EXPERIMENT NO-1

AIM: Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines.

ALGORITHM:

Step 1: Tokenization ie., Dividing the program into valid tokens.

Step 2: Remove white space characters.

Step 3: Remove comments.

Step 4: It also provides help in generating error message by providing row number and column number.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void final();
int Isiden(char ch);
int Isop(char ch);
int Isdel(char ch);
int Iskey(char * str);
char op[8]={'+','-','*','/','=','<','>','%'};
char del[8]={'}','{',';','(',')','[',']',','};
char *key[]={"int","void","main","char","float","printf","include","return"};
char *operato[]={"+","-","/","*","<",">","=","%","<=",">=","++"};
int idi=0,idj=0,k,opi=0,opj=0,deli=0,uqdi=0,uqidi=0,uqoperi=0,kdi=0,liti=0,ci=0;
int uqdeli[20],uqopi[20],uqideni[20],l=0,j;
char uqdel[20],uqiden[20][20],uqop[20][20],keyword[20][20];
char iden[20][20],oper[20][20],delem[20],litral[20][20],lit[20],constant[20][20];
void lexanalysis(char *str) {
  int i=0;
  while(str[i]!='\0') {
if(Isiden(str[i]))     //for identifiers
```

```
      {
while(Isiden(str[i]) {
iden[idi][idj++]=str[i++];
       }
iden[idi][idj]='\0';
idi++;idj=0;
      }
     else
     if(str[i]=="") {        //for literal
       lit[l++]=str[i];
       for(j=i+1;str[j]!="";j++) {
         lit[l++]=str[j];
        }
       lit[l++]=str[j];lit[l]='\0';
strcpy(litral[liti++],lit);
i=j+1;
       }
     else
if(Isop(str[i]))  {      // for operators
while(Isop(str[i]))
          {
oper[opi][opj++]=str[i++];
          }
oper[opi][opj]='\0';
opi++;opj=0;
        }
      else
if(Isdel(str[i]))  {   //for delemeters
while(Isdel(str[i]) {
delem[deli++]=str[i++];
          }
```

```
          }
        else {
i++;
        }
     }
final();
}
int Isiden(char ch)
 {
   if(isalpha(ch)||ch=='_'||isdigit(ch)||ch=='.')
   return 1;
   else
   return 0;
 }

int Isop(char ch)
 {
  int f=0,i;
  for(i=0;i<8&&!f;i++){
    if(ch==op[i])
     f=1;
   }
 return f;
}

int Isdel(char ch){
  int f=0,i;
  for(i=0;i<8&&!f;i++){
    if(ch==del[i])
     f=1;
   }
```

```
 return f;
 }

int Iskey(char * str) {
  int i,f=0;
  for(i=0;i<5;i++) {
    if(!strcmp(key[i],str))
      f=1;
    }
  return f;
}
void final() {
  int i=0;
idi=0;
  for(i=0;i<uqidi;i++) {
if(Iskey(uqiden[i]))        //identifying keywords
strcpy(keyword[kdi++],uqiden[i]);
     else
      if(isdigit(uqiden[i][0]))    //identifying constants
strcpy(constant[ci++],uqiden[i]);
     else
strcpy(iden[idi++],uqiden[i]);
   }

// printing the outputs

printf("\n\tDelemeter are : \n");
for(i=0;i<uqdi;i++)
printf("\t%c\n",uqdel[i]);
printf("\n\tOperators are : \n");
for(i=0;i<uqoperi;i++) {
```

```
printf("\t");
 puts(uqop[i]);
 }
printf("\n\tIdentifiers are : \n");
for(i=0;i<idi;i++) {
printf("\t");
  puts(iden[i]);
 }
printf("\n\tKeywords are : \n");
for(i=0;i<kdi;i++)  {
printf("\t");
  puts(keyword[i]);
 }
printf("\n\tConstants are :\n");
for(i=0;i<ci;i++) {
printf("\t");
  puts(constant[i]);
  }
printf("\n\tLiterals are :\n");
for(i=0;i<liti;i++) {
printf("\t");
   puts(litral[i]);
 }
}
void main()  {
  char str[50];
nclrscr();
printf("\nEnter the string : ");
scanf("%[^\n]c",str);
lexanalysis(str);
getch();
```

}

INPUT:

```
void main(){
int a,b;
int c=a+b;
return c;
}
```

OUTPUT:

Delemetersare :

( ) { , ; }

Operators are :

= +

Identifiers are :

a b c

Keywords are :

Int main void return

Constants are:

Literals are :

Process returned 0 (0x0) execution time : 26.408 s

Press any key to continue.

EXPERIMENT NO-2

AIM: Simulate First and Follow of a Grammar.

ALGORITHM:

FIRST :

Step 1: If x is a terminal, then FIRST(x) = { 'x' }

Step 2: If x-> Є, is a production rule, then add Є to FIRST(x).

Step 3: If X->Y1 Y2 Y3….Yn is a production,

i. FIRST(X) = FIRST(Y1)
ii. If FIRST(Y1) contains Є then FIRST(X) = { FIRST(Y1) – Є } U { FIRST(Y2) }
iii. If FIRST (Yi) contains Є for all i = 1 to n, then add Є to FIRST(X).

FOLLOW :

Step 1: First put $ (the end of input marker) in Follow(S) (S is the start symbol)

Step 2: If there is a production A → aBb, (where a can be a whole string) **then** everything in FIRST(b) except for ε is placed in FOLLOW(B).

Step 3: If there is a production A → aB, then everything in FOLLOW(A) is in FOLLOW(B)

Step 4: If there is a production A → aBb, where FIRST(b) contains ε, **then** everything in FOLLOW(A) is in FOLLOW(B)

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main(){
char fin[10][20],st[10][20],ft[20][20],fol[20][20];
int a=0,e,i,b,c,n,k,l=0,j,s,m,p;
clrscr();
printf("enter the no. of coordinates\n");
scanf("%d",&n);
```

```
printf("enter the productions in a grammar\n");
for(i=0;i<n;i++)
scanf("%s",st[i]);
for(i=0;i<n;i++)
fol[i][0]='\0';
for(s=0;s<n;s++){
for(i=0;i<n;i++){
j=3;
l=0;
a=0;
l1: if(!((st[i][j]>64)&&(st[i][j]<91))){
for(m=0;m<l;m++){
if(ft[i][m]==st[i][j])
goto s1;
}
ft[i][l]=st[i][j];
l=l+1;
s1:j=j+1;
}
else{
if(s>0){
while(st[i][j]!=st[a][0]){
a++;
}
b=0;
while(ft[a][b]!='\0'){
for(m=0;m<l;m++){
if (ft[i][m]==ft[a][b])
goto s2;
}
ft[i][l]=ft[a][b];
```

```
l=l+1;
s2:b=b+1;
}
}
}
while(st[i][j]!='\0'){
if(st[i][j]=='|'){
j=j+1;
goto l1;
}
j=j+1;
}
ft[i][l]='\0';
}
}
printf("First pos\n");
for(i=0;i<n;i++)
printf("FIRS[%c]=%s\n",st[i][0],ft[i]);
fol[0][0]='$';
for(i=0;i<n;i++){
k=0;
j=3;
if(i==0)
l=1;
else
l=0;
k1:while((st[i][0]!=st[k][j])&&(k<n)){
if(st[k][j]=='\0'){
k++;
j=2;
}
```

```
j++;
}
j=j+1;
if(st[i][0]==st[k][j-1]){
if((st[k][j]!='|')&&(st[k][j]!='\0')){
a=0;
if(!((st[k][j]>64)&&(st[k][j]<91))){
for(m=0;m<l;m++){
if(fol[i][m]==st[k][j])
goto q3;
}
fol[i][l]=st[k][j];
q3:l++;
}
else{
while(st[k][j]!=st[a][0]){
a++;
}
p=0;
while(ft[a][p]!='\0'){
if(ft[a][p]!='@'){
for(m=0;m<l;m++){
if(fol[i][m]==ft[a][p])
goto q2;
}
fol[i][l]=ft[a][p];
l=l+1;
}
else
e=1;
q2:p++;
```

```
}
if(e==1){
e=0;
goto a1;
}
}
}
else{
a1:c=0;
a=0;
while(st[k][0]!=st[a][0]){
a++;
}
while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0])){
for(m=0;m<l;m++){
if(fol[i][m]==fol[a][c])
goto q1;
}
fol[i][l]=fol[a][c];
l++;
q1:c++;
}
}
goto k1;
}
fol[i][l]='\0';
}
printf("Follow pos\n");
for(i=0;i<n;i++)
printf("FOLLOW[%c]=%s\n",st[i][0],fol[i]);
printf("\n");
```

```
getch();
}
```

OUTPUT:

```
enter the no. of coordinates
2
enter the productions in a grammar
S ->CC
C-eC|d
First pos
FIRS[S]=ed
FIRS[C]=ed
Follow pos
FOLLOW[S]=$
FOLLOW[C]=ed$
```

EXPERIMENT NO-03

AIM: Construct a LL(1) parser for an expression

DESCRIPTION:

An LL(1) grammar should allow us to disambiguate the choice of two rules by looking at the next token in the input string (the lookahead token). Whenever we have$A \longrightarrow \alpha \mid \beta$then

1. $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$
2. $\varepsilon \in FIRST(\alpha)$, i.e., A is nullable, then $FIRST(\beta) \cap FOLLOW(A) = \emptyset$

ALGORITHM:

Input:

string ω

parsing table M for grammar G

Output:

If ω is in L(G) then left-most derivation of ω,

error otherwise.

Initial State : $S on stack (with S being start symbol)

ω$ in the input buffer

SET ip to point the first symbol of ω$.

repeat

let X be the top stack symbol and a the symbol pointed by ip.

if X∈ Vt or $

if X = a

POP X and advance ip.

else

error()

endif

else /* X is non-terminal */

if M[X,a] = X → Y1, Y2,... Yk

POP X

PUSH Yk, Yk-1,... Y1 /* Y1 on top */

Output the production X → Y1, Y2,...Yk

else

error()

endif

endif

until X = $ /* empty stack */

PROGRAM:

```
#include<conio.h>

#include<string.h>

char s[20],stack[20];

void main(){

char m[5][6][3]={"tb"," "," ","tb"," "," "," ","+tb"," "," ","n","n","fc"," "," ","fc"," "," "," ","n"
,"*fc"," a","n","n","i"," "," ","(e)"," "," "};

int size[5][6]={2,0,0,2,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,0,0,3,0,0};

int i,j,k,n,str1,str2;

printf("\n Enter the input string: ");

scanf("%s",s);

strcat(s,"$");
```

```
n=strlen(s);
stack[0]='$';
stack[1]='e';
i=1;
 j=0;
printf("\nStack Input\n");
printf("_____\n");
while((stack[i]!='$')&&(s[j]!='$')) {
if(stack[i]==s[j]){
i--;
j++;
  }
switch(stack[i]){
case 'e': str1=0;
break;
case 'b': str1=1;
break;
case 't': str1=2;
break;
case 'c': str1=3;
break;
case 'f': str1=4;
break;
  }
switch(s[j]){
```

```
case 'i': str2=0;
break;
case '+': str2=1;
break;
case '*': str2=2;
break;
case '(': str2=3;
break;
case ')': str2=4;
break;
case '$': str2=5;
break;
}
if(m[str1][str2][0]=='\0'){
printf("\nERROR");
exit(0);
}
else if(m[str1][str2][0]=='n')
i--;
else if(m[str1][str2][0]=='i')
stack[i]='i';
else{
for(k=size[str1][str2]-1;k>=0;k--){
stack[i]=m[str1][str2][k];
i++;
```

```
}
i--;
}
for(k=0;k<=i;k++)
printf(" %c",stack[k]);
printf(" ");
for(k=j;k<=n;k++)
printf("%c",s[k]);
printf(" \n ");
}
printf("\n SUCCESS");
}
```

OUTPUT:

INPUT & OUTPUT:

Enter the input string: i*i+i

Stack INPUT

$bti*i+i$

$bcfi*i+i$

$bcii*i+i$

$bc *i+i$

$bcf* *i+i$

$bcfi+i$

$bcii+i$

$bc +i$

$b +i$

$bt+ +i$

bti

$bcfi$

$ bcii$

$bc $

$b $

$ $

Success.

EXPERIMENT NO-4

AIM: Design predictive parser for the given language.

ALGORITHM:

Step-1: Repeat: **for** each production Aα of the grammar do

Step-2:foreach terminal a in FIRST(α)add Aα to M[A, a]

Step-3:if FIRST(α) contains€add Aα to M[A, b]for each b in FOLLOW(A)

Step-4:if€ is in FIRST(α) and $ is in FOLLOW(A) add Aα to M[A, $]make each undefined entry of M be error.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main() {
char fin[10][20],st[10][20],ft[20][20],fol[20][20];
int a=0,e,i,t,b,c,n,k,l=0,j,s,m,p;
clrscr();
printf("enter the no. of coordinates\n");
scanf("%d",&n);
printf("enter the productions in a grammar\n");
for(i=0;i<n;i++)
scanf("%s",st[i]);
for(i=0;i<n;i++)
fol[i][0]='\0';
for(s=0;s<n;s++) {
for(i=0;i<n;i++) {
j=3;
l=0;
a=0;
l1:if(!((st[i][j]>64)&&(st[i][j]<91))) {
```

```
for(m=0;m<l;m++)  {
if(ft[i][m]==st[i][j])
goto s1;
}
ft[i][l]=st[i][j];
l=l+1;
s1:j=j+1;
}
else  {
if(s>0)  {
while(st[i][j]!=st[a][0])  {
a++;
}
b=0;
while(ft[a][b]!='\0')  {
for(m=0;m<l;m++)  {
if(ft[i][m]==ft[a][b])
goto s2;
}
ft[i][l]=ft[a][b];
l=l+1;
s2:b=b+1;
}
}
}
while(st[i][j]!='\0')  {
if(st[i][j]=='|')  {
j=j+1;
goto l1;
}
j=j+1;
```

```
}
ft[i][l]='\0';
}
}
printf("first pos\n");
for(i=0;i<n;i++)
printf("FIRST[%c]=%s\n",st[i][0],ft[i]);
fol[0][0]='$';
for(i=0;i<n;i++)  {
k=0;
j=3;
if(i==0)
l=1;
else
l=0;
k1:while((st[i][0]!=st[k][j])&&(k<n))  {
if(st[k][j]=='\0')  {
k++;
j=2;
}
j++;
}
j=j+1;
if(st[i][0]==st[k][j-1])  {
if((st[k][j]!='|')&&(st[k][j]!='\0'))  {
a=0;
if(!((st[k][j]>64)&&(st[k][j]<91)))  {
for(m=0;m<l;m++)  {
if(fol[i][m]==st[k][j])
goto q3;
}
```

```
fol[i][l]=st[k][j];
q3:l++;
}
else {
while(st[k][j]!=st[a][0]) {
a++;
}
p=0;
while(ft[a][p]!='\0') {
if(ft[a][p]!='@') {
for(m=0;m<l;m++) {
if(fol[i][m]==ft[a][p])
goto q2;
}
fol[i][l]=ft[a][p];
l=l+1;
}
else
e=1;
q2:p++;
}
if(e==1) {
e=0;
goto a1;
}
else {
a1:c=0;
a=0;
while(st[k][0]!=st[a][0]) {
a++;
}
```

```
}

while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0])) {
for(m=0;m<l;m++)  {
if(fol[i][m]==fol[a][c])
goto q1;
}
fol[i][l]=fol[a][c];
l++;
q1:c++;
}
}
goto k1;
}
fol[i][l]='\0';
}
printf("Follow pos\n");
for(i=0;i<n;i++)
printf("FOLLOW[%c]=%s\n",st[i][0],fol[i]);
printf("\n");
s=0;
for(i=0;i<n;i++) {
j=3;
while(st[i][j]!='\0') {
if((st[i][j-1]=='|')||(j==3))  {
for(p=0;p<=2;p++) {
fin[s][p]=st[i][p];
}
t=j;
for(p=3;((st[i][j]!='|')&&(st[i][j]!='\0'));p++)  {
fin[s][p]=st[i][j];
j++;
```

```
}
fin[s][p]='\0';
if(st[i][k]=='@') {
b=0;
a=0;
while(st[a][0]!=st[i][0]) {
a++;
}
while(fol[a][b]!='\0')  {
printf("M[%c,%c]=%s\n",st[i][0],fol[a][b],fin[s]);
b++;
}
}
else if(!((st[i][t]>64)&&(st[i][t]<91)))
printf("M[%c,%c]=%s\n",st[i][0],st[i][t],fin[s]);
else {
b=0;
a=0;
while(st[a][0]!=st[i][3]) {
a++;
}
while(ft[a][b]!='\0') {
printf("M[%c,%c]=%s\n",st[i][0],ft[a][b],fin[s]);
b++;
}
}
s++;
}
if(st[i][j]=='|')
j++;
}
```

```
}
getch();
}
}
```

OUTPUT:

```
Enter the no. of co-ordinates
2
Enter the productions in a grammar
S->CC
C->eC | d

First pos
FIRST[S] = ed
FIRST[C] = ed

Follow pos
FOLLOW[S] =$
FOLLOW[C] =ed$

    M [S , e] =S->CC
    M [S , d] =S->CC
    M [C , e] =C->eC
    M [C , d] =C->d
```

EXPERIMENT 1:

AIM : Become familiar with MATLAB /OCTAVE Open Source Basic commands

A. Read and display Image

```
a=imread('peppers.png');    %% reads the image
imshow(a);                  %% diplays the image
```

B. Resize given image

```
 img=imread('peppers.png');
 img2=imresize(img, [230,240]);
figure,subplot(1,2,1),imshow(img);
subplot(1,2,2),imshow(img2);
```

C. **Convert given color image into gray-scale image**

```
A=imread('peppers.png');
i=rgb2gray(A);
imshow(i)
```

(OR)

```
i = imread('peppers.png');
R = i(:, :, 1);
G = i(:, :, 2);
B = i(:, :, 3);
newImage = zeros(size(i,1), size(i,2),'uint8');
for x=1:size(i,1)
for y=1:size(i,2)
newImage(x,y) = (R(x,y)*.3)+(G(x,y)*.6)+(B(x,y)*.1);
```

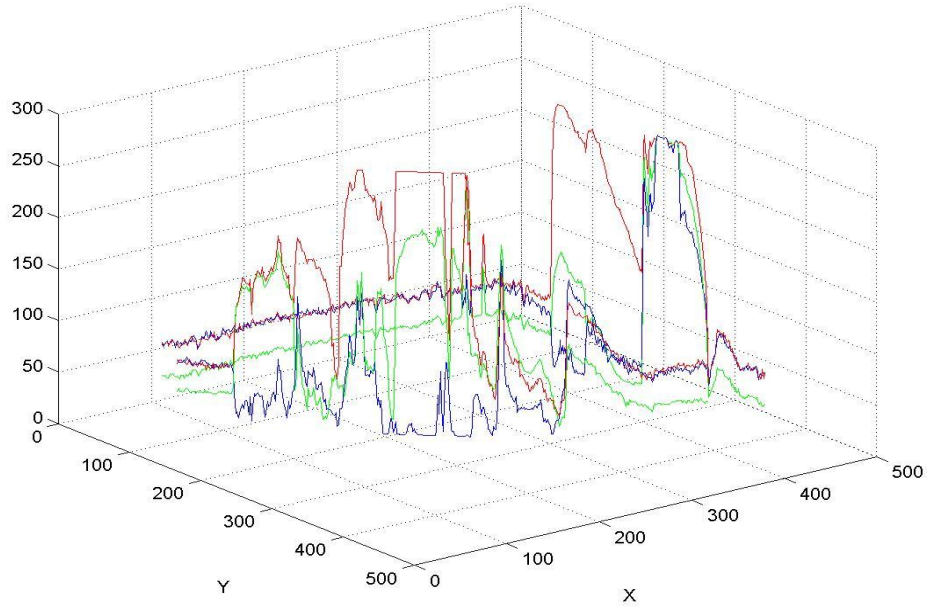
```
end
end
imshow(newImage);
```

D. Convert given color/gray-scale image into black & white image

```
A=imread('peppers.png');
i=im2bw(A);
imshow(i);
```

E. Draw image profile

```
I=imread('peppers.png');
x=[75 477 426 79];
y=[69 452 67 42];
improfile(I,x,y),grid on;
```



F. Create color image using R, G and B three separate planes

```
A=imread('peppers.png');
row = size(A, 1);
col = size(A, 2);
R = zeros(row, col, 3);
G = zeros(row, col, 3);
B = zeros(row, col, 3);
R= A(:, :, 1);
```

```
G= A(:, :, 2);
B= A(:, :, 3);
allblack=zeros(size(A, 1), size(A, 2), 'uint8');
red=cat(3,R,allblack,allblack);
green=cat(3,allblack,G,allblack);
blue=cat(3,allblack,allblack,B);
figure,subplot(1,3,1), imshow(uint8(red)),title('red image');
subplot(1,3,2), imshow(uint8(green)),title('green image');
subplot(1,3,3), imshow(uint8(blue)),title('blue image');
```

G. Brightness supression

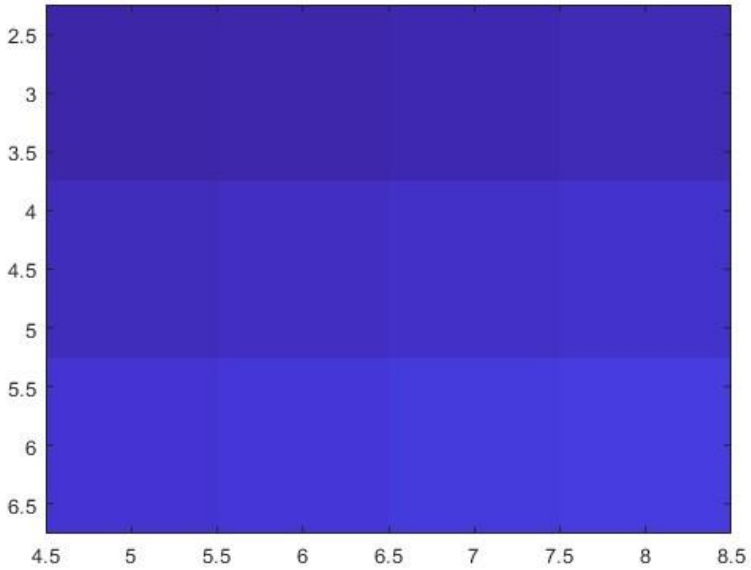
```
clc;
close;
a = imread('peppers.png');
b = double(a) -150;
b = uint8(b);
figure,subplot(1,2,1)
imshow(a);
title( 'Original Image')
subplot(1,2,2),imshow(b);
title('Brightness Supressed Image');
```

Original Image

Brightness Supressed Image

H. Write given 2-D data in image file

```
x = [5 8];
y = [3 6];
C = [0 2 4 6; 8 10 12 14; 16 18 20 22];
image(x,y,C)
```

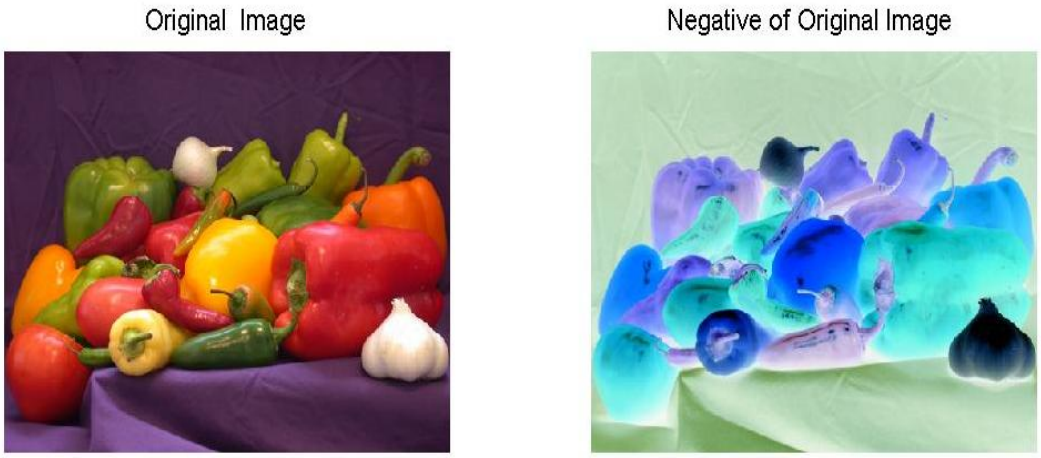


EXPERIMENT 2 :

Aim: To write and execute image processing programs using point processing method

A. Obtain Negative image

```
clc;
close;
a = imread( 'peppers.png');
k = 255-double(a);
k = uint8(k);
figure,subplot(1,2,1),imshow(a);
title( 'Original  Image')
subplot(1,2,2),imshow(k);
title( 'Negative of Original Image ')
```



B. Obtain Flip image

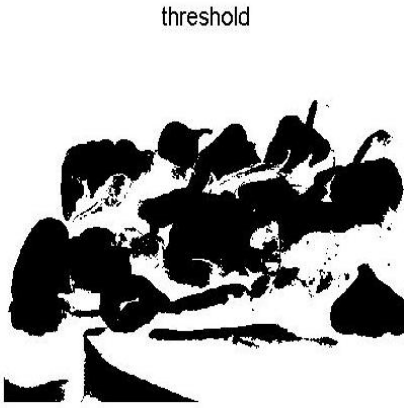
```
clear all;
x= imread('peppers.png');
figure,subplot(1,2,1),imshow(x),title('orginalimage');

b=flipdim(x,2);
subplot(1,2,2),imshow(b),title('Flippedimage')
```

orginalimage

Flippedimage

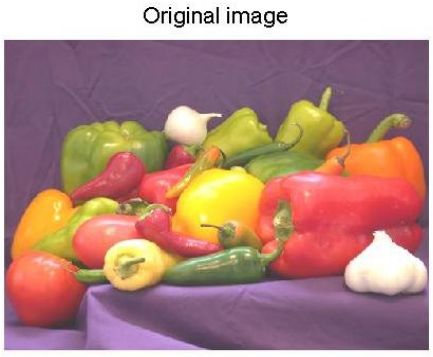
C. Thresholding

```
x= imread('peppers.png');
figure,subplot(1,2,1),imshow(x),title('orginalimage');
x=rgb2gray(x);
x=double(x);
tot=0;
[a,b]=size(x);
y=zeros(a,b);
for i=1:a
        for j=1:b
        y(i,j)=0;
        end
end
for i=1:a
        for j=1:b
        tot=tot+x(i,j);
        end
end
thr=tot/(a*b);
for i=1:a
        for j=1:b
        if x(i,j) > thr
                y(i,j)=0;
        else
                y(i,j)=1;
end
        end
end
subplot(1,2,2),imshow(y),title('threshold')
```



D. Contrast stretching

```
I = imread('peppers.png');
figure
subplot(1,2,1),imshow(I+50),title('Original image')
J = imadjust(I,stretchlim(I),[]);
subplot(1,2,2),imshow(J),title('Contrasted Image')
```



EXPERIMENT 3:

AIM : To write and execute program for geometric transformation of image

A. Translation

```
I = imread('peppers.png');
se = translate(strel(1), [30,65]);
J = imdilate(I,se);
figure,subplot(1,2,1),imshow(I), title('Original')
subplot(1,2,2),imshow(J), title('Translated');
```

B. Scaling

```
img=imread('peppers.png');
img2=imresize(img, [230,240]);
figure,subplot(1,2,1),imshow(img);
subplot(1,2,2),imshow(img2);
```

C. Rotation

```
I = imread('peppers.png');
theta=45
k=imrotate(J,theta);
subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(k);
```

D. Shrinking

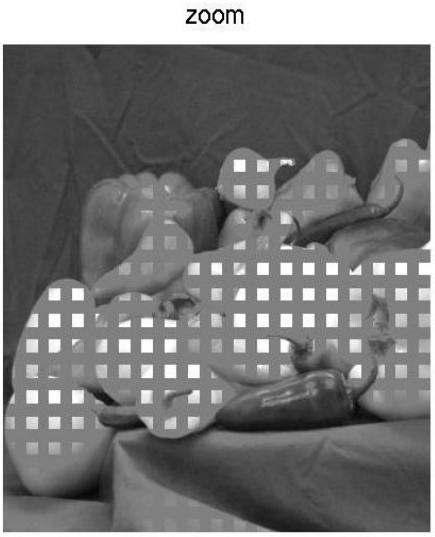
```
clc;
clear all;
close all;
A=imread('peppers.png');
display('Input Image ==> peppers.png');
f=input('Enter the shrinking factor of the image: ');
s=size(A);
s1=s/f;
k=1;
l=1;
for i=1:s1
        for j=1:s1
        B(i,j)=A(k,l);
        l=l+f;
        end
        l=1;
        k=k+f;
end
figure,imshow(A)
title('Original Image');
figure,imshow(B)
title('Shrinked Version');
subplot(1,2,1),imshow(A),title('orginal image');
```

```
subplot(1,2,2),imshow(B),title('shrinked version')
```

E. Zooming

```
clc;
clear all;
close all;
A=imread('peppers.png');
display('Input Image ==> peppers.png');
f1=input('Enter the factor by which the image is to be Zoomed: ');
s=size(A);
s2=s*f1;
k=1;
l=1;
for (i=1:f1:s2)
        for( j=1:f1:s2)
        C(i,j)= A(k,l);
        l=l+1;
        end
        l=1;
        k=k+1;
end
for (i=1:f1:s2)
        for (j=2:f1:s2-1)
        C(i,j)= [C(i,j-1)+ C(i, j+1)]*0.5;
        end
end
for(j=1:f1:s2)
        for(i=2:f1:s2-1)
```

```
        C(i,j)=[C(i-1,j)+C(i+1,j)]*0.5;
        end
end
for (i=2:f1:s2-1)
        for (j=2:f1:s2-1)
        C(i,j)= [C(i,j-1)+ C(i, j+1)]*0.5;
        end
end
figure,subplot(1,2,1),imshow(A),title('orginal');
subplot(1,2,2),imshow(C),title('zoom');
```

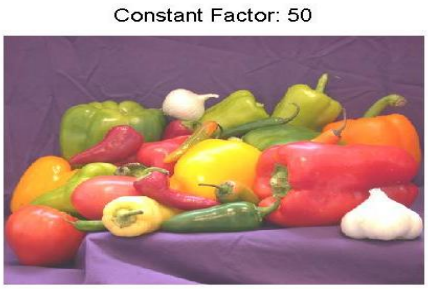


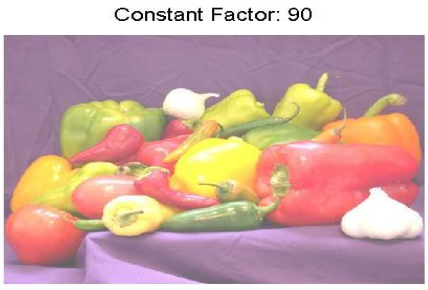
EXPERIMENT 4

Aim : To Perform algebraic operations such as addition, subtraction, multiplication, and division. These operations can be used to perform operations on images such as noise reduction by using averages, movement detection, and algebraic masking.

A. Addition

```
I = imread('peppers.png');
J = imadd(I,50);
K = imadd(I,90);
subplot(2,2,1), imshow(I), title('peppers.png');
subplot(2,2,2), imshow(J),title('Constant Factor: 50');
subplot(2,2,3), imshow(K),title('Constant Factor: 90');
```





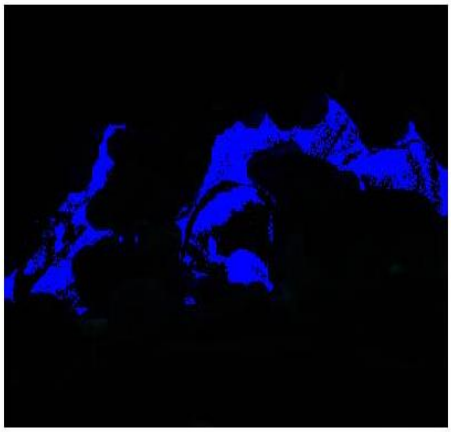
B. Multiply

```
I = imread('peppers.png');
```

```
J = immultiply(I,2.5);
figure,subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(J)
```

C. **Division**:

```
I = imread('peppers.png');
background = imopen(I, strel('disk',15));
Ip = imdivide(I,background);
figure,subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(Ip)
```



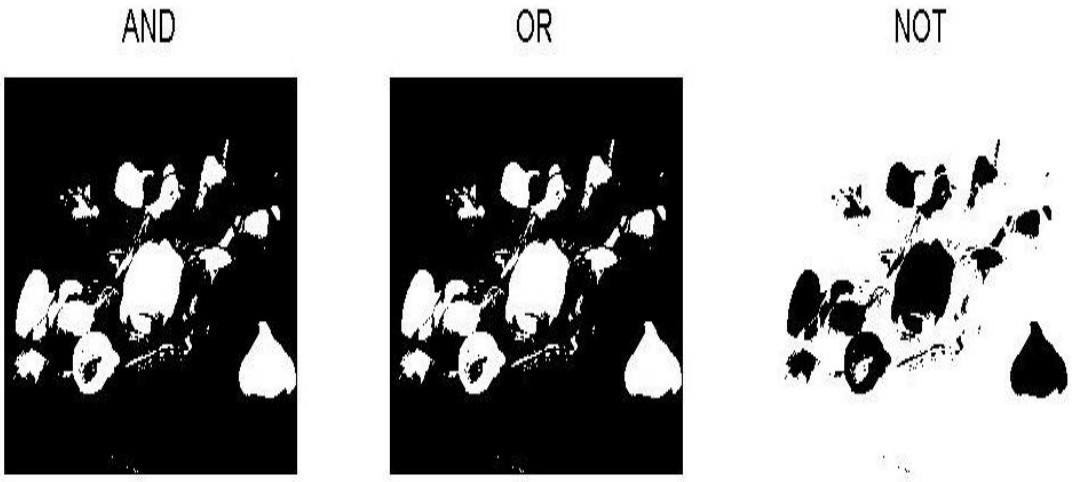
D. Subtraction

```
I = imread('peppers.png');
background = imopen(I, strel('disk',15));
Ip = imsubtract(I,background);
figure,subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(Ip)
```

Experiment 5

Aim : To write and execute programs for image Logical operations

```
close all;
a=imread('peppers.png');
b=imread('peppers.png');
i=im2bw(a);
j=im2bw(b);
d1=and(i,j);
d2=or(i,j);
d3=not(i);
subplot(1,3,1),imshow(d1),title('AND');
subplot(1,3,2),imshow(d2),title('OR');
subplot(1,3,3),imshow(d3),title('NOT')
```



Experiment 6

Aim: To perform operations on images and introduce different noises

A. Salt and pepper Noise

```
I = imread('peppers.png');
imshow(I)
J = imnoise(I,'salt & pepper',0.02);
imshow(J)
```

B. Gaussian Noise

```
I = imread('peppers.png');
imshow(I)
J = imnoise(I,'gaussian',0.02);
imshow(J)
```

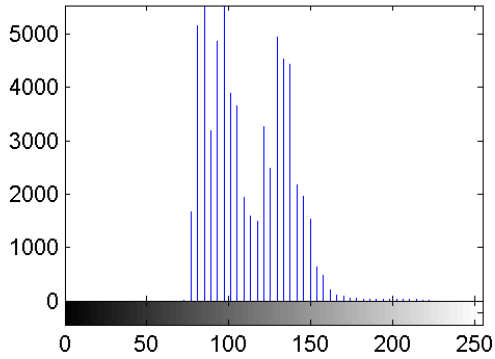
C. Speckle Noise

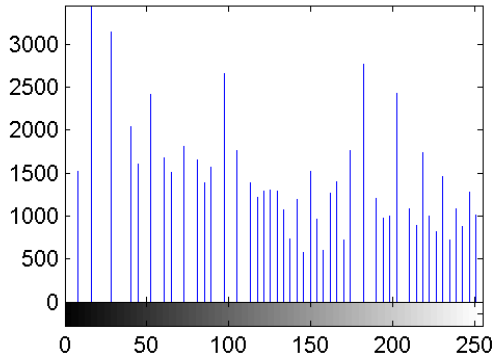
```
I = imread('peppers.png');
imshow(I)
J = imnoise(I,'speckle',0.02);
imshow(J)
```

Experiment 7

Aim : To perform Histogram equalization on images

```
I = imread('pout.tif');
figure
subplot(2,2,1)
imshow(I)
subplot(2,2,2)
imhist(I,64)
J = histeq(I);
subplot(2,2,3)
imshow(J)
subplot(2,2,4)
imhist(J,64)
```





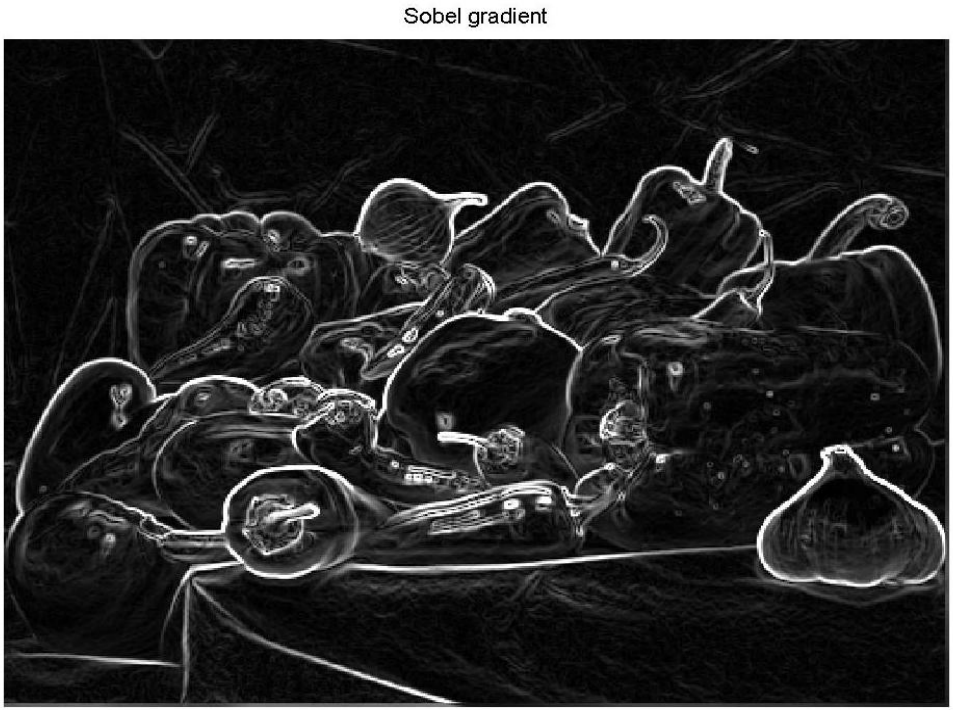
Experiment 8

Aim : To Implement

A) Edge Detection
B) Line Detection

A. Edge Detection

```
A=imread('peppers.png');
B=rgb2gray(A);
C=double(B);
for i=1:size(C,1)-2
        for j=1:size(C,2)-2
%Sobel mask for x-direction:
                Gx=((2*C(i+2,j+1)+C(i+2,j)+C(i+2,j+2))-(2*C(i,j+1)+C(i,j)+C(i,j+2)));
 %Sobel mask for y-direction:
        Gy=((2*C(i+1,j+2)+C(i,j+2)+C(i+2,j+2))-(2*C(i+1,j)+C(i,j)+C(i+2,j)));

%The gradient of the image
%B(i,j)=abs(Gx)+abs(Gy);
 B(i,j)=sqrt(Gx.^2+Gy.^2);

end
end
figure,imshow(B),title('Sobel gradient');
```



B. Line Detection

```
I  = imread('cameraman.tif');
fig1 = imshow(rotI);
BW = edge(rotI,'canny');
[H,theta,rho] = hough(BW);
xlabel('\theta (degrees)'), ylabel('\rho');
axis on, axis normal, hold on;
colormap(hot)
P = houghpeaks(H,5,'threshold',ceil(0.3*max(H(:))));
x = theta(P(:,2));
y = rho(P(:,1));
plot(x,y,'s','color','black');
lines = houghlines(BW,theta,rho,P,'FillGap',5,'MinLength',7);
figure,subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(rotI), hold on
max_len = 0;
for k = 1:length(lines)
        xy = [lines(k).point1; lines(k).point2];
        plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
        % Plot beginnings and ends of lines
        plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
        plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');
        % Determine the endpoints of the longest line segment
        len = norm(lines(k).point1 - lines(k).point2);
```

```
if ( len > max_len)
            max_len = len;
 xy_long = xy;
            end
end
% highlight the longest line segment
plot(xy_long(:,1),xy_long(:,2),'LineWidth',2,'Color','red');
```