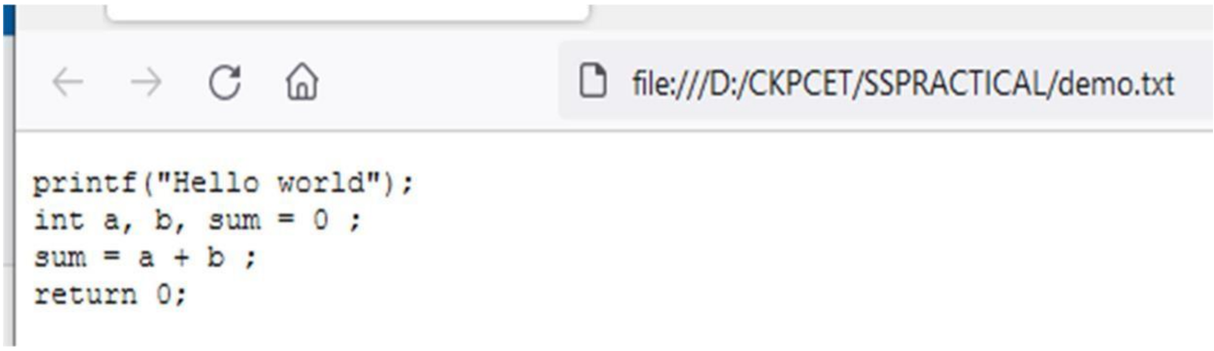
## PRACTICAL: 1

# Write a program to implement a lexical analyzer for the ‘C’ language.

## CODE:

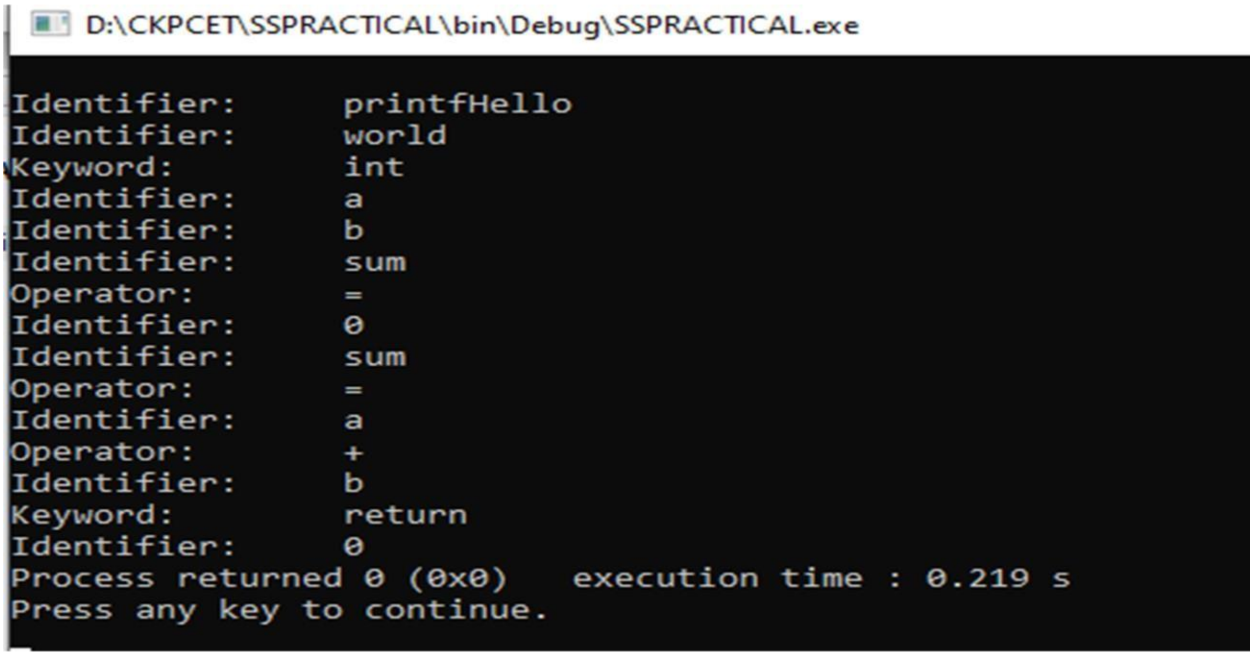
```
#include  <stdio.h>
#include <conio.h>
#include  <ctype.h>
#include <string.h>
#include <stdlib.h>
int keyword_library(char temp[]);
int main()
{
   char ch, temp[40], operators[] = "=+%*/-";
   FILE *fp;
   int count, x = 0;
   fp= fopen("D:/CKPCET/SSPRACTICAL/demo.txt", "r");
   if (fp == NULL)
   {
      printf("The file could not be opened.\n");
      exit(0);
   }
   while ((ch = fgetc(fp)) != EOF)
   {
      count = 0;
      while (count <= 5)
      {
         if (ch == operators[count])
         {
            printf("\nOperator:\t%c", ch);
         }
         count = count + 1;
      }
      if (isalnum(ch))
      {
         temp[x++] = ch;
      }
      else if ((ch == '\n' || ch == ' ') && (x != 0))
      {
         temp[x] = '\0';
         x = 0;
         if (keyword_library(temp) == 1)
         {
                printf("\nKeyword:\t%s", temp);
            }
            else
            {
                printf("\nIdentifier:\t%s", temp);
            }
        }
    }
    fclose(fp);
    return 0;
}
int keyword_library(char temp[])
{
    int count = 0, flag = 0;
    char keywords[14][10] = {"return", "continue","switch", "char", "else", "if", "while",
"float", "double", "for",
                            "break", "void", "int", "do"};
    while (count <= 13)
    {
        if (strcmp(keywords[count], temp) == 0)
        {
            flag = 1;
            break;
        }
        count = count + 1;
}
return (flag);
}
```

**OUTPUT:**



file:///D:/CKPCET/SSPRACTICAL/demo.txt

```
printf("Hello world");
int a, b, sum = 0 ;
sum = a + b ;
return 0;
```



D:\CKPCET\SSPRACTICAL\bin\Debug\SSPRACTICAL.exe

```
Identifier:     printfHello
Identifier:     world
Keyword:        int
Identifier:     a
Identifier:     b
Identifier:     sum
Operator:       =
Identifier:     0
Identifier:     sum
Operator:       =
Identifier:     a
Operator:       +
Identifier:     b
Keyword:        return
Identifier:     0
Process returned 0 (0x0)   execution time : 0.219 s
Press any key to continue.
```

## PRACTICAL: 2

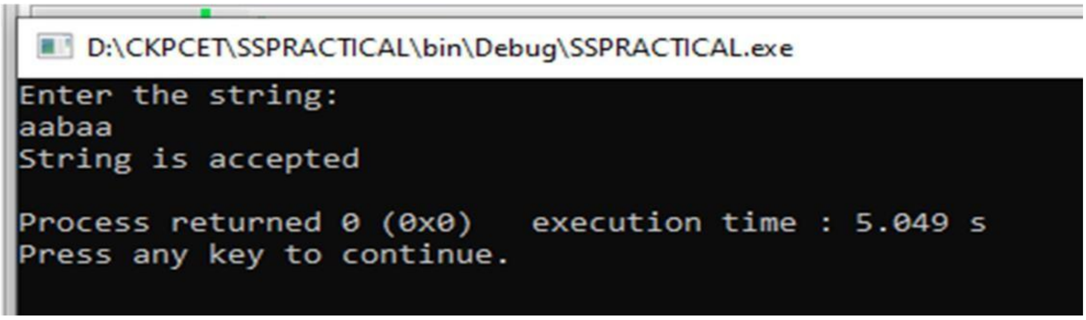
### Write a program to check the validity of the input string for a fixed Finite Automata.

**CODE:**

```
//DFA for regular expression (a+aab*)*
#include<stdio.h>
#include<string.h>
int main()
{
   char input[100];
   int len,i,status_a=0,status_b=0;
   printf("Enter the string: \n");
   scanf("%s",input);
   len=strlen(input);
   for(i=0;i<len;i++)
   {
      if(input[i]!='a'&&input[i]!='b')
      {
         printf("you enter wrong input\n");
         break;
      }
      else
      {
         if(input[i]=='a')
         {
            status_a=1;
            status_b=0;
         }
         else
         {
            if(status_b==1 || status_a==0)
            {
               printf("String is not accepted\n");
               break;
            }
            else
            {
               status_b=1;
               status_a=0;
            }
            }
        }
    if(i==len-1)
    {
        printf("String is accepted\n");
    }
    }
    return 0;
}
```

**OUTPUT:**

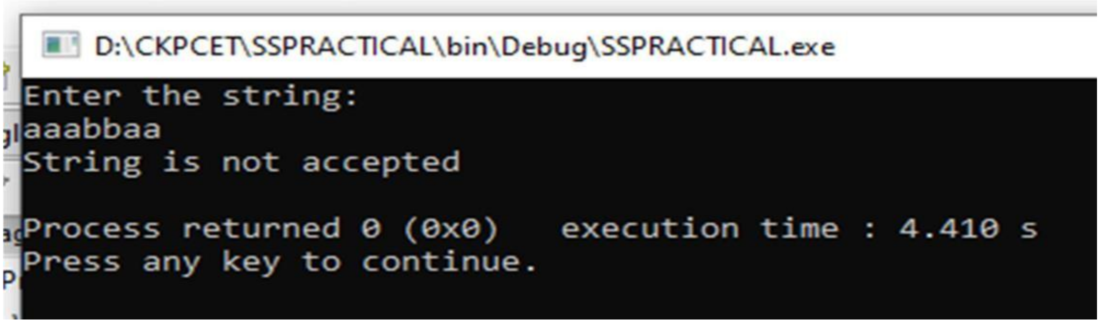
//DFA for regular expression (a+aab*)*



D:\CKPCET\SSPRACTICAL\bin\Debug\SSPRACTICAL.exe

```
Enter the string:
aabaa
String is accepted

Process returned 0 (0x0)   execution time : 5.049 s
Press any key to continue.
```



D:\CKPCET\SSPRACTICAL\bin\Debug\SSPRACTICAL.exe

```
Enter the string:
aaabbaa
String is not accepted

Process returned 0 (0x0)   execution time : 4.410 s
Press any key to continue.
```

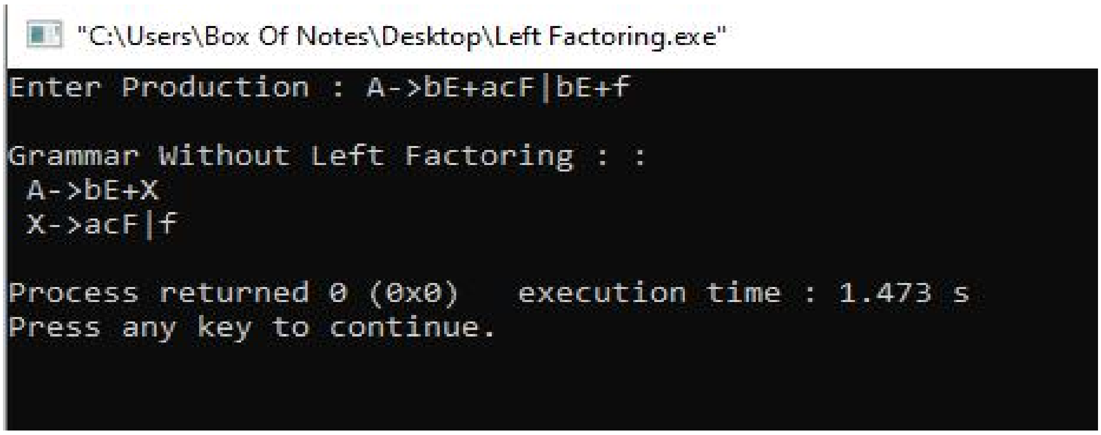
## PRACTICAL : 3

# Write a program to left factor the given grammar.

## CODE:

```
#include <stdio.h>
#include <string.h>
int main()
{
   char gram[20], part1[20], part2[20], modifiedGram[20], newGram[20],
tempGram[20];
   int i, j = 0, k = 0, l = 0, pos;
   printf("Enter Production : A->");
   gets(gram);
   for (i = 0; gram[i] != '|'; i++, j++)
      part1[j] = gram[i];
   part1[j] = '\0';
   for (j = ++i, i = 0; gram[j] != '\0'; j++, i++)
      part2[i] = gram[j];
   part2[i] = '\0';
   for (i = 0; i < strlen(part1) || i < strlen(part2); i++)
   {
      if (part1[i] == part2[i])
      {
         modifiedGram[k] =  part1[i];
         k++;
         pos = i + 1;
      }
   }
   for (i = pos, j = 0; part1[i] != '\0'; i++, j++)
   {
      newGram[j] = part1[i];
   }
   newGram[j++] = '|';
   for (i = pos; part2[i] != '\0'; i++, j++)
   {
      newGram[j] = part2[i];
   }
   modifiedGram[k] = 'X';
   modifiedGram[++k] = '\0';
   newGram[j] = '\0';
   printf("\nGrammar Without Left Factoring : : \n");
   printf(" A->%s", modifiedGram);
   printf("\n  X->%s\n",  newGram);
}
```

## OUTPUT:



"C:\Users\Box Of Notes\Desktop\Left Factoring.exe"

```
Enter Production : A->bE+acF|bE+f

Grammar Without Left Factoring : :
 A->bE+X
 X->acF|f

Process returned 0 (0x0)   execution time : 1.473 s
Press any key to continue.
```

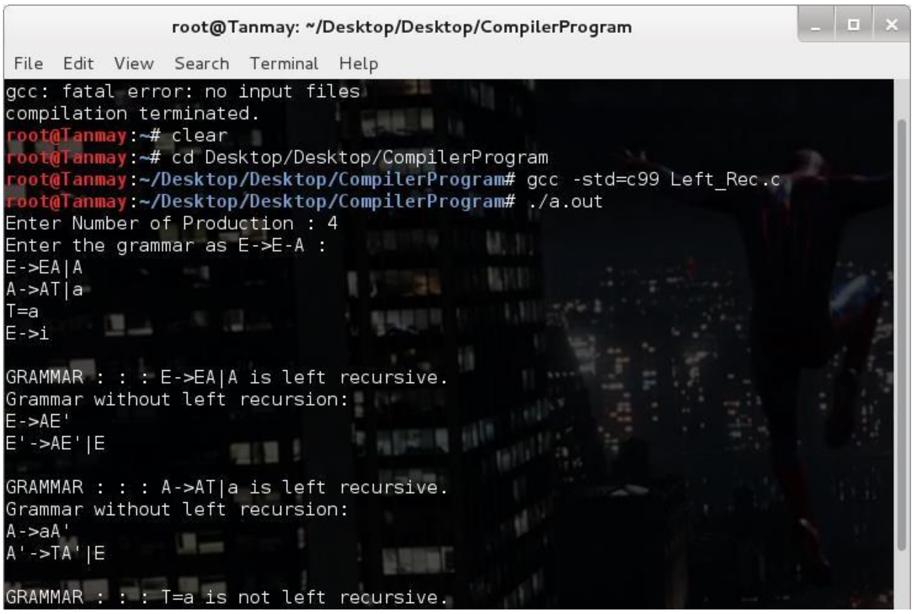
## PRACTICAL: 4

### Write a program to remove the Left Recursion from a given grammar.

**CODE:**

```
#include <stdio.h>
#include <string.h>
#define SIZE  10
int main()
{
   char non_terminal;
   char beta, alpha;
   int num;
   char production[10][SIZE];
   int index = 3; /* starting of the string following "->" */
   printf("Enter Number of Production : ");
   scanf("%d", &num);
   printf("Enter the grammar as E->E-A :\n");
   for (int i = 0; i < num; i++)
   {
      scanf("%s", production[i]);
   }
   for (int i = 0; i < num; i++)
   {
      printf("\nGRAMMAR : : : %s", production[i]);
      non_terminal = production[i][0];
      if (non_terminal == production[i][index])
      {
         alpha = production[i][index + 1];
         printf(" is left recursive.\n");
         while (production[i][index] != 0 && production[i][index] != '|')
            index++;
            if (production[i][index] != 0)
            {
                beta = production[i][index + 1];
                printf("Grammar without left recursion:\n");
                printf("%c->%c%c\'", non_terminal, beta, non_terminal);
                printf("\n%c\'->%c%c\'|E\n", non_terminal, alpha, non_terminal);
            }
            else
                printf(" can't be reduced\n");
        }
        else
            printf(" is not left recursive.\n");
        index = 3;
    }
}
```

**OUTPUT:**



root@Tanmay: ~/Desktop/Desktop/CompilerProgram
File Edit View Search Terminal Help

```
gcc: fatal error: no input files
compilation terminated.
root@Tanmay:~# clear
root@Tanmay:~# cd Desktop/Desktop/CompilerProgram
root@Tanmay:~/Desktop/Desktop/CompilerProgram# gcc -std=c99 Left_Rec.c
root@Tanmay:~/Desktop/Desktop/CompilerProgram# ./a.out
Enter Number of Production : 4
Enter the grammar as E->E-A :
E->EA|A
A->AT|a
T=a
E->i

GRAMMAR : : : E->EA|A is left recursive.
Grammar without left recursion:
E->AE'
E'->AE'|E

GRAMMAR : : : A->AT|a is left recursive.
Grammar without left recursion:
A->aA'
A'->TA'|E

GRAMMAR : : : T=a is not left recursive.
```

# PRACTICAL: 5

## Write a program to find First and Follow from the given set of production rules.

**CODE:**

```
// C program to calculate the First and
// Follow sets of a given grammar
#include <ctype.h>
#include <stdio.h>
#include <string.h>

// Functions to calculate Follow
void followfirst(char, int, int);
void follow(char c);

// Function to calculate First
void findfirst(char, int, int);

int count, n = 0;

// Stores the final result
// of the First Sets
char calc_first[10][100];

// Stores the final result
// of the Follow Sets
char calc_follow[10][100];
int m = 0;

// Stores the production rules
char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;

int main(int argc, char** argv)
{
        int  jm  =  0;
        int  km  =  0;
        int i, choice;
```

```
char c, ch;
count = 8;

// The Input grammar
strcpy(production[0], "X=TnS");
strcpy(production[1], "X=Rm");
strcpy(production[2], "T=q");
strcpy(production[3], "T=#");
strcpy(production[4], "S=p");
strcpy(production[5], "S=#");
strcpy(production[6], "R=om");
strcpy(production[7], "R=ST");

int kay;
char done[count];
int ptr = -1;

// Initializing the calc_first array
for (k = 0; k < count; k++) {
        for (kay = 0; kay < 100; kay++)
                {calc_first[k][kay] = '!';
        }
}
int point1 = 0, point2, xxx;

for (k = 0; k < count; k++)
        {c = production[k][0];
        point2 = 0;
        xxx = 0;

        // Checking if First of c has
        // already been calculated
        for (kay = 0; kay <= ptr; kay++)
                if (c == done[kay])
                        xxx = 1;

        if (xxx == 1)
                continue;

        // Function call
        findfirst(c, 0, 0);
        ptr += 1;
        // Adding c to the calculated list
        done[ptr] = c;
        printf("\n First(%c) = { ", c);
        calc_first[point1][point2++] = c;
        // Printing the First Sets of the grammar
        for (i = 0 + jm; i < n; i++) {
            int lark = 0, chk = 0;

            for (lark = 0; lark < point2; lark++) {

                if (first[i] == calc_first[point1][lark])
                    {chk = 1;
                    break;
                }
            }
            if (chk == 0) {
                printf("%c, ", first[i]);
                calc_first[point1][point2++] = first[i];
            }
        }
        printf("}\n");
        jm = n;
        point1++;
    }
    printf("\n");
    printf("_______________________________________"
        "\n\n");
    char donee[count];
    ptr = -1;

    // Initializing the calc_follow array
    for (k = 0; k < count; k++) {
        for (kay = 0; kay < 100; kay++)
            {calc_follow[k][kay] = '!';
        }
    }
    point1  =  0;
    int land = 0;
    for  (e  =  0;  e  <  count;  e++)
        {            ck            =
        production[e][0];
        point2 = 0;
        xxx = 0;

        // Checking if Follow of ck
        // has already been calculated
        for (kay = 0; kay <= ptr; kay++)
            if (ck == donee[kay])
                xxx = 1;

        if (xxx == 1)
            continue;
        land += 1;

        // Function call
        follow(ck);
        ptr += 1;

        // Adding ck to the calculated list
        donee[ptr] = ck;
        printf(" Follow(%c) = { ", ck);
        calc_follow[point1][point2++] = ck;

        // Printing the Follow Sets of the grammar
        for (i = 0 + km; i < m; i++) {
            int lark = 0, chk = 0;
            for (lark = 0; lark < point2; lark++) {
                if (f[i] == calc_follow[point1][lark])
                    {chk = 1;
                    break;
                }
            }
            if (chk == 0) {
                printf("%c, ", f[i]);
                calc_follow[point1][point2++] = f[i];
            }
        }
        printf(" }\n\n");
        km = m;
        point1++;
    }
}

void follow(char c)
```

```
{
        int i, j;

        // Adding "$" to the follow
        // set of the start symbol
        if (production[0][0] == c)
                {f[m++] = '$';
        }
        for (i = 0; i < 10; i++) {
                for (j = 2; j < 10; j++) {
                        if (production[i][j] == c) {
                                if (production[i][j + 1] != '\0') {
                                        // Calculate the first of the next
                                        // Non-Terminal in the production
                                        followfirst(production[i][j + 1], i,
                                                                (j + 2));
                                }

                                if (production[i][j + 1] == '\0'
                                        && c != production[i][0]) {
                                        // Calculate the follow of the
                                        // Non-Terminal in the L.H.S. of the
                                        // production
                                        follow(production[i][0]);
                                }
                        }
                }
        }
}

void findfirst(char c, int q1, int q2)
{
        int j;

        // The case where we
        // encounter a Terminal
        if (!(isupper(c))) {
                first[n++] = c;
        }
        for (j = 0; j < count; j++) {
                if (production[j][0] == c) {
                        if (production[j][2] == '#') {
                                if (production[q1][q2] == '\0')
```

```
                    first[n++] = '#';
                else if (production[q1][q2] != '\0'
                        && (q1 != 0 || q2 != 0)) {
                    // Recursion to calculate First of New
                    // Non-Terminal we encounter after
                    // epsilon
                    findfirst(production[q1][q2], q1,
                            (q2 + 1));
                }
                else
                    first[n++] = '#';
            }
            else if (!isupper(production[j][2]))
                {first[n++] = production[j][2];
            }
            else {
                // Recursion to calculate First of
                // New Non-Terminal we encounter
                // at the beginning
                findfirst(production[j][2], j, 3);
            }
        }
    }
}

void followfirst(char c, int c1, int c2)
{
    int k;

    // The case where we encounter
    // a Terminal
    if (!(isupper(c)))
        f[m++] = c;
    else {
        int i = 0, j = 1;
        for (i = 0; i < count; i++) {
            if (calc_first[i][0] == c)
                break;
        }

        // Including the First set of the
        // Non-Terminal in the Follow of
        // the original query
        while (calc_first[i][j] != '!') {
            if (calc_first[i][j] != '#') { f[m++]
                = calc_first[i][j];
            }
            else {
                if (production[c1][c2] == '\0') {
                    // Case where we reach the
                    // end of a production
                    follow(production[c1][0]);
                }
                else {
                    // Recursion to the next symbol
                    // in case we encounter a "#"
                    followfirst(production[c1][c2], c1,
                                c2 + 1);
                }
            }
            j++;
        }
    }
}
```

**OUTPUT:**

```
First(X) = {q, n, o, p, #,m}
First(T) = {q, #, }
First(S) = {p, #, }
First(R) = {o, p, q, #, }
-------------------------------
Follow(X) = {$,  }
Follow(T) = {n, m, }
Follow(S) = {$, q, m,  }
Follow(R) = {m,  }
```

# PRACTICAL: 6

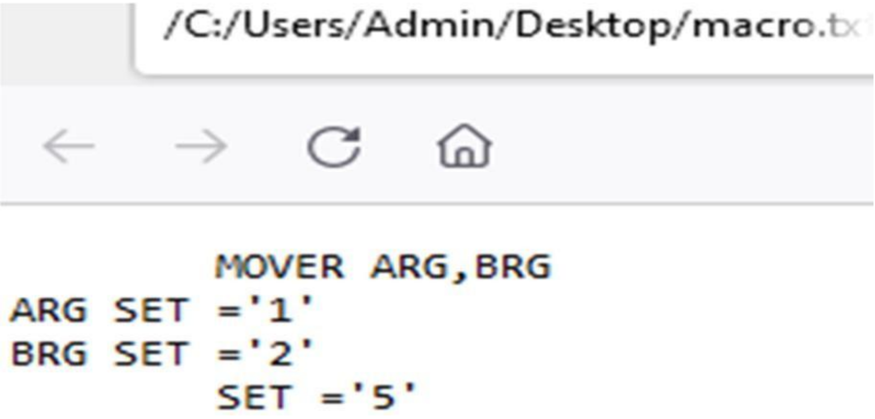
## Write an Assembly Language program in a text file and generate Symbol Table, Literal Table and Pool Table.

**CODE:**

```
 #include<stdio.h>
#include<stdlib.h>
struct sys{
  char n[20];
  int ad;
};
int main()
{
  char ch[50],c;int adr=0,f=0,fl=0,sp=0,lp=0;struct sys stb[20],lt[20];
  FILE *fr1; fr1=fopen("macro.txt","r");
  printf("The ap is \n");
  while((c=fgetc(fr1))!=EOF)
  {
    printf("%c",c);
  }
  FILE *fr;
  fr=fopen("ap.txt","r");
  c=fgetc(fr);
  printf("\n\nThe sys and lt is \n\n");
  while((c)!=EOF)
  {
    if(c=='\t'&&f==0)
            {
              f =1;
            }
   else if(c!='\t'&&f==0&&fl==0)
    {
      int j=0;
```

```
        while(c!='\t'&&(c)!=EOF){
        stb[sp].n[j++]=c;c=fgetc(fr);
        }
        stb[sp].n[j]='\0';
        stb[sp++].ad=adr;
        fl=1;
      }
     else if(c=='=')
      {
        int j=0;
        while(c!='\n'&&(c)!=EOF){
        lt[lp].n[j++]=c;
        c=fgetc(fr);
      }
     lt[lp].n[j-1]='\0';
     lt[lp++].ad=adr;
     f=1;
    }
   else if(c=='\n')
    {
      adr++; f=0;fl=0;
    }
    c=fgetc(fr);
   }
  printf("\n"); fclose(fr);
  int  i; for(i=0;i<sp;i++)
    printf("%s %d\n",stb[i].n,stb[i].ad);
  printf("\nlt table\n");
  for(i=0;i<lp;i++)
    printf("%s %d\n",lt[i].n,lt[i].ad);return 0;
  }
```

**OUTPUT:**



```
        MOVER ARG,BRG
ARG SET ='1'
BRG SET ='2'
        SET ='5'
```

## PRACTICAL: 7

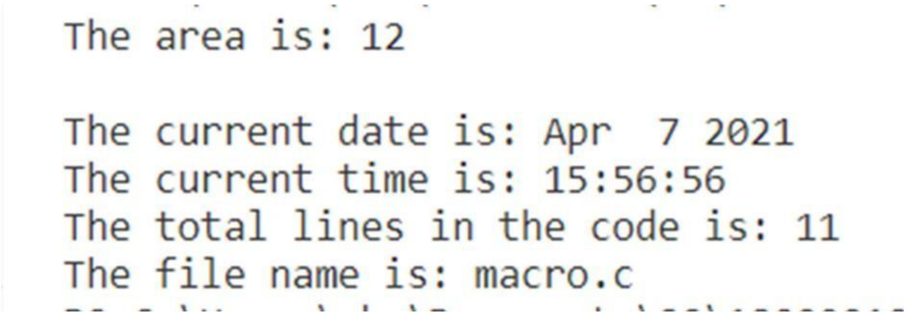
## Write a program to demonstrate the use of Macro.

**CODE:**

```
#include<stdio.h>
#define RECTANGLE(l,b)l*b
int main()
{
  int length = 3, breadth = 4;
  int area = RECTANGLE(length,breadth);

  printf("The area is: %d\n\n", area);
  printf("The current date is: %s\n",              DATE__);
  printf("The current time is: %s\n",   TIME__);
  printf("The total lines in the code is: %d\n",           LINE );
  printf("The file name is: %s\n",       FILE    );
return 0;
}
```

**OUTPUT:**



```
The area is: 12

The current date is: Apr  7 2021
The current time is: 15:56:56
The total lines in the code is: 11
The file name is: macro.c
```

# PRACTICAL: 8

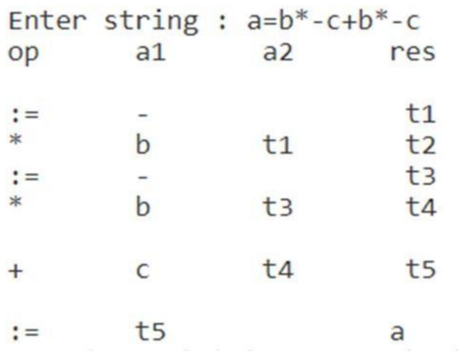
## Write a program that generates a Quadruple Table for the given postfix String.

**CODE:**

```
#include<stdio.h>
#include<string.h>
void main() {
 char line[20];
 int s[20];
 int t=1;
 int i=0;
 printf("Enter string : ");
 gets(line);
 for(i=0;i<20;i++)s[i]=0;
 printf("op\ta1\ta2\tres\n");
 for(i=2;line[i]!='\0';i++)
 {
   if(line[i]=='/' || line[i]=='*')
   {
     printf("\n");
     if(s[i]==0)
     {
       if(s[i+1]==0)
       {
         printf(":=\t%c\t\t t%d\n",line[i+1],t);
         s[i+1]=t++;
       }
      printf("%c\t",line[i]);
      (s[i-1]==0)?printf("%c\t",line[i-1]):printf("t%d\t",s[i-1]);
      printf("t%d \t t%d",s[i+1],t);
      s[i-1]=s[i+1]=t++;
      s[i]=1;
```

```
        }
      }
    }
    for(i=2;line[i]!='\0';i++)
    {
      if(line[i]=='+' || line[i]=='-')
       {
         printf("\n");
         if(s[i]==0)
          {
            if(s[i+1]==0)
            {
              printf(":=\t%c\t\t t%d\n",line[i+1],t);
              s[i+1]=t++;
            }
           printf("%c\t",line[i]);
           (s[i-1]==0)?printf("%c\t",line[i-1]):printf("t%d\t",s[i-1]);
           printf("t%d \t t%d",s[i+1],t);
           s[i-1]=s[i+1]=t++;
           s[i]=1;}}}
   printf("\n:=\tt%d\t\t%c",t-1,line[0]);
   }
```

**OUTPUT:**



```
Enter string : a=b*-c+b*-c
op      a1      a2      res

:=      -                t1
*       b       t1       t2
:=      -                t3
*       b       t3       t4

+       c       t4       t5

:=      t5              a
```

# PRACTICAL: 9

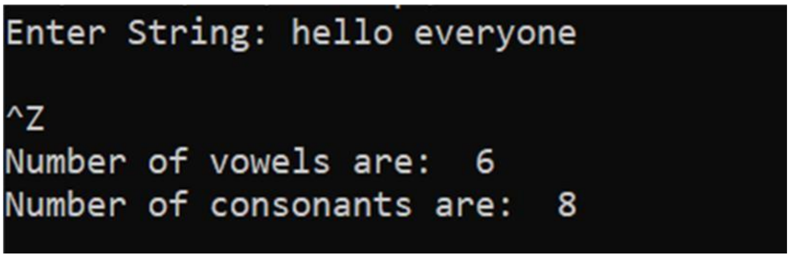
## Write a lex program to count the number of vowels and consonants in a given string.

## CODE:

```
%{
  #include<stdio.h>
  #include<string.h>
  int vcount=0, ccount=0;
%}

%%
[a|e|i|o|u|A|E|I|O|U] {vcount++;}
[a-z A-Z (^a|e|i|o|u|A|E|I|O|U)] {ccount++;}
%%
int yywrap(void){}
int main()
{
  printf("Enter String: "); yylex();
  printf("Number of vowels are: %d\n", vcount);
  printf("Number of consonants are: %d\n", ccount);
  return 0;
}
```

## OUTPUT:



```
Enter String: hello everyone

^Z
Number of vowels are:  6
Number of consonants are:  8
```

## PRACTICAL: 10

**Write a lex program to count the number of characters, words, spaces, end of lines.**

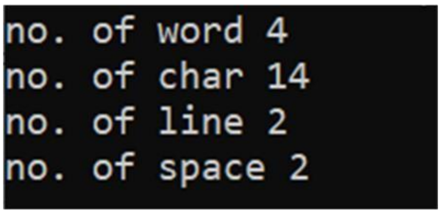
## CODE:

```
%{
#include <stdio.h> int c=0,w=0,s=0,l=1;
%}
word [^ \t\n,\.:]+ eol [\n]
blank [ ]
%%
{word} {w++; c=c+yyleng;}
{eol} {l++;}
{blank} {s++;}
%%
void main(int argc,char *argv[])
{
     if (argc!=2) {
    printf("usage : ./a.out in.txt \n"); exit(0);
     }
    yyin=fopen(argv[1],"r"); yylex();
printf("no. of word %d \n",w); printf("no. of char %d \n",c); printf("no. of line %d \n",l);
printf("no. of space %d \n",s);
}
int yywrap()  {  return 1; }
```

**INPUT:**

Hello
I am Nirali

**OUTPUT:**



```
no. of word 4
no. of char 14
no. of line 2
no. of space 2
```

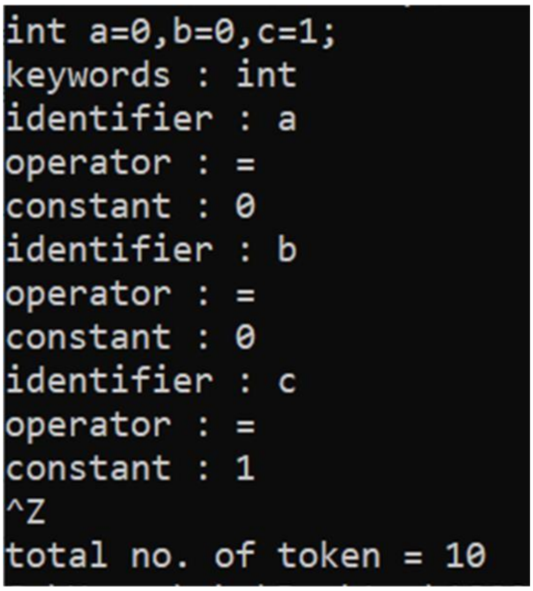
# PRACTICAL: 11

## Write a lex program to identify identifiers, constants, and keywords (int, float) for C language.

## CODE:

```
%{
  int n = 0 ;
%}
%%
"while"|"if"|"else"|"int"|"float" {n++;printf("keywords : %s", yytext);}
[a-zA-Z_][a-zA-Z0-9_]* {n++;printf("\nidentifier : %s", yytext);}
"<="|"=="|"="|"++"|"-"|"*"|"+" {n++;printf("\noperator : %s", yytext);}
[0-9]+ {n++;printf("\nconstant : %s", yytext);}
. ;
%%
int main()
 {yylex();
 printf("total no. of token = %d", n);
}
int yywrap(){  return(1);  }
```

**OUTPUT:**



```
int a=0,b=0,c=1;
keywords : int
identifier : a
operator : =
constant : 0
identifier : b
operator : =
constant : 0
identifier : c
operator : =
constant : 1
^Z
total no. of token = 10
```

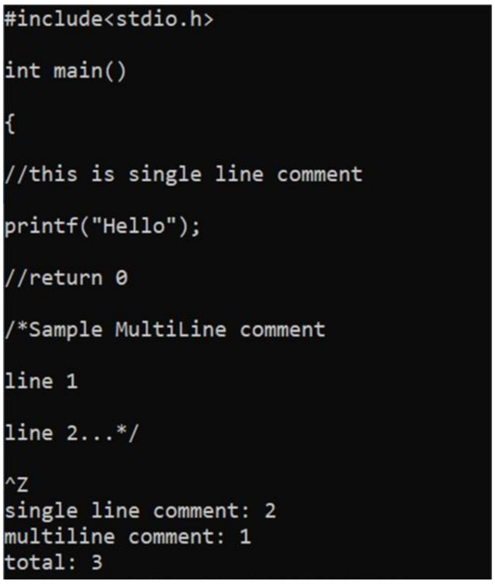
## PRACTICAL: 12

### Write a lex program to count and display Single line and Multiline comments for a C language.

**CODE:**

```
%{    #include<stdio.h>
      #include<stdlib.h>
      int a=0,c=0,d,e=1;
%}
%%
"/*" {if(e==1)e++;}
"*/" {if(e==1)e=1;c++;}
"//".* {if(e==1)a++;}
. {if(e==0)ECHO;}
%%
void main(int argc)
  {yylex();
  printf("single line comment: %d \nmultiline comment: %d \n",a,c);
  d=a+c;
  printf("total: %d \n",d);
}
int yywrap()  {  return(1);}
```

**OUTPUT:**



```
#include<stdio.h>

int main()

{

//this is single line comment

printf("Hello");

//return 0

/*Sample MultiLine comment

line 1

line 2...*/

^Z
single line comment: 2
multiline comment: 1
total: 3
```