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

```
#include<windows.h>
#include<GL/glut.h>
#include<math.h>
void draw()
{
  glClearColor(1,1,1,1);            //Background Color(#,#,#) and Transparency (#)
  glPointSize(4);                   //Point Size
  gluOrtho2D(0,50,0,50);            //Specifying the 2D Space with Start and End Cordinates
  int x1=30,y1=20,x2=20,y2=30;              //Coordinates
  float x,y,dx,dy,m,p,temp;
  glClear(GL_COLOR_BUFFER_BIT);         //Clearing the previous BG
  if(x1!=x2)
    m=(y2-y1)/(x2-x1);              //Finding Slope. If x1==x2, then slope is zero
  else
    m=999;
  x=x1;              //Updating Variable
  y=y1;             //Updating Variable
  if(fabs(m)<1) //Positive Slope and zero slope
  {
     if(x1>x2)
     {
        temp=x1;x1=x2;x2=temp;
        temp=y1;y1=y2;y2=temp;
     }
     dx=fabs(x2-x1);
     dy=fabs(y2-y1);
     x=x1;
```

```
    y=y1;
    p=2*dy-dx;
    while(x<=x2)
    {
       glBegin(GL_POINTS);
       glColor3f(1,0,0);
       glVertex2f(x,y);
       glEnd();
       x=x+1;
       if(p>=0)
       {
          if(m>=0&&m<1)
             y=y+1;
          else
             y=y-1;
          p=p+2*dy-2*dx;
       }
       else
          p=p+2*dy;
    }
}
if(fabs(m)>=1)            //Negative and Infinite Slope
{
    if(y1>y2)
    {
       temp=x1;x1=x2;x2=temp;
       temp=y1;y1=y2;y2=temp;
    }
    dx=fabs(x2-x1);
```

```
    dy=fabs(y2-y1);
    x=x1;
    y=y1;
    p=2*dy-dx;
    while(y<=y2)
    {
      glBegin(GL_POINTS);
      glColor3f(1,0,0);
      glVertex2f(x,y);
      glEnd();
      y=y+1;
      if(p>=0)
      {
        if(m>=1)
          x=x+1;
        else
          x=x-1;
        p=p+2*dx-2*dy;
      }
      else
        p=p+2*dx;
    }
  }
  glFlush();       //Refreshing the window
}
int main(int C,char *V[])
{
  glutInit(&C,V);
  glutInitWindowSize(480,480);
```

```
   glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
   glutCreateWindow("Bresenham's Algorithm");
   glutDisplayFunc(draw);
   glutMainLoop();
   return 0;
}
```

---

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

```
#include<windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
GLfloat P[3][2]={{0,0.5},{-0.5,-0.5},{0.5,-0.5}}, NP[3][2];
GLfloat d,r,xr,yr;
int i;
void draw()
{
 glClearColor(1,1,1,1);
 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]=xr+(P[i][0]-xr)*cos(r)-(P[i][1]-yr)*sin(r);
```

```
    NP[i][1]=yr+(P[i][0]-yr)*sin(r)+(P[i][1]-xr)*cos(r);
  }
  glColor3f(1,0,1);
  glBegin(GL_LINE_LOOP);
    glVertex2fv(NP[0]);
    glVertex2fv(NP[1]);
    glVertex2fv(NP[2]);
 glEnd();
 glFlush();
}
int main(int argc, char *argv[])
{
   printf("Enter the angle\n");
   scanf("%f",&d);
   printf("Enter the points for rotation\n");
   scanf("%f,%f",&xr,&yr);
   glutInit(&argc, argv);
   glutInitWindowSize(640,480);
   glutInitWindowPosition(0,0);
   glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE );
   glutCreateWindow("Triangle Rotation");
   glutDisplayFunc(draw);
   glutMainLoop();
   return 0;
}
```

---

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

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

```
#include <math.h>
#include <stdio.h>
GLfloat v[8][3]={{0,0,0},{0,0.5,0},{0.5,0.5,0},{0.5,0,0},{0,0,0.5},{0,0.5,0.5},{0.5,0.5,0.5},{0.5,0,0.5}},nv[8][3];
GLfloat d=0,r;
char a;
void spin()
{
   d=d+1;
   if(d>360)
      d=0;
   glutPostRedisplay();
}
void drawface(int a,int b,int c,int d)
{
   glBegin(GL_POLYGON);
     glVertex3fv(nv[a]);
     glVertex3fv(nv[b]);
     glVertex3fv(nv[c]);
     glVertex3fv(nv[d]);
   glEnd();
   glFlush();
}
void draw()
{
   int i;
   glClearColor(1,1,1,1);
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   r= d*3.14/180;
   if(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')
{
   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);
    }
}
if(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];
    }
}
glColor3f(0,0,0);
drawface(7,6,5,4);
glColor3f(0,0,1);
```

```
    drawface(6,2,1,5);
    glColor3f(0,1,0);
    drawface(0,3,2,1);
    glColor3f(0,1,1);
    drawface(4,7,3,0);
    glColor3f(1,0,0);
    drawface(7,3,2,6);
    glColor3f(1,0,1);
    drawface(4,0,1,5);
    glutSwapBuffers();
}
int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    printf("Enter the axis\n");
    scanf("%c",&a);
    glutInitWindowSize(640,480);
    glutInitWindowPosition(0,0);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutCreateWindow("Cube");
    glutDisplayFunc(draw);
    glutIdleFunc(spin);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
    return 0;
}
```

---

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

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

```
GLfloat v[8][3]={{0,0,0},{0,0.5,0},{0.5,0.5,0},{0.5,0,0},{0,0,0.5},{0,0.5,0.5},{0.5,0.5,0.5},{0.5,0,0.5}};
GLfloat camx=0,camy=0,camz=4;
void drawface(int a,int b,int c,int d)
{
   glBegin(GL_POLYGON);
      glVertex3fv(v[a]);
      glVertex3fv(v[b]);
      glVertex3fv(v[c]);
      glVertex3fv(v[d]);
   glEnd();
}
void draw()
{
   glColor3f(0,0,0);
   drawface(7,6,5,4);
   glColor3f(0,0,1);
   drawface(6,2,1,5);
   glColor3f(0,1,0);
   drawface(0,3,2,1);
   glColor3f(0,1,1);
   drawface(4,7,3,0);
   glColor3f(1,0,0);
   drawface(7,3,2,6);
   glColor3f(1,0,1);
   drawface(4,0,1,5);
}
void display()
{
   glClearColor(1,1,1,1);
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(camx,camy,camz,.5,.5,.5,0,1,0);
    draw();
    glutSwapBuffers();
}
void key(unsigned char ch,int x,int y)
{
    switch(ch)
    {
        case 'x': camx-=0.5;
                  break;
        case 'X': camx+=0.5;
                  break;
        case 'y': camy-=0.5;
                  break;
        case 'Y': camy+=0.5;
                  break;
        case 'z': camz-=0.5;
                  break;
        case 'Z': camz+=0.5;
                  break;
    }
    glutPostRedisplay();
}
int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(640,480);
```

```
    glutInitWindowPosition(0,0);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutCreateWindow("Cube");
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1,1,-1,1,2,20);
    glMatrixMode(GL_MODELVIEW);
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
    return 0;
}
```

---

5. Clip a lines using Cohen-Sutherland algorithm

```
#include<windows.h>
#include<GL/glut.h>
GLfloat xmin=10,ymin=10,xmax=50,ymax=50;
GLfloat x1=5,y1=2,x2=70,y2=80,m;
int left=1,right=2,bottom=4,top=8,flag=0,ac=1,c1,c2;
int getcode(GLfloat x, GLfloat y)
{
    int code=0;
    if(x<xmin)
        code=code|left;
    if(x>xmax)
        code=code|right;
    if(y<ymin)
        code=code|bottom;
    if(y>ymax)
```

```
    code=code|top;
  return code;
}
void clip()
{
  GLfloat x,y;
  int c;
  if(c1)
    c=c1;
  else
    c=c2;
  m=(y2-y1)/(x2-x1);
  if(c&left)
  {
    x=xmin;
    y=y1+m*(xmin-x1);
  }
  if(c&right)
  {
    x=xmax;
    y=y1+m*(xmax-x1);
  }
  if(c&bottom)
  {
    y=ymin;
    x=x1+(ymin-y1)/m;
  }
  if(c&top)
  {
```

```
      y=ymax;
      x=x1+(ymax-y1)/m;
   }
   if(c==c1)
   {
      x1=x;y1=y;
   }
   else
   {
      x2=x;y2=y;
   }
}
void cohensutherland()
{
   glClearColor(1,1,1,1);
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1,0,0);
   glBegin(GL_LINE_LOOP);
   glVertex2f(xmin,ymin);
   glVertex2f(xmax,ymin);
   glVertex2f(xmax,ymax);
   glVertex2f(xmin,ymax);
   glEnd();
   if(ac)
   {
   glColor3f(0,0,1);
   glBegin(GL_LINES);
   glVertex2f(x1,y1);
   glVertex2f(x2,y2);
```

```
    glEnd();
    }
    while(1&flag)
    {
        c1=getcode(x1,y1);
        c2=getcode(x2,y2);
        if((c1|c2)==0)
        {
            ac=1;
            break;
        }
        if((c1&c2)!=0)
        {
            ac=0;
            break;
        }
        if((c1&c2)==0)
            clip();
    }
    glFlush();
}
void Key(unsigned char ch, int x, int y)
{
    if(ch=='c')
        flag=1;
    glutPostRedisplay();
}
void init()
{
```

```
    gluOrtho2D(0,100,0,100);
}
int main(int C,char *V[])
{
    glutInit(&C,V);
    glutInitWindowSize(480,480);
    glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
    glutCreateWindow("Cohen Sutherland");
    init();
    glutDisplayFunc(cohensutherland);
    glutKeyboardFunc(Key);
    glutMainLoop();
    return 0;
}
```

---

6. To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.

```
#include<windows.h>
#include <GL/glut.h>
GLfloat d=0;
void spin()
{
    d=d+0.01;
    if(d>360)
        d=0;
    glutPostRedisplay();
}
void display()
{
    GLfloat L[]={1,1,1};
```

```
  GLfloat P[]={0,1,-1,0};
  GLfloat D1[]={1,0,0};
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  gluLookAt(0,1,3,0,0,0,0,1,0);
  glLightfv(GL_LIGHT0,GL_AMBIENT,L);
  glLightfv(GL_LIGHT0,GL_POSITION,P);
  glRotatef(d,0,1,0);
  glPushMatrix();
    glScalef(1,0.05,1);
    glutSolidCube(1);
  glPopMatrix();
  glPushMatrix();
    glTranslatef(-0.5,-0.5,-0.5);
    glScalef(0.05,1,0.05);
    glutSolidCube(1);
  glPopMatrix();
 glPushMatrix();
   glTranslatef(-0.5,-0.5,0.5);
   glScalef(0.05,1,0.05);
   glutSolidCube(1);
 glPopMatrix();
 glPushMatrix();
   glTranslatef(0.5,-0.5,-0.5);
   glScalef(0.05,1,0.05);
   glutSolidCube(1);
 glPopMatrix();
glPushMatrix();
   glTranslatef(0.5,-0.5,0.5);
```

```
    glScalef(0.05,1,0.05);
    glutSolidCube(1);
 glPopMatrix();
 glPushAttrib(GL_ALL_ATTRIB_BITS);
   glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,D1);
   glPushMatrix();
     glTranslatef(0,0.2,0);
     glutSolidTeapot(0.22);
   glPopMatrix();
 glPopAttrib();
 glutSwapBuffers();
}
int main(int argc, char *argv[])
{
   glutInit(&argc, argv);
   glutInitWindowSize(640,640);
   glutInitWindowPosition(10,10);
   glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
   glutCreateWindow("Teapot");
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum(-1,1,-1,1,2,10);
   glMatrixMode(GL_MODELVIEW);
   glutDisplayFunc(display);
   glutIdleFunc(spin);
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glutMainLoop();
```

```
    return 0;
}
```

---

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<windows.h>
#include<GL/glut.h>
GLfloat d=0;
void spin()
{
   d=d+0.5;
   if(d>360)
      d=0;
   glutPostRedisplay();
}
void tri(GLfloat a[3],GLfloat b[3],GLfloat c[3])
{
   glBegin(GL_TRIANGLES);
      glVertex3fv(a);
      glVertex3fv(b);
      glVertex3fv(c);
   glEnd();
}
void tetra(GLfloat a[3],GLfloat b[3],GLfloat c[3],GLfloat d[3])
{
   glColor3f(0,0,0);
   tri(a,b,c);
   glColor3f(1,0,0);
   tri(a,b,d);
   glColor3f(0,1,0);
```

```
    tri(b,c,d);
    glColor3f(0,0,1);
    tri(a,c,d);
}
void div(GLfloat p0[3],GLfloat p1[3],GLfloat p2[3],GLfloat p3[3],int n)
{
    GLfloat p01[3],p12[3],p20[3],p03[3],p13[3],p23[3];
    if(n==0)
        tetra(p0,p1,p2,p3);
    else
    {
        p01[0]=(p0[0]+p1[0])/2;
        p01[1]=(p0[1]+p1[1])/2;
        p01[2]=(p0[2]+p1[2])/2;

        p12[0]=(p1[0]+p2[0])/2;
        p12[1]=(p1[1]+p2[1])/2;
        p12[2]=(p1[2]+p2[2])/2;

        p20[0]=(p2[0]+p0 [0])/2;
        p20[1]=(p2[1]+p0[1])/2;
        p20[2]=(p2[2]+p0[2])/2;

        p03[0]=(p0[0]+p3[0])/2;
        p03[1]=(p0[1]+p3[1])/2;
        p03[2]=(p0[2]+p3[2])/2;

        p13[0]=(p1[0]+p3 [0])/2;
        p13[1]=(p1[1]+p3[1])/2;
```

```
        p13[2]=(p1[2]+p3[2])/2;

        p23[0]=(p2[0]+p3 [0])/2;
        p23[1]=(p2[1]+p3[1])/2;
        p23[2]=(p2[2]+p3[2])/2;

        div(p0,p01,p20,p03,n-1);
        div(p01,p1,p12,p13,n-1);
        div(p12,p2,p20,p23,n-1);
        div(p03,p13,p23,p3,n-1);
    }
}
void draw()
{
    GLfloat p[4][3]={ {-0.5,-0.5,0.5},{0.5,-0.5,0.5},{0,0.5,0.5},{0,0,-0.5} };
    glClearColor(1,1,1,1);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(d,0,1,0);
    div(p[0],p[1],p[2],p[3],3);
    glutSwapBuffers();
}
int main(int c,char *v[])
{
    glutInit(&c,v);
    glutInitWindowPosition(200,150);
    glutInitWindowSize(648,480);
    glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
    glutCreateWindow("Sierpinski Gasket");
```

```
    glutDisplayFunc(draw);
    glutIdleFunc(spin);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
    return 0;
}
```

---

8. Develop a menu driven program to animate a flag using Bezier Curve algorithm.

```
#include <windows.h>
#include <GL/glut.h>
#include <math.h>
int Anflag=0, yFlag=1, xFlag=1;
float yC=-50,xC=-10;
float x[4],Y1[4],Y2[4],Y3[4];
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];
  glClearColor(1,1,1,1);
  glClear(GL_COLOR_BUFFER_BIT);
  x[0]=300-xC;   x[1]=200;   x[2]=200;   x[3]=100;
  Y1[0]=450;   Y1[1]=450+yC;   Y1[2]=450-yC;   Y1[3]=450;
  Y2[0]=400;   Y2[1]=400+yC;   Y2[2]=400-yC;   Y2[3]=400;
  Y3[0]=350;   Y3[1]=350+yC;   Y3[2]=350-yC;   Y3[3]=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(1,1,0);
   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(0.5,0.5,0.5);
  glRectf(85,460,100,0);
  glFlush();
}
void MyInit()
{
 glMatrixMode(GL_PROJECTION);
```

```
  glLoadIdentity();
  gluOrtho2D(0,500,0,500);
  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);
   glutInitWindowSize(900,900);
   glutInitWindowPosition(10,10);
   glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
   glutCreateWindow("Bezier Curve Flag Animation");
   MyInit();
   glutDisplayFunc(Draw);
   glutIdleFunc(Idle);
   glutMainLoop();
   return 0;
}
```

---

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

```
#include<windows.h>
#include<stdlib.h>
#include<GL/glut.h>
float x1=200,x2=100,x3=200,x4=300,y1=200,y2=300,
y3=400,y4=300;
int le[500],re[500],flag=0;
```

```
void edgedetect(float x1,float y1,float x2,
float y2,int *le,int *re)
{
	float mx,x,temp;
	int i;
	if((y2-y1)<0)
	{
		temp=y1;y1=y2;y2=temp;
		temp=x1;x1=x2;x2=temp;
	}
	if((y2-y1)!=0)
		mx=(x2-x1)/(y2-y1);
	else
		mx=x2-x1;
	x=x1;
	for(i=y1;i<=y2;i++)
	{
		if(x<(float)le[i])
			le[i]=(int)x;
		if(x>(float)re[i])
			re[i]=(int)x;
		x+=mx;
	}
}
void scanfill()
{
	int i,y;
	for(i=0;i<500;i++)
	{
```

```
                le[i]=500;
                re[i]=0;
        }
        edgedetect(x1,y1,x2,y2,le,re);
        edgedetect(x2,y2,x3,y3,le,re);
        edgedetect(x3,y3,x4,y4,le,re);
        edgedetect(x4,y4,x1,y1,le,re);
        for(y=0;y<500;y++)
        {
                if(le[y]<=re[y])
                        for(i=(int)le[y];i<(int)re[y];i++)
        {
         glColor3f(0.0,0.0,1.0);
         glBegin(GL_POINTS);
         glVertex2i(i,y);
         glEnd();
         }
        }
}
void display()
{
   glClearColor(1.0,1.0,1.0,1.0);
        glClear(GL_COLOR_BUFFER_BIT);
        glPointSize(3.0);
        glColor3f(1.0,0.0,0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(x1,y1);
        glVertex2f(x2,y2);
        glVertex2f(x3,y3);
```

```
        glVertex2f(x4,y4);
        glEnd();

        if(flag)
        scanfill();
        glFlush();
}
void mymenu(int id)
{
switch(id)
{
case 1: flag=1;
        break;
case 2: flag=0;
        break;
case 3: exit(0);
}
glutPostRedisplay();
}
int main(int argc,char**argv)
{
        int sub;
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Filling a Polygon using Scan-line Algorithm");
        glutDisplayFunc(display);
        gluOrtho2D(0.0,499.0,0.0,499.0);
```

```
	sub=glutCreateMenu(mymenu);
	glutAddMenuEntry("YES",1);
	glutAddMenuEntry("NO",2);
	glutCreateMenu(mymenu);
	glutAddSubMenu("Polygon Fill",sub);
	glutAddMenuEntry("Exit",3);
	glutAttachMenu(GLUT_RIGHT_BUTTON);
	glutMainLoop();
}
```