 Yacc-Lex code for +, -, * and div of integers with precedence specified within CFG.

Source Code:

q13.l

```
%{
#include<stdlib.h>
int yylval;
#include "y.tab.h"
%}
%%
[0-9]+ {yylval = atoi(yytext); return digit;}
[-+*/\n]    return *yytext;
. ;
%%
int yywrap(void){return 1;}
```

q13.y

```
%{
#include<stdio.h>
int yylex(void);
void yyerror(char *);
%}
%token digit
%%
S:S E'\n' {$$=$2; printf("\n\nOutput = %d \n\n",$$);}
 | ;
E:E '+' T {$$=$1+$3;}
 |E '-' T {$$=$1-$3;}
 |T       {$$=$1;}
 ;
T:T '*' F {$$=$1*$3;}
 |T '/' F {$$=$1/$3;}
```

```
 |F      {$$=$1;}
 ;
F:digit   {$$=$1;}
%%
int main()
{
   printf("Enter Arithmetic Expressions : \n\n");
   yyparse();
   return 0;
}
void yyerror(char *msg)
{
   printf("\n\n%s", msg);
   printf("\n\n Invalid Arithmetic Expression\n\n");
}
```

OUTPUT

```
ubuntu $ cd ..
ubuntu $ cd Himanshu
ubuntu $ lex q13.l
ubuntu $ yacc -d q13.y
ubuntu $ gcc lex.yy.c y.tab.c -o a.out -ll
ubuntu $ ./a.out
Enter Arithmetic Expressions :

1+3*5-2


Output = 14

1**123


syntax error

 Invalid Arithmetic Expression
```

PRACTICAL-14

Design YACC/LEX code that translates infix expression to postfix expression.

SourceCode:

Q14.l

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

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

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

Q14.y

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

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

%%
S: E {printf("\n\n");}
```

```
  ;
E: E '+' E  {printf("+");}
  | E '*' E {printf("*");}
  | E '-' E {printf("-");}
  | E '/' E {printf("/");}
  | '(' E ')'
  | '-' E %prec NEGATIVE {printf("-");}
  | NUM   {printf("%d", yyval);}
  ;
%%

int main()
{
   printf("Input infix expression : \n\n");
   yyparse();
}

int yyerror(char *msg)
{
   return printf("\n\nError YACC : %s \n\n", msg);
}
```

OUTPUT

```
ubuntu $ cd Himanshu
ubuntu $ lex q14.l
ubuntu $ yacc -d q14.y
ubuntu $ gcc lex.yy.c y.tab.c -o a.out -ll
ubuntu $ ./a.out
Input infix expression :

1+3-5*10
13+510*-
```

PRACTICAL-15

Design Desk Calculator using YACC/LEX code.

Source Code:

Q15.l

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

Q15.y

```
%{
#include<stdio.h>
void yyerror(char *);
int yylex(void);
int sym[26];
%}
%token id digit
%left '+' '-'
%left '*' '/'
%%
P: P S '\n'
| ;
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 {$$ = $1 * $3; }
| E '/' E {$$ = $1 / $3; }
| '(' E ')' { $$ = $2; }
;
%%
void yyerror(char *s){ fprintf(stderr, "%s\n", s);}
int main(void)
{
   printf("Enter Expression to Evaluate : \n\n");
   yyparse();
   return 0;
}
```

OUTPUT

```
ubuntu $ cd Himanshu
ubuntu $ lex q15.l
ubuntu $ yacc -d q15.y
ubuntu $ gcc lex.yy.c y.tab.c -o a.out -ll

ubuntu $ ./a.out
Enter Expression to Evaluate :

1+4*10-8/2+3
Output : 40
```