DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

COMPUTER GRAPHICS LABORATORY

REFERENCE GUIDE

18CSL67

AS PER REVISED VTU CBCS SCHEME

Prepared by:

Mr. Akhilesh S Narayan,
Asst. Professor

Mr. Navile Nageshwara Naveen,
Asst. Professor

Dept. of CSE

★ **PROGRAM 1**: Implement Bresenham's line drawing algorithm for all types of slope.

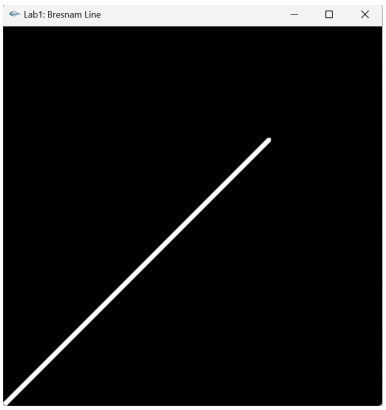
Note: Compilation command - gcc <filename>.c -lGL -lGLU -lglut -lm

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

int xstart, ystart, xend, yend;

void init()
{
    gluOrtho2D(0, 500, 0, 500);
}

void draw_pixel(int x, int y)
{
    glColor3f(1, 1, 1);
    glPointSize(5);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}

void LineBres(int xstart, int ystart, int xend, int yend)
{
    int dx = fabs(xend - xstart);
    int dy = fabs(yend - ystart);
    int twody = 2 * dy, twodyminusdx = 2 * (dy - dx);
    int p = 2 * dy - dx;
    int x, y;

    if (xstart > xend)
    {
        x = xend;
        y = yend;
        xend = xstart;
    }
    else
    {
        x = xstart;
        y = ystart;

    }
    draw_pixel(x, y);

    while (x < xend)
    {
        x++;
        if (p < 0)
```

```
            p += twody;
        else
        {
            y++;
            p += twodyminusdx;

        }

        draw_pixel(x, y);

    }

}

void Display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(0, 0, 0, 1);

    LineBres(xstart, ystart, xend, yend);
    glEnd();
    glFlush();
}

int main(int argc, char** argv)
{
    printf("Enter 2 points\n");
    scanf("%d%d%d%d", &xstart, &ystart, &xend, &yend);

    glutInit(&argc, argv);
    glutInitWindowPosition(50, 50);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Lab1: Bresnam Line");
    init();
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

EXPECTED OUTPUT:

```
Enter 2 points
0 0
350 350
```



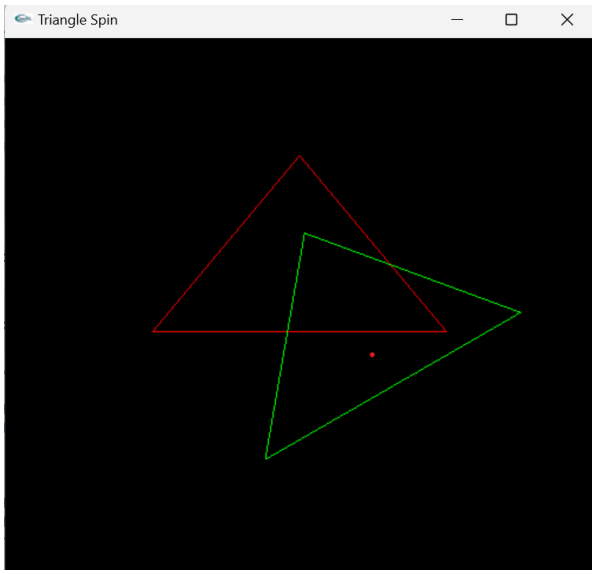
★ **PROGRAM 2**: Create and rotate a triangle about the origin & a fixed point.

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

GLfloat d;
GLfloat rX=0,rY=0;

void Spin()
{
    d = d + 0.1;
    if(d > 360)
    d = 0;
    glutPostRedisplay();
}

void Draw()
{
    GLfloat P[3][2] = {{-0.5,0},{0.5,0},{0,0.6}};
    GLfloat nP[3][2],r;
    int i;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1,0,0);
    glBegin(GL_LINE_LOOP);
    glVertex2fv(P[0]);
    glVertex2fv(P[1]);
    glVertex2fv(P[2]);
    glEnd();
    r = d * 3.14/180;
    for(i=0;i<3;i++)
    {
        nP[i][0] = P[i][0] * cos(r) - P[i][1] * sin(r) - rX*cos(r) +
rY*sin(r) + rX;
        nP[i][1] = P[i][0] * sin(r) + P[i][1] * cos(r) - rX*sin(r) -
rY*cos(r) + rY;

    }

    glColor3f(0,1,0);
    glBegin(GL_LINE_LOOP);
    glVertex2fv(nP[0]);
    glVertex2fv(nP[1]);
    glVertex2fv(nP[2]);

    glEnd();
    glFlush();
}

int main(int argC, char *argV[])
{
    printf("Enter the pivot point for rotation: ");
    scanf("%f%f", &rX, &rY);
```

```
    printf("\n Enter the degree of rotation: ");
    scanf("%f", &d);
    glutInit(&argC, argV);
    glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
    glutInitWindowPosition(0,0);
    glutInitWindowSize(500,500);
    glutCreateWindow("Triangle Spin");
    glutDisplayFunc(Draw);
    glutIdleFunc(Draw);
    glutMainLoop();
    return 0;
}
```

EXPECTED OUTPUT:

```
Enter the pivot point for rotation: 0.5 0.5

 Enter the degree of rotation: 30
```



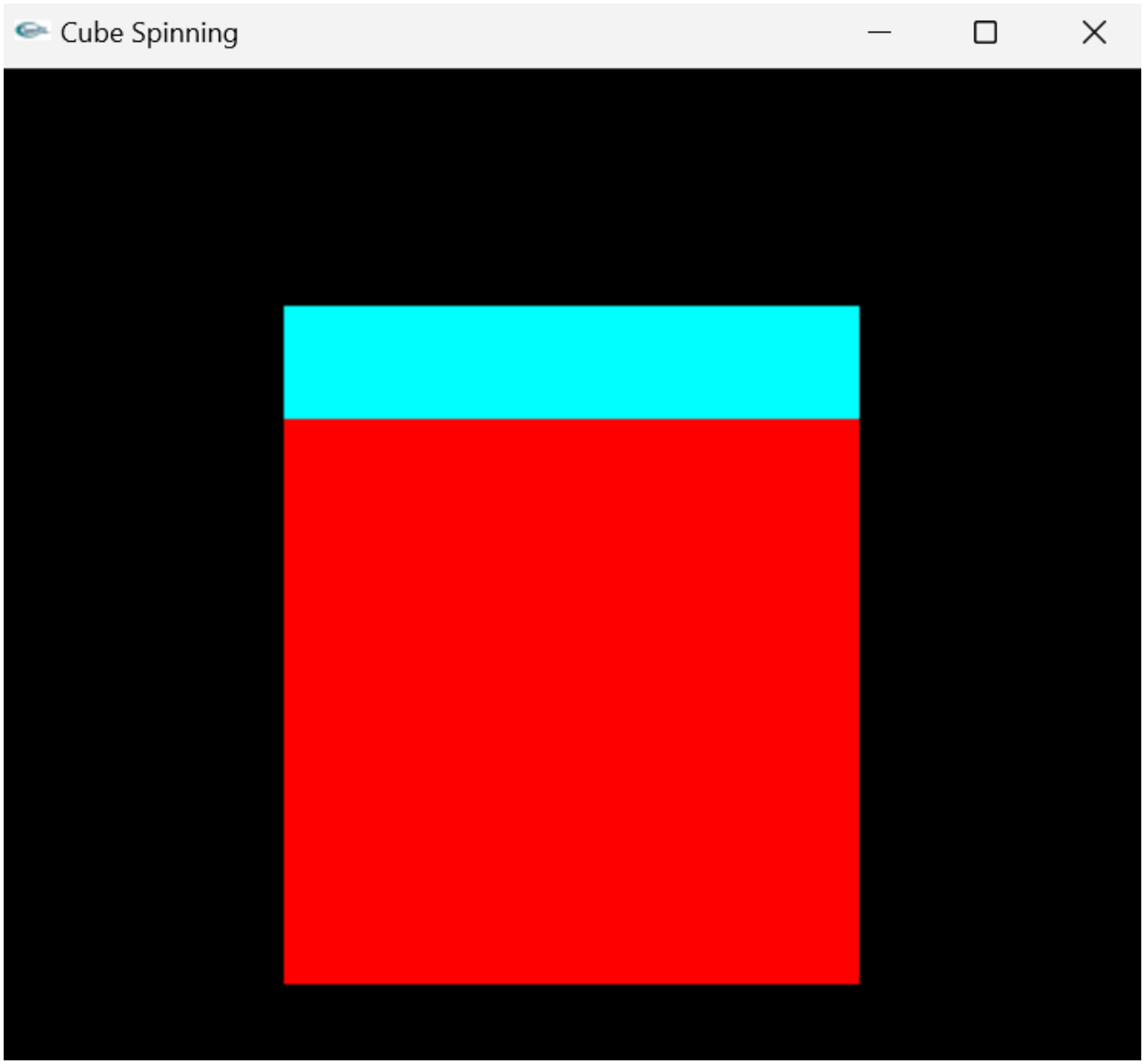
★ **PROGRAM 3**: Draw a color cube & spin it using transformation matrices.

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

GLfloat d = 0;
char a;

void Spin()
{
    d = d + 1;
    if(d > 360)
    d = 0;
    glutPostRedisplay();
}

void Face(GLfloat A[3], GLfloat B[3], GLfloat C[3], GLfloat D[3])
{
    glBegin(GL_QUADS);
    glVertex3fv(A);
    glVertex3fv(B);
    glVertex3fv(C);
    glVertex3fv(D);
    glEnd();
}

void Cube(GLfloat P1[3], GLfloat P2[3], GLfloat P3[3], GLfloat
P4[3], GLfloat P5[3], GLfloat P6[3], GLfloat P7[3], GLfloat P8[3])
{
    glColor3f(1.0, 0.0, 0.0);
    Face(P1, P2, P3, P4);
    glColor3f(0,1,0);
    Face(P5, P6, P7, P8);
    glColor3f(0.0, 0.0, 1.0);
    Face(P1, P5, P8, P4);
    glColor3f(1.0, 1.0, 0.0);
    Face(P2, P6, P7, P3);
    glColor3f(1.0, 0.0, 1.0);
    Face(P1, P2, P6, P5);
    glColor3f(0.0, 1.0, 1.0);
    Face(P4, P3, P7, P8);
}

void Draw()
{
    GLfloat V[8][3] = {
        {-0.5, -0.5, -0.5},
        {0.5, -0.5, -0.5},
        {0.5, 0.5, -0.5},
        {-0.5, 0.5, -0.5},
        {-0.5, -0.5, 0.5},
        {0.5, -0.5, 0.5},
```

```
        {0.5, 0.5, 0.5},
        {-0.5, 0.5, 0.5}
    };
    GLfloat nV[8][3], r;
    int i;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    r = d*3.14/180;
    if(a == 'z' || a == 'Z')
    {
        for (i=0;i<8;i++)
        {
            nV[i][0] = V[i][0] * cos(r) - V[i][1]*sin(r);
            nV[i][1] = V[i][0] * sin(r) + V[i][1]*cos(r);
            nV[i][2] = V[i][2];
        }
    }
    if(a == 'x' || a == 'X')
    {
        for (i=0;i<8;i++)
        {
            nV[i][0] = V[i][0];
            nV[i][1] = V[i][1] * cos(r) - V[i][2]*sin(r);
            nV[i][2] = V[i][1] * sin(r) + V[i][2]*cos(r);
        }
    }
    if(a == 'y' || a == 'Y')
    {
        for (i=0;i<8;i++)
        {
            nV[i][0] = V[i][0]*cos(r) + V[i][2]*sin(r);
            nV[i][1] = V[i][1];
            nV[i][2] = -V[i][0]*sin(r) + V[i][2]*cos(r);
        }
    }

    Cube(nV[0], nV[1], nV[2], nV[3], nV[4], nV[5], nV[6], nV[7]);
    glutSwapBuffers();

}

int main(int argC, char *argV[])
{
    printf("\n Enter the Axis of Rotation: ");
    scanf("%c", &a);
    glutInit(&argC, argV);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowPosition(0,0);
    glutInitWindowSize(500,500);
    glutCreateWindow("Cube Spinning");
    glutDisplayFunc(Draw);
    glutIdleFunc(Spin);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
    return 0; }
```

EXPECTED OUTPUT:

```
PS C:\Users\AKHILESH\GL> .\a.exe

 Enter the Axis of Rotation: x
```



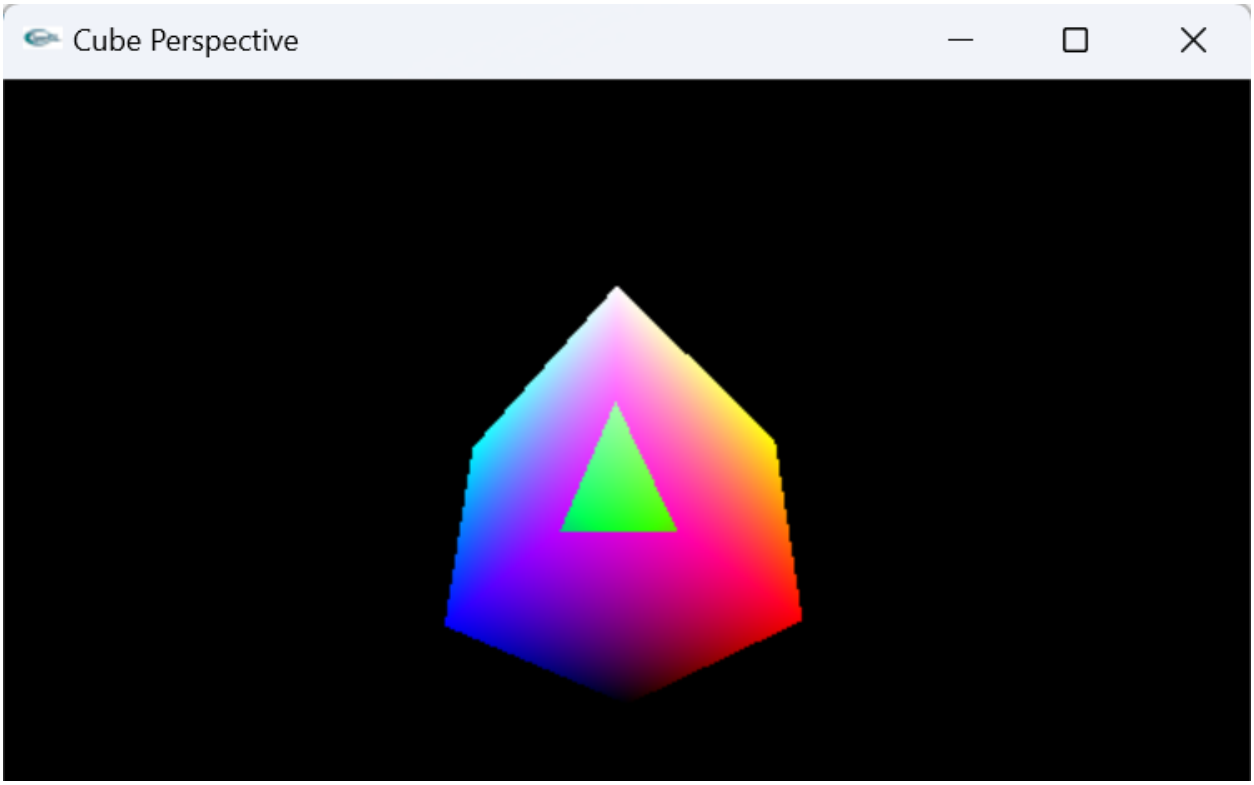
★ **PROGRAM 4**: Draw a color cube & allow the user to move the camera suitably to experiment with perspective viewing.

```
#include "stdio.h"
#include<GL/glut.h>
float
ver[8][3]={{0,0,0},{1,0,0},{1,1,0},{0,1,0},{0,0,1},{1,0,1},{1,1,1},{
0,1,1}};
float v1[3]={0,0,5};
void polygon(int a,int b,int c,int d);
void polygon1();
void display()
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(v1[0],v1[1],v1[2],0,0,0,0,1,0);
    polygon1();
    glFlush();
}
void init()
{
    glClearColor(0.0,0.0,0.0,1.0);
}
void polygon1()
{
    polygon(0,1,2,3);
            polygon(4,5,6,7);
                polygon(5,1,2,6);
                    polygon(4,0,3,7);
                        polygon(4,5,1,0);
                            polygon(7,6,2,3);
}
void polygon(int a,int b,int c,int d)
{
glBegin(GL_POLYGON);
glColor3fv(ver[a]);
glVertex3fv(ver[a]);
glColor3fv(ver[b]);
glVertex3fv(ver[b]);
glColor3fv(ver[c]);
glVertex3fv(ver[c]);
glColor3fv(ver[d]);
glVertex3fv(ver[d]);
glEnd();
}
void key(unsigned char f,int x,int y)
{
    if(f=='x')v1[0]-=.10;
    if(f=='X')v1[0]+=.10;
    if(f=='y')v1[1]-=.10;
    if(f=='Y')v1[1]+=.10;
    if(f=='z')v1[2]-=.10;
```

```
    if(f=='Z')v1[2]+=.10;
    display();
}
void Reshape(int w,int h)
{
    glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if(w<=h)
        glFrustum(-2.0,2.0,-2.0*w/h,2.0*w/h,2.0,20);
    else
        glFrustum(-2.0,2.0,-2.0*w/h,2.0*w/h,2.0,20);
        glMatrixMode(GL_MODELVIEW);
}
    void main()
    {
        glutInitDisplayMode(GLUT_SINGLE|GLUT_DEPTH);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(10,10);
        glutCreateWindow("Cube Perspective");
        init();
        glutDisplayFunc(display);
        glutKeyboardFunc(key);
        glEnable(GL_DEPTH_TEST);
        glutReshapeFunc(Reshape);
        glutMainLoop();
    }
```

Note: x, X, y, Y, z, Z are the keys to change cube perspective.

EXPECTED OUTPUT:

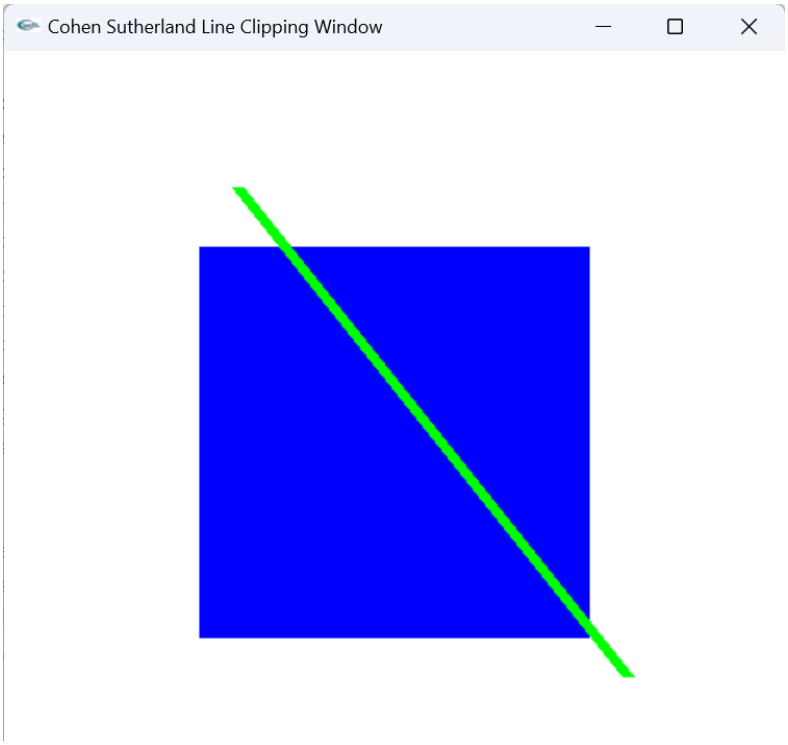


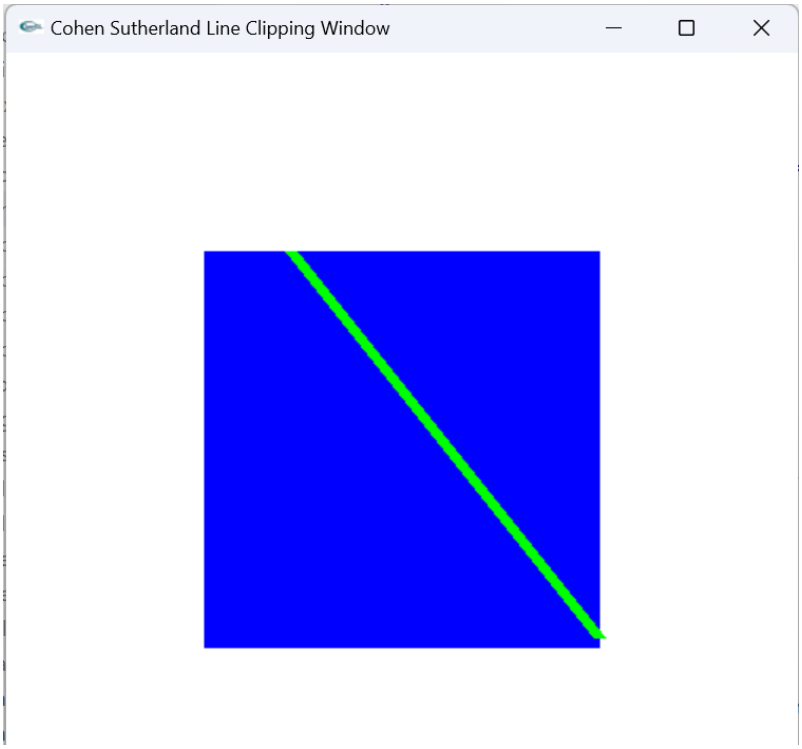
★ **PROGRAM 5**: Clip lines using Cohen-Sutherland algorithm.

```
#include<GL/glut.h>
GLfloat Xmin = -0.5, Xmax = 0.5, Ymin = -0.5, Ymax = 0.5;
GLfloat X1 = -0.4, Y1 = 0.65, X2 = 0.6, Y2 = -0.6;
int Left=1, Right=2, Bottom=4, Top=8;
int C1, C2;
int Flag = 0;
int Get_Code(GLfloat x, GLfloat y)
{
    int C = 0;
    if(x < Xmin)
    C = C | Left;
    if(x > Xmax)
    C = C | Right;
    if(y < Ymin)
    C = C | Bottom;
    if(y > Ymax)
    C = C | Top;
    return C;
}

void Clip()
{
    GLfloat X,Y;
    int C;
    if(C1)
    C = C1;
    else
    C = C2;
    if(C & Left)
    {
        X = Xmin;
        Y = Y1 + (Y2-Y1) * ((Xmin - X1) / (X2-X1));

    }
    if(C & Right)
    {
        X = Xmax;
        Y = Y1 + (Y2-Y1) * ((Xmax - X1) / (X2-X1));
    }
    if (C & Bottom)
    {
        Y = Ymin;
        X = X1 + (X2-X1) * ((Ymin - Y1) / (Y2-Y1));
    }
    if (C & Top)
    {
        Y = Ymax;
        X = X1 + (X2-X1) * ((Ymax - Y1) / (Y2-Y1));
    }
    if (C == C1)
    {
```

```
            X1 = X;
            Y1 = Y;
        }
        else
        {
            X2 = X;
            Y2 = Y;
        }

}

void Display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0,0.0,1.0);
    glRectf(Xmin,Ymin,Xmax,Ymax);
    glColor3f(0.0,1.0,0.0);
    glLineWidth(8);
    glBegin(GL_LINES);
    glVertex2f(X1,Y1);
    glVertex2f(X2,Y2);
    glEnd();
    while (1 && Flag)
    {
        C1 = Get_Code(X1,Y1);
        C2 = Get_Code(X2,Y2);
        if((C1|C2) == 0)
        break;
        else if ((C1 & C2) != 0)
        break;
        else
        Clip();
        }
        glFlush();
}

void Key(unsigned char ch, int x, int y)
{
    Flag = 1;
    glutPostRedisplay();
}

int main(int argC, char *argV[])
{
    glutInit(&argC,argV);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Cohen Sutherland Line Clipping Window");
    glClearColor(1,1,1,1);
    glutDisplayFunc(Display);
    glutKeyboardFunc(Key);
    glutMainLoop();
    return 0; }
```

EXPECTED OUTPUT:



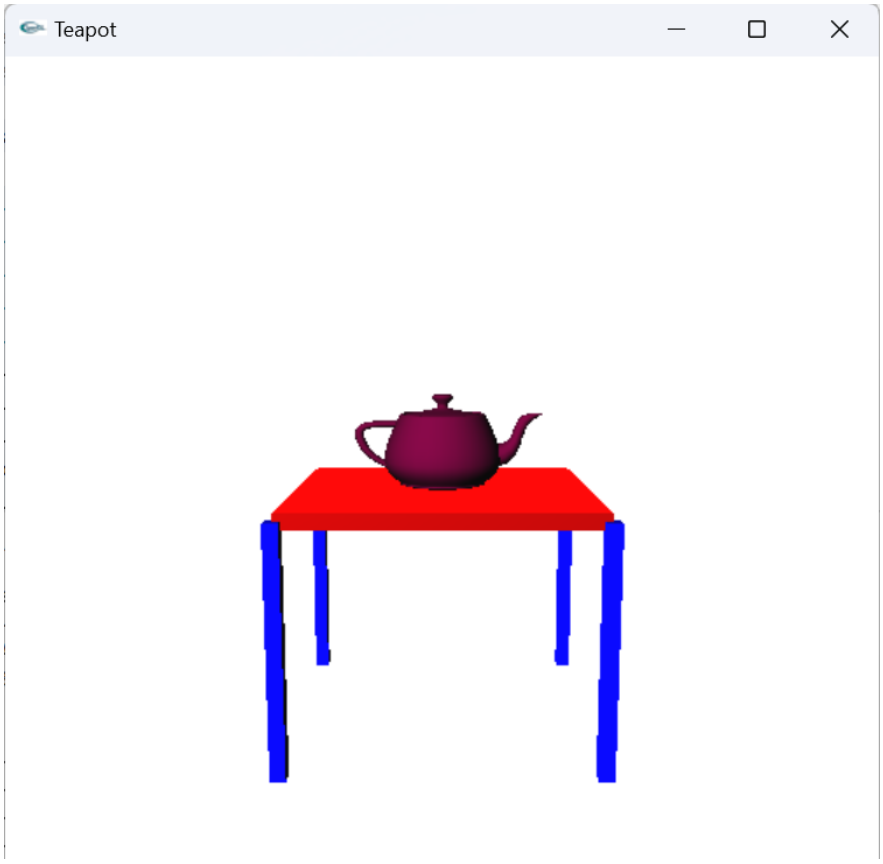


★ **PROGRAM 6**: Create a simple shaded scene of a teapot on a table.

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

void Display()
{
    GLfloat Pos[] = {-1,1,0,1};
    GLfloat S[] = {0.5,0,0.25,1};
    GLfloat D1[] = {1,0,0,1};
    GLfloat D2[] = {0,0,1,1};
    GLfloat D3[] = {0.5,0,0.25,1};
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLightfv(GL_LIGHT0,GL_POSITION,Pos);
    glLightfv(GL_LIGHT0,GL_SPECULAR,S);
    glLoadIdentity();
    gluLookAt(0,0.5,3,0,0,0,0,1,0);
    glPushAttrib(GL_ALL_ATTRIB_BITS);
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,D1);
    glPushMatrix();
    glScalef(1,0.05,1);
    glutSolidCube(1);
    glPopMatrix();
    glPopAttrib();
    glPushAttrib(GL_ALL_ATTRIB_BITS);
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,D2);
    glPushMatrix();
    glTranslatef(-0.5,-0.4,-0.5);
    glScalef(0.05,0.8,0.05);
    glutSolidCube(1); //leg1
    glPopMatrix();
    glPushMatrix();
    glTranslatef(0.5,-0.4,-0.5);
    glScalef(0.05,0.8,0.05);
    glutSolidCube(1); //leg2
    glPopMatrix();
    glPushMatrix();
    glTranslatef(0.5,-0.4,0.5);
    glScalef(0.05,0.8,0.05);
    glutSolidCube(1); //leg3
    glPopMatrix();
    glPushMatrix();
    glTranslatef(-0.5,-0.4,0.5);
    glScalef(0.05,0.8,0.05);
    glutSolidCube(1);
    glPopMatrix();
    glPopAttrib(); //leg4
    glPushAttrib(GL_ALL_ATTRIB_BITS);
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,D3);
    glPushMatrix();
    glTranslatef(0,0.18,0);
    glutSolidTeapot(0.2);
    glPopMatrix();
    glPopAttrib();
```

```
    glutSwapBuffers();
    }

int main(int argC, char *argV[])
{
    glutInit(&argC, argV);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Teapot");
    glutDisplayFunc(Display);
    glClearColor(1,1,1,1);
    glEnable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1,1,-1,1,2,20);
    glMatrixMode(GL_MODELVIEW);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glutMainLoop();
    return 0;

}
```

EXPECTED OUTPUT:



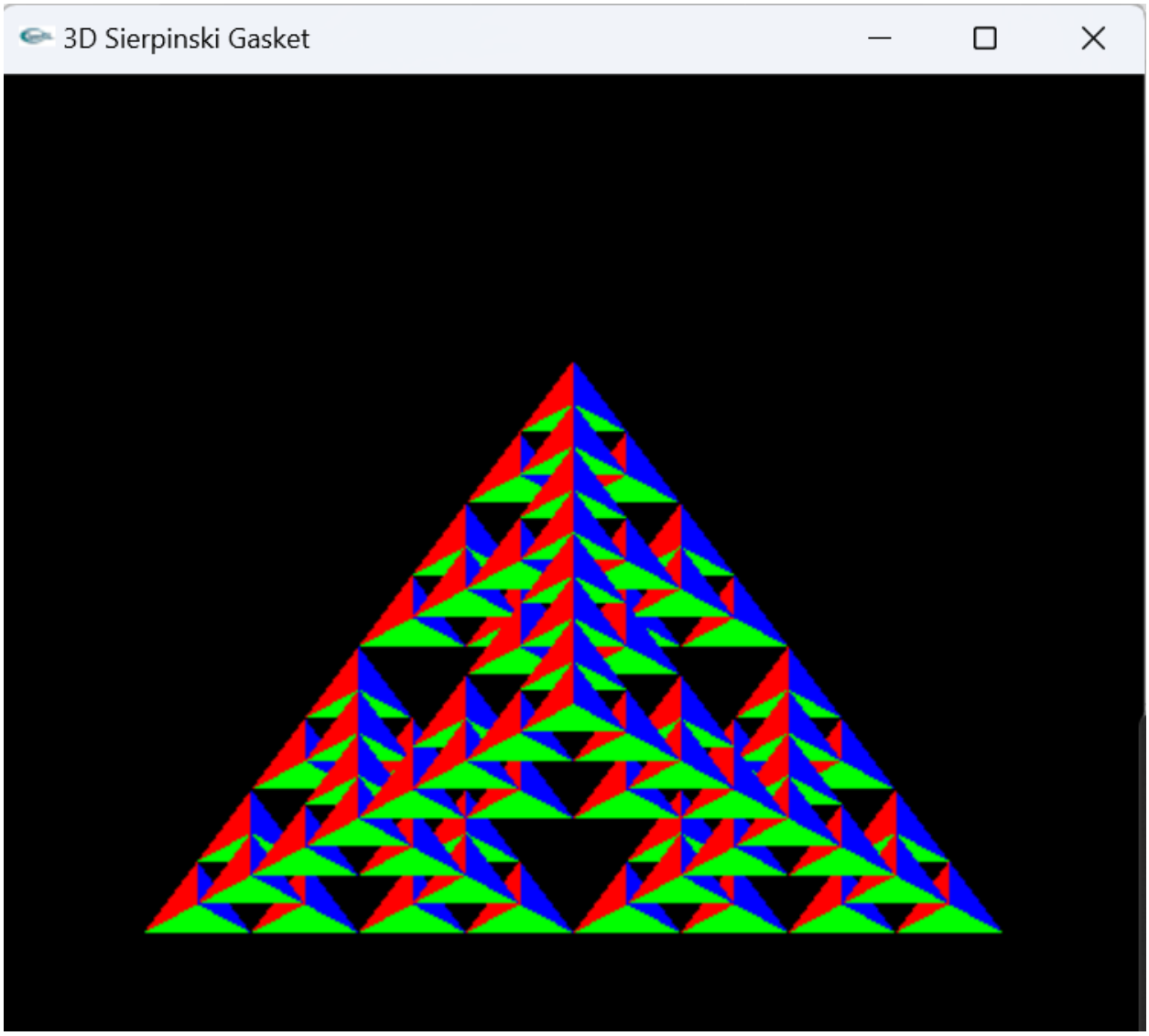
★ **PROGRAM 7**: Recursively subdivide a tetrahedron to form a 3D Sierpinski Gasket. The number of recursive steps is to be specified by the user.

```
#include<stdio.h>
#include<GL/glut.h>
int n;

void Triangle(GLfloat A[], GLfloat B[], GLfloat C[])
{
    glBegin(GL_TRIANGLES);
    glVertex3fv(A);
    glVertex3fv(B);
    glVertex3fv(C);
    glEnd();
}

void Tetra(GLfloat P1[], GLfloat P2[], GLfloat P3[], GLfloat P4[])
{
    glColor3f(1,1,1);
    Triangle(P1,P2,P3);
    glColor3f(0,1,0);
    Triangle(P1,P2,P4);
    glColor3f(0,0,1);
    Triangle(P2,P3,P4);
    glColor3f(1,0,0);
    Triangle(P1,P4,P3);
}

void Div(GLfloat P1[], GLfloat P2[], GLfloat P3[], GLfloat P4[], int
n)
{
    GLfloat P12[3], P23[3], P31[3], P14[3], P24[3], P34[3];
    if(n>0)
    {
        P12[0] = (P1[0] + P2[0])/2;
        P12[1] = (P1[1] + P2[1])/2;
        P12[2] = (P1[2] + P2[2])/2;

        P23[0] = (P2[0] + P3[0])/2;
        P23[1] = (P2[1] + P3[1])/2;
        P23[2] = (P2[2] + P3[2])/2;

        P31[0] = (P3[0] + P1[0])/2;
        P31[1] = (P3[1] + P1[1])/2;
        P31[2] = (P3[2] + P1[2])/2;

        P14[0] = (P1[0] + P4[0])/2;
        P14[1] = (P1[1] + P4[1])/2;
        P14[2] = (P1[2] + P4[2])/2;

        P24[0] = (P2[0] + P4[0])/2;
        P24[1] = (P2[1] + P4[1])/2;
```

```
        P24[2] = (P2[2] + P4[2])/2;

        P34[0] = (P3[0] + P4[0])/2;
        P34[1] = (P3[1] + P4[1])/2;
        P34[2] = (P3[2] + P4[2])/2;

        Div(P1,P12,P31,P14,n-1);
        Div(P12,P2,P23,P24,n-1);
        Div(P31,P23,P3,P34,n-1);
        Div(P14,P24,P34,P4,n-1);

    }
    else
    Tetra(P1,P2,P3,P4);
}

void Display()
{
    GLfloat V[4][3] = {
        {-0.75,-0.5,-0.5},
        {0.75,-0.5,-0.5},
        {0,0.5,-0.5},
        {0,-0.1,0.5}
    };
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    Div(V[0],V[1],V[2],V[3],n);
    glutSwapBuffers();
}

int main(int c, char *v[])
{
    printf("\n\t Enter the number of divisions: ");
    scanf("%d", &n);
    glutInit(&c,v);
    glutInitDisplayMode(GLUT_RGB|GLUT_DOUBLE|GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("3D Sierpinski Gasket");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

Note: Limit max. Recursive steps to 6.

EXPECTED OUTPUT:



Note: Execution command - ./a.out

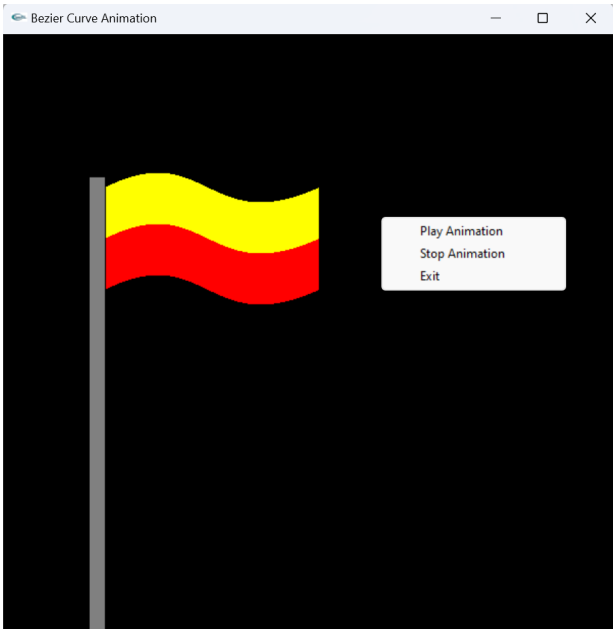
★ **PROGRAM 8**: Develop a menu driven program to animate a flag using Bezier Curve algorithm.

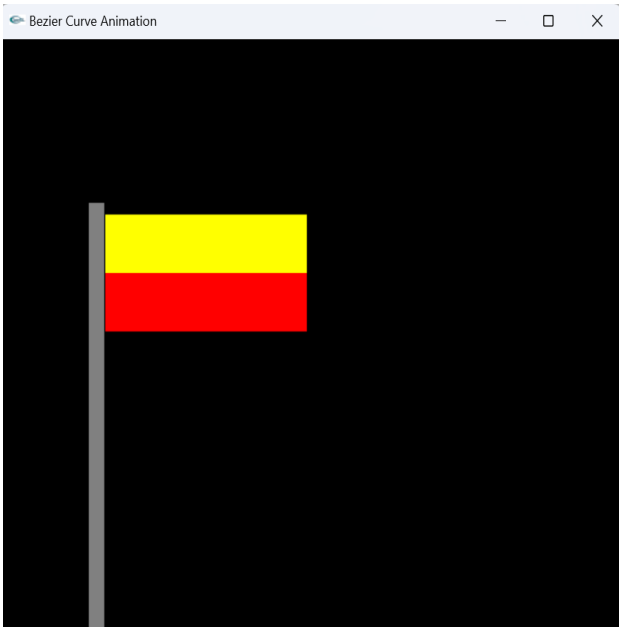
```
#include<GL/glut.h>
#include<math.h>

int AnFlag = 0;
int yFlag = 1, xFlag = 1;
float yC = -50, xC = -10;
float x[5], Y1[5], y2[5], y3[5];

void Menu(int Id)
{
    switch(Id)
    {
        case 1:
        AnFlag = 1;
        break;
        case 2:
        AnFlag = 0;
        break;
        case 3:
        exit(0);
    }
}

void Idle()
{
    if(AnFlag == 1)
    {
        if(yC<50 && yFlag == 1)
        yC = yC + 0.2;
        if(yC>=50 && yFlag == 1)
        yFlag = 0;
        if(yC>-50 && yFlag == 0)
        yC = yC - 0.2;
        if(yC<=-50 && yFlag == 0)
        yFlag = 1;
        if(xC<20 && xFlag == 1)
        xC = xC + 0.2;
        if(xC>=20 && xFlag == 1)
        xFlag = 0;
        if(xC>-20 && xFlag == 0)
        xC = xC - 0.2;
        if(xC<=-20 && xFlag == 0)
        xFlag = 1;
    }
    glutPostRedisplay();
}

void Draw()
{
```

```
    int i;
    double t, xt[200], y1t[200], y2t[200], y3t[200];
    glClear(GL_COLOR_BUFFER_BIT);
    x[0] = 300 - xC;
    Y1[0] = 450;
    y2[0] = 400;
    y3[0] = 350;
    x[1] = 200;
    Y1[1] = 450 + yC;
    y2[1] = 400 + yC;
    y3[1] = 350 + yC;
    x[2] = 200;
    Y1[2] = 450 - yC;
    y2[2] = 400 - yC;
    y3[2] = 350 - yC;
    x[3] = 100;
    Y1[3] = 450;
    y2[3] = 400;
    y3[3] = 350;
    x[4] = 0;
    Y1[4] = 450;
    y2[4] = 400;
    y3[4] = 350;
    i = 0;

    for(t = 0.0; t < 1.0; t += 0.005)
    {
        xt[i] = pow(1-t,3)*x[0] + 3*t*pow(1-t,2)*x[1] +
3*pow(t,2)*(1-t)*x[2] + pow(t,3)*x[3];
        y1t[i] = pow(1-t,3)*Y1[0] + 3*t*pow(1-t,2)*Y1[1] +
3*pow(t,2)*(1-t)*Y1[2] + pow(t,3)*Y1[3];
        y2t[i] = pow(1-t,3)*y2[0] + 3*t*pow(1-t,2)*y2[1] +
3*pow(t,2)*(1-t)*y2[2] + pow(t,3)*y2[3];
        y3t[i] = pow(1-t,3)*y3[0] + 3*t*pow(1-t,2)*y3[1] +
3*pow(t,2)*(1-t)*y3[2] + pow(t,3)*y3[3];
        i++;
    }

    glColor3f(1.0,1.0,0.0);
    glBegin(GL_QUAD_STRIP);
    for(i=0;i<200;i++)
    {
        glVertex2d(xt[i], y1t[i]);
        glVertex2d(xt[i],y2t[i]);
    }
    glEnd();
    glColor3f(1,0,0);
    glBegin(GL_QUAD_STRIP);
    for(i=0;i<200;i++)
    {
        glVertex2d(xt[i],y2t[i]);
        glVertex2d(xt[i],y3t[i]);
    }
    glEnd();
```

```
    glColor3f(0,0.5,0);
    glColor3f(0.5,0.5,0.5);
    glRectf(85,460,100,0);
    glFlush();
    }

void MyInit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0,600,0,600);
    glMatrixMode(GL_MODELVIEW);
    glutCreateMenu(Menu);
    glutAddMenuEntry("Play Animation",1);
    glutAddMenuEntry("Stop Animation",2);
    glutAddMenuEntry("Exit",3);
    glutAttachMenu(GLUT_RIGHT_BUTTON);

}

int main(int argC, char *argV[])
{
    glutInit(&argC, argV);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600,600);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Bezier Curve Animation");
    MyInit();
    glutDisplayFunc(Draw);
    glutIdleFunc(Idle);
    glutMainLoop();
    return 0;
}
```

EXPECTED OUTPUT:





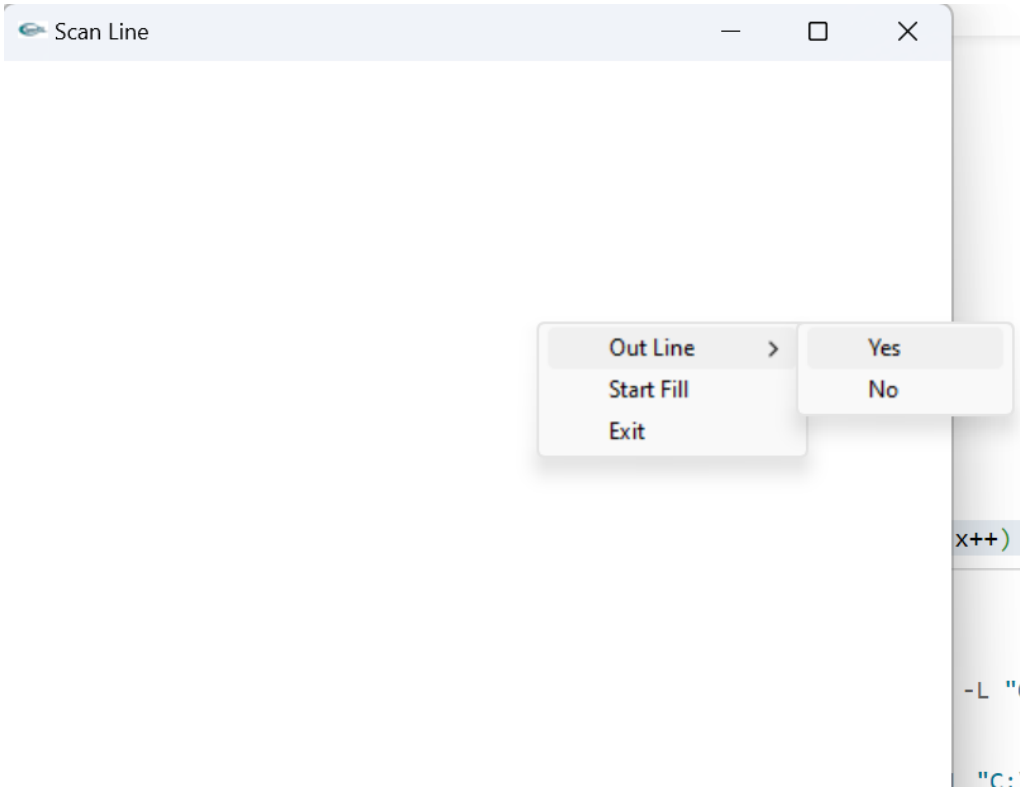
★ **PROGRAM 9**: Develop a menu driven program to fill the polygon using scan line algorithm.

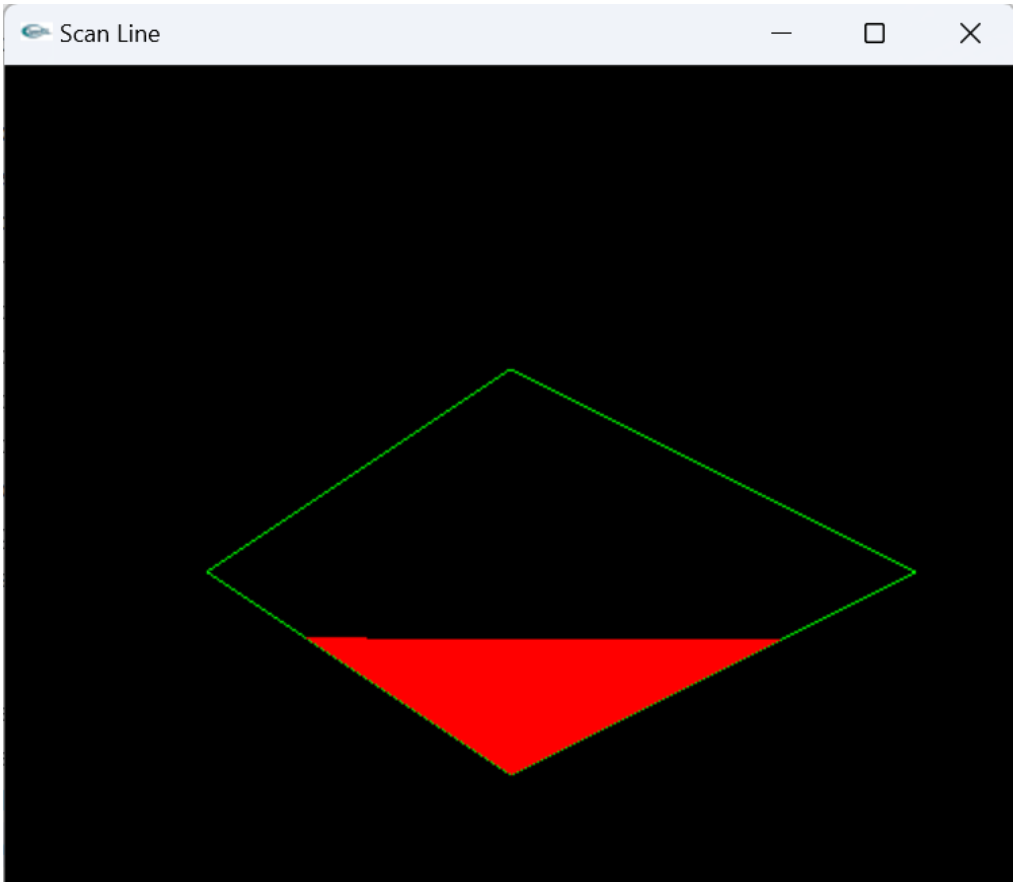
```
#include<GL/glut.h>

int LE[500], RE[500];
int Fill_Flag = 0, Line_Flag = 0;

void Intersection(GLfloat x1, GLfloat y1, GLfloat x2, GLfloat y2)
{
    GLfloat temp, M, x;
    int i;
    if(y1>y2)
    {
        temp = y1;
        y1 = y2;
        y2 = temp;
        temp = x1;
        x1 = x2;
        x2 = temp;
    }
    if(y2-y1 == 0)
    M = x2 - x1;
    else
    M = (x2-x1)/(y2-y1);
    x = x1;
    for(i=y1;i<=y2;i++)
    {
        if(x < LE[i])
        {
            LE[i] = x;
        }
        if(x > RE[i])
        {
            RE[i] = x;
        }
        x = x + M;
    }
}

void Display()
{
    GLfloat x1 = 250, y1 = 150;
    GLfloat x2 = 450, y2 = 250;
    GLfloat x3 = 250, y3 = 350;
    GLfloat x4 = 100, y4 = 250;
    int i;
    GLint x,y;
    glClear(GL_COLOR_BUFFER_BIT);
    for(i=0; i<500; i++)
    {
        LE[i] = 500;
```

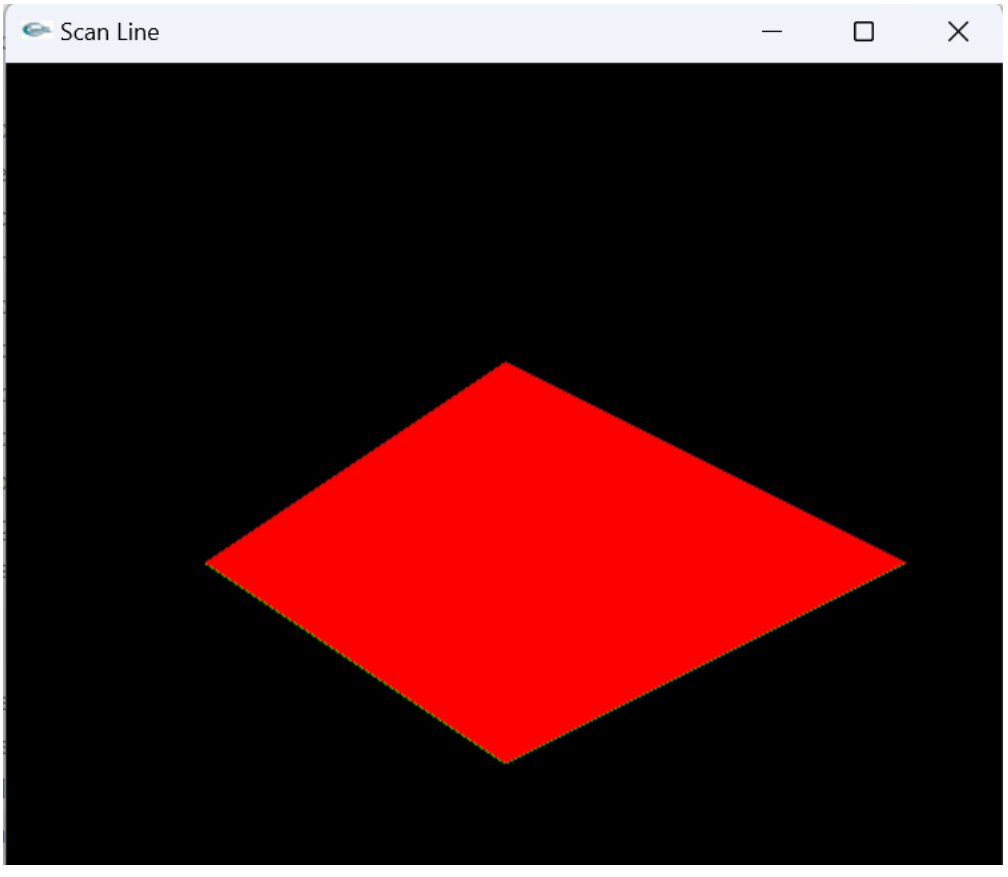
```
        RE[i] = 0;
    }
    if(Line_Flag == 1)
    {
        glColor3f(0,1,0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(x1,y1);
        glVertex2f(x2,y2);
        glVertex2f(x3,y3);
        glVertex2f(x4,y4);
        glEnd();
    }
    glColor3f(1,0,0);
    Intersection(x1,y1,x2,y2);
    Intersection(x2,y2,x3,y3);
    Intersection(x3,y3,x4,y4);
    Intersection(x4,y4,x1,y1);
    if(Fill_Flag == 1)
    {
        for(y=0; y<500; y++)
        {
            if(LE[y]<=RE[y])
            {
                for(x=LE[y]; x<=RE[y]; x++)
                {
                    glBegin(GL_POINTS);
                    glVertex2f(x,y);
                    glEnd();
                    glFlush();
                }
            }
        }
    }
}

void Line_Menu(int Id)
{
    if(Id == 1)
    Line_Flag = 1;
    if(Id == 2)
    Line_Flag = 2;
    glutPostRedisplay();
}

void Main_Menu(int Id)
{
    if(Id == 1)
    Fill_Flag = 1;
    if(Id == 2)
    exit(0);
    glutPostRedisplay();
}
```

```
int main(int argC, char *argV[])
{
    int Id;
    glutInit(&argC, argV);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Scan Line");
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0,500,0,500);
    glMatrixMode(GL_MODELVIEW);
    Id = glutCreateMenu(Line_Menu);
    glutAddMenuEntry("Yes", 1);
    glutAddMenuEntry("No", 2);
    glutCreateMenu(Main_Menu);
    glutAddSubMenu("Out Line", Id);
    glutAddMenuEntry("Start Fill", 1);
    glutAddMenuEntry("Exit", 2);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

EXPECTED OUTPUT:




Scan Line


Scan Line