```
#include<GL/glut.h>
#include<stdio.h>
int x1, y1, x2, y2;
void draw_pixel(int x, int y)
{
 glColor3f(1.0,0.0,0.0);
 glBegin(GL_POINTS);
 glVertex2i(x, y);
 glEnd();
}
void bresenhams_line_draw(int x1, int y1, int x2, int y2)
{
 int dx = x2 - x1;
 int dy = y2 - y1;
 int m = dy/dx;
 if (m < 1)
 {
 int decision_parameter = 2*dy - dx;
 int x = x1;
 int y = y1;
 if (dx < 0)
 {
 x = x2;
 y = y2;
 x2 = x1;
 }
 draw_pixel (x, y);
 while (x < x2)
 {
 if (decision_parameter >= 0)
 {
 x = x+1;
 y = y+1;
 decision_parameter = decision_parameter + 2*dy - 2*dx * (y+1 - y);
 }
 else
```

```
{
x = x+1;
y = y;
decision_parameter = decision_parameter + 2*dy - 2*dx * (y - y);
}
draw_pixel (x, y);
}
}
else if (m > 1)
{
int decision_parameter = 2*dx - dy;
int x = x1; // initial x
int y = y1; // initial y
if (dy < 0)
{
x = x2;
y = y2;
y2 = y1;
}
draw_pixel (x, y);
while (y < y2)
{
if (decision_parameter >= 0)
{
x = x+1;
y = y+1;
decision_parameter = decision_parameter + 2*dx - 2*dy * (x+1 - x);
}
else
{
y = y+1;
x = x;
decision_parameter = decision_parameter + 2*dx - 2*dy * (x- x);
}
draw_pixel(x, y);
}
}
else if (m == 1)
{
```

```
 int x = x1;
 int y = y1;
 draw_pixel (x, y);
 while (x < x2)
 {
 x = x+1;
 y = y+1;
 draw_pixel (x, y);
 }
 }
}
void init()
{
 glClearColor(1,1,1,1);
 gluOrtho2D(0.0, 500.0, 0.0, 500.0);
}
void display()
{
 glClear(GL_COLOR_BUFFER_BIT);
 bresenhams_line_draw(x1, y1, x2, y2);
 glFlush();
}
int main(int argc, char **argv)
{
 printf( "Enter Start Points (x1,y1)\n");
 scanf("%d %d", &x1, &y1);
 printf( "Enter End Points (x2,y2)\n");
 scanf("%d %d", &x2, &y2);
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
 glutInitWindowSize(500, 500);
 glutInitWindowPosition(220, 200);
 glutCreateWindow("Bresenham's Line Drawing");
 init();
 glutDisplayFunc(display);
 glutMainLoop();
}
```

********** PROGRAM 2**********

```
#include<GL/glut.h>
#include<stdio.h>
int x,y;
int where_to_rotate=0; // don't rotate initially
float rotate_angle=0; // initial angle
float translate_x=0,translate_y=0; // initial translation
void draw_pixel(float x1, float y1)
{
 glPointSize(5);
 glBegin(GL_POINTS);
 glVertex2f(x1,y1); // plot a single point
 glEnd();
}
void triangle(int x, int y)
{
 glColor3f(1,0,0);
 glBegin(GL_POLYGON); // drawing a Triangle
 glVertex2f(x,y);
 glVertex2f(x+400,y+300);
 glVertex2f(x+300,y+0);
 glEnd();
}
void display()
{
 glClear(GL_COLOR_BUFFER_BIT);
 glLoadIdentity();
 glColor3f(1,1,1); // mark origin point as white dot
 draw_pixel(0,0); // plot origin - white colour
 if (where_to_rotate == 1) // rotate around origin
 {
 translate_x = 0; // no translation for rotation around origin
 translate_y = 0;
 rotate_angle += 1; // the amount of rotation angle
 }
 if (where_to_rotate == 2) // rotate around Fixed Point
 {
```

```
 translate_x = x; // SET the translation to wherever the customer says
 translate_y = y;
 rotate_angle += 1; // the amount of rotation angle
 glColor3f(0,0,1); // mark the customer coordinate as blue dot
 draw_pixel(x,y); // plot the customer coordinate - blue colour
 }

glTranslatef(translate_x, translate_y, 0); // ACTUAL translation +ve
 glRotatef(rotate_angle, 0, 0, 1); // rotate
 glTranslatef(-translate_x, -translate_y, 0); // ACTUAL translation -ve
 triangle(translate_x,translate_y); // what to rotate? – TRIANGLE boss
 glutPostRedisplay(); // call display function again and again
 glutSwapBuffers(); // show the output
}
void init()
{
 glClearColor(0,0,0,1); //setting to black
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 gluOrtho2D(-800, 800, -800, 800);
 glMatrixMode(GL_MODELVIEW);
}
void rotateMenu (int option)
{
 if(option==1)
 where_to_rotate=1; // rotate around origin
 if(option==2)
 where_to_rotate=2; // rotate around customer's coordinates
 if(option==3)
 where_to_rotate=3; // stop rotation
}
int main(int argc, char **argv)
{
 printf( "Enter Fixed Points (x,y) for Rotation: \n");
 scanf("%d %d", &x, &y); // getting the user's coordinates to rotate
 glutInit(&argc, argv); // initialize the graphics system
 glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB); // SINGLE also works
 glutInitWindowSize(800, 800); // 800 by 800 size..you can change it
 glutInitWindowPosition(0, 0); // where do you wanna see your window
```

```
 glutCreateWindow("Create and Rotate Triangle"); // title
 init(); // initialize the canvas
 glutDisplayFunc(display); // call display function
 glutCreateMenu(rotateMenu); // menu items
 glutAddMenuEntry("Rotate around ORIGIN",1);
 glutAddMenuEntry("Rotate around FIXED POINT",2);
glutAddMenuEntry("Stop Rotation",3);
 glutAttachMenu(GLUT_RIGHT_BUTTON);
 glutMainLoop(); // run forever
}
```

********** PROGRAM 3**********

```
#include<stdlib.h>
#include<GL/glut.h>
GLfloat vertices[] = { -1, -1, -1,
 1, -1, -1,
 1, 1, -1,
 -1, 1, -1,
 -1, -1, 1,
 1, -1, 1,
 1, 1, 1,
 -1, 1, 1
 };
GLfloat colors[] = { 0, 0, 0, // white color
 1, 0, 0, // red color .. so on for eight faces of cube
 1, 1, 0,
 0, 1, 0,
 0, 0, 1,
 1, 0, 1,
 1, 1, 1,
 0, 1, 1
 };
GLubyte cubeIndices[] = {0, 3, 2, 1,
 2, 3, 7, 6,
 0, 4, 7, 3,
 1, 2, 6, 5,
 4, 5, 6, 7,
 0, 1, 5, 4
```

```
 };
static GLfloat theta[]= {0, 0, 0}; // initial angles
static GLint axis=2; // let us assume the right mouse button has been clicked initially
void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glRotatef (theta[0], 1, 0, 0); // first angle rotation via x axis
 glRotatef (theta[1], 0, 1, 0); // second angle rotation via y axis
 glRotatef (theta[2], 0, 0, 1); // third angle rotation via z axis
 glDrawElements(GL_QUADS,24,GL_UNSIGNED_BYTE,cubeIndices); // draw the cube
 glutSwapBuffers(); // show the output

void spinCube()
{
 theta[axis] += 2; // rotate every 2 degrees
 if (theta[axis] > 360) // it the rotation angle crosses 360 degrees, make it 0 degree
 theta[axis] -= 360;
 glutPostRedisplay(); // call display again
}
void mouse(int btn, int state, int x, int y)
{
 if (btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
 axis=0; // x axis rotation
 if (btn==GLUT_MIDDLE_BUTTON && state==GLUT_DOWN)
 axis=1; // y axis rotation
 if (btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
 axis=2; // z axis rotation
}
void myReshape(int w, int h)
{
 glViewport(0,0,w,h);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 if(w<=h)
 glOrtho (-2, 2, -2*(GLfloat)h/(GLfloat)w, 2*(GLfloat)h / (GLfloat)w, -10, 10);
 else
 glOrtho (-2*(GLfloat)w/(GLfloat)h, 2*(GLfloat)w / (GLfloat)h, -2, 2, -10, 10);
 glMatrixMode(GL_MODELVIEW);
```

```
}
int main(int argc, char **argv)
{
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
 glutInitWindowSize(500, 500);
 glutCreateWindow("Spin a color cube");
 glutReshapeFunc(myReshape); // calls myReshape whenever we change the window size
 glutDisplayFunc(display); // call display function
 glutIdleFunc(spinCube);
}
glutMouseFunc(mouse); // calls mouse function whenever we interact with mouse
 glEnable(GL_DEPTH_TEST); // enables depth – for 3D
 glEnableClientState(GL_COLOR_ARRAY); // enables colour and vertex properties
 glEnableClientState(GL_VERTEX_ARRAY);
 glVertexPointer(3, GL_FLOAT, 0, vertices); // glVertexPointer(size,type,stride,pointer)
 glColorPointer(3, GL_FLOAT, 0, colors); // glColorPointer(size,type,stride,pointer)
 glColor3f(1, 1, 1);
 glutMainLoop();
}

**************************************************PROGRAM
4**************************************************
#include <stdlib.h>
#include <GL/glut.h>
GLfloat vertices[][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}
 };
GLfloat colors[][3] = { {1, 0, 0},
 {1, 1, 0},
 {0, 1, 0},
 {0, 0, 1},
 {1, 0, 1},
```

```
 {1, 1, 1},
 {0, 1, 1},
 {0.5, 0.5, 0.5}
 };
GLfloat theta[] = {0, 0, 0};
GLint axis = 2;
GLdouble viewer[]= {0, 0, 5}; // initial viewer location //
void polygon(int a, int b, int c, int d)
{
 glBegin(GL_POLYGON);
 glColor3fv(colors[a]);
 glVertex3fv(vertices[a]);
 glColor3fv(colors[b]);
 glVertex3fv(vertices[b]);
 glColor3fv(colors[c]);
 glVertex3fv(vertices[c]);
 glColor3fv(colors[d]);
 glVertex3fv(vertices[d]);
 glEnd();
}
void colorcube(void)
{
 polygon (0,3,2,1);
 polygon (0,4,7,3);
 polygon (5,4,0,1);
 polygon (2,3,7,6);
 polygon (1,2,6,5);
 polygon (4,5,6,7);
}
void display(void)
{
 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[0], 1, 0, 0);
 glRotatef (theta[1], 0, 1, 0);
 glRotatef (theta[2], 0, 0, 1);
 colorcube();
 glFlush();
```

```
 glutSwapBuffers();
}
void mouse(int btn, int state, int x, int y)
{
 if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
axis = 0;
 if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
axis = 1;
 if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
axis = 2;
 theta[axis] += 2;
 if( theta[axis] > 360 )
theta[axis] -= 360;
 display();
}
void keys(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;
 display();
}
void myReshape(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, 2, -2 * (GLfloat) w/ (GLfloat) h, 2* (GLfloat) w / (GLfloat) h, 2, 20);
 glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char **argv)
{
 glutInit(&argc, argv);
```

```
 glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
 glutInitWindowSize(500, 500);
 glutCreateWindow("Colorcube Viewer");
 glutReshapeFunc(myReshape);
 glutDisplayFunc(display);
 glutMouseFunc(mouse);
 glutKeyboardFunc(keys);
 glEnable(GL_DEPTH_TEST);
 glutMainLoop();
}
**********************************************PROGRAM
5**********************************************************
#include <stdio.h>
#include <GL/glut.h>
double xmin = 50, ymin = 50, xmax = 100, ymax = 100; //window coordinates
double xvmin = 200, yvmin = 200, xvmax = 300, yvmax = 300; //viewport coordinates
const int LEFT = 1; // assuming code words for LEFT, RIGHT, BOTTOM & TOP.
const int RIGHT = 2;
const int BOTTOM = 4;
const int TOP = 8;
int ComputeOutCode (double x, double y)
{
 int code = 0;
 if (y > ymax) //above the clip window
 code |= TOP;
 else if (y < ymin) //below the clip window
 code |= BOTTOM;
 if (x > xmax) //to the right of clip window
 code |= RIGHT;
 else if (x < xmin) //to the left of clip window
 code |= LEFT;
 return code; //return the calculated code
}
void CohenSutherland(double x0, double y0, double x1, double y1)
{
 int outcode0, outcode1, outcodeOut;
 bool accept = false, done = false;
 outcode0 = ComputeOutCode (x0, y0); //calculate the region of 1st point
 outcode1 = ComputeOutCode (x1, y1); //calculate the region of 2nd point
```

```
do
{
if ( ! (outcode0 | outcode1))
{
accept = true; //both the points
done = true; are inside the window
}
else if (outcode0 & outcode1)
done = true; //both are outside
else
{
double x, y;
double m = (y1 - y0) / (x1 - x0);
outcodeOut = outcode0 ? outcode0: outcode1;
if (outcodeOut & TOP)
{
x = x0 + (1/m) * (ymax - y0);
y = ymax;
}
else if (outcodeOut & BOTTOM)
{
x = x0 + (1/m) * (ymin - y0);
y = ymin;
}
else if (outcodeOut & RIGHT)
{
y = y0 + m * (xmax - x0);
x = xmax;
}
else
{
y = y0 + m * (xmin - x0);
x = xmin;
}
/* Intersection calculations are done,
go ahead and mark the clipped line */
if (outcodeOut == outcode0)
{
x0 = x;
```

```
 y0 = y;
 outcode0 = ComputeOutCode (x0, y0);
 }
 else
 {
 x1 = x;
 y1 = y;
 outcode1 = ComputeOutCode (x1, y1);
 }
 }
 }
 while (!done);

 if (accept)
 {
 double sx = (xvmax - xvmin) / (xmax - xmin);
 double sy = (yvmax - yvmin) / (ymax - ymin);
 double vx0 = xvmin + (x0 - xmin) * sx;
 double vy0 = yvmin + (y0 - ymin) * sy;
 double vx1 = xvmin + (x1 - xmin) * sx;
 double vy1 = yvmin + (y1 - ymin) * sy;
glBegin(GL_LINE_LOOP); // draw the zoomed rectangle
 glVertex2f (xvmin, yvmin);
 glVertex2f (xvmax, yvmin);
 glVertex2f (xvmax, yvmax);
 glVertex2f (xvmin, yvmax);
 glEnd();
 glBegin(GL_LINES); // draw the zoomed clipped line
 glVertex2d (vx0, vy0);
 glVertex2d (vx1, vy1);
 glEnd();
 }
}
 void display()
{
 double x0 = 60, y0 = 20, x1 = 80, y1 = 120; // the line coordinates
 glClear (GL_COLOR_BUFFER_BIT);
 glColor3f(1, 1, 1); // white colour to draw line
 glBegin (GL_LINES);
```

```
 glVertex2d (x0, y0); // draw the line that has to be clipped
 glVertex2d (x1, y1);
 glEnd ();
 glBegin (GL_LINE_LOOP); // draw the clipping / viewing rectangle window
 glVertex2f (xmin, ymin);
 glVertex2f (xmax, ymin);
 glVertex2f (xmax, ymax);
 glVertex2f (xmin, ymax);
 glEnd ();
 CohenSutherland (x0, y0, x1, y1); // call the algorithm
 glFlush (); // show the output
}
void init()
{
 glClearColor (0, 0, 0, 1); //black background colour
 gluOrtho2D (0, 500, 0, 500);
}
int main(int argc, char **argv)
{
 glutInit (&argc,argv);
 glutInitDisplayMode (GLUT_SINGLE|GLUT_RGB);
 glutInitWindowSize (500, 500);
 glutInitWindowPosition (0, 0);
 glutCreateWindow ("Cohen Sutherland Line Clipping Algorithm");
 init();
 glutDisplayFunc(display);
 glutMainLoop();
}

********************************************** PROGRAM
6************************************************************
#include<GL/glut.h>
void teapot(GLfloat x, GLfloat y, GLfloat z)
{
 glPushMatrix (); //save the current state
 glTranslatef (x, y, z); //move your item appropriately
 glutSolidTeapot (0.1); //render your teapot
 glPopMatrix (); //get back your state with the recent changes that you have done
}
```

```
void tableTop(GLfloat x, GLfloat y, GLfloat z) // table top which is actually a CUBE
{
 glPushMatrix ();
 glTranslatef (x, y, z);
 glScalef (0.6, 0.02, 0.5);
 glutSolidCube (1);
 glPopMatrix ();
}
void tableLeg(GLfloat x, GLfloat y, GLfloat z) // table leg which is actually a CUBE
{
 glPushMatrix ();
 glTranslatef (x, y, z);
 glScalef (0.02, 0.3, 0.02);
 glutSolidCube (1);
 glPopMatrix ();
}
void wall(GLfloat x, GLfloat y, GLfloat z) // wall which is actually a CUBE
{
 glPushMatrix ();
 glTranslatef (x, y, z);
 glScalef (1, 1, 0.02);
 glutSolidCube (1);
 glPopMatrix ();
}
void light() // set the lighting arrangements
{
 GLfloat mat_ambient[] = {1, 1, 1, 1}; // ambient colour
 GLfloat mat_diffuse[] = {0.5, 0.5, 0.5, 1};
 GLfloat mat_specular[] = {1, 1, 1, 1};
 GLfloat mat_shininess[] = {50.0f};

glMaterialfv (GL_FRONT, GL_AMBIENT, mat_ambient);
 glMaterialfv (GL_FRONT, GL_DIFFUSE, mat_diffuse);
 glMaterialfv (GL_FRONT, GL_SPECULAR, mat_specular);
 glMaterialfv (GL_FRONT, GL_SHININESS, mat_shininess);
 GLfloat light_position[] = {2, 6, 3, 1};
 GLfloat light_intensity[] = {0.7, 0.7, 0.7, 1};
 glLightfv (GL_LIGHT0, GL_POSITION, light_position);
 glLightfv (GL_LIGHT0, GL_DIFFUSE, light_intensity);
```

```
}
void display()
{
 GLfloat teapotP = -0.07, tabletopP = -0.15, tablelegP = 0.2, wallP = 0.5;
 glClear (GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 gluLookAt (-2, 2, 5, 0, 0, 0, 0, 1, 0); // camera position & viewing
 light (); //Adding light source to your project
 teapot (0, teapotP, 0); //Create teapot
 tableTop (0, tabletopP, 0); //Create table's top
 tableLeg (tablelegP, -0.3, tablelegP); //Create 1st leg
 tableLeg (-tablelegP, -0.3, tablelegP); //Create 2nd leg
 tableLeg (-tablelegP, -0.3, -tablelegP); //Create 3rd leg
 tableLeg (tablelegP, -0.3, -tablelegP); //Create 4th leg
 wall (0, 0, -wallP); //Create 1st wall
 glRotatef (90, 1, 0, 0);
 wall (0, 0, wallP); //Create 2nd wall
 glRotatef (90, 0, 1, 0);
 wall (0, 0, wallP); //Create 3rd wall
 glFlush (); // show the output to the user
}
void init()
{
 glClearColor (0, 0, 0, 1); // black colour background
 glMatrixMode (GL_PROJECTION);
 glLoadIdentity ();
 glOrtho (-1, 1, -1, 1, -1, 10);
 glMatrixMode (GL_MODELVIEW);
}
int main (int argc, char **argv)
{
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
 glutInitWindowSize(500, 500);
 glutInitWindowPosition(0, 0);
 glutCreateWindow("Teapot on a table");
 init();
 glutDisplayFunc(display);
 glEnable(GL_LIGHTING); // enable the lighting properties
```

```
 glEnable(GL_LIGHT0); // enable the light source
 glShadeModel(GL_SMOOTH); // for smooth shading (select flat or smooth shading)
 glEnable(GL_NORMALIZE); // If enabled and no vertex shader is active, normal vectors
are normalized to unit length after transformation and before
lighting.
 glEnable(GL_DEPTH_TEST); // do depth comparisons and update the depth buffer.
 glutMainLoop();
}

**************************************************PROGRAM
7**************************************************
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
typedef float point[3];
point v[]= {{0, 0, 1}, {0, 1, 0}, {-1, -0.5, 0}, {1, -0.5, 0}};
int n;
void triangle(point a, point b, point c)
{
 glBegin(GL_POLYGON);
 glVertex3fv(a);
 glVertex3fv(b);
 glVertex3fv(c);
 glEnd();
}
void divide_triangle(point a, point b, point c, int n)
{
 point v1,v2,v3;
 int j;
 if(n>0)
 {
 for(j=0; j<3; j++)
 v1[j] = (a[j]+b[j])/2; // calculate mid-point between a and b
 for(j=0; j<3; j++)
 v2[j] = (a[j]+c[j])/2; // calculate mid-point between a and c
 for(j=0; j<3; j++)
 v3[j] = (c[j]+b[j])/2; // calculate mid-point between c and b
 divide_triangle(a,v1,v2,n-1); // divide triangle between points a, ab/2, ac/2 recursively
```

```
divide_triangle(c,v2,v3,n-1);

 divide_triangle(b,v3,v1,n-1);

 }
 else
triangle (a,b,c);// draw triangle
}
void tetrahedron(int n)
{
 glColor3f(1, 0, 0); // assign color for each of the side
 divide_triangle(v[0], v[1], v[2], n); // draw triangle between a, b, c
 glColor3f(0, 1, 0);
 divide_triangle(v[3], v[2], v[1], n);
 glColor3f(0, 0, 1);
 divide_triangle(v[0], v[3], v[1], n);
 glColor3f(0, 0, 0);
 divide_triangle(v[0], v[2], v[3], n);
}
void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 tetrahedron(n);
 glFlush(); // show the output
}
void myReshape(int w,int h) // please see the earlier program for explanation on this
{
 glViewport(0, 0, w, h);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 if(w<=h)
 glOrtho(-2, 2, -2*(GLfloat)h/(GLfloat)w, 2*(GLfloat)h/(GLfloat)w, -10, 10);
 else
 glOrtho(-2*(GLfloat)w/(GLfloat)h, 2*(GLfloat)w/(GLfloat)h, -2, 2, -10, 10);
 glMatrixMode(GL_MODELVIEW);
 glutPostRedisplay();
}
int main(int argc,char ** argv)
```

```
{
 printf("No of Recursive steps/Division: ");
 scanf("%d",&n);
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
 glutCreateWindow(" 3D Sierpinski gasket");
 glutReshapeFunc(myReshape);
glutDisplayFunc(display); // call display function
 glEnable(GL_DEPTH_TEST); // do depth comparisons and update the depth buffer.
 glClearColor(1, 1, 1, 0);
 glutMainLoop();
 return 0;
}
```

********** PROGRAM 8**********

```
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#define PI 3.1416
float theta = 0;
struct point
{
GLfloat x, y, z;
};
int factorial (int n)
{
if (n<=1)
return (1);
else
n = n * factorial ( n-1 );
return n;
}
void computeNcR (int n, int *hold_ncr_values)
{
int r;
for (r=0; r<=n; r++) //start from nC0, then nC1, nC2, nC3 till nCn
{
hold_ncr_values [r] = factorial (n) / ( factorial (n-r) * factorial (r) );
```

```
}
}
void computeBezierPoints (float t, point *actual_bezier_point, int number_of_control_points,
 point *control_points_array, int *hold_ncr_values) // 5 parameters
{
int i, n = number_of_control_points - 1;
float bernstein_polynomial;
actual_bezier_point -> x = 0;
actual_bezier_point -> y = 0;
actual_bezier_point -> z = 0;
for ( i=0; i<number_of_control_points; i++ )
{
bernstein_polynomial = hold_ncr_values [i] * pow(t, i) * pow( 1-t, n-i);
actual_bezier_point->x += bernstein_polynomial * control_points_array [i].x;
actual_bezier_point->y += bernstein_polynomial * control_points_array [i].y;
actual_bezier_point->z += bernstein_polynomial * control_points_array [i].z;
 }
}
void Bezier (point *control_points_array, int number_of_control_points, int
number_of_bezier_points)
{
point actual_bezier_point;
float t;
int *hold_ncr_values, i;
hold_ncr_values = new int [number_of_control_points]; // to hold the nCr values
computeNcR (number_of_control_points - 1, hold_ncr_values); // calculate nCr values
glBegin (GL_LINE_STRIP);
for(i=0; i<=number_of_bezier_points; i++)
{
t=float (i) / float (number_of_bezier_points);
computeBezierPoints ( t, &actual_bezier_point, number_of_control_points,
 control_points_array, hold_ncr_values );// 5 parameters
glVertex2f (actual_bezier_point.x, actual_bezier_point.y);
}
glEnd ();
delete [] hold_ncr_values;
}
void display()
{
```

```
glClear (GL_COLOR_BUFFER_BIT);
int number_of_control_points= 4, number_of_bezier_points= 20;
point control_points_array[4]= {{100, 400, 0}, {150, 450, 0}, {250, 350, 0},{300, 400, 0}};
control_points_array[1].x += 50 * sin (theta * PI/180.0); // for animating the flag
control_points_array[1].y += 25 * sin (theta * PI/180.0);
control_points_array[2].x -= 50 * sin ((theta+30) * PI/180.0);
control_points_array[2].y -= 50 * sin ((theta+30) * PI/180.0);
control_points_array[3].x -= 25 * sin ((theta-30) * PI/180.0);
control_points_array[3].y += sin ((theta-30) * PI/180.0);
theta += 2; //animating speed
glPushMatrix ();
glPointSize (5);
glColor3f (1, 0.4, 0.2); //Indian flag: Saffron color code
for (int i=0; i<50; i++)
{
glTranslatef(0, -0.8, 0 );
bezier(control_points_array, number_of_control_points, number_of_bezier_points);
}
glColor3f(1, 1, 1); //Indian flag: white color code
for(int i=0; i<50; i++)
{
glTranslatef(0, -0.8, 0);
bezier(control_points_array, number_of_control_points, number_of_bezier_points);
}
glColor3f(0, 1, 0); //Indian flag: green color code
for(int i=0; i<50; i++)
{
glTranslatef(0, -0.8, 0);
bezier(control_points_array, number_of_control_points, number_of_bezier_points);
}
glPopMatrix();
glLineWidth(5);
glColor3f(0.7, 0.5,0.3); //pole colour
glBegin(GL_LINES);
glVertex2f(100,400);
glVertex2f(100,40);
glEnd();
glutPostRedisplay(); // call display again
glutSwapBuffers(); // show the output
```

```
}
void init ()
{
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0,500,0,500);
}
int main(int argc, char ** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowPosition(0, 0);
glutInitWindowSize(500,500);
glutCreateWindow ("Bezier Curve - updated");
init ();
glutDisplayFunc (display);
glutMainLoop ();
}
```

********** PROGRAM 9

```
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
float x1, x2, x3, x4, y1, y2, y3, y4; // our polygon has 4 lines - so 8 coordinates
void edgedetect(float x1, float y1, float x2, float y2, int *left_edge, int *right_edge)
{
 float x_slope, x, temp;
 int i;
 if ((y2-y1)<0) // decide where to start
 {
 temp = y1;
 y1 = y2;
 y2 = temp;
 temp = x1;
 x1 = x2;
 x2 = temp;
 }
 if ((y2-y1)!=0) // compute the values
```

```
 x_slope = (x2 - x1) / (y2 - y1);
 else
 x_slope = x2 - x1;
 x = x1;
 for (i = y1; i <= y2; i++) // fill the values
 {
 if (x < left_edge[i])
 left_edge[i] = x;
 if (x > right_edge[i])
 right_edge[i] = x;
 x = x + x_slope;
 }
}
void draw_pixel (int x, int y) // fill the polygon point by point (pixel by pixel)
{
 glColor3f (1, 1, 0); // fill the RHOMBUS in yellow colour
 glBegin (GL_POINTS);
 glVertex2i (x, y);
 glEnd ();
}
void scanfill (float x1, float y1, float x2, float y2, float x3, float y3, float x4, float y4)
{
 int left_edge[500], right_edge[500];
 int i, y;
 for (i = 0; i <= 500; i++)
 {
 left_edge [i] = 500; // fill all the left_edge values as 500 initially
 right_edge [i] = 0; // fill all the right_edge values as 0 initially
 }
 edgedetect (x1, y1, x2, y2, left_edge, right_edge); // first line
 edgedetect (x2, y2, x3, y3, left_edge, right_edge); // second line
 edgedetect (x3, y3, x4, y4, left_edge, right_edge); // third line
 edgedetect (x4, y4, x1, y1, left_edge, right_edge); // fourth line
 for (y = 0; y <= 500; y++) // now that you have calculated all values, start filling
 { from left edge to right edge row by row pixel by pixel
 if (left_edge[y] <= right_edge[y])
 {
 for (i = left_edge[y]; i <= right_edge[y]; i++)
 {
```

```
 draw_pixel (i, y);
 glFlush ();
 }
 }
 }
}
void display()
{
 x1 = 200, y1 = 200; // RHOMBUS coordinates
 x2 = 100, y2 = 300;
 x3 = 200, y3 = 400;
 x4 = 300, y4 = 300;
 glClear (GL_COLOR_BUFFER_BIT);
 glColor3f (0, 0, 1); // blue RHOMBUS
 glBegin (GL_LINE_LOOP); // draw the RHOMBUS
 glVertex2f (x1, y1);
 glVertex2f (x2, y2);
 glVertex2f (x3, y3);
 glVertex2f (x4, y4);
 glEnd ();
scanfill (x1, y1, x2, y2, x3, y3, x4, y4); // FILL the RHOMBUS
}
void init()
{
 glClearColor (1, 1, 1, 1);
 gluOrtho2D (0, 499, 0, 499);
}
int main (int argc, char** argv)
{
 glutInit (&argc, argv);
 glutInitDisplayMode (GLUT_SINGLE|GLUT_RGB);
 glutInitWindowSize (500, 500);
 glutInitWindowPosition (0, 0);
 glutCreateWindow ("Filling a Polygon using Scan-line Algorithm");
 init ();
 glutDisplayFunc (display);
 glutMainLoop ();
}
```

****