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

```
#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, 0, 0);
	glBegin(GL_POINTS);
	glVertex2i(x, y);
	glEnd();
	glFlush();
}
void LineBres(int xstart, int ystart, int xend, int yend)
{
	int dx = abs(xend - xstart);
	int dy = abs(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 (x1, y1, x2, y2)\n");
	scanf("%d%d%d%d", &xstart, &ystart, &xend, &yend);
	glutInit(&argc, argv);
	glutInitWindowPosition(50, 50);
```

```
        glutInitWindowSize(500, 500);
        glutCreateWindow("Bresenham's Line Drawing");
        init();
        glutDisplayFunc(Display);
        glutMainLoop();
        return 0;
}


OUTPUT:ENTER X1,X2,Y1,Y2
240,240,460,510
```

PROGRAM - 2

Create and rotate a triangle about the origin and a fixed point.

```
#include<stdio.h>
#include<GL/glut.h>
int x,y;
int where_to_rotate=0;
float translate_x=0.0,translate_y=0.0,rotate_angle=0.0;
void draw_pixel(float x1,float y1)
{
        glPointSize(5.0);
        glBegin(GL_POINTS);
                glVertex2f(x1,y1);
        glEnd();
}
void triangle(int x,int y)
{
        glColor3f(0.0,1.0,0.0); // set interior color of triangle to green
        glBegin(GL_POLYGON);
                glVertex2f(x,y);
                glVertex2f(x+400,y+400);
                glVertex2f(x+300,y+0);
        glEnd();
        glFlush();
}
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glLoadIdentity();
        glColor3f(1.0,0.0,0.0); //color of point
        draw_pixel(0.0,0.0);
        if(where_to_rotate==1)
        {
                translate_x=0.0;
                translate_y=0.0;
                rotate_angle+=0.9;
        }
        if(where_to_rotate==2)
        {
                translate_x=x;
                translate_y=y;
                rotate_angle+=0.9;
                glColor3f(0.0,0.0,1.0);
                draw_pixel(x,y);
        }
        glTranslatef(translate_x,translate_y,0.0);
        glRotatef(rotate_angle,0.0,0.0,1.0);
        glTranslatef(-translate_x,-translate_y,0.0);
        triangle(translate_x,translate_y);
        glutPostRedisplay();
        glutSwapBuffers();
}
void myInit()
{
        glClearColor(1.0,1.0,1.0,1.0); //background color to white
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-800.0,800.0,-800.0,800.0);
        glMatrixMode(GL_MODELVIEW);
}
void rotate_menu(int option)
{
        if(option==1)
                where_to_rotate=1;
        if(option==2)
                where_to_rotate=2;
        if(option==3)
                where_to_rotate=3;
        display();
}
int main(int argc,char **argv)
{
        printf("\nEnter fixed points for rotation (x,y) : ");
        scanf("%d%d",&x,&y);
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
        glutInitWindowSize(800,800);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Rotate Created Triangle");
                myInit();
        glutDisplayFunc(display);
        glutCreateMenu(rotate_menu);
                glutAddMenuEntry("Rotate Around Origin",1);
                glutAddMenuEntry("Rotate Around Fixed Points",2);
                glutAddMenuEntry("Stop Rotation",3);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        glutMainLoop();
}
```

PROGRAM - 3

Draw a colour cube and spin it using OpenGL transformation matrices.

```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
float v[][3] = { { -1,-1,-1 } ,{ -1,1,-1 } ,{ 1,1,-1 } ,{ 1,-1,-1 } ,{ -1,-1,1 } ,
{ -1,1,1 } ,{ 1,1,1 } ,{ 1,-1,1 } }; // 8 vertices of the cube with origin as its centroid
int t[] = { 0,0,0 };
int ax = 2;
void init()
{
        glMatrixMode(GL_PROJECTION);
        glOrtho(-4, 4, -4, 4, -10, 10);
        glMatrixMode(GL_MODELVIEW);
}
void polygon(int a, int b, int c, int d)  // function used to draw one face of a cube at
a time
{
        glBegin(GL_POLYGON);
        glVertex3fv(v[a]);
        glVertex3fv(v[b]);
        glVertex3fv(v[c]);
        glVertex3fv(v[d]);
        glEnd();
}
void colorcube()
}
        glColor3f(0, 0, 1);
        polygon(0, 1, 2, 3);
        glColor3f(0, 1, 1);
        polygon(4, 5, 6, 7);
        glColor3f(0, 1, 0);
        polygon(0, 1, 5, 4);
        glColor3f(1, 0, 0);
        polygon(2, 6, 7, 3);
        glColor3f(1, 1, 0);
        polygon(0, 4, 7, 3);
        glColor3f(1, 0, 1);
        polygon(1, 5, 6, 2);
}
void spincube()
{
        t[ax] += 1;
        if (t[ax] == 360)
                t[ax] -= 360;
        glutPostRedisplay();
}
void mouse(int btn, int state, int x, int y)
{
        if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
                ax =
0;
        if (btn == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
                ax =
1;
        if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
                ax =
2;
}
void display()  // display function
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glClearColor(1, 1, 1, 1);
        glLoadIdentity();
        glRotatef(t[0], 1, 0, 0);
        glRotatef(t[1], 0, 1, 0);
        glRotatef(t[2], 0, 0, 1);
        colorcube();
        glutSwapBuffers();
        glFlush();
}
int main(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
        glutInitWindowPosition(100, 100);
        glutInitWindowSize(500, 500);
        glutCreateWindow("Cube rotation");
        init();
        glutIdleFunc(spincube);
        glutMouseFunc(mouse);
        glEnable(GL_DEPTH_TEST);
        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```

PROGRAM - 4
Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing

```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
float pts[8][3] = {{-1,-1,-1},{-1,1,-1},{1,1,-1},{1,-1,-1},{-1,-1,1},{-1,1,1},{1,1,1},-
{1,-1,1}};
```

```
float theta[] ={0,0,0};
int axis = 2;
float viewer[]={5,0,0};
void myInit()
{
        glMatrixMode(GL_PROJECTION);
        glFrustum(-2,2,-2,2,2,10);
        glMatrixMode(GL_MODELVIEW);
}
void draw_polygon(int a, int b, int c, int d)
{
        glBegin(GL_QUADS);
        glVertex3fv(pts[a]);
        glVertex3fv(pts[b]);
        glVertex3fv(pts[c]);
        glVertex3fv(pts[d]);
        glEnd();
}
void draw_cube(float pts[8][3])
{
        glColor3f(0,0,1);
        draw_polygon(0,1,2,3); //front face
        glColor3f(0,1,0);
        draw_polygon(4,5,6,7);  //behind face
        glColor3f(1,0,0);
        draw_polygon(0,1,5,4);  //left face
        glColor3f(0,0,0);
        draw_polygon(3,2,6,7);  //right face
        glColor3f(0,1,1);
        draw_polygon(0,4,7,3);  //bottom face
        glColor3f(1,0,1);
        draw_polygon(1,5,6,2);  //top face
}
void myDisplay()
{
        glClearColor(1,1,1,1);
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookAt(viewer[0],viewer[1],viewer[2],0,0,0,0,1,0);
        glRotatef(theta[2],0,0,1);
        glRotatef(theta[1],0,1,0);
        glRotatef(theta[0],1,0,0);
        draw_cube(pts);
        glFlush();
        glutSwapBuffers();
}
void spincube()
{
        theta[axis] = theta[axis]+4;
        if(theta[axis]>360)
                theta[axis]=0;
        glutPostRedisplay();
}
void mouse(int btn , int state , int x , int y)
{
        if((btn==GLUT_LEFT_BUTTON)&&(state==GLUT_DOWN))
                axis=0;
        if((btn==GLUT_RIGHT_BUTTON)&&(state==GLUT_DOWN))
                axis=2;
        if((btn==GLUT_MIDDLE_BUTTON)&&(state==GLUT_DOWN))
                axis=1;
        spincube();
}
void keyboard(unsigned char key, int x, int y)
{
        if(key=='X') viewer[0]+=1;
        if(key=='x') viewer[0]-=1;
        if(key=='Y') viewer[1]+=1;
        if(key=='y') viewer[1]-=1;
        if(key=='Z') viewer[2]+=1;
        if(key=='z') viewer[2]-=1;
        glutPostRedisplay();
}
int main (int argc, char ** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode( GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowPosition(50,50);
    glutInitWindowSize(500,500);
        glutCreateWindow("Positioning of Camera");
        myInit();
        glEnable(GL_DEPTH_TEST);
        glutDisplayFunc(myDisplay);
        glutKeyboardFunc(keyboard);
        glutMouseFunc(mouse);
    glutMainLoop();
}
```

PROGRAM - 5

Clip a lines using Cohen-Sutherland algorithm.

```
#include<stdio.h>
#include<GL/glut.h>
#define true 1;
#define false 0;
#define bool int;
double x,y;
int xmin=50,xmax=100,ymin=50,ymax=100;
const int RIGHT=8,LEFT=2,TOP=4,BOTTOM=1;
int outcode0,outcode1,outcodeout,done,accept;
int computeoutcode(double x,double y)
{
  int code=0;
  if(y>ymax)
    code|=TOP;
  else if(y<ymin)
    code|=BOTTOM;
  if(x>xmax)
    code|=RIGHT;
  else if(x<xmin)
    code|=LEFT;
  return code;
}
void LineClip(double x0,double y0,double x1,double y1)
{
  int accept=false;
  int done=false;
  outcode0=computeoutcode(x0,y0);
  outcode1=computeoutcode(x1,y1);
  do{
    if(!(outcode0|outcode1))
    {
      accept=true;
      done=true;
    }
    else if(outcode0&outcode1)
    {
      done=true;
    }
    else
    {
      outcodeout=outcode0?outcode0:outcode1;
      if(outcodeout & TOP)
      {
        x=x0+(x1-x0)*(ymax-y0)/(y1-y0);
        y=ymax;
      }
      else if(outcodeout & BOTTOM)
      {
        x=x0+(x1-x0)*(ymin-y0)/(y1-y0);
        y=ymin;
      }
      else if(outcodeout & RIGHT)
      {
        y=y0+(y1-y0)*(xmax-x0)/(x1-x0);
        x=xmax;
      }
      else
      {
        y=y0+(y1-y0)*(xmin-x0)/(x1-x0);
        x=xmin;
      }
      if(outcodeout==outcode0)
      {
        x0=x;y0=y;outcode0=computeoutcode(x0,y0);
      }
      else
      {
        x1=x;y1=y;outcode1=computeoutcode(x1,y1);
      }
    }
  }while(!done);
  if(accept)
  {
    glPushMatrix();
    glTranslatef(100,100,0);
    glColor3f(1.0,0.0,0.0);
    glBegin(GL_LINE_LOOP);
    glVertex2i(50,50);
    glVertex2i(100,50);
    glVertex2i(100,100);
    glVertex2i(50,100);
    glEnd();
    glColor3f(1.0,0.0,1.0);
    glBegin(GL_LINES);
    glVertex2i(x0,y0);
    glVertex2i(x1,y1);
    glEnd();
    glPopMatrix();
    glFlush();
  }
}
void display()
{
  glClearColor(1,1,1,1);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0,0.0,0.0);
  glBegin(GL_LINE_LOOP);
  glVertex2i(50,50);
  glVertex2i(100,50);
  glVertex2i(100,100);
  glVertex2i(50,100);
  glEnd();
  glColor3f(1.0,0.0,1.0);
  glBegin(GL_LINES);
  glVertex2i(60,20);
  glVertex2i(80,120);
  glVertex2i(80,20);
  glVertex2i(60,120);
  glEnd();
  LineClip(60,20,80,120);
  LineClip(80,20,60,120);
  glFlush();
}
voi0d init()
{
  glMatrixMode(GL_PROJECTION);
  gluOrtho2D(0,300,0,300);
  0glMatrixMode(GL_MODELVIEW);
00}
int main(int argc,char** argv)
{
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowPosition(0,0);
  glutInitWindowSize(500,500);
  glutCreateWindow("Cohen Sutherland line and drawing algorithm");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
}
```

PROGRAM6
TEA POt on a table

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

void obj(double tx,double ty,double tz,double sx,double sy,double sz)
{
    glRotated(50,0,1,0);
        glRotated(10,-1,0,0);
        glRotated(11.7,0,0,-1);
        glTranslated(tx,ty,tz);
        glScaled(sx,sy,sz);
        glutSolidCube(1);
        glLoadIdentity();
}

void display()
{
        glViewport(0,0,700,700);
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        obj(0,0,0.5,1,1,0.04);
        obj(0,-0.5,0,1,0.04,1);
        obj(-0.5,0,0,0.04,1,1);
        obj(0,-0.3,0,0.02,0.2,0.02);
        obj(0,-0.3,-0.4,0.02,0.2,0.02);
        obj(0.4,-0.3,0,0.02,0.2,0.02);
        obj(0.4,-0.3,-0.4,0.02,0.2,0.02);
        obj(0.2,-0.18,-0.2,0.6,0.02,0.6);
        glRotated(50,0,1,0);
        glRotated(10,-1,0,0);
        glRotated(11.7,0,0,-1);
        glTranslated(0.3,-0.1,-0.3);
        glutSolidTeapot(0.09);
        glFlush();
        glLoadIdentity();
}

int main(int argc,char **argv)
{
        glutInit(&argc,argv);
        float ambient[]={1,1,1,1};
        float light_pos[]={27,80,2,3};
        glutInitWindowSize(700,700);
        glutCreateWindow("Tea Pot");
        glutDisplayFunc(display);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glMaterialfv(GL_FRONT,GL_AMBIENT,ambient);
        glLightfv(GL_LIGHT0,GL_POSITION,light_pos);
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
}
```

PROGRAM - 7

Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.

```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
float v[4][3] = { { 0.0,0.0,1.0 },{ 0,1,-1 },{ -0.8,-0.4,-1 },{ 0.8,-0.4,-1 } };
int n;

void triangle(float a[], float b[], float c[])
{
        glBegin(GL_POLYGON);
        glVertex3fv(a);
        glVertex3fv(b);
        glVertex3fv(c);
        glEnd();
}

void divide_triangle(float a[], float b[], float c[], int m)
{
        float v1[3], v2[3], v3[3];
        int i;
        if (m>0)
        {
                for (i = 0; i<3; i++) v1[i] = (a[i] + b[i]) / 2;
                for (i = 0; i<3; i++) v2[i] = (a[i] + c[i]) / 2;
                for (i = 0; i<3; i++) v3[i] = (b[i] + c[i]) / 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));
}

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(1.0, 1.0, 0.0);
        divide_triangle(v[0], v[2], v[3], m);
}

void display()
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-2.0, 2.0, -2.0, 2.0, -10.0, 10.0);
        glMatrixMode(GL_MODELVIEW);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        tetrahedron(n);
        glFlush();
        glutPostRedisplay();
}

int main(int argc, char* argv[])
{
         printf("Enter the number of divisions: ");
        scanf("%d",&n);
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(0, 0);
        glutCreateWindow("3D Gasket");
        glutDisplayFunc(display);
        glEnable(GL_DEPTH_TEST);
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glutMainLoop();
        return 0;
}
```

PROGRAM - 9
Develop a menu driven program to fill the polygon using scan line algorithm

```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
int le[500], re[500], flag=0 ,m;

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

void edge(int x0, int y0, int x1, int y1)
{
        if (y1<y0)
        {
                int tmp;
                tmp = y1;
                y1 = y0;
                y0 = tmp;
                tmp = x1;
                x1 = x0;
                x0 = tmp;
        }
        int x = x0;
        m = (y1 - y0) / (x1 - x0);
        for (int i = y0; i<y1; i++)
        {
                if (x<le[i])
                        le[i] = x;
                if (x>re[i])
                        re[i] = x;
                x += (1 / m);
        }
}
void draw_pixel (int x, int y)
{
    glColor3f (1, 1, 0);
    glBegin (GL_POINTS);
    glVertex2i (x, y);
    glEnd ();
}
```

```
void display()
{
        glClearColor(1, 1, 1, 1);
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0, 0, 1);
        glBegin(GL_LINE_LOOP);
        glVertex2f(200, 100);
        glVertex2f(100, 200);
        glVertex2f(200, 300);
        glVertex2f(300, 200);
        glEnd();
        for (int i = 0; i<500; i++)
        {
                le[i] = 500;
                re[i] = 0;
        }
        edge(200, 100, 100, 200);
        edge(100, 200, 200, 300);
        edge(200, 300, 300, 200);
        edge(300, 200, 200, 100);
        if (flag == 1)
        {
                for (int i = 0; i < 500; i++)
                {
                        if (le[i] < re[i])
                        {
                                for (int j = le[i]; j < re[i]; j++)
                                {
                                        glColor3f(1, 0, 0);
                                        glBegin(GL_POINTS);
                                        glVertex2f(j, i);
                                        glEnd();
                                }
                        }
                }
        }
        glFlush();
}

void ScanMenu(int id)
{
        if (id == 1) {
                flag = 1;
        }
        else if (id == 2) {
                flag = 0;
        }
        else { exit(0); }
        glutPostRedisplay();
}

int main(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitWindowPosition(100, 100);
        glutInitWindowSize(500, 500);
        glutCreateWindow("scan line");
        init();
        glutDisplayFunc(display);
        glutCreateMenu(ScanMenu);
        glutAddMenuEntry("scanfill", 1);
        glutAddMenuEntry("clear", 2);
        glutAddMenuEntry("exit", 3);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        glutMainLoop();
        return 0;
}
```