

boundary fill algorithm implementaion using traingle drawing :

```
#include<windows.h>
#include <iostream>
#include <math.h>
#include <time.h>
#include <GL/glut.h>
```

```
using namespace std;
```

```
void delay(float ms){
    clock_t goal = ms + clock();
    while(goal>clock());
}
```

```
void init(){
    glClearColor(0.0,1.0,1.0,0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0,640,0,480);
}
```

```
void bound_it(int x, int y, float* fillColor, float* bc){
    float color[3];
    glReadPixels(x,y,1.0,1.0,GL_RGB,GL_FLOAT,color);
    if((color[0]!=bc[0] || color[1]!=bc[1] || color[2]!=bc[2])&&(
        color[0]!=fillColor[0] || color[1]!=fillColor[1] || color[2]!=fillColor[2])){
        glColor3f(fillColor[0],fillColor[1],fillColor[2]);
        glBegin(GL_POINTS);
            glVertex2i(x,y);
        glEnd();
        glFlush();
        bound_it(x+1,y,fillColor,bc);
        bound_it(x-2,y,fillColor,bc);
        bound_it(x,y+2,fillColor,bc);
        bound_it(x,y-2,fillColor,bc);
    }
}
```

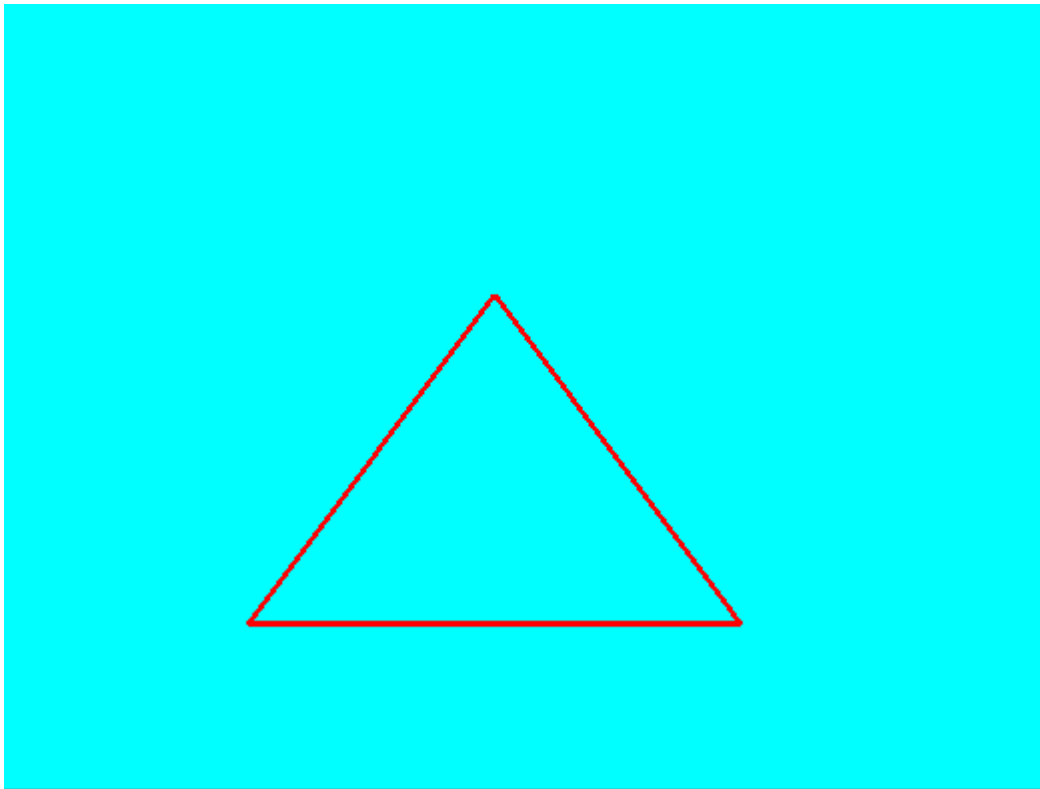
```
void mouse(int btn, int state, int x, int y){
    y = 480-y;
    if(btn==GLUT_LEFT_BUTTON)
    {
        if(state==GLUT_DOWN)
        {
            float bCol[] = {1,0,0};
            float color[] = {0,0,1};
            //glReadPixels(x,y,1.0,1.0,GL_RGB,GL_FLOAT,intCol);
            bound_it(x,y,color,bCol);
        }
    }
}
```

```
    }  
  }  
}
```

```
void world(){  
    glLineWidth(3);  
    glPointSize(2);  
    glClear(GL_COLOR_BUFFER_BIT);  
    glColor3f(1,0,0);  
    glBegin(GL_LINE_LOOP);  
        glVertex2i(150,100);  
        glVertex2i(300,300);  
        glVertex2i(450,100);  
    glEnd();  
    glFlush();  
}
```

```
int main(int argc, char** argv){  
    glutInit(&argc, argv);  
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);  
    glutInitWindowSize(640,480);  
    glutInitWindowPosition(200,200);  
    glutCreateWindow("Boundary Fill Algorithm Implementation");  
    glutDisplayFunc(world);  
    glutMouseFunc(mouse);  
    init();  
    glutMainLoop();  
    return 0;  
}
```

output:



boundary fill circle

```
#include <windows.h>
#include <bits/stdc++.h>
#include <math.h>
#include <gl/glut.h>

struct Point {
    GLint x;
    GLint y;
};

struct Color {
    GLfloat r;
    GLfloat g;
    GLfloat b;
};

void init() {
    glClearColor(0.0, 1.0, 1.0, 0.0);
    glColor3f(1.0, 0.0, 0.0);
    glPointSize(3.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 800, 0, 600);
}
```

```

Color getPixelColor(GLint x, GLint y) {
    Color color;
    glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color);
    return color;
}

void setPixelColor(GLint x, GLint y, Color color) {
    glColor3f(color.g, color.r, color.b);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}

void BoundaryFill(int x, int y, Color fillColor, Color boundaryColor) {
    Color currentColor = getPixelColor(x, y);
    if(currentColor.r != boundaryColor.r && currentColor.g != boundaryColor.g &&
    currentColor.b != boundaryColor.b) {
        setPixelColor(x, y, fillColor);
        BoundaryFill(x+1, y, fillColor, boundaryColor);
        BoundaryFill(x-1, y, fillColor, boundaryColor);
        BoundaryFill(x, y+1, fillColor, boundaryColor);
        BoundaryFill(x, y-1, fillColor, boundaryColor);
    }
}

void onMouseClick(int button, int state, int x, int y)
{
    Color fillColor = {1.0f, 0.0f, 0.0f};           // red color will be filled
    Color boundaryColor = {0.0f, 0.0f, 0.0f}; // black- boundary

    Point p = {321, 241}; // a point inside the circle

    BoundaryFill(p.x, p.y, fillColor, boundaryColor);
}

void draw_circle(Point pC, GLfloat radius) {
    GLfloat step = 1/radius;
    GLfloat x, y;

    for(GLfloat theta = 0; theta <= 360; theta += step) {
        x = pC.x + (radius * cos(theta));
        y = pC.y + (radius * sin(theta));
        glVertex2i(x, y);
    }
}

```

```

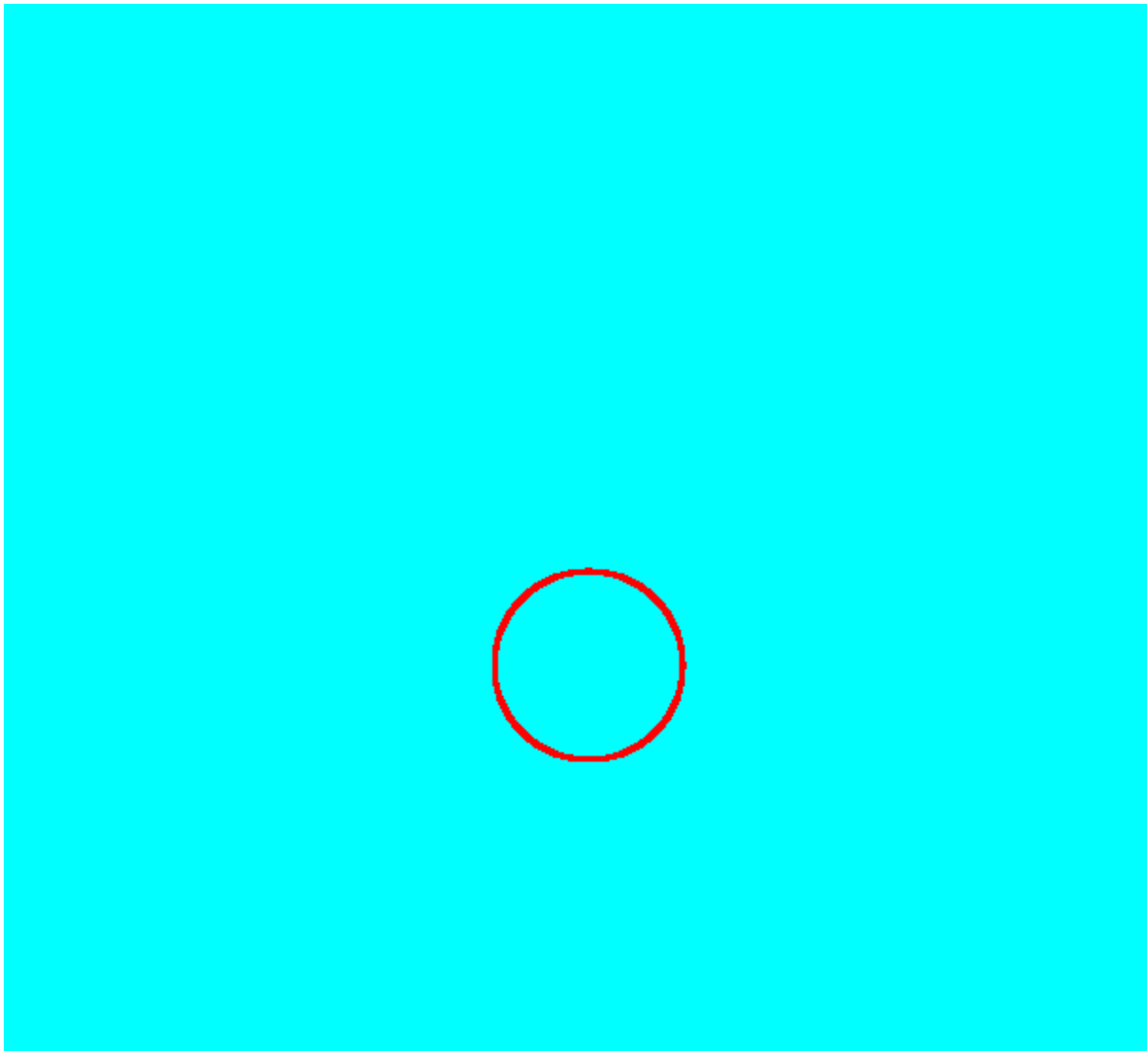
void display(void) {
    Point pt = {320, 240};
    GLfloat radius = 20;

    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POINTS);
        draw_circle(pt, 50);
    glEnd();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutInitWindowPosition(200, 200);
    glutCreateWindow("Boundary Fill Circle");
    init();
    glutDisplayFunc(display);
    glutMouseFunc(onMouseClicked);
    glutMainLoop();
    return 0;
}

```

Output :



Polygon clipping algorithm:

```
#include <windows.h>
```

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

```
struct Point{  
    float x,y;  
} w[4],oVer[4];  
int Nout;
```

```
void drawPoly(Point p[],int n){  
    glBegin(GL_POLYGON);  
    for(int i=0;i<n;i++)  
        glVertex2f(p[i].x,p[i].y);  
    glEnd();  
}
```

```

bool insideVer(Point p){
    if((p.x>=w[0].x)&&(p.x<=w[2].x))
        if((p.y>=w[0].y)&&(p.y<=w[2].y))
            return true;
    return false;
}

```

```

void addVer(Point p){
    oVer[Nout]=p;
    Nout=Nout+1;
}

```

```

Point getInterSect(Point s,Point p,int edge){
    Point in;
    float m;
    if(w[edge].x==w[(edge+1)%4].x){ //Vertical Line
        m=(p.y-s.y)/(p.x-s.x);
        in.x=w[edge].x;
        in.y=in.x*m+s.y;
    }
    else{//Horizontal Line
        m=(p.y-s.y)/(p.x-s.x);
        in.y=w[edge].y;
        in.x=(in.y-s.y)/m;
    }
    return in;
}

```

```

void clipAndDraw(Point inVer[],int Nin){
    Point s,p,interSec;
    for(int i=0;i<4;i++)
    {
        Nout=0;
        s=inVer[Nin-1];
        for(int j=0;j<Nin;j++)
        {
            p=inVer[j];
            if(insideVer(p)==true){
                if(insideVer(s)==true){
                    addVer(p);
                }
            }
            else{
                interSec=getInterSect(s,p,i);
                addVer(interSec);
                addVer(p);
            }
        }
    }
    else{

```

```

        if(insideVer(s)==true){
            interSec=getInterSect(s,p,i);
            addVer(interSec);
        }
    }
    s=p;
}
inVer=oVer;
Nin=Nout;
}
drawPoly(oVer,4);
}

```

```

void init(){
    glClearColor(0.0f,0.0f,0.0f,0.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0,100.0,0.0,100.0,0.0,100.0);
    glClear(GL_COLOR_BUFFER_BIT);
    w[0].x =20,w[0].y=10;
    w[1].x =20,w[1].y=80;
    w[2].x =80,w[2].y=80;
    w[3].x =80,w[3].y=10;
}

```

```

void display(void){
    Point inVer[4];
    init();
    // As Window for Clipping
    glColor3f(0.0f,1.0f,1.0f);
    drawPoly(w,4);
    // As Rect
    glColor3f(0.5f,0.0f,1.0f);
    inVer[0].x =10,inVer[0].y=40;
    inVer[1].x =10,inVer[1].y=60;
    inVer[2].x =60,inVer[2].y=60;
    inVer[3].x =60,inVer[3].y=40;
    drawPoly(inVer,4);
    // As Rect
    glColor3f(1.0f,0.0f,0.0f);
    clipAndDraw(inVer,4);
    // Print
    glFlush();
}

```

```

int main(int argc,char *argv[]){
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(400,400);
}

```

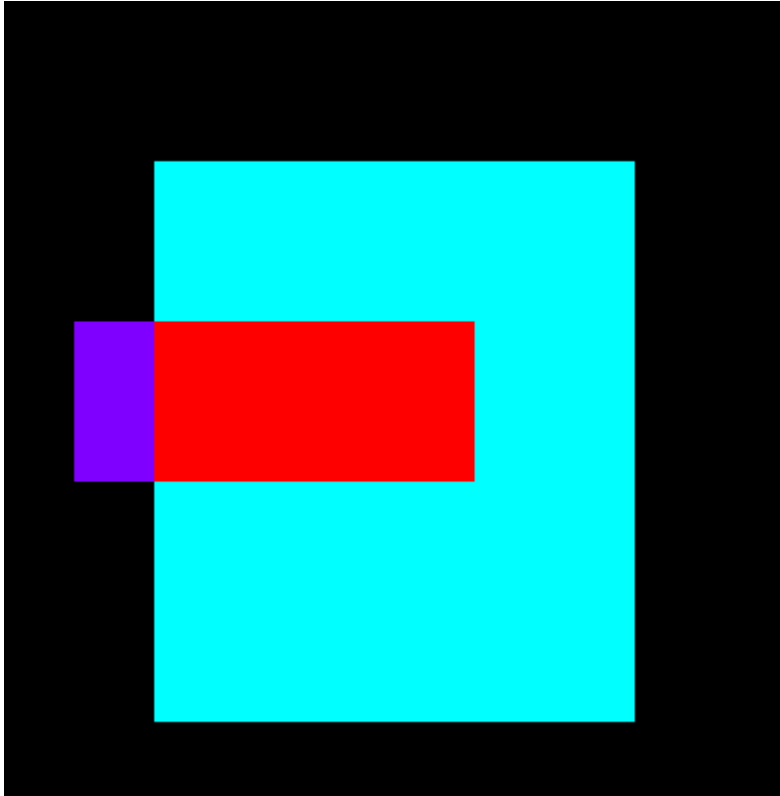


```

glutInitWindowPosition(100,100);
glutCreateWindow("Polygon Clipping!");
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```

Output:



bresenhams circle drawing :

```

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

// Center of the circle = (320, 240)
int xc = 320, yc = 240;

// Plot eight points using circle's symmetrical property
void plot_point(int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(xc+x, yc+y);
    glVertex2i(xc+x, yc-y);
    glVertex2i(xc+y, yc+x);
    glVertex2i(xc+y, yc-x);
    glVertex2i(xc-x, yc-y);
    glVertex2i(xc-x, yc+y);
    glVertex2i(xc-y, yc-x);
    glVertex2i(xc-y, yc+y);
}

```

```

    glVertex2i(xc-x, yc+y);
    glVertex2i(xc-y, yc+x);
    glEnd();
}

// Function to draw a circle using bresenham's
// circle drawing algorithm
void bresenham_circle(int r)
{
    int x=0,y=r;
    float pk=(5.0/4.0)-r;

    /* Plot the points */
    /* Plot the first point */
    plot_point(x,y);
// int k;
    /* Find all vertices till x=y */
    while(x < y)
    {
        x = x + 1;
        if(pk < 0)
            pk = pk + 2*x+1;
        else
        {
            y = y - 1;
            pk = pk + 2*(x - y) + 1;
        }
        plot_point(x,y);
    }
    glFlush();
}

// Function to draw two concentric circles
void concentric_circles(void)
{
    //Clears buffers to preset values
    glClear(GL_COLOR_BUFFER_BIT);
    int radius = 100;
    bresenham_circle(radius);
}

void Init()
{
    glClearColor(0.0,1.0,1.0,0.0); //clear values for RGBA
    glColor3f(1.0f,0.0f,0.0f); //set current color - RGB
    glPointSize(3.0); //diameters of rasterized points
    glMatrixMode(GL_PROJECTION); // specifies current matrix

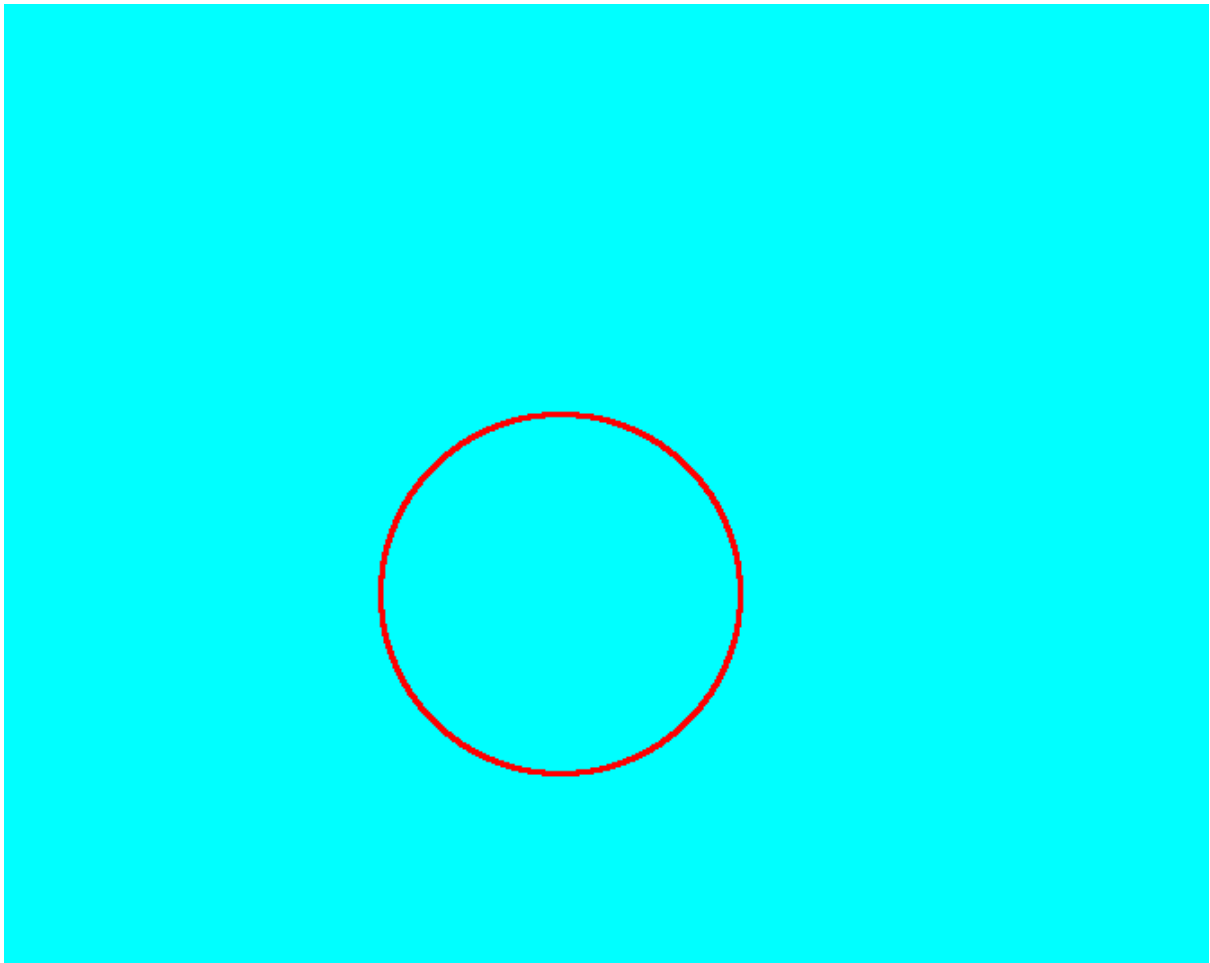
```

```

    glLoadIdentity();
    gluOrtho2D(0.0,800.0,0.0,600.0);
}

int main(int argc, char **argv)
{
    /* Initialise GLUT library */
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); //Set the initial display mode
    glutInitWindowPosition(0,0);
    glutInitWindowSize(800,600);
    glutCreateWindow("Bresenham Circle Drawing");
    /* Initialize drawing colors */
    Init();
    /* Call the displaying function */
    glutDisplayFunc(concentric_circles);
    /* Keep displaying untill the program is closed */
    glutMainLoop();
}

```



C curve:  
#include<windows.h>

```

#include<GL/glut.h>
#include<bits/stdc++.h>
using namespace std;
float x, y, len, alpha;
int n;

void line (float x1, float y1, float x2, float y2)
{
    glVertex2f(x1,y1);
    glVertex2f(x2,y2);
}

void c_curve (float x, float y, float len, float alpha, int n)
{
    if(n > 0){
        len = len / sqrt(2.0);
        c_curve(x, y, len, alpha+45, n-1);

        x += len*cos(alpha+45);
        y += len*sin(alpha+45);
        c_curve(x, y, len, alpha-45, n-1);
    }
    else{
        line(x, y, x+len*cos(alpha), y+len*sin(alpha));
    }
}

void myDisplay(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f (1.0, 0.0, 0.0);
    glPointSize(1);
    glBegin(GL_LINES);
    c_curve(x, y, len, alpha, n);
    glEnd();
    glFlush ();
}

void init (void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(0.7,0.7,0.7,0.7);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-200,500,-200,500);
}

int main(int argc, char** argv)

```

```

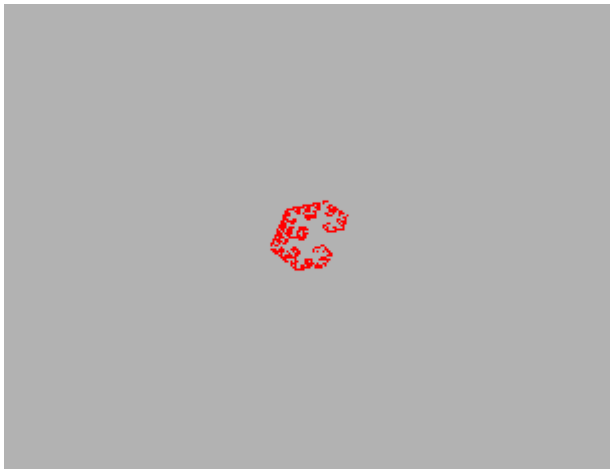
{
    cout<<"Co-ordinate of C(x,y): ";
    cin>>x>>y;
    cout<<"\nLength: ";
    cin>>len;
    cout<<"\nAngle: ";
    cin>>alpha;
    cout<<"\nValue of n: ";
    cin>>n;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(850, 600);
    glutInitWindowPosition(100, 50);
    glutCreateWindow("C CURVE");
    init();
    glutDisplayFunc(myDisplay);
    glutMainLoop();

    return 0;
}

```

Output:



cubic bezier curve:

```

#include <windows.h>
#include <GL/glut.h>
#include <math.h>
#include <stdio.h>
#define CTRL_COUNT 100
int ctrlPointsCount;
int ctrlPointsX[CTRL_COUNT], ctrlPointsY[CTRL_COUNT];
int X1[3]={20,25,20}, Y1[3]={5,24,38}; //first point(x1[0],y1[0]) second(x1[1],y1[1])
third(x1[2],y1[2])

```

```

void myInit()
{
glClearColor(0.0,0.0,0.0,0.0);
glColor3f(1.0,0.0,0.0);
glPointSize(8.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,128.0,0.0,96.0);
}
//p(t)=(1-t)^3*p0+3t(1-t)^2*p1+3t^2(1-t)p2+t^3p3
float getNextBezierPointX(float t)
{
float x=0.0;

for(int i=0; i<ctrlPointsCount; i++)
{
int c;
if(i==0 || i==ctrlPointsCount-1)
    c = 1;
else
{
    c = ctrlPointsCount-1;
}
x += c * pow(t, i) * pow(1-t, ctrlPointsCount-1-i) * ctrlPointsX[i];
}

return x;
}

float getNextBezierPointY(float t)
{
float y=0.0;

for(int i=0; i<ctrlPointsCount; i++)
{
int c;
if(i==0 || i==ctrlPointsCount-1)
    c = 1;
else
{
    c = ctrlPointsCount-1;
}
y += c * pow(t, i) * pow(1-t, ctrlPointsCount-1-i) * ctrlPointsY[i];
}
return y;
}

```

```

void drawline()
{
// draw control points using red color
for(int i=0; i < 3; i++)
{
glBegin(GL_POINTS);
glVertex2i(ctrlPointsX[i], ctrlPointsY[i]);
glEnd();
glFlush();
}
// draw bezier curve using control points by calculating next points using cubic bezier curve
formula
float oldX=ctrlPointsX[0], oldY=ctrlPointsY[0];
for(double t = 0.0;t <= 1.0; t += 0.01) {

float x = getNextBezierPointX(t);
float y = getNextBezierPointY(t);
//glColor3f(1.0,t,1.0);
glColor3f(1.0,1.0,1.0);
glBegin(GL_LINES);
glVertex2f(oldX, oldY);
glVertex2f(x, y);
glEnd();
glFlush();

oldX = x;
oldY = y;
}
}

void myDisplay()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0,0.0);
ctrlPointsCount=3;
for(int i=0;i<3;i++)
{
ctrlPointsX[i] = X1[i];
ctrlPointsY[i] = Y1[i];
}
drawline();

glFlush();
}

int main(int argc, char *argv[])
{

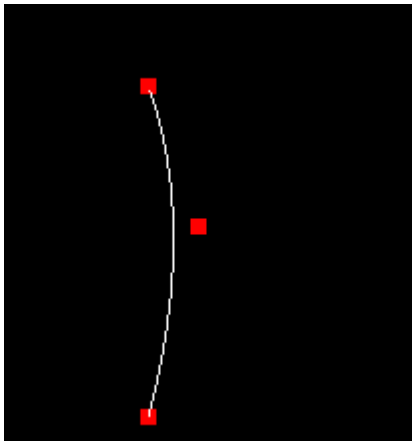
```

```

glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(640,480);
glutInitWindowPosition(100,150);
glutCreateWindow("Cubic Bezier Curve");
glutDisplayFunc(myDisplay);
myInit();
glutMainLoop();
return 0;
}

```

output:



Flood fill circle algorithm:

```

#include <windows.h>
#include <bits/stdc++.h>
#include <math.h>
#include <gl/glut.h>

```

```

struct Point {
    GLint x;
    GLint y;
};

```

```

struct Color {
    GLfloat r;
    GLfloat g;
    GLfloat b;
};

```

```

void init() {
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.0, 0.0, 0.0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
}

```



```

        glLoadIdentity();
        gluOrtho2D(0, 640, 0, 480);
    }

    Color getPixelColor(GLint x, GLint y) {
        Color color;
        glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color);
        return color;
    }

    void setPixelColor(GLint x, GLint y, Color color) {
        glColor3f(color.r, color.g, color.b);
        glBegin(GL_POINTS);
        glVertex2i(x, y);
        glEnd();
        glFlush();
    }

    void floodFill(GLint x, GLint y, Color oldColor, Color newColor) {
        Color color;
        color = getPixelColor(x, y);

        if(color.r == oldColor.r && color.g == oldColor.g && color.b == oldColor.b)
        {
            setPixelColor(x, y, newColor);
            floodFill(x+1, y, oldColor, newColor);
            floodFill(x, y+1, oldColor, newColor);
            floodFill(x-1, y, oldColor, newColor);
            floodFill(x, y-1, oldColor, newColor);
        }
        return;
    }

    void onMouseClick(int button, int state, int x, int y)
    {
        Color newColor = {1.0f, 0.0f, 0.0f};
        Color oldColor = {1.0f, 1.0f, 1.0f};

        floodFill(320, 240, oldColor, newColor);
    }

    void draw_circle(Point pC, GLfloat radius) {
        GLfloat step = 1/radius;
        GLfloat x, y;

        for(GLfloat theta = 0; theta <= 360; theta += step) {
            x = pC.x + (radius * cos(theta));
            y = pC.y + (radius * sin(theta));
        }
    }

```

```

        glVertex2i(x, y);
    }
}

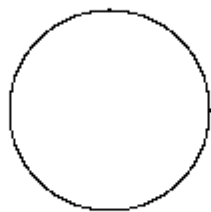
void display(void) {
    Point pt = {320, 240};
    GLfloat radius = 50;

    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POINTS);
        draw_circle(pt, radius);
    glEnd();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(200, 200);
    glutCreateWindow("Flood Fill Circle");
    init();
    glutDisplayFunc(display);
    glutMouseFunc(onMouseClicked);
    glutMainLoop();
    return 0;
}

```

Output:



Flood fill square:

```

#include <windows.h>
#include <bits/stdc++.h>

```

```

#include <math.h>
#include <gl/glut.h>

struct Point {
    GLint x;
    GLint y;
};

struct Color {
    GLfloat r;
    GLfloat g;
    GLfloat b;
};

void draw_dda(Point p1, Point p2) {
    GLfloat dx = p2.x - p1.x;
    GLfloat dy = p2.y - p1.y;

    GLfloat x1 = p1.x;
    GLfloat y1 = p1.y;

    GLfloat step = 0;

    if(abs(dx) > abs(dy)) {
        step = abs(dx);
    } else {
        step = abs(dy);
    }

    GLfloat xInc = dx/step;
    GLfloat yInc = dy/step;

    for(float i = 1; i <= step; i++) {
        glVertex2i(x1, y1);
        x1 += xInc;
        y1 += yInc;
    }
}

void init() {
    glClearColor(0.0, 1.0, 1.0, 0.0);
    glColor3f(0.0, 0.0, 0.0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 640, 0, 480);
}

```

```

Color getPixelColor(GLint x, GLint y) {
    Color color;
    glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color);
    return color;
}

void setPixelColor(GLint x, GLint y, Color color) {
    glColor3f(color.r, color.g, color.b);
    glBegin(GL_POINTS);
        glVertex2i(x, y);
    glEnd();
    glFlush();
}

void floodFill(GLint x, GLint y, Color oldColor, Color newColor) {
    Color color;
    color = getPixelColor(x, y);

    if(color.r == oldColor.r && color.g == oldColor.g && color.b == oldColor.b)
    {
        setPixelColor(x, y, newColor);
        floodFill(x+1, y, oldColor, newColor);
        floodFill(x, y+1, oldColor, newColor);
        floodFill(x-1, y, oldColor, newColor);
        floodFill(x, y-1, oldColor, newColor);
    }
    return;
}

void onMouseClick(int button, int state, int x, int y)
{
    Color newColor = {1.0f, 0.0f, 0.0f};
    Color oldColor = {0.0f, 1.0f, 1.0f};

    floodFill(101, 199, oldColor, newColor);
}

void display(void) {
    Point p1 = {100, 100}, // bottom-right
           p2 = {200, 100}, // bottom-left
           p3 = {200, 200}, // top-right
           p4 = {100, 200}; // top-left

    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POINTS);
        draw_dda(p1, p2);
        draw_dda(p2, p3);
        draw_dda(p3, p4);

```

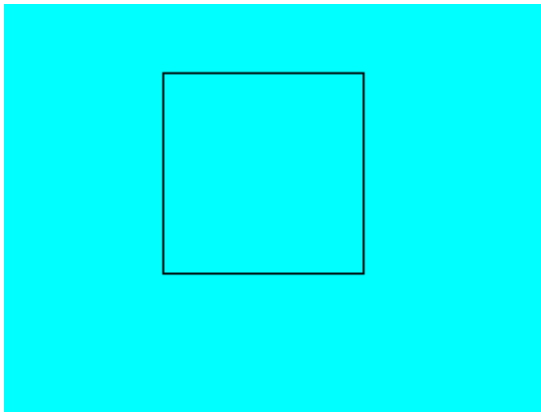
```

        draw_dds(p4, p1);
    glEnd();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(200, 200);
    glutCreateWindow("Flood Fill Square");
    init();
    glutDisplayFunc(display);
    glutMouseFunc(onMouseClicked);
    glutMainLoop();
    return 0;
}

```

Output:



Implement Cohen Sutherland polygon clipping method to clip the polygon with respect the:

```

#include<windows.h>
#include <stdio.h>
#include<iostream>
#include<GL/glut.h>
#include<math.h>
#include<bits/stdc++.h>

using namespace std ;
int result;
int xmin, ymin, xmax, ymax, pt[30][2], w[30][2], n=0, flg=0;

int leftClip(int limit, int xm)
{
    int i, j = 0, x1, y1, x2, y2;
    float m;

```

```

for (i = 0; i < limit; i++)
{
    x1 = pt[i][0];
    y1 = pt[i][1];
    x2 = pt[(i + 1) % limit][0];
    y2 = pt[(i + 1) % limit][1];
    if (x2 - x1)
        m = (y2 - y1) * 1.0 / (x2 - x1);

    if (x1 < xm && x2 < xm)
        continue;
    if (x1 > xm && x2 > xm)
    {
        w[j][0] = x2;
        w[j++][1] = y2;
        continue;
    }
    if (x1 > xm && x2 < xm)
    {
        w[j][0] = xm;
        w[j++][1] = y1 + m * (xm - x1);
        continue;
    }
    if (x1 < xm && x2 > xm)
    {
        w[j][0] = xm;
        w[j++][1] = y1 + m * (xm - x1);
        w[j][0] = x2;
        w[j++][1] = y2;
    }
}

for (i = 0; i < j; i++)
{
    pt[i][0] = w[i][0];
    pt[i][1] = w[i][1];
    w[i][0] = w[i][1] = 0;
}

if (j < limit)
    for (; i < limit; i++)
        pt[i][0] = pt[i][1] = 0;

return j;
}

int topClip(int limit, int ym)
{

```

```

int i, j = 0, x1, y1, x2, y2;
float m;
for (i = 0; i < limit; i++)
{
    x1 = pt[i][0];
    y1 = pt[i][1];
    x2 = pt[(i + 1) % limit][0];
    y2 = pt[(i + 1) % limit][1];
    if (x2 - x1)
        m = (y2 - y1) * 1.0 / (x2 - x1);

    if (y1 < ym && y2 < ym)
        continue;
    if (y1 > ym && y2 > ym)
    {
        w[j][0] = x2;
        w[j++][1] = y2;
        continue;
    }
    if (y1 > ym && y2 < ym)
    {
        w[j][0] = x1 + (ym - y1) / m;
        w[j++][1] = ym;
        continue;
    }
    if (y1 < ym && y2 > ym)
    {
        w[j][0] = x1 + (ym - y1) / m;
        w[j++][1] = ym;
        w[j][0] = x2;
        w[j++][1] = y2;
    }
}

for (i = 0; i < j; i++)
{
    pt[i][0] = w[i][0];
    pt[i][1] = w[i][1];
    w[i][0] = w[i][1] = 0;
}

if (j < limit)
    for (; i < limit; i++)
        pt[i][0] = pt[i][1] = 0;

return j;
}

```

```

int rightClip(int limit, int xm)
{
    int i, j = 0, x1, y1, x2, y2;
    float m;
    for (i = 0; i < limit; i++)
    {
        x1 = pt[i][0];
        y1 = pt[i][1];
        x2 = pt[(i + 1) % limit][0];
        y2 = pt[(i + 1) % limit][1];
        if (x2 - x1)
            m = (y2 - y1) * 1.0 / (x2 - x1);

        if (x1 > xm && x2 > xm)
            continue;
        if (x1 < xm && x2 < xm)
        {
            w[j][0] = x2;
            w[j++][1] = y2;
            continue;
        }
        if (x1 < xm && x2 > xm)
        {
            w[j][0] = xm;
            w[j++][1] = y1 + m * (xm - x1);
            continue;
        }
        if (x1 > xm && x2 < xm)
        {
            w[j][0] = xm;
            w[j++][1] = y1 + m * (xm - x1);
            w[j][0] = x2;
            w[j++][1] = y2;
        }
    }

    for (i = 0; i < j; i++)
    {
        pt[i][0] = w[i][0];
        pt[i][1] = w[i][1];
        w[i][0] = w[i][1] = 0;
    }

    if (j < limit)
        for (; i < limit; i++)
            pt[i][0] = pt[i][1] = 0;

    return j;
}

```



```
}
```

```
int bottomClip(int limit, int ym)
```

```
{
```

```
    int i, j = 0, x1, y1, x2, y2;
```

```
    float m;
```

```
    for (i = 0; i < limit; i++)
```

```
    {
```

```
        x1 = pt[i][0];
```

```
        y1 = pt[i][1];
```

```
        x2 = pt[(i + 1) % limit][0];
```

```
        y2 = pt[(i + 1) % limit][1];
```

```
        if (x2 - x1)
```

```
            m = (y2 - y1) * 1.0 / (x2 - x1);
```

```
        if (y1 > ym && y2 > ym)
```

```
            continue;
```

```
        if (y1 < ym && y2 < ym)
```

```
        {
```

```
            w[j][0] = x2;
```

```
            w[j++][1] = y2;
```

```
            continue;
```

```
        }
```

```
        if (y1 < ym && y2 > ym)
```

```
        {
```

```
            w[j][0] = x1 + (ym - y1) / m;
```

```
            w[j++][1] = ym;
```

```
            continue;
```

```
        }
```

```
        if (y1 > ym && y2 < ym)
```

```
        {
```

```
            w[j][0] = x1 + (ym - y1) / m;
```

```
            w[j++][1] = ym;
```

```
            w[j][0] = x2;
```

```
            w[j++][1] = y2;
```

```
        }
```

```
    }
```

```
    for (i = 0; i < j; i++)
```

```
    {
```

```
        pt[i][0] = w[i][0];
```

```
        pt[i][1] = w[i][1];
```

```
        w[i][0] = w[i][1] = 0;
```

```
    }
```

```
    if (j < limit)
```

```
        for (; i < limit; i++)
```

```
            pt[i][0] = pt[i][1] = 0;
```

```

    return j;
}

void display(void)
{
}

void init()
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear (GL_COLOR_BUFFER_BIT);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,0.0,0.0);
    glPointSize(2.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0,700,0,700) ;
}

void menu(int c)
{
    if(c==1)
    {
        result = leftClip(n, xmin);
        result = topClip(result, ymin);
        result = rightClip(result, xmax);
        result = bottomClip(result, ymax);
    }
    if(c==2)
    {
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0,0.0,0.0);
        glBegin(GL_POINTS);
        glVertex2i(0,0);
        glEnd();
        glFlush();
        glColor3f(1.0,1.0,0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2i(xmin,ymin);
        glVertex2i(xmax,ymin);
        glVertex2i(xmax,ymax);
        glVertex2i(xmin,ymax);
        glEnd();
        glFlush();
        for (int i = 0; i < result; i++)
        {

```

```

        glColor3f(0.0,0.0,1.0);
        glBegin(GL_LINE_STRIP);
        glVertex2i(pt[i][0],pt[i][1]);
        glVertex2i(pt[(i+1)%result][0],pt[(i+1)%result][1]);
        glEnd();
        glFlush();
    }
}

}

void mouse(int button, int state, int cx, int cy )
{
    if(state==GLUT_DOWN)
    {
        if(button==GLUT_LEFT_BUTTON)
        {
            pt[n][0] = cx ;
            pt[n][1] = 700-cy ;
            n++;
            if(n>1)
            {
                glColor3f(1.0,0.0,0.0);
                glBegin(GL_LINE_STRIP);
                glVertex2i(pt[n-2][0],pt[n-2][1]);
                glVertex2i(pt[n-1][0],pt[n-1][1]);
                glEnd();
                glFlush();
            }
        }
        if(button==GLUT_RIGHT_BUTTON)
        {
            if(flag==0)
            {
                cout<<"RIGHT CLICK 1 - DONE"<<endl;
                xmin=cx;
                ymin=700-cy;
                flag++;
            }
            else
            {
                cout<<"RIGHT CLICK 2 - DONE"<<endl;
                xmax=cx;
                ymax=700-cy;
                glColor3f(1.0,1.0,0.0);
                glBegin(GL_LINE_LOOP);
                glVertex2i(xmin,ymin);
                glVertex2i(xmax,ymin);
            }
        }
    }
}

```

```

        glVertex2i(xmax,ymax);
        glVertex2i(xmin,ymax);
        glEnd();
        glFlush();
    }
}
}

int main(int argc, char *argv[])
{
    glutInit(&argc,argv);
    glutInitWindowSize(700,700);
    glutInitWindowPosition(500,50);
    glutCreateWindow(" COHEN SUTHERLAND POLYGON CLIPPING ");
    cout<<"PLEASE FOLLOW THESE STEPS:"<<endl;
    cout<<"[1] MAKE POLYGON by USING LEFT BUTTON CLICK"<<endl;
    cout<<"[2] SELECT WINDOW COORDINATES by USING RIGHT BUTTON CLICK:
where"<<endl;
    cout<<"FIRST RIGHT CLICK - SELECT Xmin and Ymin"<<endl;
    cout<<"SECOND RIGHT CLICK - SELECT Xmax and Ymax"<<endl<<endl;
    cout<<"[3]In MENU, please first execute and then clip"<<endl<<endl;
    init();

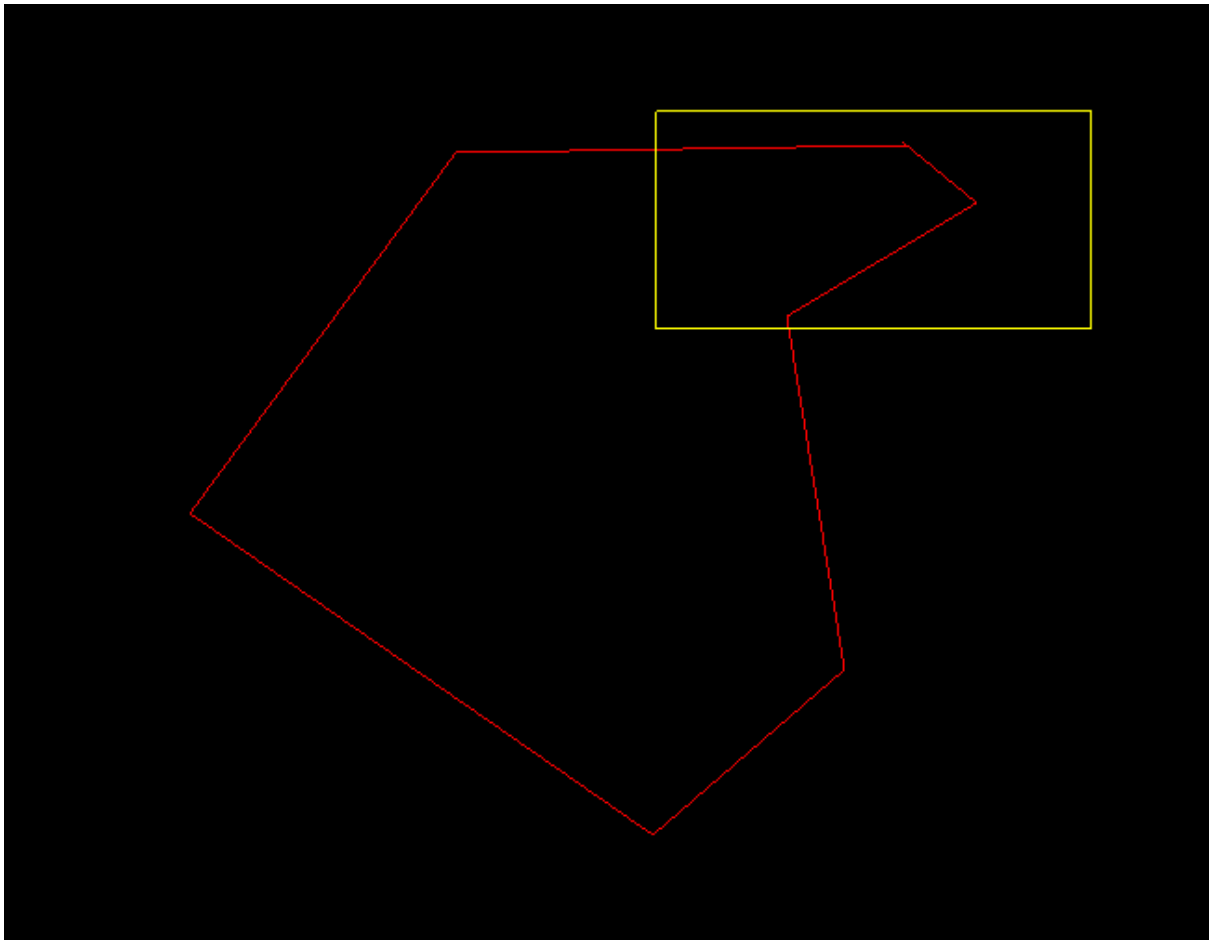
    glutMouseFunc(mouse);
    glutDisplayFunc(display);
    glutCreateMenu(menu);
    glutAddMenuEntry("EXECUTE",1);
    glutAddMenuEntry("SHOW CLIPPED",2);
    glutAttachMenu(GLUT_MIDDLE_BUTTON);

    glutMainLoop();
    return 0;

}

```

Output:



```
#include<windows.h>
#include<iostream>
#include<GL/glut.h>
#include<math.h>
#include<bits/stdc++.h>

using namespace std ;

int m[20][3], n = 0 ;

void setpixel(GLint x, GLint y)
{
    glColor3f(0.0,0.0,1.0);
    glBegin(GL_POINTS);
    glVertex2f(x,y);
    glEnd();
    glFlush();
}

void choice()
{
```

```

int i;
glPointSize(2.0);
for(i=-700; i<700; i++)
{
    setpixel(0,i);
    setpixel(i,0);
}
}

void setpcolor(double r1, double b1, double g1 )
{
    glColor3f(r1,b1,g1);
}

void conect(int x, int y, int px, int py)
{
    glPointSize(2);
    glBegin(GL_LINE_STRIP);
    glVertex2i(x,y);
    glVertex2i(px,py);
    glEnd();
    glFlush();
}

void translation(int tx,int ty)
{
    int tm[3][3] = {{1,0,tx},{0,1,ty},{0,0,1}},ne[3]= {} ;
    for(int i=0 ; i<n; i++)
    {
        ne [0] = tm[0][0]*m[i][0] + tm[0][1]*m[i][1] + tm[0][2]*m[i][2] ;
        ne [1] = tm[1][0]*m[i][0] + tm[1][1]*m[i][1] + tm[1][2]*m[i][2] ;
        ne [2] = tm[2][0]*m[i][0] + tm[2][1]*m[i][1] + tm[2][2]*m[i][2] ;
        m[i][0] = ne[0] ;
        m[i][1] = ne[1] ;
        m[i][2] = ne[2] ;
    }
    for(int i=0 ; i<n; i++)
    {
        int ni = (i+1)%n;
        setpcolor(1,1,0) ;
        conect(m[i][0],m[i][1],m[ni][0],m[ni][1]) ;
    }

}

void rotation(double rot, int xm, int ym )
{

```

```

double pi = 3.14159265 ;
double rad = (pi/180.00) ;
rad *= rot ;
double rm[3][3] =
{{cos(rad),sin(rad),0},{-sin(rad),cos(rad),0},{-xm*cos(rad)+ym*sin(rad)+xm,-xm*sin(rad)-ym*cos(rad)+ym,1} } ;
int ne[3]= {} ;
for(int i=0 ; i<n; i++)
{
    ne [0] = rm[0][0]*m[i][0] + rm[0][1]*m[i][1] + rm[0][2]*m[i][2] ;
    ne [1] = rm[1][0]*m[i][0] + rm[1][1]*m[i][1] + rm[1][2]*m[i][2] ;
    ne [2] = rm[2][0]*m[i][0] + rm[2][1]*m[i][1] + rm[2][2]*m[i][2] ;
    m[i][0] = ne[0] ;
    m[i][1] = ne[1] ;
    m[i][2] = ne[2] ;
}
for(int i=0 ; i<n; i++)
{
    int ni = (i+1)%n;
    setpcolor(1,1,0);
    conect(m[i][0],m[i][1],m[ni][0],m[ni][1]) ;
}
}

```

```

void scale(int sx, int sy )
{
    int sm[3][3] = {{sx,0,0},{0,sy,0},{0,0,1} } ;
    int ne[3]= {} ;
    for(int i=0 ; i<n; i++)
    {
        ne [0] = sm[0][0]*m[i][0] + sm[0][1]*m[i][1] + sm[0][2]*m[i][2] ;
        ne [1] = sm[1][0]*m[i][0] + sm[1][1]*m[i][1] + sm[1][2]*m[i][2] ;
        ne [2] = sm[2][0]*m[i][0] + sm[2][1]*m[i][1] + sm[2][2]*m[i][2] ;
        m[i][0] = ne[0] ;
        m[i][1] = ne[1] ;
        m[i][2] = ne[2] ;
    }
    for(int i=0 ; i<n; i++)
    {
        int ni = (i+1)%n;
        setpcolor(1,1,0) ;
        conect(m[i][0],m[i][1],m[ni][0],m[ni][1]) ;
    }
}

```

```

void reflect(char c)
{

```

```

int sm[3][3] = {{1,0,0},{0,1,0},{0,0,1}} ;
if(c=='x'||c=='X')
{
    sm[1][1]=-1 ;
}
else
{
    sm[0][0] = - 1 ;
}
int ne[3]= {} ;
for(int i=0 ; i<n; i++)
{
    ne [0] = sm[0][0]*m[i][0] + sm[0][1]*m[i][1] + sm[0][2]*m[i][2] ;
    ne [1] = sm[1][0]*m[i][0] + sm[1][1]*m[i][1] + sm[1][2]*m[i][2] ;
    ne [2] = sm[2][0]*m[i][0] + sm[2][1]*m[i][1] + sm[2][2]*m[i][2] ;
    m[i][0] = ne[0] ;
    m[i][1] = ne[1] ;
    m[i][2] = ne[2] ;
}
for(int i=0 ; i<n; i++)
{
    int ni = (i+1)%n;
    setpcolor(1,1,0) ;
    conect(m[i][0],m[i][1],m[ni][0],m[ni][1]) ;
}
}

```

```

void init()
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear (GL_COLOR_BUFFER_BIT);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,0.0,0.0);
    glPointSize(2.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-350,350,-350,350) ;
}

```

```

void menu(int c)
{
    for(int i=0 ; i<n; i++)
    {
        int ni = (i+1)%n;
        setpcolor(1,1,1) ;
        conect(m[i][0],m[i][1],m[ni][0],m[ni][1]) ;
    }
}

```



```

}
glFlush() ;

if (c==1)
{
    for(int i=0; i<n; i++)
    {
        cout<<m[i][0]<<" "<<m[i][1]<<endl;
    }
}
else if(c==2)
{
    int tx = 0, ty = 0 ;
    cout <<"Enter x-translation factor : " ;
    cin >> tx ;
    cout <<"Enter y-translation factor : " ;
    cin >> ty ;
    translation(tx,ty) ;
}
else if(c==3)
{
    double rot ;
    int flg = 1,ym,xm ;
    cout <<"Enter the arbitrary point x : " ;
    cin >> xm ;
    cout <<"Enter the arbitrary point y : " ;
    cin >> ym ;
    cout <<"Enter 1 for clockwise else enter 0 for anti-clock wise : " ;
    cin >> flg ;
    cout <<"Enter by how much degree the object is to be rotated : " ;
    cin >> rot ;
    if(flg)
    {
        rot = -rot ;
    }
    rotation(rot,xm,ym) ;
}
else if(c==4)
{
    int sx = 1, sy = 1 ;
    cout <<"Enter the horizontal scaling factor : " ;
    cin >> sx ;
    cout <<"Enter the vertical scaling factor : " ;
    cin >> sy ;
    scale(sx,sy) ;
}
else if(c==5)
{

```

```

        char c ;
        cout <<"Enter the axis of reflection : (X | Y |)" ;
        cin >> c ;
        reflect(c) ;
    }

}

```

```

void mouse(int button, int state, int cx, int cy )
{
    cx -= 350 ;
    cy -= 350 ;
    cy = - cy ;
    if(state==GLUT_DOWN)
    {
        if(button==GLUT_LEFT_BUTTON)
        {
            m[n][0] = cx ;
            m[n][1] = cy ;
            m[n][2] = 1 ;
            n++;
            if(n>1)
            {
                glColor3f(1.0,0.0,0.0);
                glBegin(GL_LINE_STRIP);
                glVertex2i(m[n-2][0],m[n-2][1]);
                glVertex2i(m[n-1][0],m[n-1][1]);
                glEnd();
                glFlush();
            }
        }
    }
}

```

```

int main(int argc, char *argv[])
{
    glutInit(&argc,argv);
    glutInitWindowSize(700,700);
    glutInitWindowPosition(500,50);
    glutCreateWindow(" 2D TRANSFORMATION ");
    cout<<"PLEASE FOLLOW THESE STEPS:"<<endl;
    cout<<"MAKE POLYGON by USING LEFT BUTTON CLICK"<<endl;
    cout<<"FOR MENU, use the RIGHT button of the mouse"<<endl;
    init();
    glutDisplayFunc(choice);

    glutMouseFunc(mouse);
}

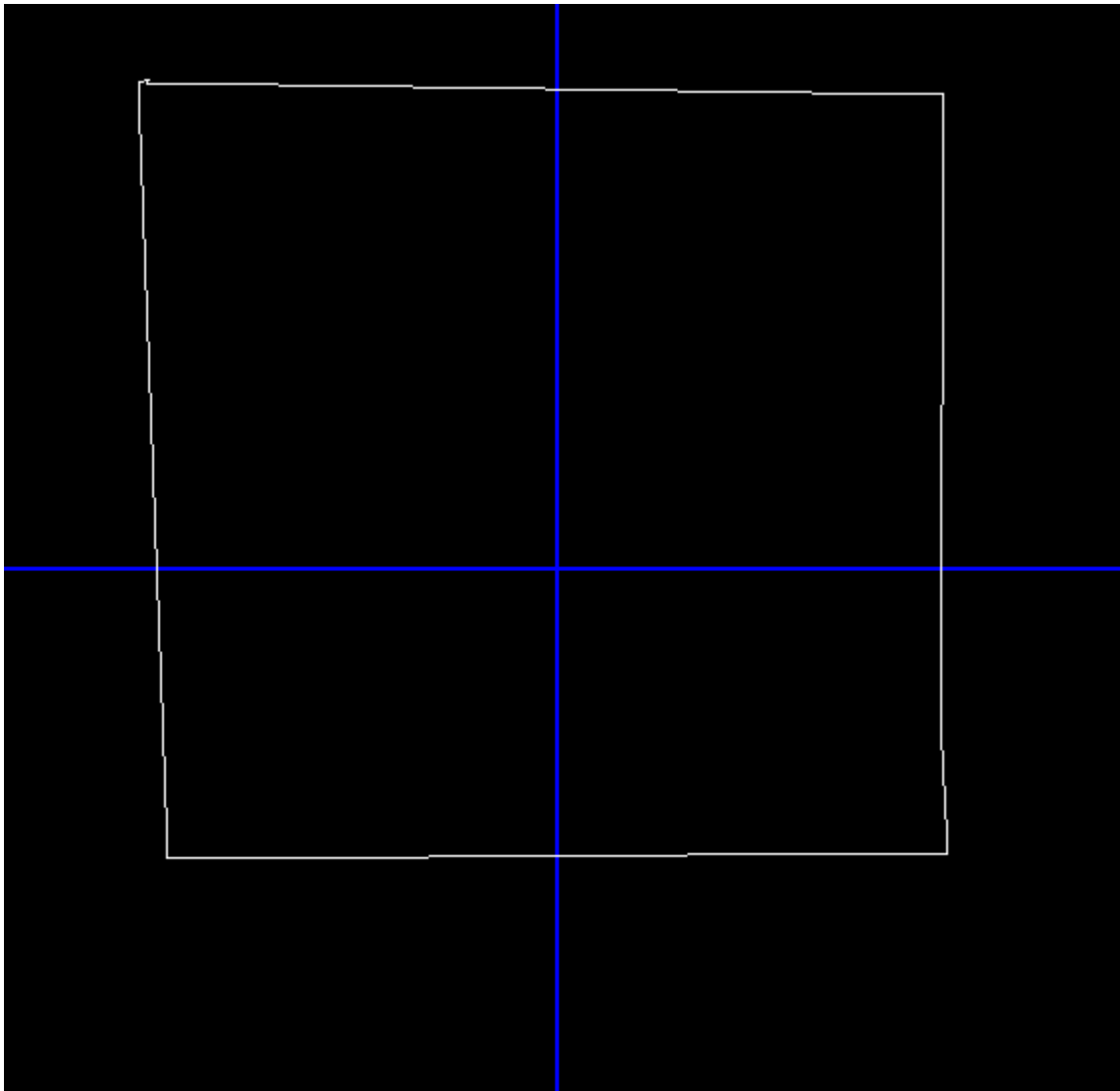
```

```
glutCreateMenu(menu);
glutAddMenuEntry("DISPLAY AXES OF POLYGON",1);
glutAddMenuEntry("TRANSLATION",2);
glutAddMenuEntry("ROTATION",3);
glutAddMenuEntry("SCALING",4);
glutAddMenuEntry("REFLECTION",5);

glutAttachMenu(GLUT_RIGHT_BUTTON);

glutMainLoop();
return 0;
}
```

A - display axes of polygon:

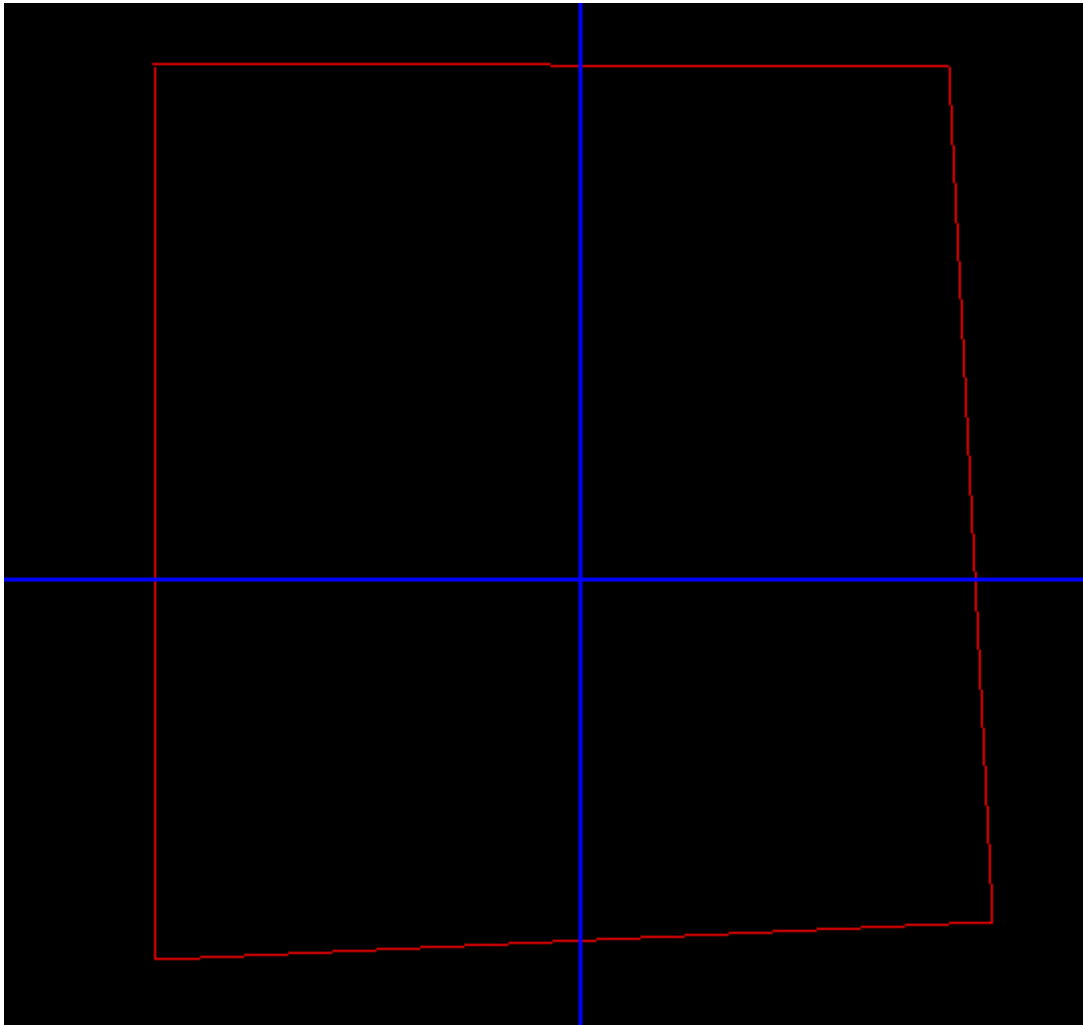


```

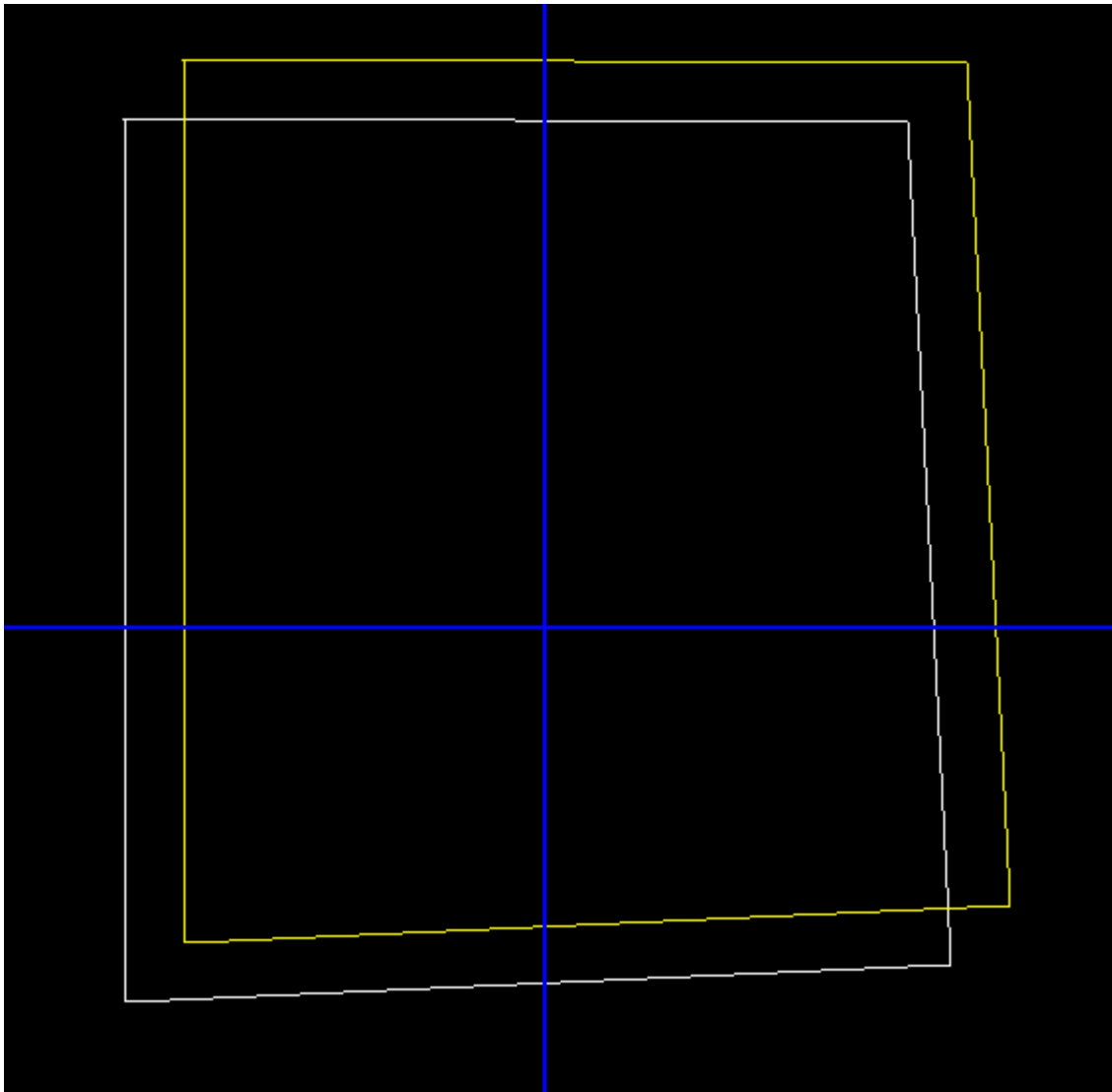
-208 246
196 240
194 -75
198 -144
-197 -147
-212 246
-206 248

```

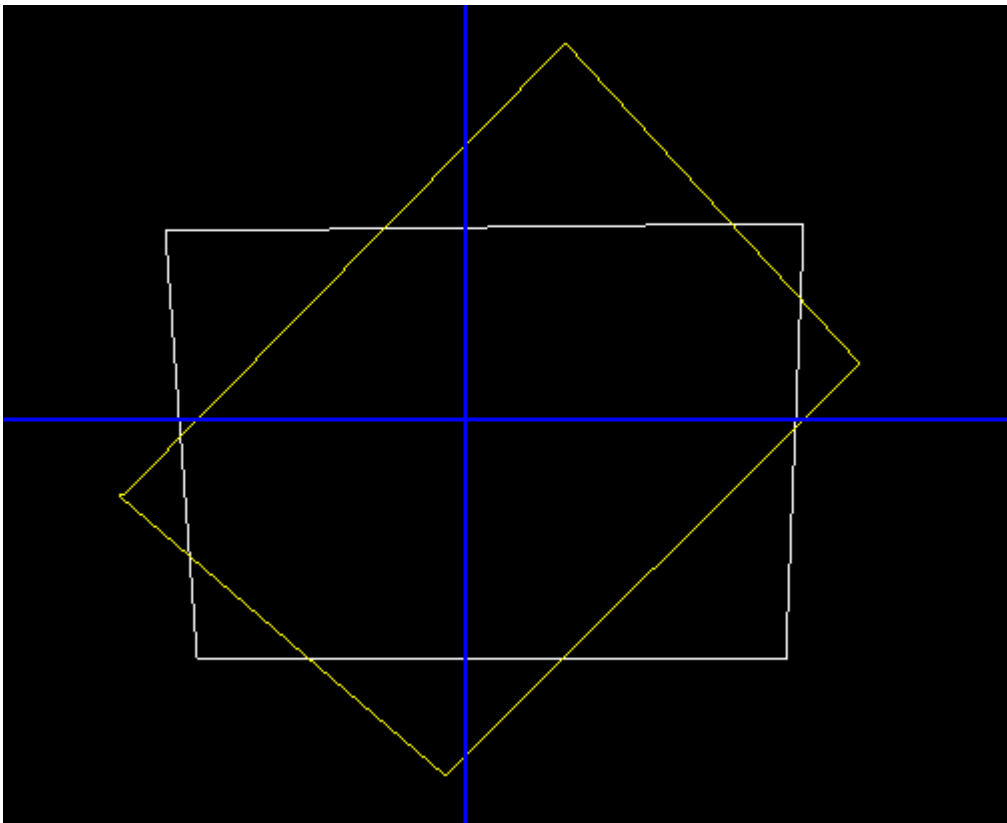
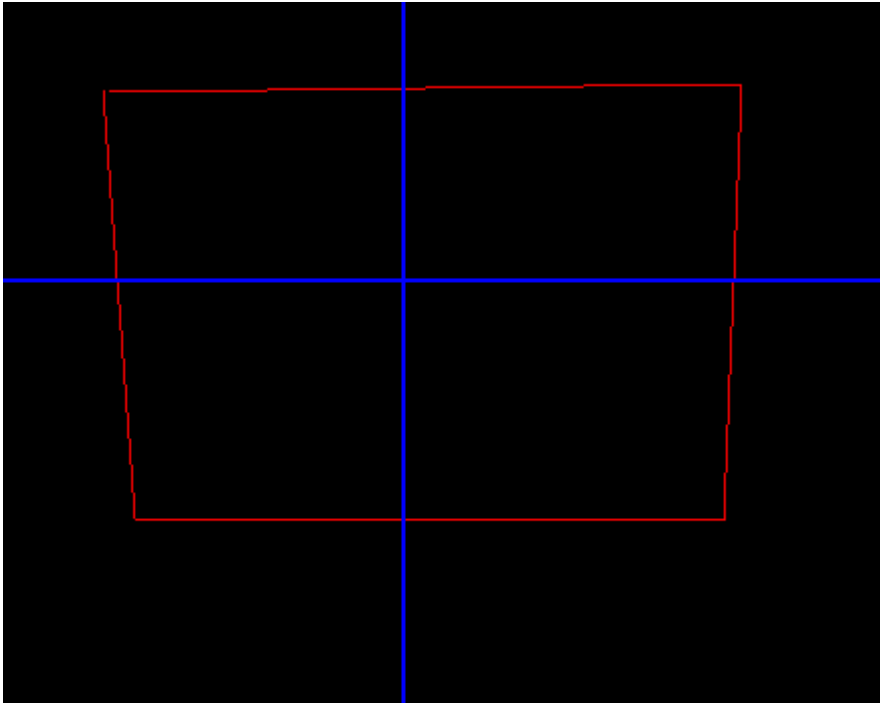
This are the coordinates of the polygon



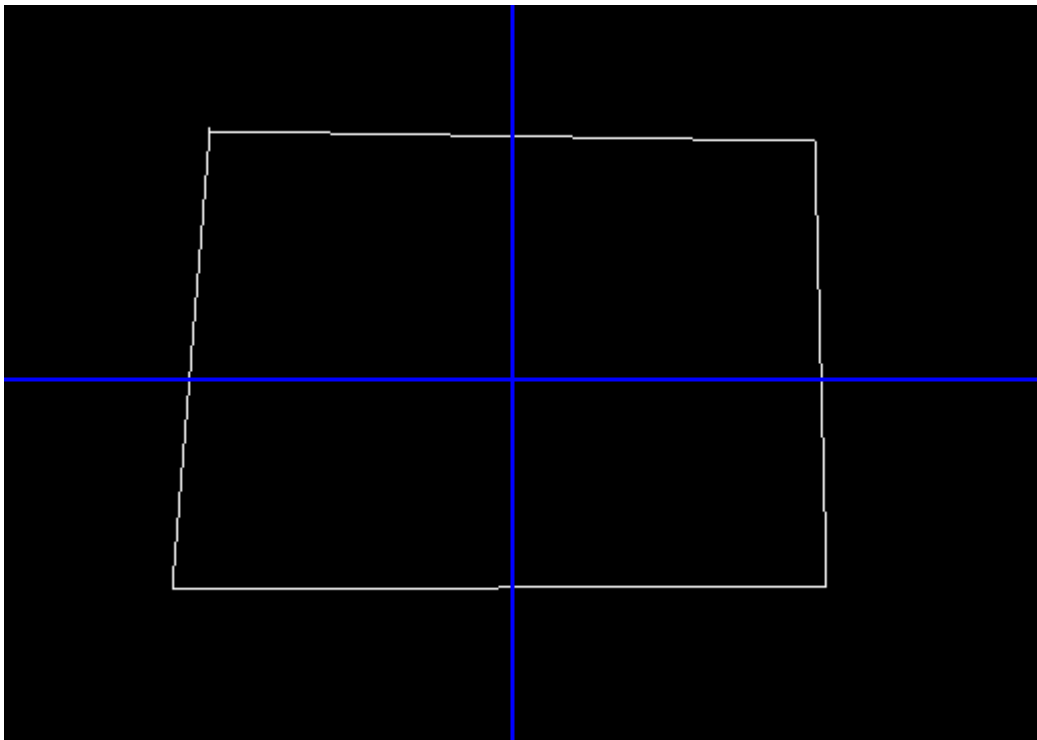
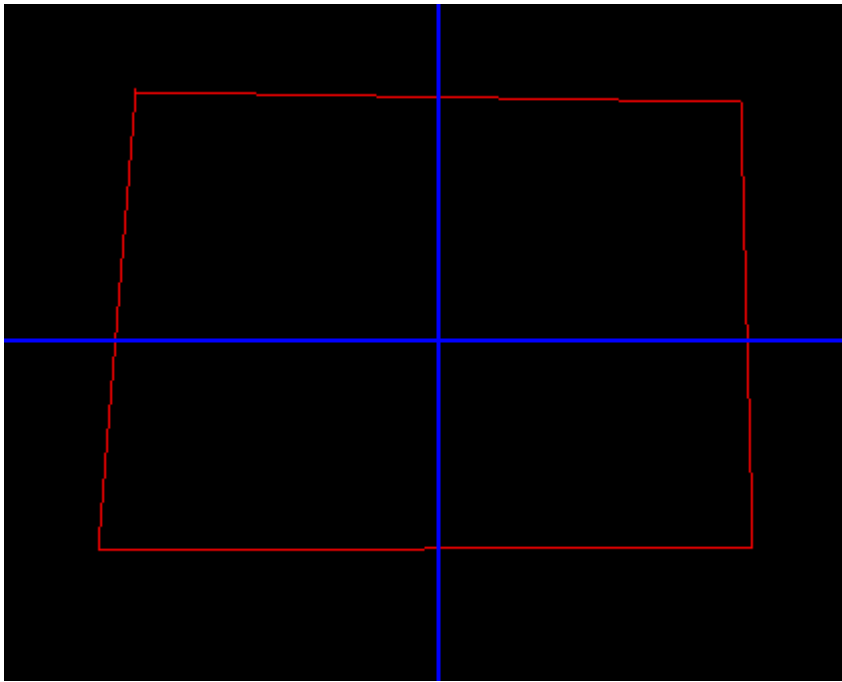
B- translation



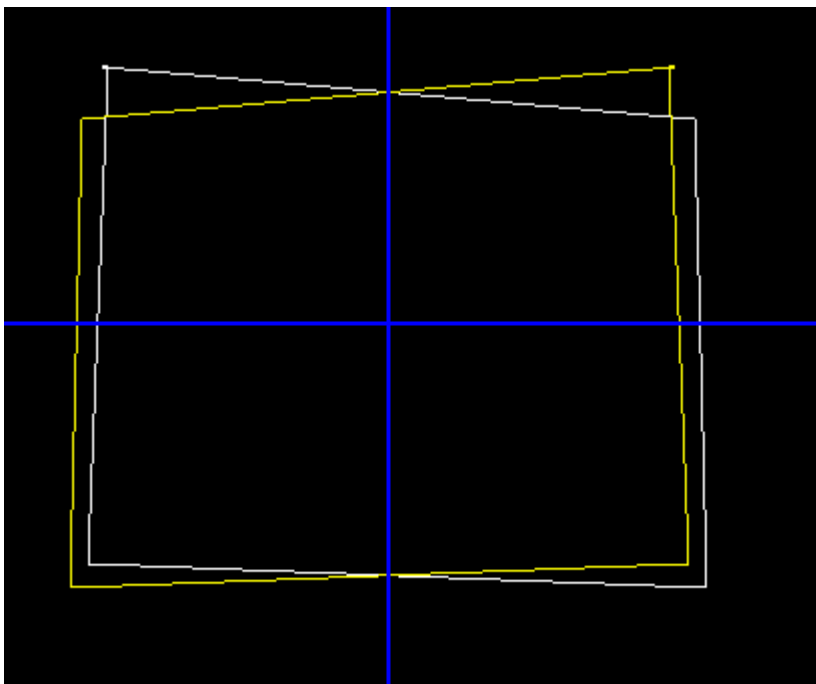
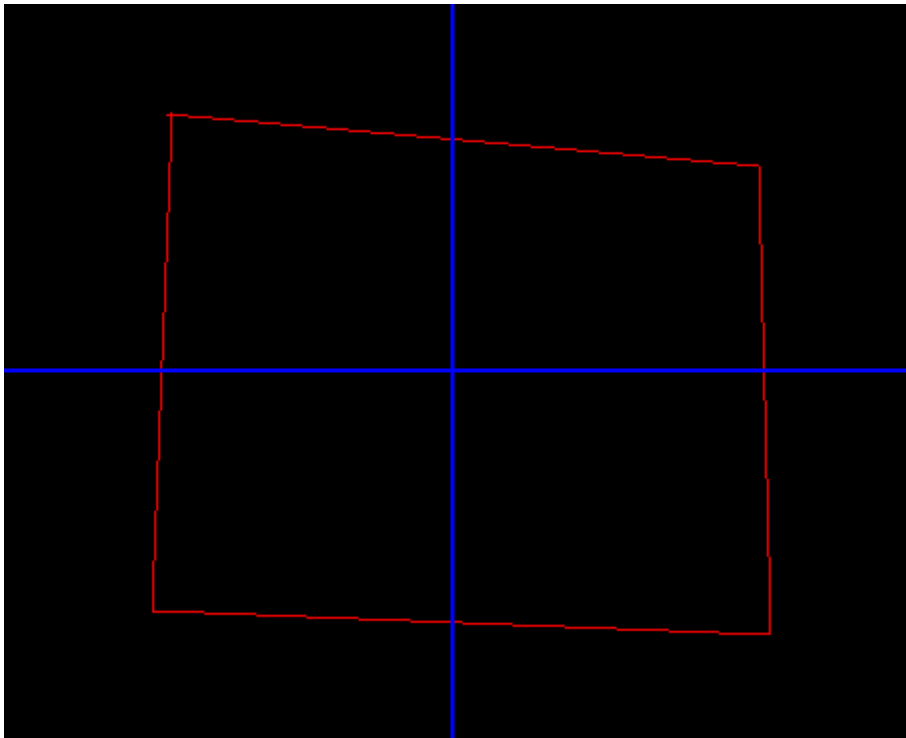
C - rotation



D - scaling



E - reflection



Koch curve:

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

```
using namespace std;
float x1,x2,y1,y2,n;
```



```

void getdata()
{
    cout<<"Enter Start & End Points of Line: ";
    cin>>x1>>y1>>x2>>y2;
    cout<<"Enter no. of Iteration: ";
    cin>>n;
}

void koch(float x1,float y1,float x2,float y2,float n)
{
    float ang=60;ang=ang*3.14/180;
    float x3=(2*x1+x2)/3;
    float y3=(2*y1+y2)/3;
    float x4=(x1+2*x2)/3;
    float y4=(y1+2*y2)/3;
    float x=x3+(x4-x3)*0.5+(y4-y3)*0.8660;
    float y=y3-(x4-x3)*0.8660+(y4-y3)*0.5;
    if(n>0)
    {
        koch(x1,y1,x3,y3,n-1);
        koch(x3,y3,x,y,n-1);
        koch(x,y,x4,y4,n-1);
        koch(x4,y4,x2,y2,n-1);
    }
    else
    {
        glBegin(GL_LINE_STRIP);
        glClearColor(1.0,1.0,1.0,0.0);
        glColor3f(0.0,1.0,1.0);
        glVertex2f(x1,y1);
        glColor3f(0.0,1.0,1.0);
        glVertex2f(x3,y3);
        glColor3f(1.0,1.0,0.0);
        glVertex2f(x,y);
        glColor3f(1.0,0.0,1.0);
        glVertex2f(x4,y4);
        glColor3f(1.0,1.0,1.0);
        glVertex2f(x2,y2);
        glEnd();
    }
}

void Init()
{
    glClearColor(0.0,0.0,0.0,0.0);
    glColor3f(0.0,0.0,0.0);
    gluOrtho2D(0.0,640.0,480.0,0.0);
}

```

```

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    koch(x1,y1,x2,y2,n);
    glFlush();
}
int main(int argv,char **argc)
{

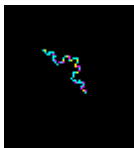
    getdata();

    glutInit(&argv,argc);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(640,480);
    glutCreateWindow("Koch Curve Implementation");
    Init();
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}

```

Output:



Line clipping algorithm:

```

#include <windows.h>
#include<GL/glut.h>
#include<math.h>
#include<stdio.h>
#include<iostream>

void display();
using namespace std;
float xmin=-100;
float ymin=-100;
float xmax=100;
float ymax=100;
float xd1,yd1,xd2,yd2;

void init(void)
{

    glClearColor(0.0,0,0,0);

```

```

    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-300,300,-300,300);

}

int code(float x,float y)
{
    int c=0;
    if(y>ymax)c=8;
    if(y<ymin)c=4;
    if(x>xmax)c=2;
    if(x<xmin)c=1;
    return c;
}

void cohen_Line(float x1,float y1,float x2,float y2)
{
    int c1=code(x1,y1);
    int c2=code(x2,y2);
    float m=(y2-y1)/(x2-x1);
    while((c1|c2)>0)
    {
        if((c1 & c2)>0)
        {
            exit(0);
        }

        float xi=x1;float yi=y1;
        int c=c1;
        if(c==0)
        {
            c=c2;
            xi=x2;
            yi=y2;
        }
        float x,y;
        if((c & 8)>0)
        {
            y=ymax;
            x=xi+ 1.0/m*(ymax-yi);
        }
        else
            if((c & 4)>0)
            {
                y=ymin;
                x=xi+1.0/m*(ymin-yi);
            }
        else

```

```

        if((c & 2)>0)
        {
            x=xmax;
            y=yi+m*(xmax-xi);
        }
        else
        if((c & 1)>0)
        {
            x=xmin;
            y=yi+m*(xmin-xi);
        }

        if(c==c1)
        {
            xd1=x;
            yd1=y;
            c1=code(xd1,yd1);
        }

        if(c==c2)
        {
            xd2=x;
            yd2=y;
            c2=code(xd2,yd2);
        }
    }

    display();

}

void mykey(unsigned char key,int x,int y)
{
    if(key=='c')
    {
        cohen_Line(xd1,yd1,xd2,yd2);
        glFlush();
    }
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0,1.0,0.0);

    glBegin(GL_LINE_LOOP);
    glVertex2i(xmin,ymin);
    glVertex2i(xmin,ymax);

```

```

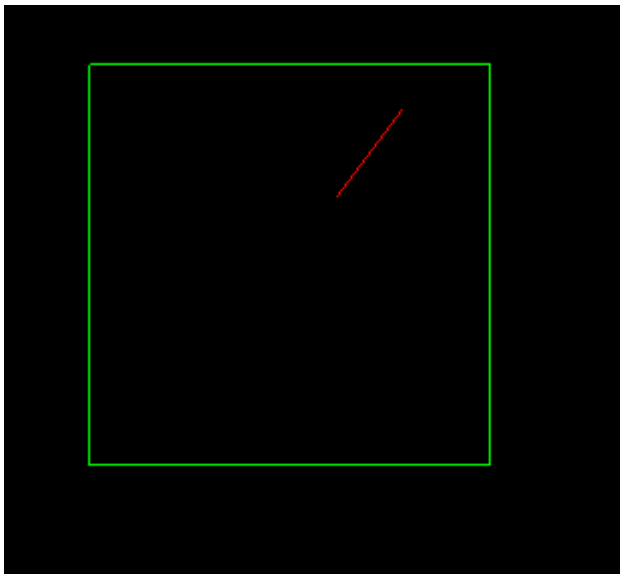
glVertex2i(xmax,ymax);
glVertex2i(xmax,ymin);
glEnd();
glColor3f(1.0,0.0,0.0);
glBegin(GL_LINES);
glVertex2i(xd1,yd1);
glVertex2i(xd2,yd2);
glEnd();
glFlush();

}

int main(int argc,char** argv)
{
    printf("Enter line co-ordinates:");
    cin>>xd1>>y1>>xd2>>y2;
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(600,600);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Line Clipping");
    glutDisplayFunc(display);
    glutKeyboardFunc(mykey);
    init();
    glutMainLoop();
    return 0;
}

```

Output:



Line program:

```
#include<windows.h>
#include<GL/glut.h>
#include<stdlib.h>
#include<stdio.h>
GLint x0,y0,xEnd,yEnd;
inline GLint round(const GLfloat a)
{
    return GLint(a+0.5);
}
void myInit(void)
{
    glClearColor(0.0,1.0,1.0,0.0);
    glColor3f(1.0f,0.0f,0.0f);
    glPointSize(3.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,640.0,0.0,480.0);
}
void readInput()
{
    printf("Enter x0, y0, xEnd, yEnd: ");
    scanf("%i %i %i %i",&x0,&y0,&xEnd,&yEnd);
}
void setPixel(GLint xcoordinate, GLint ycoordinate)
{
    glBegin(GL_POINTS);
    glVertex2i(xcoordinate,ycoordinate);
    glEnd();
    glFlush(); //forces execution in finite time
}
void lineDDA(GLint x0,GLint y0,GLint xEnd,GLint yEnd)
{
    GLint dx = abs(xEnd-x0);
    GLint dy = abs(yEnd-y0);
    GLint steps,k;
    GLfloat xIncrement,yIncrement,x=x0,y=y0;

    if(dx>dy)
        steps = dx;
    else
        steps = dy;

    xIncrement = GLfloat(dx) / GLfloat(steps);
```

```

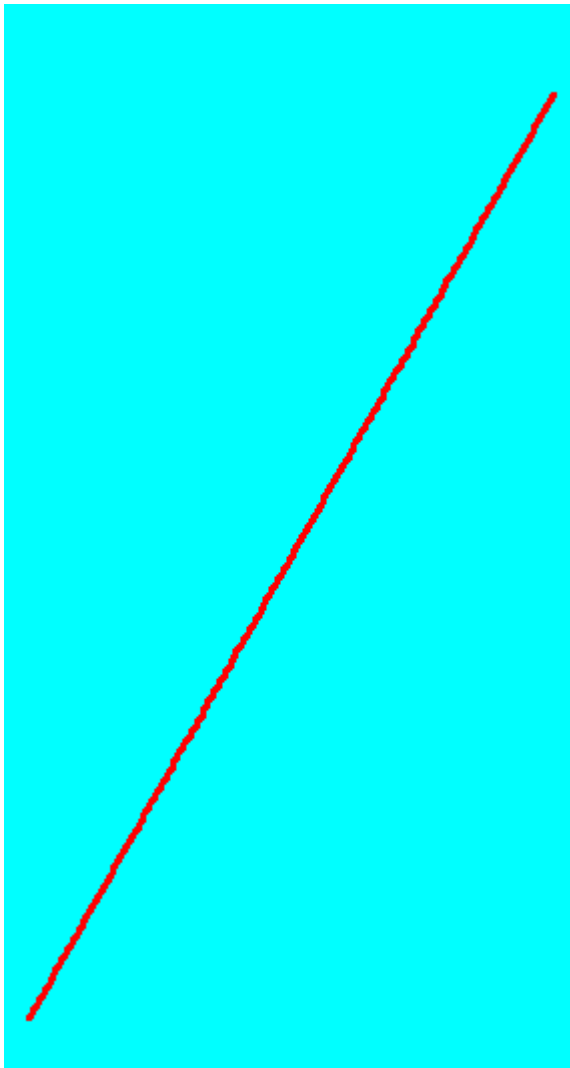
yIncrement = GLfloat(dy) / GLfloat(steps);
setPixel(round(x),round(y));

for(k=1;k<steps;k++)
{
    x+= xIncrement;
    y+= yIncrement;
    setPixel(round(x),round(y));
}

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    lineDDA(x0,y0,xEnd,yEnd);
}

int main(int argc,char *argv[])
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600,600);
    glutInitWindowPosition(50,50);
    glutCreateWindow("DDA Line Algorithm");
    readInput();
    glutDisplayFunc(Display);
    myInit();
    glutMainLoop();
    return EXIT_SUCCESS;
}
Output:

```



Mid point circle drawing:

```
#include<windows.h>
#include<GL/glut.h>
#include<stdio.h>
GLint xc,yc,r;
void myInit(void)
{
    glClearColor(0.0,1.0,1.0,0.0); //clear values for RGBA
    glColor3f(1.0f,0.0f,0.0f); //set current color - RGB
    glPointSize(3.0); //diameters of rasterized points
    glMatrixMode(GL_PROJECTION); // specifies current matrix
    glLoadIdentity();
    gluOrtho2D(0.0,800.0,0.0,600.0); //left, right, top, bottom
    //glOrtho(-sizes/2,sizes/2,-sizes/2,sizes/2,-1,1);
}
void readInput()
{
    printf("Enter xc, yc, radius: ");
```



```

    scanf("%i %i %i",&xc,&yc,&r);

}

void setPixel(GLint xcoordinate, GLint ycoordinate)
{
    glBegin(GL_POINTS);
    glVertex2i(xcoordinate,ycoordinate);
    glEnd();
    glFlush();
}

/* void draw_axis()
{
    GLint i=(-sizes)/2;
    for(;i<(sizes/2);i++)
    {
        setPixel(i,0);
        setPixel(0,i);
    }

} */

void draw_in_each_oct(GLint xk,GLint yk, GLint xc,GLint yc)
{
    setPixel(xc+xk,yc+yk);
    setPixel(xc+yk,yc+xk);
    setPixel(xc-yk,yc+xk);
    setPixel(xc-xk,yc+yk);
    setPixel(xc-xk,yc-yk);
    setPixel(xc-yk,yc-xk);
    setPixel(xc+yk,yc-xk);
    setPixel(xc+xk,yc-yk);
}

void midPtCircle(GLint xc,GLint yc,GLint r)
{
    GLint pk,xk,yk;
    pk=1-r;
    xk=0;
    yk=r;
    draw_in_each_oct(xk,yk,xc,yc);
    while(xk<=yk)
    {
        if(pk<0)
        {
            xk=xk+1;
            pk=pk+(2*xk)+1;
        }
        else
        {

```

```

        xk=xk+1;
        yk=yk-1;
        pk=pk+(2*xk)+1-(2*yk);
    }
    draw_in_each_oct(xk,yk,xc,yc);
}
}

```

```

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    //draw_axis();
    midPtCircle(xc,yc,r);
}

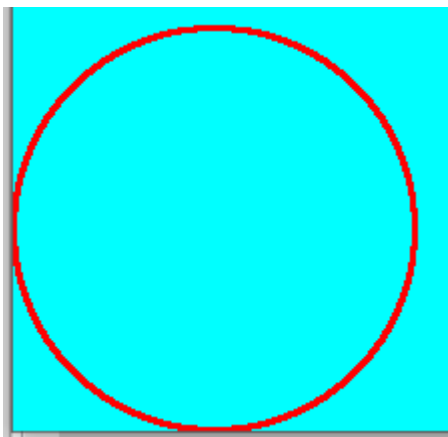
```

```

int main(int argc,char *argv[])
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(100,50);
    glutCreateWindow("Mid Point Circle");
    readInput();
    glutDisplayFunc(Display);
    myInit();
    glutMainLoop();
    return 0;
}

```

Output:



Mid point ellipse:

```

#include<windows.h>
#include<bits/stdc++.h>

```

```

#include<GL/glut.h>

using namespace std;

int xc,yc,a,b;
void pp(int x,int y)
{
    glBegin(GL_POINTS);
        glVertex2i(x,y);
    glEnd();
}
void getpixel(int x,int y)
{
    pp(300+x,300+y);
    pp(300-x,300+y);
    pp(300-x,300-y);
    pp(300+x,300-y);
}

void display ()
{
    glClear(GL_COLOR_BUFFER_BIT);
    int x=0;
    int y=b;
    int e=a*a;
    int g=b*b;
    int fx=0;
    int fy=2*e*b;

    int p0=g-e*b+0.25*e;
    while(fx<fy){
        getpixel(x,y);
        x++;
        fx=fx+2*g;
        if(p0<0){
            p0=p0+fx+g;
        }
        else{
            y--;
            fy=fy-2*e;
            p0=p0+fx+g-fy;
        }
    }
    getpixel(x,y);
    p0=g*(x+0.5)*(x+0.5)+e*(y-1)*(y-1)-e*g;
    while(y>0)

```

```

{
    y--;
    fy=fy-2*e;
    if(p0>=0)
        p0=p0-fy+e;
    else{
        x++;
        fx=fx+2*e;
        p0=p0+fx-fy+e;
    }
    getpixel(x,y);
}
glFlush();
}

void init(void)
{
    glClearColor(0.0,0.0,0.0,0.0); //clear values for RGBA
    glColor3f(1.0f,1.0f,1.0f); //set current color - RGB
    glPointSize(3.0); //diameters of rasterized points
    glMatrixMode(GL_PROJECTION); // specifies current matrix
    glLoadIdentity();
    gluOrtho2D(0.0,800.0,0.0,600.0); //left, right, top, bottom
}

int main (int argc, char **argv)
{
    cout<<"enter values of a and b:"<<endl;
    cin>>a>>b;

    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(100,50);
    glutCreateWindow("MidPoint Ellipse Algorithm");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
}

```

Output:

