```
/*1. Implement Bresenham's line drawing algorithm for all types of slope.*/
#include<math.h>
#include<stdio.h>
#include<GL/glut.h>

int x1, y11, x2, y2,dx,dy;
void display();
void init();
void bresenhams(int,int,int,int);
void main(int argc,char**argv)
{
        glutInit(&argc,argv);
        printf("enter the end points of the line");
        scanf("%d%d%d%d", &x1, &y11, &x2, &y2);
        glutCreateWindow("Bresenhams Line Drawing");
        init();
        glutDisplayFunc(display);
        glutMainLoop();
}
void init()
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-500, 500, -500, 500);
        glMatrixMode(GL_MODELVIEW);
}
void display()
{
        glClearColor(1, 1, 1, 0);
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1, 0, 0);
        bresenhams(x1, y11, x2, y2);
        glFlush();

}
void plotline(int x, int y)
{
        glPointSize(2);
        glBegin(GL_POINTS);
                glVertex2f(x,y);
        glEnd();
}
void bresenhams(int x1, int y11, int x2, int y2)
{
        int dx, dy,pk,xinc,yinc,x,y;
        dx = x2 - x1;
        dy = y2 - y11;
        x = x1, y = y11;
        plotline(x, y);
        if (dx > 0)
        xinc = 1;
    else
        xinc = -1;
    if (dy > 0)
        yinc = 1;
    else
        yinc = -1;
    if (fabs(dx) > fabs(dy))
    {
        pk = 2 * fabs(dy) - fabs(dx);
        for (int i = 0; i <= fabs(dx) - 1; i++)
        {
            if (pk > 0)
            {
                pk = pk + 2 * fabs(dy) - 2 * fabs(dx);
                y = y+yinc;
            }
            else
            {
                pk = pk + 2 * fabs(dy);
                y = y;
            }
            x = x + xinc;
            plotline(x, y);
        }
    }
    else
    {
        pk = 2 * fabs(dx) - fabs(dy);
        for (int i = 0; i <= fabs(dy) - 1; i++)
        {

            if (pk > 0)
            {
                pk = pk + 2 * fabs(dx) - 2 * fabs(dy);
                x = x + xinc;
            }
            else
            {
                pk = pk + 2 * fabs(dx);
                x = x;
            }
            y = y + yinc;
            plotline(x, y);
        }
    }
}
```

```
/*2. Write a program in OpenGL that demonstrates basic 2D geometric transformations
such as transalation, rotation and scaling. Allow the user to interactively apply these
transformations to a 2D object  */
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
void init()
{
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(-500, 500, -500, 500);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}
void display()
{
	glClearColor(0,0,0,0);
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(1,1,0);
	glBegin(GL_TRIANGLES);
		glVertex2f(-100,-100);
		glVertex2f(0,200);
		glVertex2f(100,-100);
		glVertex2f(-100,100);
		glVertex2f(0,-200);
		glVertex2f(100,100);
	glEnd();
	glFlush();
}
void menu(int id)
{
	if (id == 1)
		glTranslatef(-10,0,0);
	if (id == 2)
		glTranslatef(10,0,0);
	if (id == 3)
		glTranslatef(0,10,0);
	if (id == 4)
		glTranslatef(0,-10,0);
	if (id==5)
		glRotatef(10,0,0,1);
	if (id==6)
		glRotatef(-10,0,0,1);
	if (id==7)
		glScalef(0.5,0.5,0);
	if (id==8)
		glScalef(1.5,1.5,0);
```

```
        glutPostRedisplay();

}

void main(int argc,char**argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(10,10);
        glutCreateWindow("2D Transformations");
        init();
        glutDisplayFunc(display);
        int translate=glutCreateMenu(menu);
        glutAddMenuEntry("Left",1);
        glutAddMenuEntry("Right",2);
        glutAddMenuEntry("Up",3);
        glutAddMenuEntry("Down",4);
        int rotation=glutCreateMenu(menu);
        glutAddMenuEntry("Anticlockwise",5);
        glutAddMenuEntry("Clockwise",6);
        int scaling=glutCreateMenu(menu);
        glutAddMenuEntry("Minimize",7);
        glutAddMenuEntry("Maximize",8);
        glutCreateMenu(menu);
        glutAddSubMenu("Translate",translate);
        glutAddSubMenu("Rotate",rotation);
        glutAddSubMenu("Scaling",scaling);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        glutMainLoop();
}
```

```
/*3. Develop a program to demonstarte 3D transformation on 3D objets */
#include<stdio.h>
#include<stdlib.h>
#include<GL/glut.h>
GLfloat
v[8][3]={{-200,-200,200},{200,-200,200},{200,200,200},{-200,200,200},{-200,-200,-200},
{200,-200,-200},{200,200,-200},{-200,200,-200}};
void drawcube(GLfloat *,GLfloat *,GLfloat *,GLfloat *);

void init()
{
        glClearColor(0.0,0.0,0.0,0.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-500,500,-500,500,-500,2000);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}
void display()
{

        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        //glLoadIdentity();
        glRotatef(0.01,1.0,0.0,1.0);
        glColor3f(1.0,0.6,0.3);
        drawcube(v[0],v[1],v[2],v[3]);
        glColor3f(1.0,0.7,0.3);
        drawcube(v[1],v[5],v[6],v[2]);
        glColor3f(1.0,0.0,0.0);
        drawcube(v[3],v[2],v[6],v[7]);
        glColor3f(0.0,1.0,0.0);
        drawcube(v[4],v[5],v[1],v[0]);
        glColor3f(0.0,0.0,1.0);
        drawcube(v[7],v[6],v[5],v[4]);
        glColor3f(1.0,1.0,0.3);
        drawcube(v[3],v[7],v[4],v[0]);
        glFlush();
}
void drawcube(GLfloat *a,GLfloat *b,GLfloat *c,GLfloat *d)
{
        glBegin(GL_POLYGON);
                glVertex3fv(a);
                glVertex3fv(b);
                glVertex3fv(c);
                glVertex3fv(d);
        glEnd();
}
void keys(unsigned char k,int x,int y)
{
        if(k=='s')
                glScalef(0.5,0.5,0.5);
        if(k=='S')
                glScalef(1.5,1.5,1.5);
        if(k=='t')
                glTranslatef(10,10,10);
        if(k=='T')
                glTranslatef(-10,-10,-10);
        glutPostRedisplay();
}
void spincube()
{
        glutPostRedisplay();
}
void main(int argc, char *argv[])
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowPosition(10,10);
        glutInitWindowSize(500,500);
        glutCreateWindow("3D Transformation");
        init();
        glutDisplayFunc(display);
        glutKeyboardFunc(keys);
        glutIdleFunc(spincube);
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
}
```

```
/*4. Write a program that takes an RGB color as input and converts it to its corresponding
CMY values*/
#include <stdio.h>

int main()
{

        float R,G,B;
        float C,M,Y,K,W,Rf,Gf,Bf,max;
        printf("Enter the values of R,G & B: ");
        scanf("%f%f%f",&R,&G,&B);
        if (R<0||R>255)
        {
        printf("Enter R within limit\n");
        scanf("%f",&R);
        }
        if (G<0||G>255)
        {
        printf("Enter G within limits\n");
        scanf("%f",&G);
        }
        if (B<0||B>255)
        {
        printf("Enter B within limits\n");
        scanf("%f",&B);
        }
        printf("\nR,G,B: %f,%f,%f\n",R,G,B);

        Rf = R/255;
        Gf = G/255;
        Bf = B/255;

        printf("***********************************\n");
        printf("RGB to CMY Values\n");
        W = 1;
        printf("White: %f\n", W);
        C = W-Rf;
        M = W-Gf;
```

```
    Y = W-Bf;
    printf("The value of Cyan: %f\n", C);
    printf("The value of Magenta: %f\n", M);
    printf("The value of Yellow: %f\n", Y);
    printf("****************************************\n");

    printf("RGB to CMYK Values\n");
    if (R == 0 && G == 0 && B == 0)
    {
        printf("The value of Cyan: 0\n");
        printf("The value of Magenta: 0\n");
        printf("The value of Yellow: 0\n");
        printf("The value of Black: 1\n");
    }
    else
    {
        max = Rf;
        if (max<Gf)
           max = Gf;
        if (max<Bf)
           max = Bf;
        W = max;
        printf("White: %f\n", W);
        C = (W-Rf)/W;
        M = (W-Gf)/W;
        Y = (W-Bf)/W;
        K = 1- W;
        printf("The value of Cyan: %f\n", C);
        printf("The value of Magenta: %f\n", M);
        printf("The value of Yellow: %f\n", Y);
        printf("The value of Black: %f\n", K);
    }
    printf("****************************************\n");
}
```

```
#5. Create a program that captures user input to dynamically adjust the
#the properties of a shape(eg.,size,color)
import cv2
import numpy as np

# Callback function for trackbars
def on_change(value):
    img = cv2.rectangle(image, (0,0), (550,550), (0,0,0), -1)
    cv2.imshow("Shape", img)
    pass

# Create a window
cv2.namedWindow("Shape Properties")
image = np.zeros((550,550,3), np.uint8)
# Initial values
initial_width = 200
initial_height = 100
initial_color = (0, 255, 0)  # Green
choice=1
# Create trackbars
cv2.createTrackbar("Width", "Shape Properties", initial_width, 500, on_change)
cv2.createTrackbar("Height", "Shape Properties", initial_height, 500, on_change)
cv2.createTrackbar("Red", "Shape Properties", initial_color[0], 255, on_change)
cv2.createTrackbar("Green", "Shape Properties", initial_color[1], 255, on_change)
cv2.createTrackbar("Blue", "Shape Properties", initial_color[2], 255, on_change)

while True:
    # Get current trackbar values
    width = cv2.getTrackbarPos("Width", "Shape Properties")
    height = cv2.getTrackbarPos("Height", "Shape Properties")
    red = cv2.getTrackbarPos("Red", "Shape Properties")
    green = cv2.getTrackbarPos("Green", "Shape Properties")
    blue = cv2.getTrackbarPos("Blue", "Shape Properties")
    # Draw the rectangle with dynamically adjusted properties
    shape = (width, height)
    color = (blue, green, red)  # OpenCV uses BGR format

    img = cv2.rectangle(image, (10,10), shape, color, -1)
    # Display the image
    cv2.imshow("Shape", img)
    # Check for key press
    key = cv2.waitKey(1) & 0xFF
    if key == ord("q"):
        break
cv2.destroyAllWindows()
```

```
#6. Write a program to read a digital image.split an image into
#four quadrants up,down,right and left
import cv2

img = cv2.imread('images.jpeg')

# cv2.imread() -> takes an image as an input
h, w, channels = img.shape

half1 = w//2
half2 = h//2

# this will be the first column
up_left = img[:half2, :half1]

up_right = img[:half2, half1:]

down_left = img[half2:, :half1]

down_right = img[half2:, half1:]

cv2.imshow('Original image', img)
cv2.imshow('Up-Left part', up_left)
cv2.imshow('Up-Right part', up_right)
cv2.imshow('Down-Left part', down_left)
cv2.imshow('Down-Right part', down_right)

# saving all the images
# cv2.imwrite() function will save the image
# into your pc
```

```
cv2.imwrite('up_left.jpg', up_left)
cv2.imwrite('up_right.jpg', up_right)
cv2.imwrite('down_left.jpg', down_left)
cv2.imwrite('down_right.jpg', down_right)
```

```
#7. Read an image and extract and display low level features such as
#edges, textures using filtering techniques
import cv2
import numpy as np

# Load the image
image_path = "images.jpeg" # Replace with the path to your image
img = cv2.imread(image_path)

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Edge detection
edges = cv2.Canny(gray, 100, 200) # Use Canny edge detector

# Texture extraction
kernel = np.ones((5, 5), np.float32) / 25 # Define a 5x5 averaging kernel
texture = cv2.filter2D(gray, -1, kernel) # Apply the averaging filter for texture
                                     #extraction

# Display the original image, edges, and texture
cv2.imshow("Original Image", img)
cv2.imshow("Edges", edges)
cv2.imshow("Texture", texture)

# Wait for a key press and then close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
#8. write a program to blur and smoothing an image
import cv2
# Load the image
image = cv2.imread('images.jpeg')

# Average Blur
average_blur = cv2.blur(image, (5, 5))

# Gaussian Blur
gaussian_blur = cv2.GaussianBlur(image, (5, 5), 0)

# Median Blur
median_blur = cv2.medianBlur(image, 5)

# Bilateral Filter
bilateral_filter = cv2.bilateralFilter(image, 9, 75, 75)

# Display the original and processed images
cv2.imshow('Original Image', image)
cv2.imshow('Average Blur', average_blur)
cv2.imshow('Gaussian Blur', gaussian_blur)
cv2.imshow('Median Blur', median_blur)
cv2.imshow('Bilateral Filter', bilateral_filter)

# Wait for a key press to close the windows
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

```
#9. write a program for image segmentation by
#using edge based segmentation
import cv2
import numpy as np

# Read the image
image = cv2.imread('images.jpeg')

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply Canny edge detection
edges = cv2.Canny(gray, 100, 200)

# Find contours in the edged image
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Create a mask with the same dimensions as the image
mask = np.zeros_like(gray)

# Draw contours on the mask
cv2.drawContours(mask, contours, -1, (255), thickness=cv2.FILLED)

# Apply the mask to the original image
```

```
segmented_image = cv2.bitwise_and(image, image, mask=mask)

# Display the original image and the segmented image
cv2.imshow('Original Image', image)
cv2.imshow('Segmented Image', segmented_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```