1. Write a program to generate a line using Bresenham's line drawing technique. Consider slopes greater than one and slopes less than one. User must able to draw as many lines and specify inputs through keyboard/mouse.

Program:

```
#include <iostream>
#include <GL/glut.h>
#include <time.h>
using namespace std;
int x1, x2, yc1, y2;
int flag = 0;
void draw_pixel(int x, int y)
{
    glColor3f(1, 0, 0);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();

}
void draw_line()
{
    int dx, dy, i, e;
    int incx, incy, inc1, inc2;
    int x, y;
    dx = x2 - x1;
    dy = y2 - yc1;
    if (dx < 0)dx = -dx;
    if (dy < 0)dy = -dy;
    incx = 1;
    if (x2 < x1)
        incx = -1;
    incy = 1;
    if (y2 < yc1)
        incy = -1;
    x = x1;
    y = yc1;
    if (dx > dy)
    {
        draw_pixel(x, y);
        e = 2 * dy - dx;
        inc1 = 2 * (dy - dx);
```

```
            inc2 = 2 * dy;
            for (i = 0; i < dx; i++)
            {
                    if (e > 0)
                    {
                            y += incy;
                            e += inc1;
                    }
                    else
                            e += inc2;
                    x += incx;
                    draw_pixel(x, y);
            }
    }
    else
    {

            draw_pixel(x, y);
            e = 2 * dx - dy;
            inc1 = 2 * (dx - dy);
            inc2 = 2 * dx;
            for (i = 0; i < dy; i++)
            {
                    if (e > 0)
                    {
                            x += incx;
                            e += inc1;

                    }
                    else

                            e += inc2;
                    y += incy;

                    draw_pixel(x, y);
            }
    }
    glFlush();
}
void myinit()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1, 1, 1, 1);
    gluOrtho2D(-250, 250, -250, 250);
```

```
}
void MyMouse(int button, int state, int x, int y)
{
    switch (button)
    {
    case GLUT_LEFT_BUTTON:
        if (state == GLUT_DOWN)
        {
            if (flag == 0)
            {
                printf("Defining x1,y1");
                x1 = x - 250;
                yc1 = 250 - y;
                flag++;
                cout << x1 << " " << yc1 << " \n";
            }
            else
            {
                printf("Defining x2,y2");
                x2 = x - 250;
                y2 = 250 - y;
                flag = 0;
                cout << x2 << " " << y2 << " \n";
                draw_line();

            }

        }
        break;
    }
}
void display()
{}
int main(int ac, char* av[])
{
    /*
    //FOR KEYBOARD
    cout<<"X1\n";
    cin>>x1;
    cout<<"Y1\n";
    cin>>yc1;
    cout<<"X2\n";
    cin>>x2;
    cout<<"Y2\n";
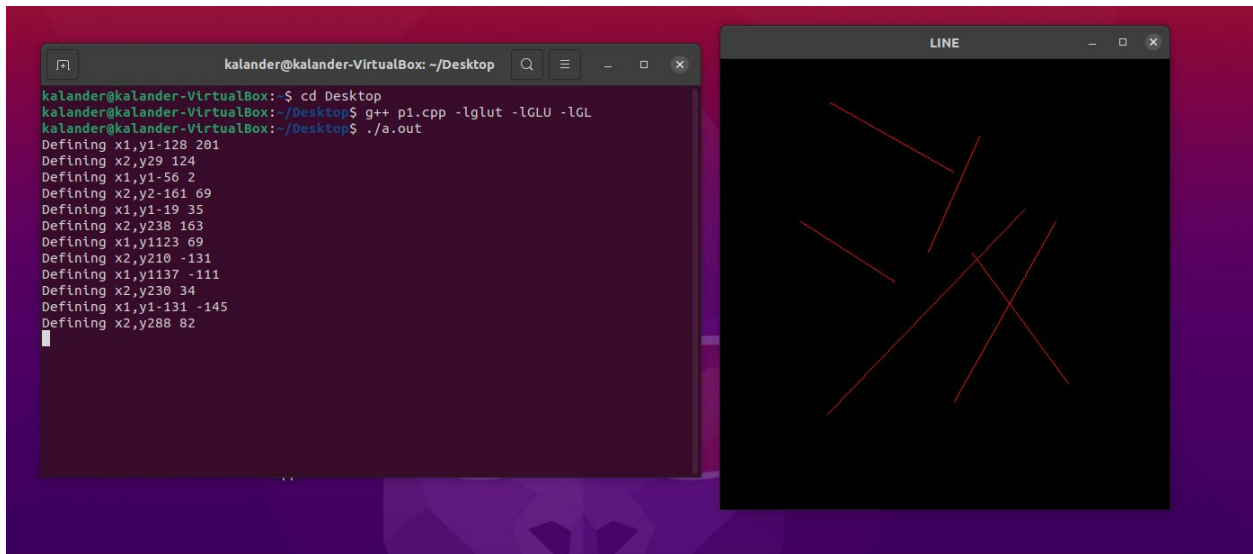```

```
    cin>>y2;
    //END KEYBOARD
    */
    glutInit(&ac, av);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 200);
    glutCreateWindow("LINE");
    myinit();
    glutMouseFunc(MyMouse); //INCLUDE TO USE MOUSE, REMOVE WHILE USING
KEYBOARD
    //draw_line(); //INCLUDE TO USE KEYBOARD, REMOVE WHILE USING MOUSE
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Output:

2) a program to generate a circle and ellipse using Bresenham's circle drawing and ellipse drawing techniques. Use two windows to draw circle in one window and ellipse in the other window. User can specify inputs through keyboard/mouse.

Code:

```c
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
int xc, yc, r;
int rx, ry, xce, yce;
void draw_circle(int xc, int yc, int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(xc + x, yc + y);
    glVertex2i(xc - x, yc + y);
    glVertex2i(xc + x, yc - y);
    glVertex2i(xc - x, yc - y);
    glVertex2i(xc + y, yc + x);
    glVertex2i(xc - y, yc + x);
    glVertex2i(xc + y, yc - x);
    glVertex2i(xc - y, yc - x);
    glEnd();
}
void circlebres()
{
    glClear(GL_COLOR_BUFFER_BIT);
    int x = 0, y = r;
    int d = 3 - 2 * r;
    while (x <= y)
    {
        draw_circle(xc, yc, x, y);
        x++;
        if (d < 0)
                d = d + 4 * x + 6;
        else
        {
                y--;
                d = d + 4 * (x - y) + 10;
        }
        draw_circle(xc, yc, x, y);
    }
    glFlush();
}
int p1_x, p2_x, p1_y, p2_y;
int point1_done = 0;
void myMouseFunccircle(int button, int state, int x, int y)
{
```

```
        if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && point1_done == 0)
        {
            p1_x = x - 250;
            p1_y = 250 - y;
            point1_done = 1;
        }
        else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        {
            p2_x = x - 250;
            p2_y = 250 - y;
            xc = p1_x;
            yc = p1_y;
            float exp = (p2_x - p1_x) * (p2_x - p1_x) + (p2_y - p1_y) * (p2_y - p1_y);
            r = (int)(sqrt(exp));
            circlebres();
            point1_done = 0;
        }
}


/////ELLIPSE//////////////
void draw_ellipse(int xce, int yce, int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(x + xce, y + yce);
    glVertex2i(-x + xce, y + yce);
    glVertex2i(x + xce, -y + yce);
    glVertex2i(-x + xce, -y + yce);
    glEnd();
}
void midptellipse()
{
    glClear(GL_COLOR_BUFFER_BIT);
    float dx, dy, d1, d2, x, y;
    x = 0;
    y = ry;

    // Initial decision parameter of region 1
    d1 = (ry * ry) - (rx * rx * ry) +
            (0.25 * rx * rx);
    dx = 2 * ry * ry * x;
    dy = 2 * rx * rx * y;

    // For region 1
```

```
while (dx < dy)
{

        // Print points based on 4-way symmetry
        draw_ellipse(xce, yce, x, y);


        // Checking and updating value of
        // decision parameter based on algorithm
        if (d1 < 0)
        {
                x++;
                dx = dx + (2 * ry * ry);
                d1 = d1 + dx + (ry * ry);
        }
        else
        {
                x++;
                y--;
                dx = dx + (2 * ry * ry);
                dy = dy - (2 * rx * rx);
                d1 = d1 + dx - dy + (ry * ry);
        }
}

// Decision parameter of region 2
d2 = ((ry * ry) * ((x + 0.5) * (x + 0.5))) +
        ((rx * rx) * ((y - 1) * (y - 1))) -
        (rx * rx * ry * ry);

// Plotting points of region 2
while (y >= 0)
{

        // Print points based on 4-way symmetry
        draw_ellipse(xce, yce, x, y);

        // Checking and updating parameter
        // value based on algorithm
        if (d2 > 0)
        {
                y--;
                dy = dy - (2 * rx * rx);
```

```c
                d2 = d2 + (rx * rx) - dy;
        }
        else
        {
                y--;
                x++;
                dx = dx + (2 * ry * ry);
                dy = dy - (2 * rx * rx);
                d2 = d2 + dx - dy + (rx * rx);
        }
    }
    glFlush();
}
int p1e_x, p2e_x, p1e_y, p2e_y, p3e_x, p3e_y;
int point1e_done = 0;
void myMouseFunc(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && point1e_done == 0)
    {
        p1e_x = x - 250;
        p1e_y = 250 - y;
        xce = p1e_x;
        yce = p1e_y;
        point1e_done = 1;
    }
    else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && point1e_done == 1)
    {
        p2e_x = x - 250;
        p2e_y = 250 - y;
        float exp = (p2e_x - p1e_x) * (p2e_x - p1e_x) + (p2e_y - p1e_y) * (p2e_y - p1e_y);
        rx = (int)(sqrt(exp));
        //midptellipse();
        point1e_done = 2;
    }
    else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && point1e_done == 2)
    {
        p3e_x = x - 250;
        p3e_y = 250 - y;
        float exp = (p3e_x - p1e_x) * (p3e_x - p1e_x) + (p3e_y - p1e_y) * (p3e_y - p1e_y);
        ry = (int)(sqrt(exp));
        midptellipse();
        point1e_done = 0;
    }
}
```

```c
void myDrawing()
{ }
void myDrawingc()
{ }

void minit()
{
    glClearColor(1, 1, 1, 1);
    glColor3f(1.0, 0.0, 0.0);
    glPointSize(3.0);
    gluOrtho2D(-250, 250, -250, 250);
}

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

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    /*
    //FOR MOUSE
    int id1 = glutCreateWindow("Circle");
    glutSetWindow(id1);
    glutMouseFunc(myMouseFunccircle);
    glutDisplayFunc(myDrawingc);
    minit();
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(600, 100);
    int id2 = glutCreateWindow("Ellipse");
    glutSetWindow(id2);
    glutMouseFunc(myMouseFunc);
    glutDisplayFunc(myDrawing);
    //END MOUSE
    */
    //FOR KEYBOARD
    printf("Enter 1 to draw circle , 2 to draw ellipse\n");
    int ch;
    scanf("%d",&ch);
    switch(ch){
    case 1:
    printf("Enter coordinates of centre of circle and radius\n");
    scanf("%d%d%d",&xc,&yc,&r);
    glutCreateWindow("Circle");
```
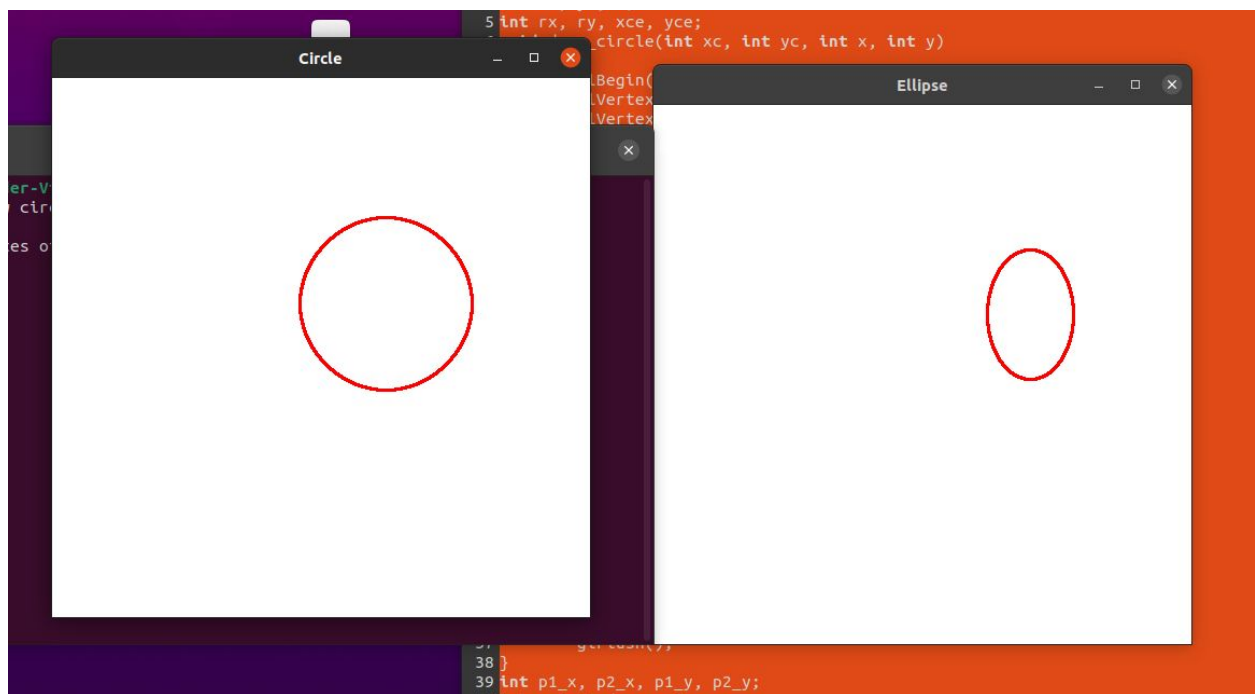
```
glutDisplayFunc(circlebres);
break;
case 2:
printf("Enter coordinates of centre of ellipse and major and minor radius\n");
scanf("%d%d%d%d",&xce,&yce,&rx,&ry);
glutCreateWindow("Ellipse");
glutDisplayFunc(midptellipse);
break;
}
//END KEYBOARD
minit();
glutMainLoop();
}
```

Output:

3) Write a program to recursively subdivides a tetrahedron to form 3D Sierpinski gasket. The number of recursive steps is to be specified at execution time.

Code:

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

int m;
typedef float point[3];
point tetra[4] = { {0,100,-100},{0,0,100},{100,-100,-100},{-100,-100,-100} };
void tetrahedron(void);
void myinit(void);
void divide_triangle(point a, point b, point c, int m);
void draw_triangle(point p1, point p2, point p3);
int main(int argv, char** argc)
{
    //int m;
    printf("Enter the number of iterations: ");
    scanf("%d", &m);
    glutInit(&argv, argc);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition(100, 200);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Seirpinski Gasket");
    glutDisplayFunc(tetrahedron);
    glEnable(GL_DEPTH_TEST);
    myinit();
    glutMainLoop();
}
void divide_triangle(point a, point b, point c, int m)
{
    point v1, v2, v3;
    int j;
    if (m > 0) {
            for (j = 0; j < 3; j++)
                    v1[j] = (a[j] + b[j]) / 2;
            for (j = 0; j < 3; j++)
                    v2[j] = (a[j] + c[j]) / 2;
            for (j = 0; j < 3; j++)
                    v3[j] = (b[j] + c[j]) / 2;

            divide_triangle(a, v1, v2, m - 1);
            divide_triangle(c, v2, v3, m - 1);
            divide_triangle(b, v3, v1, m - 1);
    }
    else
```

```
            draw_triangle(a, b, c);
}
void myinit()
{
    glClearColor(1, 1, 1, 1);

    //glFlush();
    glOrtho(-500.0, 500.0, -500.0, 500.0, -500.0, 500.0);
    //gluOrtho(-500.0,500.0,-500.0,500.0,-500.0,500.0);
}
void tetrahedron(void)
{
    //myinit();
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    divide_triangle(tetra[0], tetra[1], tetra[2], m);
    glColor3f(0.0, 1.0, 0.0);
    divide_triangle(tetra[3], tetra[2], tetra[1], m);
    glColor3f(0.0, 0.0, 1.0);
    divide_triangle(tetra[0], tetra[3], tetra[1], m);
    glColor3f(0.0, 0.0, 0.0);
    divide_triangle(tetra[0], tetra[2], tetra[3], m);
    glFlush();
}
void draw_triangle(point p1, point p2, point p3)
{
    glBegin(GL_TRIANGLES);
    glVertex3fv(p1);
    glVertex3fv(p2);
    glVertex3fv(p3);
    glEnd();
}
```

Output:
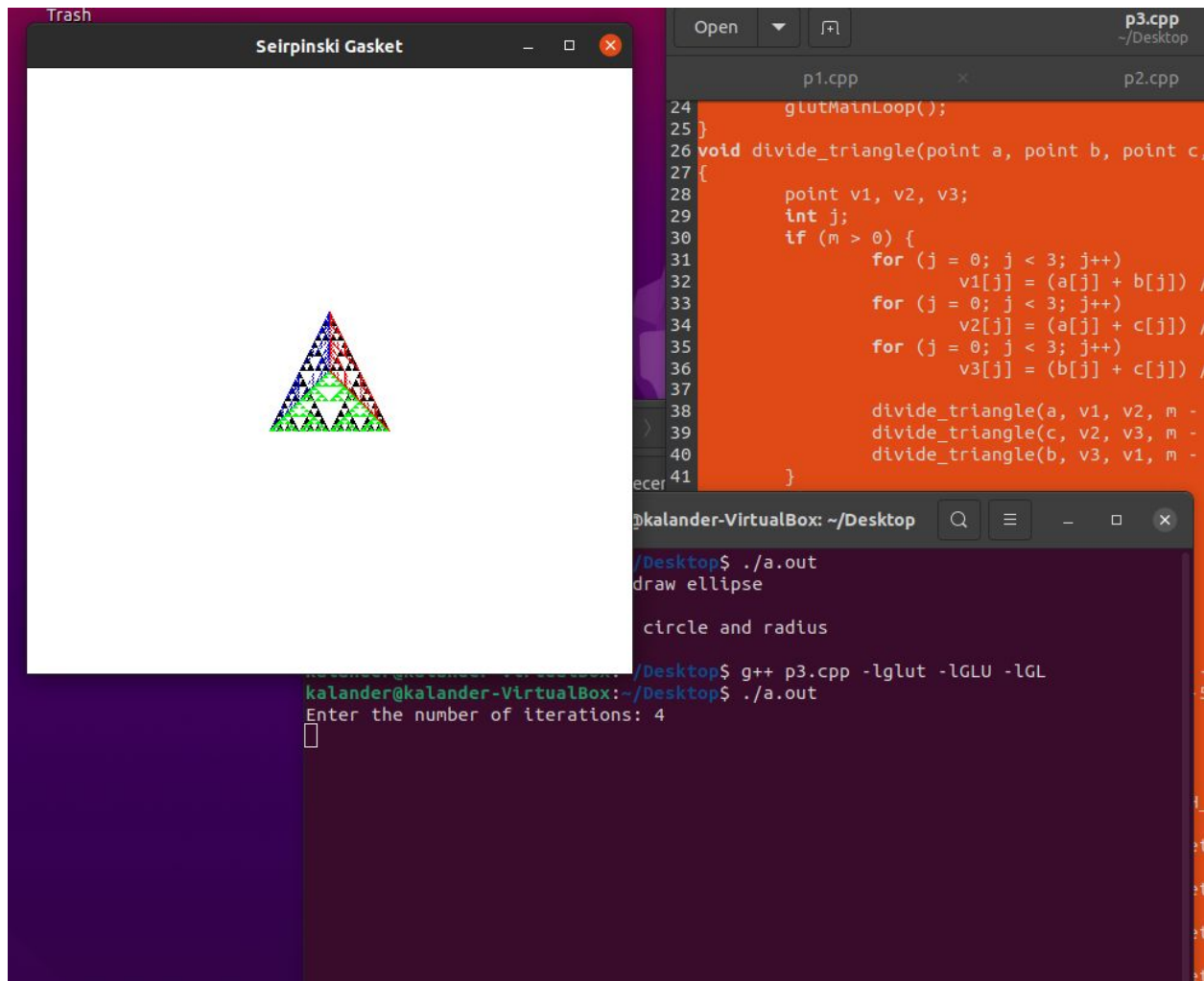
4. Write a program to fill any given polygon using scan-line area filling algorithm.

Code:

```cpp
#include <GL/glut.h>
#include <unistd.h>
float x1,x2,x3,x4,y1,y2,y3,y4;
void edgedetect(float x1,float y1,float x2,float y2,int *le,int *re)
{
float mx,x,temp;
int i;
if((y2-y1)<0)
{
temp=y1;y1=y2;y2=temp;
```

```c
temp=x1;x1=x2;x2=temp;
}
if((y2-y1)!=0)
mx=(x2-x1)/(y2-y1);
else mx=x2-x1;
x=x1;
for(i=y1;i<=y2;i++)
{
if(x<(float)le[i])
le[i]=(int)x;
if(x>(float)re[i])
re[i]=(int)x;
x+=mx;
}
}
void draw_pixel(int x,int y)
{
glColor3f(1.0,0.0,0.0);
//sleep(10);
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
glFlush();
}
void scanfill(float x1,float y1,float x2,float y2,float x3,float y3,float x4,float y4)
{
int le[500],re[500],i,y;
for(i=0;i<500;i++)
{
le[i]=500;
re[i]=0;
}
edgedetect(x1,y1,x2,y2,le,re);
edgedetect(x2,y2,x3,y3,le,re);
edgedetect(x3,y3,x4,y4,le,re);
edgedetect(x4,y4,x1,y1,le,re);
for(y=0;y<500;y++)
{
if(le[y]<=re[y])
for(i=(int)le[y];i<(int)re[y];i++)
draw_pixel(i,y);
}
}
void display()
```

```
{
x1=200.0;y1=200.0;x2=100.0;y2=300.0;x3=200.0;y3=400.0;x4=300.0;y4=300.0;
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0, 0.0, 1.0);
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);
glFlush();
}
void myinit()
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(1.0,0.0,0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,499.0,0.0,499.0);
}
int main(int argc, char** argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutCreateWindow("Filling a Polygon using Scan-line Algorithm");
glutDisplayFunc(display);
myinit();
glutMainLoop();
}
```

Output:

Filling a Polygon using Scan-line Algorithm