# System Security Labs Practical Exam Programs

## Experiment No: -3

### WAP To Count the Numbers of Words, Character, Blank Spaces & Lines Using LEX.

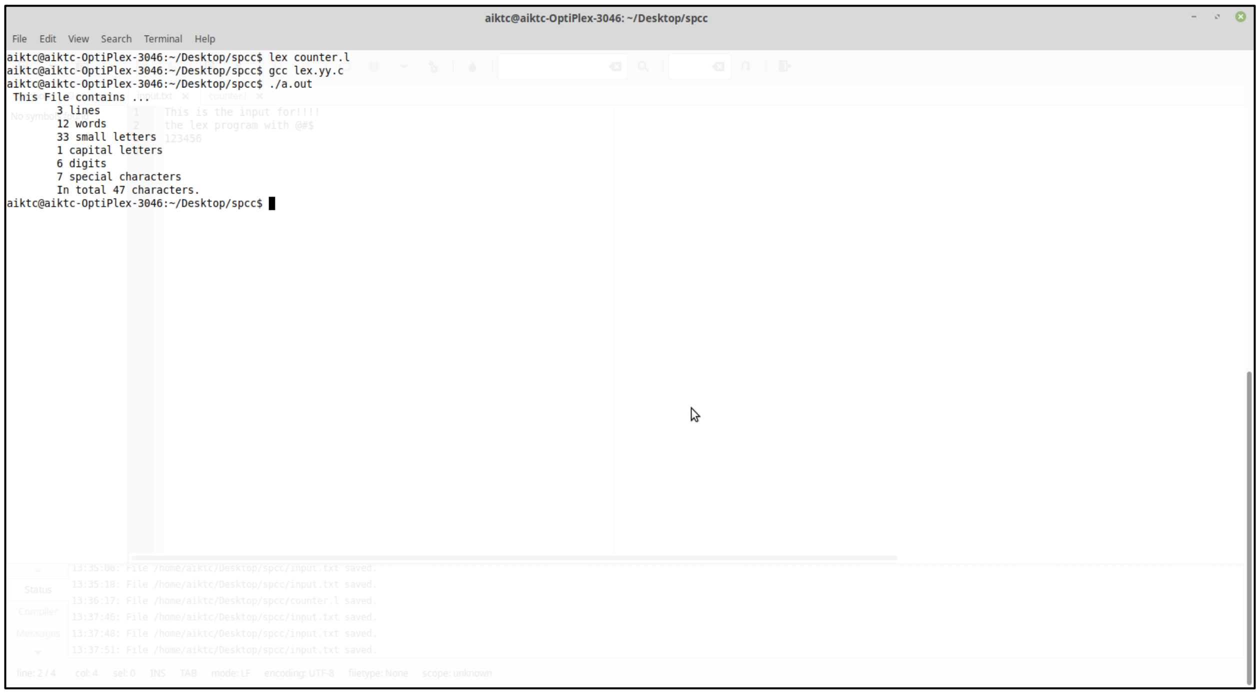
**Program Code**

```
%{
#include<stdio.h>
int         lines=0,          words=0,s_letters=0,c_letters=0,          num=0,
spl_char=0,total=0,space=0;
%}
%%

\n { lines++; words++;}
[\t ' '] words++;
(?:\t\t) space++;
[A-Z] c_letters++;
[a-z] s_letters++;
[0-9] num++;
[.@#$!%^&*()><?~`''"";:?] spl_char++;
%%

int main(void)
{
yyin= fopen("input.txt","r");
yylex();
total=s_letters+c_letters+num+spl_char;
printf(" This File contains ...");
printf("\n\t%d lines", lines);
printf("\n\t%d words",words-space);
printf("\n\t%d small letters", s_letters);
printf("\n\t%d capital letters",c_letters);
printf("\n\t%d digits", num);
printf("\n\t%d special characters",spl_char);
printf("\n\tIn total %d characters.\n",total);
}

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

**Input File**

```
This is the input for!!!!
the lex program with @#$
123456
```

**Output**



```
aiktc@aiktc-OptiPlex-3046:~/Desktop/spcc$ lex counter.l
aiktc@aiktc-OptiPlex-3046:~/Desktop/spcc$ gcc lex.yy.c
aiktc@aiktc-OptiPlex-3046:~/Desktop/spcc$ ./a.out
 This File contains ...
        3 lines
        12 words
        33 small letters
        1 capital letters
        6 digits
        7 special characters
        In total 47 characters.
aiktc@aiktc-OptiPlex-3046:~/Desktop/spcc$ 
```

## Experiment No: -4

### WAP To recognize identifiers in C using symbol table.

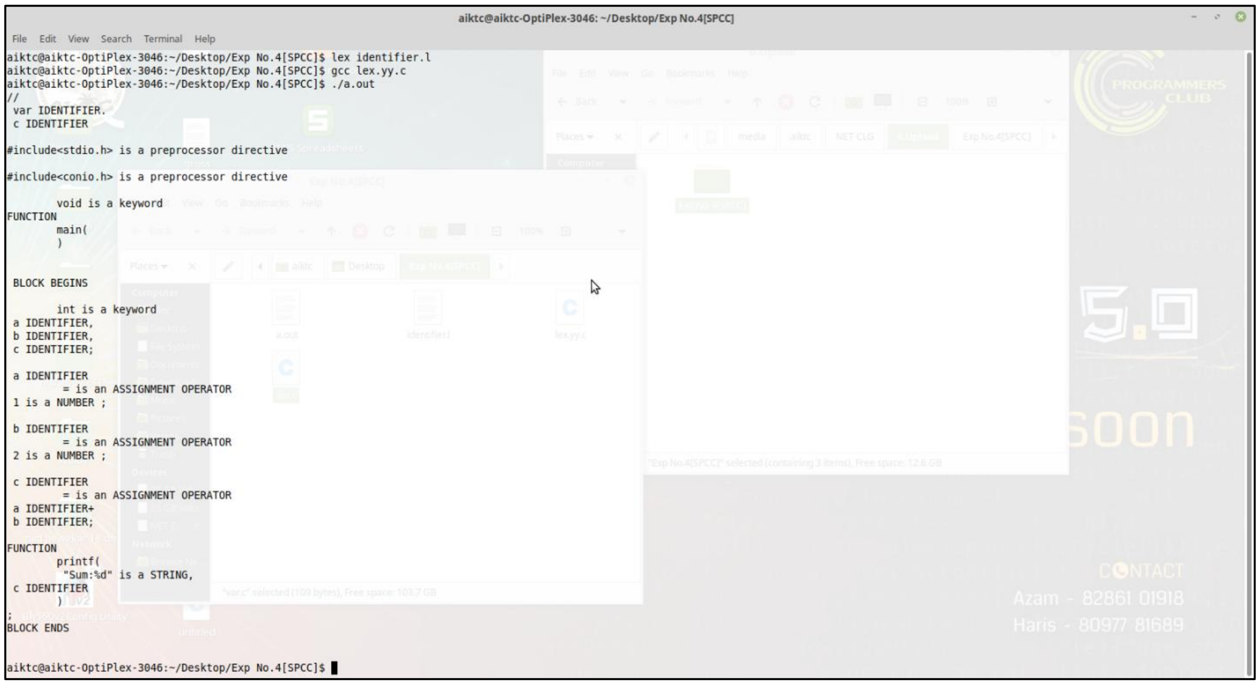
**Program Code**

```
//Implementation of Lexical Analyzer using Lex tool
%{
int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* {printf("\n%s is a preprocessor directive",yytext);}
int |
float |
char |
double |
while |
for |
struct |
typedef |
do |
if |
break |
continue |
void |
switch |
return |
else |
goto {printf("\n\t%s is a keyword",yytext);}
"/*" {COMMENT=1;}{printf("\n\t %s is a COMMENT",yytext);}
{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}
\{  {if(!COMMENT)printf("\n BLOCK BEGINS");}
\}  {if(!COMMENT)printf("BLOCK ENDS ");}
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\".*\" {if(!COMMENT)printf("\n\t %s is a STRING",yytext);}
[0-9]+ {if(!COMMENT) printf("\n %s is a NUMBER ",yytext);}
\)(\:)? {if(!COMMENT)printf("\n\t");ECHO;printf("\n");}
\( ECHO;
= {if(!COMMENT)printf("\n\t %s is an ASSIGNMENT OPERATOR",yytext);}
\<= |
\>= |
\< |
== |
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
int main(int argc, char **argv)
{
      FILE *file;
      file=fopen("var.c","r");
      if(!file)
      {
            printf("could not open the file");
            exit(0);
      }
      yyin=file;
      yylex();
      printf("\n");
      return(0);
}
int yywrap()
{
      return(1);
}
```

**Input File**

```
//var.c
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
a=1;
b=2;
c=a+b;
printf("Sum:%d",c);
}
```

**Output**


aiktc@aiktc-OptiPlex-3046: ~/Desktop/Exp No.4[SPCC]
File Edit View Search Terminal Help

```
aiktc@aiktc-OptiPlex-3046:~/Desktop/Exp No.4[SPCC]$ lex identifier.l
aiktc@aiktc-OptiPlex-3046:~/Desktop/Exp No.4[SPCC]$ gcc lex.yy.c
aiktc@aiktc-OptiPlex-3046:~/Desktop/Exp No.4[SPCC]$ ./a.out
//
 var IDENTIFIER.
 c IDENTIFIER

#include<stdio.h> is a preprocessor directive

#include<conio.h> is a preprocessor directive

        void is a keyword
FUNCTION
        main(
        )


 BLOCK BEGINS

        int is a keyword
 a IDENTIFIER,
 b IDENTIFIER,
 c IDENTIFIER;

 a IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 1 is a NUMBER ;

 b IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 2 is a NUMBER ;

 c IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 a IDENTIFIER+
 b IDENTIFIER;

FUNCTION
        printf(
         "Sum:%d" is a STRING,
 c IDENTIFIER
        )
;
BLOCK ENDS

aiktc@aiktc-OptiPlex-3046:~/Desktop/Exp No.4[SPCC]$ 
```

## Experiment No: -5

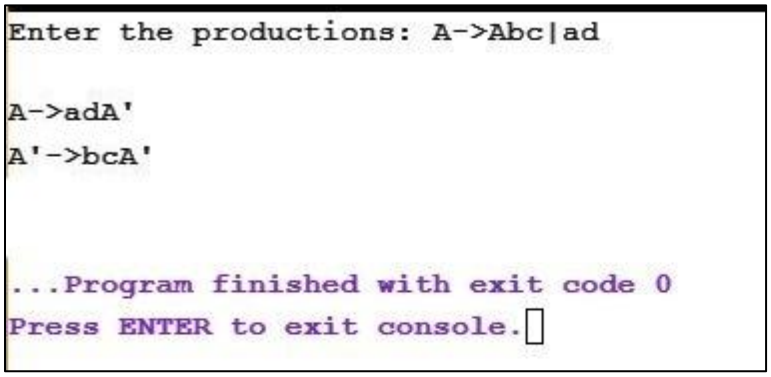
### WAP to remove the Left Recursion from a given grammar.

**Program Code**

```
#include<stdio.h>
#include<string.h>
int main()  {
    char input[100],*l,*r,*temp,tempprod[20],productions[25][50];
    int i=0,j=0,flag=0;
    printf("Enter the productions: ");
    scanf("%s",input);
    l = strtok(input,"->");
    r = strtok(NULL,"->");
    temp = strtok(r,"|");
    while(temp)  {
        if(temp[0] == l[0])  {
            flag = 1;
            sprintf(productions[i++],"%s'->%s%s'\0",l,temp+1,l);
        }
        else
            sprintf(productions[i++],"%s->%s%s'\0",l,temp,l);
        temp = strtok(NULL,"|");
    }
    sprintf(productions[i++],"%s'->\u03B5",l);
    if(flag == 0)
        printf("The given productions don't have Left Recursion");
    else
```

```
        for(j=0;j<i;j++)  {
            printf("\n%s",productions[j]);
        }
    return 0;
}
```

**Output**



```
Enter the productions: A->Abc|ad

A->adA'
A'->bcA'


...Program finished with exit code 0
Press ENTER to exit console.
```

## Experiment No: -6

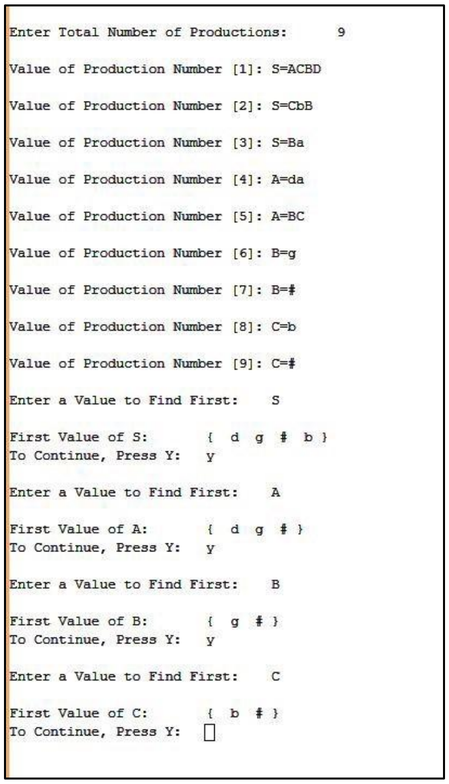
**WAP to find first () of given grammar.**

**Program Code**

```
#include<stdio.h>
#include<ctype.h>

void Find_First(char[], char);
void Array_Manipulation(char[], char);

int limit;
char production[25][25];

int main()
{
      char option;
      char ch;
      char array[25];
      int count;
      printf("\nEnter Total Number of Productions:\t");
      scanf("%d", &limit);
      for(count = 0; count < limit; count++)
      {
            printf("\nValue of Production Number [%d]:\t", count + 1);
            scanf("%s", production[count]);
      }
      do
      {
            printf("\nEnter a Value to Find First:\t");
            scanf(" %c", &ch);
            Find_First(array, ch);
            printf("\nFirst Value of %c:\t{ ", ch);
            for(count = 0; array[count] != '\0'; count++)
            {
                  printf(" %c ", array[count]);
            }
            printf("}\n");
            printf("To Continue, Press Y:\t");
            scanf(" %c", &option);
      }while(option == 'y' || option == 'Y');
      return 0;
}

void Find_First(char* array, char ch)
{
      int count, j, k;
      char temporary_result[20];
      int x;
      temporary_result[0] = '\0';
      array[0] = '\0';
      if(!(isupper(ch)))
      {
            Array_Manipulation(array, ch);
            return ;
      }
      for(count = 0; count < limit; count++)
      {
            if(production[count][0] == ch)
            {
                  if(production[count][2] == '$')
                  {
                        Array_Manipulation(array, '$');
                  }
                  else
                  {
                        j = 2;
                        while(production[count][j] != '\0')
                        {
                              x = 0;
                              Find_First(temporary_result,
production[count][j]);
                              for(k = 0; temporary_result[k] != '\0'; k++)
                              {

Array_Manipulation(array,temporary_result[k]);
                              }
                              for(k = 0; temporary_result[k] != '\0'; k++)
                              {
                                    if(temporary_result[k] == '$')
                                    {
                                          x = 1;
                                          break;
                                    }
```

```
                    }
                    if(!x)
                    {
                        break;
                    }
                    j++;
                }
            }
        }
    }
    return;
}

void Array_Manipulation(char array[], char value)
{
    int temp;
    for(temp = 0; array[temp] != '\0'; temp++)
    {
        if(array[temp] == value)
        {
            return;
        }
    }
    array[temp] = value;
    array[temp + 1] = '\0';
}
```

**Output**



```
Enter Total Number of Productions:      9

Value of Production Number [1]: S=ACBD

Value of Production Number [2]: S=CbB

Value of Production Number [3]: S=Ba

Value of Production Number [4]: A=da

Value of Production Number [5]: A=BC

Value of Production Number [6]: B=g

Value of Production Number [7]: B=#

Value of Production Number [8]: C=b

Value of Production Number [9]: C=#

Enter a Value to Find First:    S

First Value of S:       {  d  g  #  b }
To Continue, Press Y:   y

Enter a Value to Find First:    A

First Value of A:       {  d  g  # }
To Continue, Press Y:   y

Enter a Value to Find First:    B

First Value of B:       {  g  # }
To Continue, Press Y:   y

Enter a Value to Find First:    C

First Value of C:       {  b  # }
To Continue, Press Y:
```

# Experiment No: -7

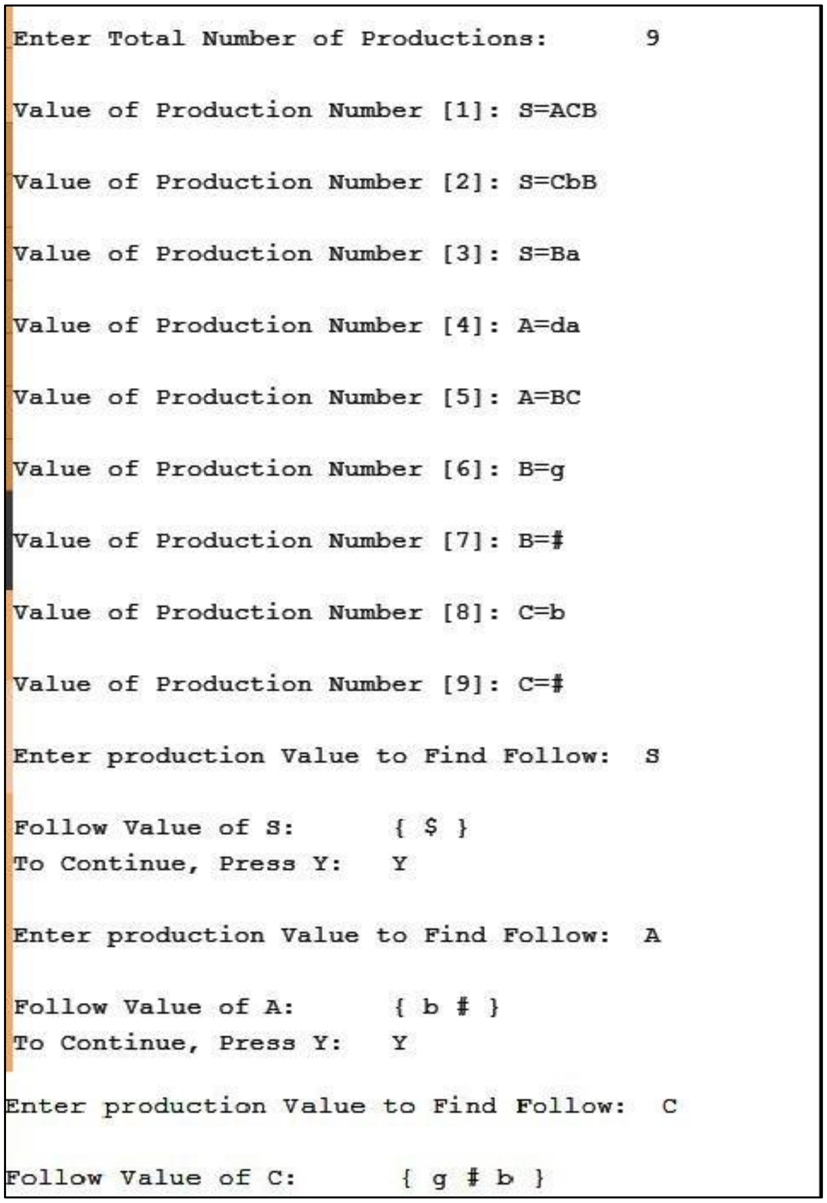
## WAP to find Follow () of given grammar

**Program Code**

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

int limit, x = 0;
char production[10][10], array[10];

void find_first(char ch);
void find_follow(char ch);
void Array_Manipulation(char ch);

int main()
{
      int count;
      char option, ch;
      printf("\nEnter Total Number of Productions:\t");
      scanf("%d", &limit);
      for(count = 0; count < limit; count++)
      {
            printf("\nValue of Production Number [%d]:\t", count + 1);
            scanf("%s", production[count]);
      }
      do
      {
            x = 0;
            printf("\nEnter production Value to Find Follow:\t");
            scanf(" %c", &ch);
            find_follow(ch);
            printf("\nFollow Value of %c:\t{ ", ch);
            for(count = 0; count < x; count++)
            {
                  printf("%c ", array[count]);
            }
            printf("}\n");
            printf("To Continue, Press Y:\t");
            scanf(" %c", &option);
      }while(option == 'y' || option == 'Y');
      return 0;
}

void find_follow(char ch)
{
      int i, j;
      int length = strlen(production[i]);
      if(production[0][0] == ch)
      {
            Array_Manipulation('$');
```

```
        }
        for(i = 0; i < limit; i++)
        {
                for(j = 2; j < length; j++)
                {
                        if(production[i][j] == ch)
                        {
                                if(production[i][j + 1] != '\0')
                                {
                                        find_first(production[i][j + 1]);
                                }
                                if(production[i][j   +   1]   ==   '\0'   &&   ch   !=
production[i][0])
                                {
                                        find_follow(production[i][0]);
                                }
                        }
                }
        }
}

void find_first(char ch)
{
        int i, k;
        if(!(isupper(ch)))
        {
                Array_Manipulation(ch);
        }
        for(k = 0; k < limit; k++)
        {
                if(production[k][0] == ch)
                {
                        if(production[k][2] == '$')
                        {
                                find_follow(production[i][0]);
                        }
                        else if(islower(production[k][2]))
                        {
                                Array_Manipulation(production[k][2]);
                        }
                        else
                        {
                                find_first(production[k][2]);
                        }
                }
        }
}

void Array_Manipulation(char ch)
{
        int count;
        for(count = 0; count <= x; count++)
        {
            if(array[count] == ch)
            {
                  return;
            }
      }
      array[x++] = ch;
}
```

**Output**



```
Enter Total Number of Productions:      9

Value of Production Number [1]: S=ACB

Value of Production Number [2]: S=CbB

Value of Production Number [3]: S=Ba

Value of Production Number [4]: A=da

Value of Production Number [5]: A=BC

Value of Production Number [6]: B=g

Value of Production Number [7]: B=#

Value of Production Number [8]: C=b

Value of Production Number [9]: C=#

Enter production Value to Find Follow:  S

Follow Value of S:      { $ }
To Continue, Press Y:   Y

Enter production Value to Find Follow:  A

Follow Value of A:      { b # }
To Continue, Press Y:   Y

Enter production Value to Find Follow:  C

Follow Value of C:      { g # b }
```

# Experiment No: -8

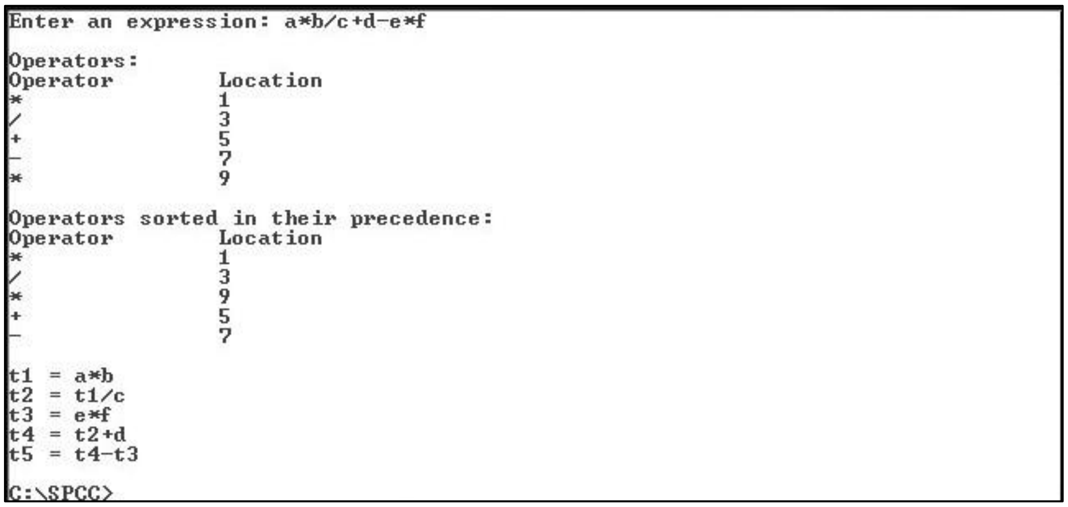
## WAP to generate 3 address code.

**Program Code**

```
import java.io.*;
class ThreeAddressCode
{
      private static final char[][] precedence = {
            {'/', '1'},
            {'*', '1'},
            {'+', '2'},
            {'-', '2'}
      };

      private static int precedenceOf(String t)
      {
            char token = t.charAt(0);
            for (int i=0; i < precedence.length; i++)
            {
                  if (token == precedence[i][0])
                  {
                        return Integer.parseInt(precedence[i][1]+"");
                  }
            }
            return -1;
      }

      public static void main(String[] args) throws Exception
      {
            int i, j, opc=0;
            char token;
            boolean processed[];
            String[][] operators = new String[10][2];
            String expr="", temp;
            BufferedReader      in      =      new      BufferedReader(new
InputStreamReader(System.in));
            System.out.print("\nEnter an expression: ");
            expr = in.readLine();
            processed = new boolean[expr.length()];
            for (i=0; i < processed.length; i++)
            {
                  processed[i] = false;
            }
            for (i=0; i < expr.length(); i++)
            {
                  token = expr.charAt(i);
                  for (j=0; j < precedence.length; j++)
                  {
                        if (token==precedence[j][0])
                        {
                              operators[opc][0] = token+"";
                              operators[opc][1] = i+"";
                              opc++;
                              break;
                        }
                  }
            }
            System.out.println("\nOperators:\nOperator\tLocation");
            for (i=0; i < opc; i++)
            {
                  System.out.println(operators[i][0]      +      "\t\t"      +
operators[i][1]);
            }
            //sort
            for (i=opc-1; i >= 0; i--)
            {
                  for (j=0; j < i; j++)
                  {
                        if          (precedenceOf(operators[j][0])           >
precedenceOf(operators[j+1][0]))
                        {
                              temp = operators[j][0];
                              operators[j][0] = operators[j+1][0];
                              operators[j+1][0] = temp;
                              temp = operators[j][1];
                              operators[j][1] = operators[j+1][1];
                              operators[j+1][1] = temp;
                        }
                  }
            }
            System.out.println("\nOperators       sorted       in       their
precedence:\nOperator\tLocation");
            for (i=0; i < opc; i++)
            {
                  System.out.println(operators[i][0]      +      "\t\t"      +
operators[i][1]);
            }
            System.out.println();
            for (i=0; i < opc; i++)
            {
                  j = Integer.parseInt(operators[i][1]+"");
                  String op1="", op2="";
                  if (processed[j-1]==true)
                  {
                        if         (precedenceOf(operators[i-1][0])         ==
precedenceOf(operators[i][0]))
                        {
                              op1 = "t"+i;
                        }
                        else
                        {
                              for (int x=0; x < opc; x++)
                              {
```

```
                                        if                    ((j-2)                    ==
Integer.parseInt(operators[x][1]))
                                        {
                                                op1 = "t"+(x+1)+"";
                                        }
                                }
                        }
                }
                else
                {
                        op1 = expr.charAt(j-1)+"";
                }
                if (processed[j+1]==true)
                {
                        for (int x=0; x < opc; x++)
                        {
                                if ((j+2) == Integer.parseInt(operators[x][1]))
                                {
                                        op2 = "t"+(x+1)+"";
                                }
                        }
                }
                else
                {
                        op2 = expr.charAt(j+1)+"";
                }
                System.out.println("t"+(i+1)+" = "+op1+operators[i][0]+op2);
                processed[j] = processed[j-1] = processed[j+1] = true;
        }
    }
}
```

**Output**



```
Enter an expression: a*b/c+d-e*f

Operators:
Operator        Location
*               1
/               3
+               5
-               7
*               9

Operators sorted in their precedence:
Operator        Location
*               1
/               3
*               9
+               5
-               7

t1 = a*b
t2 = t1/c
t3 = e*f
t4 = t2+d
t5 = t4-t3

C:\SPCC>
```