1.without whitespaces,identifying tokens

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
void lexicalAnalyzer(char *code) {
for (int i = 0; code[i]; i++) {
if (isspace(code[i])) continue;
if (code[i] == '/' && code[i + 1] == '/') while (code[i] && code[i] != '\n') i++;
else if (code[i] == '/' && code[i + 1] == '*') while (code[i] && !(code[i] == '*' &&
code[i + 1] == '/')) i++;
else if (isalpha(code[i]) || code[i] == '_') { printf("IDENTIFIER: "); while
(isalnum(code[i]) || code[i] == '_') putchar(code[i++]); printf("\n"); i--; }
else if (isdigit(code[i])) { printf("CONSTANT: "); while (isdigit(code[i]))
putchar(code[i++]); printf("\n"); i--; }
else if (strchr("+-*/=<>", code[i])) { printf("OPERATOR: %c\n", code[i]); }
}
}
int main() {
char code[1000];
fgets(code, sizeof(code), stdin);
lexicalAnalyzer(code);
return 0;
}
```

2.checking comment

```
#include <stdio.h>
#include <string.h>
void check_comment(char *line) {
if (strncmp(line, "//", 2) == 0)
printf("Single-line comment\n");
else if (strncmp(line, "/*", 2) == 0 && strstr(line, "*/") != NULL)
printf("Multi-line comment\n");
else
```

```
printf("Not a comment\n");
}
int main() {
char line[100];
fgets(line, sizeof(line), stdin);
check_comment(line);
return 0;
}
```

3.Valid operator

```
#include <stdio.h>
void check_operator(char ch) {
if (ch == '+' || ch == '-' || ch == '*' || ch == '/')
printf("Valid Operator: %c\n", ch);
else
printf("Invalid Operator\n");
}
int main() {
char ch;
scanf(" %c", &ch);
check_operator(ch);
return 0;
}
```

4.whitespaces, newlines

```
#include <stdio.h>
#include <string.h>
void count_whitespace_newlines(char *str) {
int spaces = 0, newlines = 0;
for (int i = 0; str[i] != '\0'; i++) {
if (str[i] == ' ' || str[i] == '\t')
spaces++;
else if (str[i] == '\n')
newlines++;
```

```
}

printf("Whitespaces: %d\n", spaces);
printf("Newlines: %d\n", newlines);
}
int main() {
char str[200];
fgets(str, sizeof(str), stdin);
count_whitespace_newlines(str);
return 0;
}
```

5.Valid identifier

```
#include <stdio.h>
#include <ctype.h>
int is_valid_identifier(char *str) {
    if (!isalpha(*str) && *str != '_') return 0;
    while (*++str) if (!isalnum(*str) && *str != '_') return 0;
    return 1;
}
int main() {
    char str[100];
    scanf("%s", str);
    printf("%s\n", is_valid_identifier(str) ? "Valid Identifier" : "Invalid Identifier");
    return 0;
}
```

6.eliminating left recursion

```
#include <stdio.h>
void eliminateLeftRecursion(char nt, char alpha[], char beta[]) {
    printf("%c -> %s%c'\n", nt, beta, nt + 1);
    printf("%c' -> %s%c' | ε\n", nt + 1, alpha, nt + 1);
}
int main() {
```

```
    printf("Original Grammar:\nA -> Aa | b\n\nTransformed Grammar:\n");
    eliminateLeftRecursion('A', "a", "b");
    return 0;
}
```

7.**Eliminating left factoring**

```
#include <stdio.h>
void eliminateLeftFactoring(char nt, char common[], char alpha[], char beta[]) {
    printf("%c -> %s%c'\n", nt, common, nt + 1);
    printf("%c' -> %s | %s\n", nt + 1, alpha, beta);
}
int main() {
    printf("Original Grammar:\nA -> xy | xz\n\nTransformed Grammar:\n");
    eliminateLeftFactoring('A', "x", "y", "z");
    return 0;
}
```

8.Symbol Table

```
#include <stdio.h>
#include <string.h>
#define MAX 10
struct Symbol {
char name[20];
char type[10];
int address;
} table[MAX];
int count = 0;
void insert(char name[], char type[], int address) {
strcpy(table[count].name, name);
strcpy(table[count].type, type);
table[count].address = address;
count++;
}
```

```
void display() {
printf("Symbol Table:\n");
printf("Name Type Address\n");
for (int i = 0; i < count; i++) {
printf("%-10s %-10s %-10d\n", table[i].name, table[i].type, table[i].address);
}
}
int main() {
insert("x", "int", 1000);
insert("y", "float", 1004);
insert("z", "char", 1008);
display();
return 0;
}
```

9.sentence follows the grammar

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int isValidSentence(const char *str) {
    int len = strlen(str);
    if (len == 0 || !isupper(str[0]) || str[len - 1] != '.') return 0;
    for (int i = 1; i < len - 1; i++)
        if (!isalpha(str[i]) && str[i] != ' ') return 0;
    return 1;
}
int main() {
    char input[100];
    printf("Enter a sentence: ");
    fgets(input, sizeof(input), stdin);
    input[strcspn(input, "\n")] = '\0'; // Remove newline

    printf("The given sentence %s the grammar rules.\n",
```

```
        isValidSentence(input) ? "follows" : "does NOT follow");
    return 0;
}
```

10.recursive descent parsing

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
char input[100];
int pos = 0;
void E(), T(), F();
void error() { printf("Syntax Error\n"); exit(1); }
void match(char c) { if (input[pos] == c) pos++; else error(); }
void E() { T(); if (input[pos] == '+') match('+'), E(); }
void T() { F(); if (input[pos] == '*') match('*'), T(); }
void F() { isalpha(input[pos]) ? match(input[pos]) : (input[pos] == '(' ? (match('('), E(), match(')')) :
error()); }
int main() {
    printf("Enter an expression: ");
    scanf("%s", input);
    E();
    printf(input[pos] ? "Syntax Error\n" : "Valid Expression\n");
    return 0;
}
```

11. In a class of Grade 3, Mathematics Teacher asked for the Acronym PEMDAS?. All of them are thinking for a while. A smart kid of the class Kishore of the class says it is Parentheses, Exponentiation, Multiplication, Division, Addition, Subtraction. Can you write a C Program to help the students to understand about the operator precedence parsing for an expression containing more than one operator, the order of evaluation depends on the order of operations.

Code:

```
#include <stdio.h> int main() {     int a = 5, b = 2, c = 3, d = 4;     int result;     printf("Given
Expression: %d + %d * %d - %d / %d\n", a, b, c, d, b);     int step1 = b * c;        printf("Step 1: %d *
%d = %d\n", b, c, step1);


    int step2 = d / b;        printf("Step 2: %d / %d = %d\n", d, b, step2);
```

```
    int step3 = a + step1;      printf("Step 3: %d + %d = %d\n", a, step1, step3);     result = step3 - step2;
printf("Step 4: %d - %d = %d\n", step3, step2, result);     printf("Final Result: %d\n", result);     return
0;
}
```

12.Three address code representation

```
#include <stdio.h>
int temp = 1;
void gen_TAC(char op, char a, char b, char res) {
    printf("%c = %c %c %c\n", res, a, op, b);
}
int main() {
    char t1 = 't', t2 = 't' + 1, t3 = 't' + 2;
    gen_TAC('*', 'b', 'c', t1);  // t1 = b * c
    gen_TAC('+', 'a', t1, t2);   // t2 = a + t1
    return 0;
}
```

13.count no.of characters,words,lines

```
#include <stdio.h>
void analyze(const char *s) {
    int chars = 0, words = 0, lines = 1, inWord = 0;
    while (*s) {
        chars++;
        if (*s == '\n') lines++;
        if (*s == ' ' || *s == '\t' || *s == '\n') inWord = 0;
        else if (!inWord) { words++; inWord = 1; }
        s++;
    }
    printf("Characters: %d\nWords: %d\nLines: %d\n", chars, words, lines);
}
int main() {
    char text[] = "Hello World\nThis is a test\nLexical Analyzer!";
    analyze(text);
```

```
    return 0;
}
```

14.eliminate common subexpression

```
#include <stdio.h>
void optimize() {
    int a = 5, b = 3, c = 2, d = 4;
    int t1 = a + b;  // Common subexpression
    int t2 = t1 * c;
    int t3 = t1 * d;
    printf("t1 = %d\n", t1);
    printf("t2 = %d\n", t2);
    printf("t3 = %d\n", t3);
}
int main() {
    optimize();
    return 0;
}
```

15.back end of the compiler

```
#include <stdio.h> #include <string.h>
void generateAssembly(char op, char arg1[], char arg2[], char result[]) {     switch (op) {         case '+':
printf("MOV R1, %s\nADD R1, %s\nMOV %s, R1\n", arg1, arg2, result); break;         case '-':
printf("MOV R1, %s\nSUB R1, %s\nMOV %s, R1\n", arg1, arg2, result); break;         case '*':
printf("MOV R1, %s\nMUL R1, %s\nMOV %s, R1\n", arg1, arg2, result); break;         case '/':
printf("MOV R1, %s\nDIV R1, %s\nMOV %s, R1\n", arg1, arg2, result); break;         default:
printf("Invalid Operation\n");
    }
}


int main() {     char result[] = "T1";     char arg1[] = "b";     char arg2[] = "c";     char op = '+';


    printf("Three Address Code (TAC):\n");     printf("%s = %s %c %s\n", result, arg1, op, arg2);
printf("\nGenerated Assembly Code:\n");     generateAssembly(op, arg1, arg2, result);     return 0;
}
```

1.Accept string starting with vowel

```
%{
#include <stdio.h>
%}
%%
^[AEIOUaeiou][a-zA-Z]* { printf("Valid String: %s\n", yytext); }
.* { printf("Invalid String: %s\n", yytext); }
%%
int main() {     printf("Enter a string (Ctrl+D to stop):\n");     yylex();     return 0;
} int yywrap() {     return 1;
}
```

2.Date of birth of students

```
%{
#include <stdio.h>
%}
%%
((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: %s\n", yytext);
} .* {     printf("Invalid DoB: %s\n", yytext);
}
%%
int main() {     printf("Enter DOB (DD/MM/YYYY) to validate (Ctrl+D to stop):\n");
   yylex();     return 0; } int yywrap() {     return 1;
}
```

3.Email

```
%{
#include <stdio.h>
#include <string.h>
%}
%%
[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,} { printf("Valid email: %s\n", yytext); }
. ;
```

```
%%
int main() {  printf("Enter email address: \n");  yylex();  return 0;  }  int yywrap() {  return 1;
}
```

Count positive and negative numbers

```
%{
#include <stdio.h>
int pos_count = 0, neg_count = 0;
%}
%%
[+-]?[0-9]+(\.[0-9]+)? {     if(yytext[0] == '-')          neg_count++;
   else          pos_count++;
}
\n  { printf("Positive Numbers: %d\nNegative Numbers: %d\n", pos_count, neg_count); }
.   ;
%%
int main() {     printf("Enter numbers (Ctrl+D to stop):\n");     yylex();     return 0;
} int yywrap() {     return 1;
}
```

4.URL

```
%%
((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()
{
}
```

5. The lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Write a LEX specification file to take input C program from a .c file and count the number of characters, number of lines & number of words.

```
%{

int nchar, nword, nline;

%}

%%

\n { nline++; nchar++; }

[^ \t\n]+ { nword++, nchar += yyleng; }

. { nchar++; }

%%

int yywrap(void) {

return 1; }

int main(int argc, char *argv[]) { yyin = fopen(argv[1], "r"); yylex(); printf("Number of characters = %d\n", nchar); printf("Number of words = %d\n", nword); printf("Number of lines = %d\n", nline); fclose(yyin);

}
```

6.No.of macros, header count

Code (Lex):

```
%{

#include<stdio.h> #include<stdlib.h>

int macro_count = 0, header_count = 0;

%}

macro    \#define[ ]+[a-zA-Z_][a-zA-Z0-9_]* header   \#include[ ]+<[^>]+>

%%

{macro}   { macro_count++; } {header}   { header_count++; }

.         { /* Ignore other characters */ }

%%

int main(int argc, char *argv[]) {     if(argc != 2) {         printf("Usage: %s <filename>\n", argv[0]);

       return 1;

   }

   FILE *file = fopen(argv[1], "r");

   if (!file) {         printf("Error opening file: %s\n", argv[1]);         return 1;     }     yyin = file; yylex();     fclose(file);

   printf("Number of Macros: %d\n", macro_count);     printf("Number of Header files: %d\n", header_count);     return 0; } int yywrap() {     return 1;

}
```

7.Constants

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
digit     [0-9] number     {digit}+     floatnum   {digit}+\.({digit}+)?
string    \"([^\"]|\\\")*\"
%%
{number}    { printf("Integer constant: %s\n", yytext); }
{floatnum}  { printf("Floating-point constant: %s\n", yytext); }
{string}    { printf("String constant: %s\n", yytext); }
.           { /* Ignore other characters */ }
%%
int main(int argc, char *argv[]) {     if(argc != 2) {          printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }
    FILE *file = fopen(argv[1], "r");
    if (!file) {          printf("Error opening file: %s\n", argv[1]);          return 1;      }      yyin = file;
yylex();     fclose(file);     return 0;
}

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

8.Html tags

```
%{
int tags;
%}
%%
"<"[^>]*>  { tags++; printf("%s \n", yytext); }
.|\n { }
%%
int yywrap(void) {  return 1; }  int main(void)
```

```
{
FILE *f;
char file[10];
printf("Enter File Name : ");  scanf("%s",file);  f = fopen(file,"r");  yyin = f;  yylex();  printf("\n Number of html tags: %d",tags);  fclose(yyin);
}
```

9.adds line numbers to the given C program file and display the same in the standard output.

```
int yylineno;
%}
%%
^(.*)\n printf("%4d\t%s", ++yylineno, yytext);
%%
int yywrap(void) {  return 1;  }
int main(int argc, char *argv[]) {  yyin = fopen(argv[1], "r");  yylex();  fclose(yyin);
}
```

10.Iput is digit or not

```
%{
#include <stdio.h>
%}
%%
[0-9]+ { printf("\nValid digit\n"); }
.* { printf("\nInvalid digit\n"); }
%%
int yywrap() {      return 1;
} int main() {      printf("Enter input (Ctrl+D to stop):\n");      yylex();      return 0;
}
```

11.Basic mathematical operations

```
%{
#include <stdio.h>
#include <stdlib.h>
#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 {          op2 = atof(yytext);          f2 =
1;

   }

}

{OP} {      oper = yytext[0];

} \n {      if (f1 && f2) {          eval();          printf("Result: %.2f\n", ans);          f1 = f2 = 0; // Reset for
next input

   }

}

%%

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

       case '/':

          if (op2 != 0)                ans = op1 / op2;             else                printf("Error: Division by
zero\n");             break;         default:

          printf("Invalid operator\n");

   }

} int main() {     printf("Enter arithmetic expression (e.g., 5 + 3). Press Enter to evaluate:\n");
yylex();     return 0;

}
```

12.Length of the longest word

```
%{

#include <stdio.h>

#include <string.h>

int max_length = 0;

char longest_word[100]; // Assuming words won't exceed 100 characters

%}

%%
```

```
[a-zA-Z]+ {
   int len = strlen(yytext);    if (len
> max_length) {       max_length
= len;       strcpy(longest_word,
yytext);
   }
}
[^a-zA-Z]+ { /* Ignore non-word characters */ }
%%
int main() {    printf("Enter text (Ctrl+D to stop):\n");    yylex();
printf("Longest Word: %s (Length: %d)\n", longest_word, max_length);
   return 0;
} int yywrap()
{    return 1;
}
```

13.**program to count the frequency of the given word in a given sentence**

```
%{
#include <stdio.h> #include <string.h> int count = 0;
char target[100];  // Word to search for
%}
%%
[a-zA-Z]+ {     if (strcmp(yytext, target) == 0) {         count++;
   }
}
.|\n { /* Ignore other characters */ }
%%
int main() {    printf("Enter the word to search: ");    scanf("%s", target);       printf("Enter the
sentence (Ctrl+D to stop):\n");    yylex();       printf("The word '%s' appears %d times.\n", target,
count);    return 0;
} int yywrap() {    return 1;
}
```

14.to replace a word with another word in a file.

```
%{

#include <stdio.h> #include <string.h>

char old_word[100], new_word[100]; // Words for replacement

%}

%%

[a-zA-Z]+ {     if (strcmp(yytext, old_word) == 0) {         printf("%s", new_word);  // Replace old word with new word

   } else {        printf("%s", yytext); // Print the word as it is

   }

}

. { printf("%c", yytext[0]); } // Print other characters (punctuation, spaces, etc.)

%%

int main() {     printf("Enter the word to be replaced: ");     scanf("%s", old_word);     printf("Enter the new word: ");     scanf("%s", new_word);     printf("Enter the file content (Ctrl+D to stop):\n"); yylex(); // Process the file     return 0;

} int yywrap() {     return 1;

}
```

15.to recognize a word and relational operator

```
%{

#include <stdio.h>

%}

%%

[a-zA-Z_][a-zA-Z0-9_]* { printf("Word: %s\n", yytext); }

(<=|>=|==|!=|<|>) { printf("Relational Operator: %s\n", yytext); }

[ \t\n] { /* Ignore whitespace */ }

. { printf("Other: %s\n", yytext); } // Print other symbols if needed

%%

int main() {     printf("Enter input (Ctrl+D to stop):\n");     yylex();     return 0;

} int yywrap() {     return 1;

}
```

16.No.of comment lines

```
%{   int com=0;
%}
%s COMMENT
%%
"/*" {BEGIN COMMENT;}
<COMMENT>"*/" {BEGIN 0; com++;}
<COMMENT>\n {com++;}
<COMMENT>. {;}
\/\/.* {; com++;}
.|\n {fprintf(yyout,"%s",yytext);}
%%
void main(int argc, char *argv[])
{   if(argc!=3)
{   printf("usage : a.exe input.c output.c\n");   exit(0);
}
yyin=fopen(argv[1],"r");   yyout=fopen(argv[2],"w");   yylex();   printf("\n number of comments are = %d\n",com);
}   int yywrap()
{   return 1;
}
```

17.to identify the capital words from the given input.

```
%%
;
[A-Z]+[\t\n ] { printf("%s is a capital word\n",yytext); }
.
%%
int main( )
{   printf("Enter String :\n");   yylex();   }   int yywrap( )
{   return 1;
}
```

18.to convert the substring abc to ABC from the given input string.

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

```
%}
%%
abc { printf("ABC"); }
.   { printf("%s", yytext); }
%%
int main() {  printf("Enter text: \n");  yylex();  return 0;  }  int yywrap() {  return 1;
}
```

19. The Company ABC runs with employees with several departments. The Organization manager had all the mobile numbers of employees. Assume that you are the manager and need to verify the valid mobile numbers because there may be some invalid numbers present.

Implement a LEX program to check whether the mobile number is valid or not.

Code (Lex):

```
%%
[1-9][0-9]{9} {printf("\nMobile Number Valid\n");}
.+ {printf("\nMobile Number Invalid\n");}
%%
int main()
{  printf("\nEnter Mobile Number : ");  yylex();  printf("\n");
return 0;  }  int yywrap()
{ }
```

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

Input Source Program: (sample7.c)

```
#include<stdio.h>   void main()
{
int a,b,c = 30;   printf("hello");
}
```

Code (Lex):

```
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);

}
```

21. In a class, an English teacher was teaching the vowels and consonants to the students. She says "Vowel sounds allow the air to flow freely, causing the chin to drop noticeably, whilst consonant sounds are produced by restricting the air flow". As a class activity the students are asked to identify the vowels and consonants in the given word/sentence and count the number of elements in each. Write an algorithm to help the student to count the number of vowels and consonants in the given sentence.

Code (Lex):

```
%{      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;

}
```

22. Keywords are predefined, reserved words used in programming that have special meanings to the compiler. Keywords are part of the syntax and they cannot be used as an identifier. In general there are 32 keywords. The prime function of Lexical Analyser is token Generation. Among the 6 types of tokens, differentiating Keyword and Identifier is a challenging issue. Thus write a LEX program to separate keywords and identifiers

```
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);

}
```

23. Write a LEX program to recognise numbers and words in a statement. Pooja is a small girl of age 3 always fond of games. Due to the pandemic, she was not allowed to play outside. So her mother designs a gaming event by showing a flash card. Pooja has to separate the numbers in one list and words in another list shown in the flash card.

Code (Lex):

```
%{

#include <stdio.h>

#include <stdlib.h> #include <ctype.h> int words = 0, numbers = 0;

%}

%%

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

}

[a-zA-Z]+ {      printf("WORD: %s\n", yytext);     words++;

}

[ \t\n] ;  /* Ignore whitespace */

. ;  /* Ignore other characters */

%%

int main() {     printf("Enter a statement:\n");     yylex();     printf("\nTotal Words: %d\n", words);
printf("Total Numbers: %d\n", numbers);

   return 0; } int yywrap() {     return 1;

}
```

24.positive count,negative count

```
%{
#include <stdio.h>
int pos_count = 0, neg_count = 0;
%}
%%
[+-]?[0-9]+(\.[0-9]+)? {     if(yytext[0] == '-')          neg_count++;
   else          pos_count++;
}
\n  { printf("Positive Numbers: %d\nNegative Numbers: %d\n", pos_count, neg_count); }
.   ;
%%
int main() {     printf("Enter numbers (Ctrl+D to stop):\n");     yylex();     return 0;
} int yywrap() {     return 1;
}
```