Exp. No. 31

Implement Lexical Analyzer using FLEX (Fast Lexical Analyzer). The program should separate the tokens in the given C program and display with appropriate caption.

Input Source Program: (sample.c)

```
#include<stdio.h>

void main()
{
int a,b,c = 30;

printf("hello");
}
```

Program: (token.l)

```
digit [0-9]
letter [A-Za-z]
%{
int count_id,count_key;
%}
%%
(stdio.h|conio.h) { printf("%s is a standard library\n",yytext); }
(include|void|main|printf|int) { printf("%s is a keyword\n",yytext); count_key++; }
{letter}({letter}|{digit})*  { printf("%s is a identifier\n", yytext); count_id++; }
{digit}+  { printf("%s is a number\n", yytext); }
\"(\\.|[^"\\])*\"  { printf("%s is a string literal\n", yytext); }
.|\n {  }
%%
int yywrap(void) {
return 1;
}
int main(int argc, char *argv[]) {
yyin = fopen(argv[1], "r");
yylex();
printf("number of identifiers = %d\n", count_id);
printf("number of keywords = %d\n", count_key);
fclose(yyin);
}
```

Output:

```
G:\lex>flex token.l

G:\lex>gcc lex.yy.c

G:\lex>a.exe sample.c
include is a keyword
stdio.h is a standard library
void is a keyword
main is a keyword
int is a keyword
a is a identifier
b is a identifier
c is a identifier
30 is a number
printf is a keyword
"hello" is a string literal
number of identifiers = 3
number of keywords = 5

G:\lex>
```

Exp. No. 32

Write a LEX program to count the number of vowels in the given sentence.

Program: (vowels.l)

```
%{
   int vow_count=0;
   int const_count =0;
%}

%%
[aeiouAEIOU] {vow_count++;}
[a-zA-Z] {const_count++;}
%%
int yywrap(){}
int main()
{
   printf("Enter the string of vowels and consonants:");
   yylex();
   printf("Number of vowels are:  %d\n", vow_count);
   printf("Number of consonants are:  %d\n", const_count);
   return 0;
}
```

Output:

```
G:\lex>flex vowels.l

G:\lex>gcc lex.yy.c

G:\lex>a.exe
Enter the string of vowels and consonants: Vowel sounds allow the air to flow freely,
causing the chin to drop noticeably, whilst consonant sounds are produced by
restricting the air flow
      ,     ,
Number of vowels are:  42
Number of consonants are:  77
^C
G:\lex>
```

Exp. No. 33

Write a LEX program to count the number of vowels in the given sentence.

Program: (vowels.l)

```
%{
   int vow_count=0;
   int const_count =0;
%}

%%
[aeiouAEIOU] {vow_count++;}
[a-zA-Z] {const_count++;}
%%
int yywrap(){}
int main()
{
   printf("Enter the string of vowels and consonants:");
   yylex();
   printf("Number of vowels are:  %d\n", vow_count);
   printf("Number of consonants are:  %d\n", const_count);
   return 0;
}
```

Output:

```
G:\lex>flex vowels.l

G:\lex>gcc lex.yy.c

G:\lex>a.exe
Enter the string of vowels and consonants: Vowel sounds allow the air to flow freely,
causing the chin to drop noticeably, whilst consonant sounds are produced by
restricting the air flow
     ,    ,
Number of vowels are:  42
Number of consonants are:  77
^C
G:\lex>
```

Exp. No. 34
Write a LEX program to separate the keywords and identifiers.

Input Source Program: (sample.c)

```
#include<stdio.h>

void main()
{
int a,b,c = 30;

printf("hello");
}
```

Program: (token.l)

```
digit [0-9]
letter [A-Za-z]
%{
int count_id,count_key;
%}
%%
(stdio.h|conio.h) { printf("%s is a standard library\n",yytext); }
(include|void|main|printf|int) { printf("%s is a keyword\n",yytext); count_key++; }
{letter}({letter}|{digit})*  { printf("%s is a identifier\n", yytext); count_id++; }
{digit}+  { printf("%s is a number\n", yytext); }
\"(\\.|[^"\\])*\"  { printf("%s is a string literal\n", yytext); }
.|\n {  }
%%
int yywrap(void) {
return 1;
}
int main(int argc, char *argv[]) {
yyin = fopen(argv[1], "r");
yylex();
printf("number of identifiers = %d\n", count_id);
printf("number of keywords = %d\n", count_key);
fclose(yyin);
}
```

Output:

```
G:\lex>flex token.l
G:\lex>gcc lex.yy.c
G:\lex>a.exe sample.c
include is a keyword
```

```
stdio.h is a standard library
void is a keyword
main is a keyword
int is a keyword
a is a identifier
b is a identifier
c is a identifier
30 is a number
printf is a keyword
"hello" is a string literal
number of identifiers = 3
number of keywords = 5

G:\lex>
```

Exp. No. 35

Write a LEX program to recognise numbers and words in a statement.

Program: (numbers_words.l)

```
%%
[\t ]+ ;
[0-9]+|[0-9]*\.[0-9]+ { printf("\n%s is NUMBER", yytext);}
#.* { printf("\n%s is COMMENT", yytext);}
[a-zA-Z]+ { printf("\n%s is WORD", yytext);}
\n { ECHO;}
%%
int main()
{
        while( yylex());
}

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

Output:

```
G:\lex>flex numbers_words.l

G:\lex>gcc lex.yy.c

G:\lex>a.exe
Variables A and B contains 10 and 20 respectively

Variables is WORD
A is WORD
and is WORD
B is WORD
contains is WORD
10 is NUMBER
and is WORD
20 is NUMBER
respectively is WORD
```

Exp. No. 36

Write a LEX program to identify and count positive and negative numbers.

Program: (positive_neg_nums.l)

```
%{
int positive_no = 0, negative_no = 0;
%}
%%
^[-][0-9]+ {negative_no++;
                    printf("negative number = %s\n",
                          yytext);} // negative number

[0-9]+ {positive_no++;
              printf("positive number = %s\n",
                          yytext);} // positive number
%%
int yywrap(){}
int main()
{
yylex();
printf ("number of positive numbers = %d,"
             "number of negative numbers = %d\n",
                          positive_no, negative_no);
return 0;
}
```

Output:

```
G:\lex>flex positive_neg_nums.l
G:\lex>gcc lex.yy.c
G:\lex>a.exe
-10
negative number = -10
20
positive number = 20
number of positive numbers = 1,number of negative numbers = 1
G:\lex>
```

Exp. No. 37
Write a LEX program to validate the URL.

Program: (url.l)

```
%%
((http)|(ftp))s?:\/\/[a-zA-Z0-9](.[a-z])+(.[a-zA-Z0-9+=?]*)* {printf("\nURL Valid\n");}

.+ {printf("\nURL Invalid\n");}

%%
void main()
{
        printf("\nEnter URL : ");
        yylex();
        printf("\n");
}
int yywrap()
{
}
```

Output:

```
G:\lex>flex url.l

G:\lex>gcc lex.yy.c

G:\lex>a.exe

Enter URL : https:\\www.sse.in

URL Invalid

https://www.sse.in

URL Valid

G:\lex>
```

Exp. No. 38

Write a LEX program to validate DOB of students.

Program: (dob.l)

```
%%
((0[1-9])|([1-2][0-9])|(3[0-1]))\/((0[1-9])|(1[0-2]))\/(19[0-9]{2}|2[0-9]{3})
printf("Valid DoB");
.* printf("Invalid DoB");
%%

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

Output:

```
G:\lex>flex dob.l

G:\lex>gcc lex.yy.c

G:\lex>a.exe
26/07/1995
Valid DoB



13\2\96
Invalid DoB

G:\lex>
```

Exp. No. 39
Write a LEX program to check whether the given input is digit or not.

Program: (digit_or_not.l)

```
%%
[0-9]+ {printf("\nValid digit \n");}
.* printf("\nInvalid digit\n");
%%
int yywrap(){}
int main()
{
yylex();
return 0;
}
```

Output:

```
G:\lex>flex digit_or_not.l

G:\lex>gcc lex.yy.c

G:\lex>a.exe
23
Valid digit

h56
Invalid digit

G:\lex>
```

Exp. No. 40
Write a LEX program to implement basic mathematical operations.

Program: (cal.l)

```
%{
#undef yywrap
#define yywrap() 1
int f1=0,f2=0;
char oper;
float op1=0,op2=0,ans=0;
void eval();
%}

DIGIT [0-9]
NUM {DIGIT}+(\.{DIGIT}+)?
OP [*/+-]

%%

{NUM} {
        if(f1==0)
        {
                op1=atof(yytext);
                f1=1;
        }

        else if(f2==-1)
        {
                op2=atof(yytext);
                f2=1;
        }

        if((f1==1) && (f2==1))
        {
                eval();
                f1=0;
                f2=0;

        }
}
```

```
{OP} {

        oper=(char) *yytext;
        f2=-1;
}

[\n] {

        if(f1==1 && f2==1)
        {
                eval;
                f1=0;
                f2=0;
        }
}

%%


int main()
{
        yylex();
}


void eval()
{
        switch(oper)
        {
                case '+':
                        ans=op1+op2;
                        break;

                case '-':
                        ans=op1-op2;
                        break;

                case '*':
                        ans=op1*op2;
```

```
            break;

        case '/':
            if(op2==0)
            {
                printf("ERROR");
                return;
            }
            else
            {
                ans=op1/op2;
            }
            break;
        default:
            printf("operation not available");
            break;
    }
    printf("The answer is = %lf",ans);
}
```

Output:

```
G:\lex>flex cal.l

G:\lex>gcc lex.yy.c

G:\lex>a.exe
20 + 30
 The answer is = 50.000000
25 * 5
 The answer is = 125.000000

G:\lex>
```