```
// C++ program for implementing Sutherland–Hodgman
// algorithm for polygon clipping
#include<iostream>
#include<GL/glut.h>
using namespace std;
int poly_size, poly_points[20][2], org_poly_size, org_poly_points[20]
[2], clipper_size, clipper_points[20][2];
const int MAX_POINTS = 20;


// Returns x-value of point of intersection of two
// lines

void drawPoly(int p[][2], int n) {
        glBegin(GL_POLYGON);
        for (int i = 0; i < n; i++)
                glVertex2f(p[i][0], p[i][1]);
        glEnd();
}

int x_intersect(int x1, int y1, int x2, int y2,
        int x3, int y3, int x4, int y4)
{
        int num = (x1 * y2 - y1 * x2) * (x3 - x4) -
                (x1 - x2) * (x3 * y4 - y3 * x4);
        int den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
        return num / den;
}

// Returns y-value of point of intersectipn of
// two lines
int y_intersect(int x1, int y1, int x2, int y2,
        int x3, int y3, int x4, int y4)
{
        int num = (x1 * y2 - y1 * x2) * (y3 - y4) -
                (y1 - y2) * (x3 * y4 - y3 * x4);
        int den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
        return num / den;
}

// This functions clips all the edges w.r.t one clip
// edge of clipping area
void clip(int poly_points[][2], int& poly_size,
        int x1, int y1, int x2, int y2)
{
        int new_points[MAX_POINTS][2], new_poly_size = 0;

        // (ix,iy),(kx,ky) are the co-ordinate values of
        // the points
        for (int i = 0; i < poly_size; i++)
        {
                // i and k form a line in polygon
                int k = (i + 1) % poly_size;
                int ix = poly_points[i][0], iy = poly_points[i][1];
                int kx = poly_points[k][0], ky = poly_points[k][1];
```

```
                // Calculating position of first point
                // w.r.t. clipper line
                int i_pos = (x2 - x1) * (iy - y1) - (y2 - y1) * (ix -
x1);

                // Calculating position of second point
                // w.r.t. clipper line
                int k_pos = (x2 - x1) * (ky - y1) - (y2 - y1) * (kx -
x1);

                // Case 1 : When both points are inside
                if (i_pos >= 0 && k_pos >= 0)
                {
                        //Only second point is added
                        new_points[new_poly_size][0] = kx;
                        new_points[new_poly_size][1] = ky;
                        new_poly_size++;
                }

                // Case 2: When only first point is outside
                else if (i_pos < 0 && k_pos >= 0)
                {
                        // Point of intersection with edge
                        // and the second point is added
                        new_points[new_poly_size][0] = x_intersect(x1,
                                y1, x2, y2, ix, iy, kx, ky);
                        new_points[new_poly_size][1] = y_intersect(x1,
                                y1, x2, y2, ix, iy, kx, ky);
                        new_poly_size++;

                        new_points[new_poly_size][0] = kx;
                        new_points[new_poly_size][1] = ky;
                        new_poly_size++;
                }

                // Case 3: When only second point is outside
                else if (i_pos >= 0 && k_pos < 0)
                {
                        //Only point of intersection with edge is
added
                        new_points[new_poly_size][0] = x_intersect(x1,
                                y1, x2, y2, ix, iy, kx, ky);
                        new_points[new_poly_size][1] = y_intersect(x1,
                                y1, x2, y2, ix, iy, kx, ky);
                        new_poly_size++;
                }

                // Case 4: When both points are outside
                else
                {
                        //No points are added
                }
        }

        // Copying new points into original array
        // and changing the no. of vertices
        poly_size = new_poly_size;
```

```
        for (int i = 0; i < poly_size; i++)
        {
                poly_points[i][0] = new_points[i][0];
                poly_points[i][1] = new_points[i][1];
        }
}


void init() {
        glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0.0, 500.0, 0.0, 500.0, 0.0, 500.0);
        glClear(GL_COLOR_BUFFER_BIT);
}

// Implements Sutherland Hodgman algorithm
void display()
{
        init();
        glColor3f(1.0f, 0.0f, 0.0f);
        drawPoly(clipper_points, clipper_size);

        glColor3f(0.0f, 1.0f, 0.0f);
        drawPoly(org_poly_points, org_poly_size);
        //i and k are two consecutive indexes

        for (int i = 0; i < clipper_size; i++)
        {
                int k = (i + 1) % clipper_size;

                // We pass the current array of vertices, it's size
                // and the end points of the selected clipper line
                clip(poly_points, poly_size, clipper_points[i][0],
                        clipper_points[i][1], clipper_points[k][0],
                        clipper_points[k][1]);
        }

        glColor3f(0.0f, 0.0f, 1.0f);
        drawPoly(poly_points, poly_size);
        glFlush();
}

//Driver code
int main(int argc, char* argv[])
{
        printf("Enter no. of vertices: \n");
        scanf_s("%d", &poly_size);
        org_poly_size = poly_size;
        for (int i = 0; i < poly_size; i++)
        {
                printf("Polygon Vertex:\n");
                scanf_s("%d%d", &poly_points[i][0], &poly_points[i]
[1]);
                org_poly_points[i][0] = poly_points[i][0];
                org_poly_points[i][1] = poly_points[i][1];
```

```
        }

        printf("Enter no. of vertices of clipping window:");
        scanf_s("%d", &clipper_size);
        for (int i = 0; i < clipper_size; i++)
        {
                printf("Clip Vertex:\n");
                scanf_s("%d%d", &clipper_points[i][0],
&clipper_points[i][1]);
        }

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(400, 400);
        glutInitWindowPosition(100, 100);
        glutCreateWindow("Polygon Clipping!");
        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```