PROGRAM 1 Implement Brenham's line drawing algorithm for all types of slope.

```
#include<GL/glut.h>
#include<stdio.h>
int x1, y1, x2, y2;
void draw_pixel(int x, int y)
{
glColor3f(1.0,0.0,0.0);
glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd();
}
void brenhams_line_draw(int x1, int y1, int x2, int y2)
{
int dx=x2-x1,dy=y2-y1;
int p=2*dy*dx;
int twoDy=2*dy;
int twoDyMinusDx=2*(dy-dx);
int x=x1,y=y1;
if(dx<0)
{
x=x2;
y=y2;
x2=x1;
}
draw_pixel(x, y);
while(x<x2)
{
x++;
if(p<0)
p+=twoDy;
else
```

```
{
y++;
p+=twoDyMinusDx;
}
draw_pixel(x, y);
}
}
void myInit()
{
glClearColor(0.0,0.0,0.0,1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 500.0, 0.0, 500.0);
glMatrixMode(GL_MODELVIEW);
}
void display ()
{
glClear(GL_COLOR_BUFFER_BIT);
brenhams_line_draw(x1, y1, x2, y2);
glFlush();
}
void main (int argc, char **argv)
{
printf("Enter Start Points (x1,y1)\n");
scanf("%d%d", &x1,&y1);
printf("Enter End Points (x2,y2)\n");
scanf("%d%d", &x2,&y2);
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(0, 0);
```

```
glutCreateWindow("Bresenham's Line Drawing");

myInit();

glutDisplayFunc(display);

glutMainLoop();

}
```

02.Develop a program to demonstrate basic geometric operations on the 2D object

```
#include <stdio.h>
#include <GL/glut.h>
typedef float point2[2];

/* initial triangle */

point2 v[]={{-1.0, -0.58}, {1.0, -0.58}, {0.0, 1.15}};
int n;

/* display one triangle  */
void triangle( point2 a, point2 b, point2 c)
{
   glBegin(GL_TRIANGLES);
      glVertex2fv(a);
      glVertex2fv(b);
      glVertex2fv(c);
   glEnd();
}

void divide_triangle(point2 a, point2 b, point2 c, int m)
{

/* triangle subdivision using vertex numbers */

   point2 v0, v1, v2;
   int j;
   if(m>0)
   {
      for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;
      for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;
      for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;
      divide_triangle(a, v0, v1, m-1);
      divide_triangle(c, v1, v2, m-1);
      divide_triangle(b, v2, v0, m-1);
   }
   else(triangle(a,b,c));    /* draw triangle at end of recursion */
}
void display(void)
{
```

```
   glClear(GL_COLOR_BUFFER_BIT);
   divide_triangle(v[0], v[1], v[2], n);
   glFlush();
}

void myinit()
{

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
   glMatrixMode(GL_MODELVIEW);
   glClearColor (1.0, 1.0, 1.0, 1.0);
         glColor3f(0.0,0.0,0.0);
}

void main(int argc, char **argv)
{
  printf(" No. of Subdivisions : ");
  scanf("%d",&n);
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
   glutInitWindowSize(500, 500);
   glutCreateWindow("Sierpinski Gasket 2D triangle");
   glutDisplayFunc(display);
         myinit();
   glutMainLoop();
}
```

03.Develop a program to demonstrate basic geometric operations on the 3D object

```
#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
typedef float point[3];

/* initial tetrahedron */
point v[]={{0.0, 0.0, 1.0}, {0.0, 0.942809, -0.33333},
        {-0.816497, -0.471405, -0.333333}, {0.816497, -0.471405, -0.333333}};
static GLfloat theta[] = {0.0,0.0,0.0};
int n;
void triangle( point a, point b, point c)
/* display one triangle using a line loop for wire frame, a single
normal for constant shading, or three normals for interpolative shading */
{
   glBegin(GL_POLYGON);
     glNormal3fv(a);
     glVertex3fv(a);
```

```
      glVertex3fv(b);
      glVertex3fv(c);
   glEnd();
}
/* triangle subdivision using vertex numbers
righthand rule applied to create outward pointing faces */
void divide_triangle(point a, point b, point c, int m)
{
   point v1, v2, v3;
   int j;
   if(m>0)
   {
      for(j=0; j<3; j++) v1[j]=(a[j]+b[j])/2;
      for(j=0; j<3; j++) v2[j]=(a[j]+c[j])/2;
      for(j=0; j<3; j++) v3[j]=(b[j]+c[j])/2;
      divide_triangle(a, v1, v2, m-1);
      divide_triangle(c, v2, v3, m-1);
      divide_triangle(b, v3, v1, m-1);
   }
   else(triangle(a,b,c)); /* draw triangle at end of recursion */
}
/* Apply triangle subdivision to faces of tetrahedron */
void tetrahedron( int m)
{
                     glColor3f(1.0,0.0,0.0);
                     divide_triangle(v[0], v[1], v[2], m);
                    glColor3f(0.0,1.0,0.0);
                    divide_triangle(v[3], v[2], v[1], m);
                    glColor3f(0.0,0.0,1.0);
                    divide_triangle(v[0], v[3], v[1], m);
                    glColor3f(0.0,0.0,0.0);
                    divide_triangle(v[0], v[2], v[3], m);
}

void display()
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
   tetrahedron(3);
   glFlush();
}

void myReshape(int w, int h)
{
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
   glOrtho(-2.0, 2.0, -2.0 * (GLfloat) h / (GLfloat) w,
         2.0 * (GLfloat) h / (GLfloat) w, -10.0, 10.0);
```

```
    else
        glOrtho(-2.0 * (GLfloat) w / (GLfloat) h,
            2.0 * (GLfloat) w / (GLfloat) h, -2.0, 2.0, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
    glutPostRedisplay();
}

void main(int argc, char **argv)
{
        int i = 0;
        printf("Enter value of N:");
        scanf("%d", &i);
        n = i;
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("3D tetrahedron Gasket");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glEnable(GL_DEPTH_TEST);
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glutMainLoop();
}
```

04.Develop a program to demonstrate 2D transformation on basic objects

```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>

GLfloat t[3][3]={{10.0,30.0,20.0},{20.0,20.0,40.0},{1.0,1.0,1.0}};
GLfloat rotatemat[3][3]={{0},{0},{0}};
GLfloat result[3][9]={{0},{0},{0}};

GLfloat xr=10.0;
GLfloat yr=20.0;

GLfloat theta;
GLint ch;

void multiply(){
        int i,j,k;
        for(i=0;i<3;i++)
                for(j=0;j<9;j++){
                        result[i][j]=0;
                        for(k=0;k<3;k++)
                                result[i][j]=result[i][j]+rotatemat[i][k]*t[k][j];
                }
}

void rotate_about_origin(){
```

```
        rotatemat[0][0]=cos(theta);
        rotatemat[0][1]=-sin(theta);
        rotatemat[0][2]=0;
        rotatemat[1][0]=sin(theta);
        rotatemat[1][1]=cos(theta);
        rotatemat[1][2]=0;
        rotatemat[2][0]=0;
        rotatemat[2][1]=0;
        rotatemat[2][2]=1;
        multiply();
}

void rotate_about_fixed_point(){
        GLfloat m,n;
        m=xr*(1-cos(theta))+yr*sin(theta);
        n=yr*(1-cos(theta))-xr*sin(theta);
        rotatemat[0][0]=cos(theta);
        rotatemat[0][1]=-sin(theta);
        rotatemat[0][2]=m;
        rotatemat[1][0]=sin(theta);
        rotatemat[1][1]=cos(theta);
        rotatemat[1][2]=n;
        rotatemat[2][0]=0;
        rotatemat[2][1]=0;
        rotatemat[2][2]=1;
        multiply();
}

void draw_triangle(){
        glLineWidth(10);
        glBegin(GL_LINE_LOOP);
        glColor3f(1.0,0.0,0.0);
        glVertex2f(t[0][0],t[1][0]);
        glColor3f(0.0,1.0,0.0);
        glVertex2f(t[0][1],t[1][1]);
        glColor3f(0.0,0.0,1.0);
        glVertex2f(t[0][2],t[1][2]);
        glEnd();
        glFlush();
}

void draw_rotated_triangle(){
        glLineWidth(10);
        glBegin(GL_LINE_LOOP);
        glColor3f(1.0,0.0,0.0);
        glVertex2f(result[0][0],result[1][0]);
        glColor3f(0.0,1.0,0.0);
        glVertex2f(result[0][1],result[1][1]);
        glColor3f(0.0,0.0,1.0);
        glVertex2f(result[0][2],result[1][2]);
```

```
	glEnd();
	glFlush();
}

void display(){
	glClear(GL_COLOR_BUFFER_BIT);
	if(ch==1){
		draw_triangle();
		rotate_about_origin();
		draw_rotated_triangle();
		glFlush();
	}
	if(ch==2){
		draw_triangle();
		rotate_about_fixed_point();
		draw_rotated_triangle();
		glFlush();
	}
}

void myinit(){
	glClearColor(1.0,1.0,1.0,1.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(-50.0,50.0,-50.0,50.0);
}

int main(int argc,char** argv){
	printf("***Rotation***\n1.Rotation about origin\n2.Rotation about a fixed point
(xr,yr)\n");
	printf("Enter choice\n");
	scanf("%d",&ch);
	printf("Enter the rotation angle\n");
	scanf("%f",&theta);
	theta=theta*(3.14/180);
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutInitWindowSize(500,500);
	glutInitWindowPosition(0,0);
	glutCreateWindow("Triangle Rotation\n");
	glutDisplayFunc(display);
	myinit();
	glutMainLoop();
	return 0;
}
```

05.Develop a program to demonstrate 3D transformation on basic objects

```
#include <stdlib.h>
#include <GL/glut.h>
	GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
```

```
                                {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
                                {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};

        GLfloat colors[][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},
                               {1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},
                               {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0}};

void polygon(int a, int b, int c , int d)
{
        glBegin(GL_POLYGON);
                glColor3fv(colors[a]);
            glVertex3fv(vertices[a]);
                glColor3fv(colors[b]);
                glVertex3fv(vertices[b]);
                glColor3fv(colors[c]);
                glVertex3fv(vertices[c]);
                glColor3fv(colors[d]);
                glVertex3fv(vertices[d]);
        glEnd();
}

void colorcube()
{
        polygon(0,3,2,1);
        polygon(2,3,7,6);
        polygon(0,4,7,3);
        polygon(1,2,6,5);
        polygon(4,5,6,7);
        polygon(0,1,5,4);
}

static GLfloat theta[] = {0.0,0.0,0.0};
static GLint axis = 2;
static GLdouble viewer[]= {0.0, 0.0, 5.0}; /* initial viewer location */

void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
/* Update viewer position in modelview matrix */
    glLoadIdentity();
    gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
          /* rotate cube */
glRotatef(theta[0], 1.0, 0.0, 0.0);
glRotatef(theta[1], 0.0, 1.0, 0.0);
glRotatef(theta[2], 0.0, 0.0, 1.0);
colorcube();      /* draw the rotated color cube */
glFlush();
glutSwapBuffers();
}
```

```
void mouse(int btn, int state, int x, int y)
{
if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis =1;
if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
 theta[axis] += 4.0;
if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
display();
}

void keys(unsigned char key, int x, int y)
{
/* Use x, X, y, Y, z, and Z keys to move viewer */
   if(key == 'x') viewer[0]-= 1.0;
   if(key == 'X') viewer[0]+= 1.0;
   if(key == 'y') viewer[1]-= 1.0;
   if(key == 'Y') viewer[1]+= 1.0;
   if(key == 'z') viewer[2]-= 1.0;
   if(key == 'Z') viewer[2]+= 1.0;
   display();
}

void myReshape(int w, int h)
{
 glViewport(0, 0, w, h);
/* Use a perspective view */
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
    if(w<=h) glFrustum(-2.0, 2.0, -2.0 * (GLfloat) h/ (GLfloat) w,
       2.0* (GLfloat) h / (GLfloat) w, 2.0, 20.0);
          else glFrustum(-2.0, 2.0, -2.0 * (GLfloat) w/ (GLfloat) h,
       2.0* (GLfloat) w / (GLfloat) h, 2.0, 20.0);
            /* Or we can use gluPerspective that is   gluPerspective(45.0, w/h, -10.0, 10.0); */
 glMatrixMode(GL_MODELVIEW);
}

void  main(int argc, char **argv)
{
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
 glutInitWindowSize(500, 500);
 glutCreateWindow("Colorcube Viewer");
 glutReshapeFunc(myReshape);
 glutDisplayFunc(display);
glutMouseFunc(mouse);
glutKeyboardFunc(keys);
glEnable(GL_DEPTH_TEST);
glutMainLoop();
}
```

06.Develop a program to demonstrate Animation effects on simple objects.

```
#include<GL/glut.h>

#include<stdio.h>
#include<math.h>

#define PI 3.1416

static int win,val=0,CMenu;

void CreateMenu(void);
void Menu(int value);

struct wcPt3D
{
GLfloat x, y, z;
};


GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = 0.0, xwcMax = 130.0;
GLfloat ywcMin = 0.0, ywcMax = 130.0;
void bino(GLint n, GLint *C)
{
GLint k, j;
for(k=0;k<=n;k++)
{
C[k]=1;
for(j=n;j>=k+1; j--)

C[k]*=j;
for(j=n-k;j>=2;j--)
C[k]/=j;

}
}
void computeBezPt(GLfloat u,struct wcPt3D *bezPt, GLint nCtrlPts,struct wcPt3D *ctrlPts,
GLint *C)
{
GLint k, n=nCtrlPts-1;
GLfloat bezBlendFcn;
bezPt ->x =bezPt ->y = bezPt->z=0.0;
for(k=0; k< nCtrlPts; k++)
{

bezBlendFcn = C[k] * pow(u, k) * pow( 1-u, n-k); bezPt ->x += ctrlPts[k].x * bezBlendFcn;
bezPt ->y += ctrlPts[k].y * bezBlendFcn;
bezPt ->z += ctrlPts[k].z * bezBlendFcn;
}
}
```

```
void bezier(struct wcPt3D *ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)
{
struct wcPt3D bezCurvePt;
GLfloat u;
GLint *C, k;
C= new GLint[nCtrlPts];
bino(nCtrlPts-1, C);

glBegin(GL_LINE_STRIP);
for(k=0; k<=nBezCurvePts; k++)

{
u=GLfloat(k)/GLfloat(nBezCurvePts);
computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);
glVertex2f(bezCurvePt.x, bezCurvePt.y);
}
glEnd();
delete[]C;
}
void displayFcn()
{
GLint nCtrlPts = 4, nBezCurvePts =20;

static float theta = 0;
struct wcPt3D ctrlPts[4] = {{20, 100, 0},{30, 110, 0},{50, 90, 0},{60, 100, 0}};
ctrlPts[1].x +=10*sin(theta * PI/180.0);
ctrlPts[1].y +=5*sin(theta * PI/180.0);
ctrlPts[2].x -= 10*sin((theta+30) * PI/180.0);
ctrlPts[2].y -= 10*sin((theta+30) * PI/180.0);
ctrlPts[3].x-= 4*sin((theta) * PI/180.0);
ctrlPts[3].y += sin((theta-30) * PI/180.0);
theta+=0.1;
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);

glPointSize(5);
//Indian Flag
if(val==1){
glPushMatrix();
glLineWidth(5);
glColor3f(1.0,0.5,0); //Indian flag: Orange color code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(1,1,1); //Indian flag: white color code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
```

```
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(0,1.0,0); //Indian flag: green color code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glPopMatrix();
glColor3f(0.7, 0.5,0.3);
glLineWidth(5);

glBegin(GL_LINES);
glVertex2f(20,100);

glVertex2f(20,40);
glEnd();
glFlush();
}
//Karnataka Flag
if(val==2){
glPushMatrix();
glLineWidth(5);
glColor3f(1.0, 1.0, 0.0); //Karnataka flag: Yellow color code
for(int i=0;i<12;i++)
{

glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(1, 0.0, 0.0); //Karnataka flag: Red color code
for(int i=0;i<12;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glPopMatrix();
glColor3f(0.7, 0.5,0.3);

glLineWidth(5);
glBegin(GL_LINES);
glVertex2f(20,100);
glVertex2f(20,40);
glEnd();
glFlush();
}
glutPostRedisplay();
glutSwapBuffers();
}
```

```
void winReshapeFun(GLint newWidth, GLint newHeight)
{
glViewport(0, 0, newWidth, newHeight);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax); glClear(GL_COLOR_BUFFER_BIT); }


void CreateMenu(void)
{
CMenu= glutCreateMenu(Menu);//Creaate Menu Option
glutAddMenuEntry("Indian Flag",1);
glutAddMenuEntry("Karnataka Flag",2);
glutAddMenuEntry("Exit",0);
glutAttachMenu(GLUT_RIGHT_BUTTON);
}
void Menu(int value)
{
if(value==0)
{
glutDestroyWindow(win);
exit(0);
}
else {
val=value;
}
}
int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowPosition(50, 50);
glutInitWindowSize(winWidth, winHeight);
glutCreateWindow("Prg. 8 Bezier Curve");
CreateMenu();
glutDisplayFunc(displayFcn);
glutReshapeFunc(winReshapeFun);
glutMainLoop();
```

7. Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image_pat="flower.jpeg"
img=cv2.imread(image_pat)
height,width=img.shape[:2]
quad1=img[:height//2,width//2]
quad2=img[:height//2,width//2:]
quad3=img[height//2:,:width//2]
```

```
quad4=img[height//2:,width//2:]
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.imshow(quad1)
plt.title("1")
plt.axis("off")

plt.subplot(1,2,2)
plt.imshow(quad2)
plt.title("2")
plt.axis("off")

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.imshow(quad3)
plt.title("3")
plt.axis("off")

plt.subplot(1,2,2)
plt.imshow(quad4)
plt.title("4")
plt.axis("off")

plt.show()
```

8. Write a program to show rotation, scaling, and translation on an image.

```
import cv2
import numpy as np
def translate_image(image,dx,dy):
    rows,cols=image.shape[:2]
    translation_matrix=np.float32([[1,0,dx],[0,1,dy]])
    translated_image=cv2.warpAffine(image,translation_matrix,(cols,rows))
    return translated_image
image=cv2.imread('flower.jpeg')
height,width=image.shape[:2]
center=(width//2,height//2)
rotation_value=int(input("enter the degree of rotation"))
scaling_value=int(input("enter the zooming factor:"))
rotated=cv2.getRotationMatrix2D(center=center , angle=rotation_value,scale=1)
rotated_image=cv2.warpAffine(src=image,M=rotated,dsize=(width,height))
scaled=cv2.getRotationMatrix2D(center=center , angle=0,scale=scaling_value)
scaled_image=cv2.warpAffine(src=rotated_image,M=scaled,dsize=(width,height))
h=int(input("how many pixels you want image to be translated horizontally?"))
v=int(input("how many pixels you want image to be translated vertically?"))
translated_image=translate_image(scaled_image,dx=h,dy=v)
cv2.imwrite('final_image.png',translated_image)
```

9. Read an image and extract and display low-level features such as edges, textures using filtering techniques.

```
import cv2
import numpy as np
image_path="flower.jpeg"
```

```
img=cv2.imread(image_path)
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges=cv2.Canny(gray,100,200)
kernel=np.ones((5,5),np.float32)/25
texture=cv2.filter2D(gray,-1,kernel)
cv2.imshow("Original Image",img)
cv2.imshow("Edges",edges)
cv2.imshow("texture",texture)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

10. Write a program to blur and smoothing an image.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread("flower.jpeg", cv2.IMREAD_GRAYSCALE)
image_array = np.array(img)
print(image_array)
def sharpen():
    return np.array([[1,1,0], [1,0,1], [0,1,1]])
def filtering(image, kernel):
    m, n = kernel.shape
    if (m == n):
        y, x =image.shape
        y=y-m+1
        x=x-m+1
        new_image= np.zeros((y,x))
        for i in range(y):
            for j in range(x):
                new_image[i][j] = np.sum(image[i:i+m, j:j+m]*kernel)
    return new_image
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_array,cmap='gray')
plt.title("Original Grayscale Image")
plt.axis("off")
plt.subplot(1, 2, 2)
plt.imshow(filtering(image_array, sharpen()), cmap='gray')
plt.title("Blurred Image")
plt.axis("off")
plt.show()
```

11. Write a program to contour an image.

```
import cv2
import numpy as np
image_path='flower.jpeg'
image=cv2.imread(image_path)
gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
_, binary_image=cv2.threshold(gray,127,255,cv2.THRESH_BINARY)
contours,_=cv2.findContours(binary_image,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX
_SIMPLE)
cv2.drawContours(image,contours,-1,(0,255,0),3)
```

```
cv2.imshow('Contours',image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

12. Write a program to detect a face/s in an image.

```
import cv2
face_cascade=cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalface_default
.xml')
eye_cascade=cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_eye.xml')
image_path='viratkohli.jpg'
image=cv2.imread(image_path)
gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
faces=face_cascade.detectMultiScale(gray,scaleFactor=1.3,minNeighbors=5)
for(x,y,w,h) in faces:
    cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)
    cv2.imwrite('detected_faces.jpg',image)
    cv2.imshow('Detected Faces',image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```