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

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

int xStart, yStart, xEnd, yEnd;

void myInit()
{
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0, 500, 0, 500);
      glMatrixMode(GL_MODELVIEW);
}

void setPixel(int x, int y)
{
      glBegin(GL_POINTS);
      glVertex2f(x, y);
      glEnd();
      glFlush();
}

void bresenhamLine(int x1, int y1, int x2, int y2)
{
      if (abs(y2 - y1) < abs(x2 - x1))
      {
            if (x1 > x2)
                  drawLineLow(x2, y2, x1, y1);
            else
                  drawLineLow(x1, y1, x2, y2);
      }
      else
      {
            if (y1 > y2)
                  drawLineHigh(x2, y2, x1, y1);
            else
                  drawLineHigh(x1, y1, x2, y2);
      }
}

void drawLineLow(int x1, int y1, int x2, int y2)
{
      int dx = x2 - x1;
      int dy = y2 - y1;
      int iy = 1;

      if (dy < 0)
      {
            iy = -1;
            dy = -dy;
      }
```

```
	int P = 2 * dy - dx;

	int y = y1;

	for (int x = x1; x <= x2; x++)
	{
		setPixel(x, y);
		if (P >= 0)
		{
			y = y + iy;
			P = P - 2 * dx;
		}
		P = P + 2 * dy;
	}
}

void drawLineHigh(int x1, int y1, int x2, int y2)
{
	int dx = x2 - x1;
	int dy = y2 - y1;
	int ix = 1;

	if (dx < 0)
	{
		ix = -1;
		dx = -dx;
	}

	int P = 2 * dx - dy;

	int x = x1;

	for (int y = y1; y <= y2; y++)
	{
		setPixel(x, y);
		if (P >= 0)
		{
			x = x + ix;
			P = P - 2 * dy;
		}
		P = P + 2 * dx;
	}
}

void display()
{

	glClearColor(1, 1, 1, 1);
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(1, 0, 0);
	glPointSize(2);
	bresenhamLine(xStart, yStart, xEnd, yEnd);
	glFlush();
}
```

```
int main()
{
      glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
      glutInitWindowPosition(100, 100);
      glutInitWindowSize(500, 500);
      glutCreateWindow("Bresenham's Line Drawing Algorthm");
      myInit();
      printf("Enter co-ordinates of first point: ");
      scanf_s("%d %d", &xStart, &yStart);

      printf("Enter co-ordinates of second point: ");
      scanf_s("%d %d", &xEnd, &yEnd);

      glutDisplayFunc(display);
      glutMainLoop();
}
------------------------------------------------------------------------------------------
```

2.Rotate Trianlge

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

int xr, yr;
float angle;
float vertex[3][2] = { { 20, 20 }, { 40,60 }, { 60,20 } };

void drawTriangle()
{
      glBegin(GL_TRIANGLES);
      glVertex2fv(vertex[0]);
      glVertex2fv(vertex[1]);
      glVertex2fv(vertex[2]);
      glEnd();
      glFlush();
}

void display()
{
      glClearColor(1.0, 1.0, 1.0, 1.0);
      glClear(GL_COLOR_BUFFER_BIT);
      glColor3f(1, 0, 0);
      drawTriangle();

      for (int i = 0; i < 3; i++)
      {
            int x = vertex[i][0];
            int y = vertex[i][1];
            vertex[i][0] = xr + (x - xr) * cos(angle) - (y - yr) * sin(angle);
            vertex[i][1] = yr + (x - xr) * sin(angle) + (y - yr) * cos(angle);
      }

      glColor3f(0, 1, 0);
      drawTriangle();
}
```

```
int main()
{
      glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
      glutCreateWindow("Rotate Triangle");
      gluOrtho2D(0, 150, 0, 150);

      printf("Enter Reference Point: ");
      scanf_s("%d %d", &xr, &yr);
      printf("Enter Angle in Degree: ");
      scanf_s("%f", &angle);

      angle = ((angle * 22) / 7) / 180;
      glutDisplayFunc(display);
      glutMainLoop();
}
```

----

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

```
#include <stdio.h>
#include <glut.h>

float points[][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 colors[][3] = { { 1,0,0 },{ 0,1,0 },{ 0,0,1 },{ 1,1,0 },{ 0,1,1 },{ 1,0,1 },{
0.5,0.5,0.5 },{ 0.75,0.25,1 } };
int flag = 2;
float theta[] = { 0,0,0 };

void init()
{
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-2, 2, -2, 2, -2, 2);
      glMatrixMode(GL_MODELVIEW);
}

void idleFunction()
{
      ++theta[flag];
      if (theta[flag] >= 360)
      {
            theta[flag] = 0;
      }
      for (int i = 0; i < 1000000; i++);
      glutPostRedisplay();
}
```

```
void mouse(int key, int state, int x, int y)
{
      if (state == GLUT_DOWN)
      {
            if (key == GLUT_LEFT_BUTTON)
                  flag = 2;
            else if (key == GLUT_MIDDLE_BUTTON)
                  flag = 1;
            else if (key == GLUT_RIGHT_BUTTON)
                  flag = 0;
      }
}

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

void colorCube()
{
      drawPolygon(0, 1, 2, 3);
      drawPolygon(4, 5, 6, 7);
      drawPolygon(5, 1, 2, 6);
      drawPolygon(4, 0, 3, 7);
      drawPolygon(6, 2, 3, 7);
      drawPolygon(5, 1, 0, 4);
}

void display()
{
      glClearColor(1, 1, 1, 1);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      glLoadIdentity();
      glEnable(GL_DEPTH_TEST);
      glRotatef(theta[0], 1, 0, 0);
      glRotatef(theta[1], 0, 1, 0);
      glRotatef(theta[2], 0, 0, 1);
      colorCube();
      glutSwapBuffers();
}
```

```
int main()
{
      glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
      glutCreateWindow("Rotate Cube");
      init();
      glutDisplayFunc(display);
      glutIdleFunc(idleFunction);
      glutMouseFunc(mouse);
      glutMainLoop();
}
```

----

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<glut.h>

float points[][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 colors[][3] = { { 1,0,0 },{ 0,1,0 },{ 0,0,1 },{ 1,1,0 },{ 0,1,1 },{ 1,0,1 },{
0.5,0.5,0.5 },{ 0.75,0.25,1 } };
float theta[3] = { 0,0,0 };
int flag = 2;

void myinit()
{
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-2, 2, -2, 2, 2, 20);
      glMatrixMode(GL_MODELVIEW);
}
int viewer[3] = { 0,0,2 };

void keyboardFunc(unsigned char key, int x, int y)
{
      if (key == 'x') viewer[0]--;
      if (key == 'X') viewer[0]++;
      if (key == 'y') viewer[1]--;
      if (key == 'Y') viewer[1]++;
      if (key == 'z') viewer[2]--;
      if (key == 'Z') viewer[2]++;
      glutPostRedisplay();
}
```

```
void reshape(int w, int h)
{
      glViewport(0, 0, w, h);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      if (w <= h)
            glFrustum(-2, 2, -2 * (GLfloat)h / (GLfloat)w, 2 * (GLfloat)h / (GLfloat)w,
2, 20);
      else
            glFrustum(-2 * (GLfloat)h / (GLfloat)w, 2 * (GLfloat)h / (GLfloat)w, -2, 2,
2, 20);
      glMatrixMode(GL_MODELVIEW);
}

void mouseFunc(int button, int status, int x, int y)
{
      if (status == GLUT_DOWN)
      {
            if (button == GLUT_LEFT_BUTTON)
                  flag = 2;
            if (button == GLUT_MIDDLE_BUTTON)
                  flag = 1;
            if (button == GLUT_RIGHT_BUTTON)
                  flag = 0;
      }
      theta[flag]++;
      if (theta[flag] >= 360)theta[flag] = 0;
      glutPostRedisplay();
}

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

void colorCube()
{
      drawPolygon(0, 1, 2, 3);
      drawPolygon(4, 5, 6, 7);
      drawPolygon(5, 1, 2, 6);
      drawPolygon(4, 0, 3, 7);
      drawPolygon(6, 2, 3, 7);
      drawPolygon(5, 1, 0, 4);
}
```

```
void display()
{
      glClearColor(1, 1, 1, 1);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      glColor3f(1, 0, 0);
      glEnable(GL_DEPTH_TEST);
      glLoadIdentity();
      gluLookAt(viewer[0], viewer[1], viewer[2], 0, 0, 0, 0, 1, 0);
      glRotatef(theta[0], 1, 0, 0);//x
      glRotatef(theta[1], 0, 1, 0);//y
      glRotatef(theta[2], 0, 0, 1);//z
      colorCube();
      glFlush();
      glutSwapBuffers();
}

int main()
{
      glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
      glutInitWindowPosition(100, 100);
      glutInitWindowSize(500, 500);
      glutCreateWindow("Cube with camera movement");
      myinit();
      glutDisplayFunc(display);
      glutMouseFunc(mouseFunc);
      glutKeyboardFunc(keyboardFunc);
      glutReshapeFunc(reshape);
      glutMainLoop();
}
-----------------------------------------------------------------------------------------
```

5. Clip a line using Cohen-Sutherland algorithm

```
#include <stdio.h>
#include <glut.h>
#define opcode int

float xmin, xmax, umin, umax, ymin, ymax, vmin, vmax;
enum { top = 0x8, bottom = 0x4, right = 0x2, left = 0x1 };
float x1, y1, x2, y2, prev_x1, prev_y1, prev_x2, prev_y2;

opcode generateOpcode(int x, int y)
{
      int p = 0;
      if (x < xmin) p = p | left;
      if (x > xmax) p = p | right;
      if (y < ymin) p = p | bottom;
      if (y > ymax) p = p | top;
      return p;
}
```

```
void sutherland()
{
      float m;
      opcode p1, p2, p;
      int x, y;

      x1 = prev_x1;
      y1 = prev_y1;
      x2 = prev_x2;
      y2 = prev_y2;

      bool done = false, accept = false;
      p1 = generateOpcode(x1, y1);
      p2 = generateOpcode(x2, y2);

      while (!done)
      {
            if ((p1 | p2) == 0)
            {
                  accept = true;
                  done = true;
            }
            else if ((p1 & p2) != 0)
            {
                  done = true;
            }
            else
            {
                  m = (y2 - y1) / (x2 - x1);
                  p = p1 != 0 ? p1 : p2;

                  if ((p & left) != 0)
                  {
                        x = xmin;
                        y = y1 + (xmin - x1) * m;
                  }
                  if ((p & right) != 0)
                  {
                        x = xmax;
                        y = y1 + (xmax - x1) * m;
                  }
                  if ((p & top) != 0)
                  {
                        y = ymax;
                        x = x1 + (ymax - y1) / m;
                  }
                  if ((p & bottom) != 0)
                  {
                        y = ymin;
                        x = x1 + (ymin - y1) / m;
                  }
                  if (p == p1)
                  {
                        x1 = x;
                        y1 = y;
                        p1 = generateOpcode(x1, y1);
                  }
                  if (p == p2)
                  {
                        x2 = x;
                        y2 = y;
                        p2 = generateOpcode(x2, y2);
                  }
            }
      }

      if (accept)
      {
            float sx = (umax - umin) / (xmax - xmin);
            float sy = (vmax - vmin) / (ymax - ymin);
            x1 = sx * x1 + umin - sx * xmin;
            y1 = sy * y1 + vmin - sy * ymin;
            x2 = sx * x2 + umin - sx * xmin;
            y2 = sy * y2 + vmin - sy * ymin;

            glBegin(GL_LINES);
            glVertex2f(x1, y1);
            glVertex2f(x2, y2);
            glEnd();
            glFlush();
      }
}

void display()
{
      glClearColor(1, 1, 1, 1);
      glClear(GL_COLOR_BUFFER_BIT);

      glColor3f(1, 0, 0);
      glBegin(GL_LINE_LOOP);
      glVertex2f(xmin, ymin);
      glVertex2f(xmin, ymax);
      glVertex2f(xmax, ymax);
      glVertex2f(xmax, ymin);
      glEnd();

      glColor3f(0, 1, 0);
      glBegin(GL_LINE_LOOP);
      glVertex2f(umin, vmin);
      glVertex2f(umin, vmax);
      glVertex2f(umax, vmax);
      glVertex2f(umax, vmin);
      glEnd();

      glColor3f(0, 0, 0);
      glBegin(GL_LINES);
      glVertex2f(prev_x1, prev_y1);
      glVertex2f(prev_x2, prev_y2);
      glEnd();

      glColor3f(0, 0, 1);
      sutherland();
      glFlush();
}

int main()
{
      glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
      glutInitWindowSize(1200, 1200);
      glutCreateWindow("Clipping");
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0, 1200, 0, 1200);

      glMatrixMode(GL_MODELVIEW);

      printf("Enter Clipping Window Diagonal Points: ");
      scanf_s("%f%f%f%f", &xmin, &ymin, &xmax, &ymax);

      printf("Enter Viewport Diagonal Points: ");
      scanf_s("%f%f%f%f", &umin, &vmin, &umax, &vmax);

      printf("Enter Line End Points: ");
      scanf_s("%f%f%f%f", &prev_x1, &prev_y1, &prev_x2, &prev_y2);

      glutDisplayFunc(display);
      glutMainLoop();
}
```

----

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

void myInit()
{
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-100, 200, -100, 200, -200, 200);
      glMatrixMode(GL_MODELVIEW);
}
```

```
void drawTable()
{
      // Table Top
      glPushMatrix();
      glColor3f(0, 1, 0);
      glTranslatef(50, 40, -50);
      glScalef(50, 5, 50);
      glutSolidCube(1);
      glPopMatrix();

      // 1st leg
      glPushMatrix();
      glColor3f(1, 1, 0);
      glTranslatef(30, 20, -30);
      glScalef(5, 35, 5);
      glutSolidCube(1);
      glPopMatrix();

      // 2nd leg
      glPushMatrix();
      glColor3f(1, 1, 0);
      glTranslatef(70, 20, -30);
      glScalef(5, 35, 5);
      glutSolidCube(1);
      glPopMatrix();

      // 3rd leg
      glPushMatrix();
      glColor3f(1, 1, 0);
      glTranslatef(30, 20, -70);
      glScalef(5, 35, 5);
      glutSolidCube(1);
      glPopMatrix();

      // 4th leg
      glPushMatrix();
      glColor3f(1, 1, 0);
      glTranslatef(70, 20, -70);
      glScalef(5, 35, 5);
      glutSolidCube(1);
      glPopMatrix();

      // Floor
      glPushMatrix();
      glColor3f(1, 0, 1);
      glTranslatef(50, 0, -50);
      glScalef(100, 5, 100);
      glutSolidCube(1);
      glPopMatrix();

      // Left wall
      glPushMatrix();
      glColor3f(1, 0, 0);
      glRotatef(90, 0, 0, 1);
      glTranslatef(50, 0, -50);
```

```
	glScalef(100, 5, 100);
	glutSolidCube(1);
	glPopMatrix();

	// Backside wall
	glPushMatrix();
	glColor3f(1, 0, 0);
	glTranslatef(50, 50, -100);
	glScalef(100, 100, 5);
	glutSolidCube(1);
	glPopMatrix();

	// Tea pot
	glPushMatrix();
	glTranslatef(50, 50, -50);
	glRotatef(30, 0, 1, 0);
	glutSolidTeapot(10);
	glPopMatrix();
}

void display()
{
	glClearColor(0, 0, 0, 1);
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	GLfloat mat_ambient[] = { .7,.7,.7,1 };
	GLfloat mat_diffuse[] = { .5,.5,.5,1 };
	GLfloat mat_spec[] = { 1,1,1,1 };
	GLfloat mat_shininess[] = { 50 };

	glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
	glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
	glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spec);
	glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

	GLfloat light_int[] = { .7,.7,.7,1 };
	GLfloat lightpos[] = { 100,100,100 };
	glLightfv(GL_LIGHT0, GL_POSITION, lightpos);
	glLightfv(GL_LIGHT0, GL_DIFFUSE, light_int);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	gluLookAt(25, 25, 50, 0, 0, -25, 0, 1, 0);
	glMatrixMode(GL_MODELVIEW);
	drawTable();
	glFlush();
}
```

```
int main()
{
      glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
      glutInitWindowSize(500, 500);
      glutInitWindowPosition(100, 100);
      glutCreateWindow("Tea Pot");
      myInit();
      glutDisplayFunc(display);
      glEnable(GL_LIGHTING);
      glEnable(GL_LIGHT0);
      glShadeModel(GL_SMOOTH);
      glEnable(GL_NORMALIZE);
      glEnable(GL_DEPTH_TEST);
      glutMainLoop();
}
-----------------------------------------------------------------------------------------
```

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<glut.h>

struct Point
{
      float x, y, z;

      Point()
      {
            x = y = z = 0;
      }

      Point(float x, float y, float z)
      {
            this->x = x;
            this->y = y;
            this->z = z;
      }
};

int n;
Point points[4] = { Point(0,1,0), Point(0.5,-0.5,0), Point(-0.5,-0.5,0), Point(0,0,1) };

void myInit()
{
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-2, 2, -2, 2, -2, 2);
      glMatrixMode(GL_MODELVIEW);
}
```

```
void drawTriangle(Point a, Point b, Point c)
{
      glBegin(GL_POLYGON);
      glVertex3f(a.x, a.y, a.z);
      glVertex3f(b.x, b.y, b.z);
      glVertex3f(c.x, c.y, c.z);
      glEnd();
}

void divideTriangle(Point a, Point b, Point c, int n)
{
      if (n > 0)
      {
            Point midAB = midPoint(a, b);
            Point midBC = midPoint(b, c);
            Point midCA = midPoint(c, a);

            divideTriangle(a, midAB, midCA, n - 1);
            divideTriangle(midAB, b, midBC, n - 1);
            divideTriangle(midCA, midBC, c, n - 1);
      }
      else
            drawTriangle(a, b, c);
}

Point midPoint(Point a, Point b)
{
      Point mid;
      mid.x = (a.x + b.x) / 2;
      mid.y = (a.y + b.y) / 2;
      mid.z = (a.z + b.z) / 2;
      return mid;
}

void display()
{

      glClearColor(1, 1, 1, 1);
      glClear(GL_COLOR_BUFFER_BIT);
      glColor3f(1, 0, 0);
      divideTriangle(points[0], points[1], points[2], n);
      glColor3f(0, 1, 0);
      divideTriangle(points[3], points[2], points[0], n);
      glColor3f(0, 0, 1);
      divideTriangle(points[3], points[0], points[1], n);
      glColor3f(1, 0, 1);
      divideTriangle(points[3], points[1], points[2], n);
      glFlush();
}
```

```
int main()
{
	glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
	glutInitWindowPosition(100, 100);
	glutInitWindowSize(500, 500);
	glutCreateWindow("3D SIERPINSKI PATTERN");
	printf("Enter the value of n: ");
	scanf_s("%d", &n);
	myInit();
	glutDisplayFunc(display);
	glutMainLoop();
}
```

----

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

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

#define PI 3.1416

GLsizei winWidth = 600, winHeight = 600;
GLfloat xMinWC = 0.0, xMaxWC = 130.0;
GLfloat yMinWC = 0.0, yMaxWC = 130.0;

struct Point3d
{
	GLfloat x, y, z;
};

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 computeBeziarPoint(GLfloat u, Point3d *beziarPoint, GLint nControlPoints, Point3d
controlPoints[], GLint C[])
{
	GLint k, n = nControlPoints - 1;
	GLfloat bezBlendFcn;
	beziarPoint->x = beziarPoint->y = beziarPoint->z = 0.0;
	for (k = 0; k < nControlPoints; k++)
	{
		bezBlendFcn = C[k] * pow(u, k) * pow(1 - u, n - k);
		beziarPoint->x += controlPoints[k].x * bezBlendFcn;
		beziarPoint->y += controlPoints[k].y * bezBlendFcn;
```

```
            beziarPoint->z += controlPoints[k].z * bezBlendFcn;
      }
}

void bezier(Point3d controlPoints[], GLint nControlPoints, GLint nBeziarCurvePoints)
{
      Point3d beziarCurvePoint;
      GLfloat u;
      GLint *C, k;
      C = new GLint[nControlPoints];
      bino(nControlPoints - 1, C);
      glBegin(GL_LINE_STRIP);


      for (k = 0; k <= nBeziarCurvePoints; k++)
      {
            u = GLfloat(k) / GLfloat(nBeziarCurvePoints);
            computeBeziarPoint(u, &beziarCurvePoint, nControlPoints, controlPoints, C);
            glVertex2f(beziarCurvePoint.x, beziarCurvePoint.y);
      }
      glEnd();
      delete[] C;
}

void display()
{
      GLint nCtrlPts = 4, nBezCurvePts = 20;
      static float theta = 0;
      Point3d 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);
      glPointSize(5);
      glPushMatrix();
      glLineWidth(5);


      for (int i = 0; i < 24; i++)
      {
            glTranslatef(0, -0.8, 0);
            bezier(ctrlPts, nCtrlPts, nBezCurvePts);
      }

      glPopMatrix();
      glLineWidth(5);
      glBegin(GL_LINES);
      glVertex2f(20, 100);
      glVertex2f(20, 40);
      glEnd();
      glFlush();
```

```
	glutPostRedisplay();
	glutSwapBuffers();
}

void reshape(GLint w, GLint h)
{
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(xMinWC, xMaxWC, yMinWC, yMaxWC);
	glClear(GL_COLOR_BUFFER_BIT);
}


void menu(int op)
{
	if (op == 1)
		glColor3f(1.0, 0.0, 0.0);
	else if (op == 2)
		glColor3f(0.0, 1.0, 0.0);
	else if (op == 3)
		glColor3f(0.0, 0.0, 1.0);
	else if (op == 4)
		exit(0);
	glutPostRedisplay();
}

int main()
{
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowPosition(50, 50);
	glutInitWindowSize(winWidth, winHeight);
	glutCreateWindow("Bezier Curve");

	glutCreateMenu(menu);
	glutAddMenuEntry("Red", 1);
	glutAddMenuEntry("Green", 2);
	glutAddMenuEntry("Blue", 3);
	glutAddMenuEntry("Quit", 4);
	glutAttachMenu(GLUT_RIGHT_BUTTON);

	glutDisplayFunc(display);
	glutReshapeFunc(reshape);
	glutMainLoop();
}
```

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

```
#include<stdio.h>
#include<glut.h>

void display()
{
      glClearColor(1.0, 1.0, 1.0, 1.0);
      glClear(GL_COLOR_BUFFER_BIT);
      float x1 = 200, y1 = 200, x2 = 100, y2 = 300, x3 = 200, y3 = 400, x4 = 300, y4 =
300;

      glBegin(GL_LINE_LOOP);
      glVertex2f(x1, y1);
      glVertex2f(x2, y2);
      glVertex2f(x3, y3);
      glVertex2f(x4, y4);
      glEnd();
      scanfill(x1, y1, x2, y2, x3, y3, x4, y4);
}

void scanfill(float x1, float y1, float x2, float y2, float x3, float y3, float x4, float
y4)
{
      int le[500], re[500];
      for (int 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 (int y = 0; y < 500; y++)
            if (le[y] <= re[y])
                  for (int i = le[y]; i < re[y]; i++)
                        drawPoint(i, y);
}

void edgeDetect(float x1, float y1, float x2, float y2, int *le, int *re)
{
      float mx, temp;
      if (y2 < y1)
      {
            temp = y2; y2 = y1; y1 = temp;
            temp = x2; x2 = x1; x1 = temp;
      }

      mx = (x2 - x1) / (y2 - y1);
      float x = x1;
```

```
    for (int i = (int)y1; i < y2; i++)
    {
        if (x < le[i])
        {
            le[i] = (int)x;
        }

        if (x > re[i])
        {
            re[i] = (int)x;
        }

        x += mx;
    }
}

void drawPoint(int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2f(x, y);
    glEnd();
    glFlush();
}

void menu(int choice)
{
    switch (choice)
    {
    case 0:
        glColor3f(1.0, 0.0, 0.0);
        break;
    case 1:
        glColor3f(0.0, 1.0, 0.0);
        break;
    case 2:
        glColor3f(0.0, 0.0, 1.0);
        break;
    case 3:
        exit(0);
    }
    glutPostRedisplay();
}
```

```
int main()
{
      glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
      glutCreateWindow("Title");
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0, 500, 0, 500);
      glColor3f(1.0, 0.0, 0.0);

      glutCreateMenu(menu);
      glutAddMenuEntry("Red", 0);
      glutAddMenuEntry("Green", 1);
      glutAddMenuEntry("Blue", 2);
      glutAddMenuEntry("Quit", 3);
      glutAttachMenu(GLUT_RIGHT_BUTTON);


      glutDisplayFunc(display);
      glutMainLoop();
}
```