COHEN SUTHERLAND

```
#include <graphics.h>
#include <iostream>
#include <cmath>

using namespace std;

// Define the window boundaries
#define X_MIN 100
#define Y_MIN 100
#define X_MAX 400
#define Y_MAX 400

// Define the region codes for the clipping
#define INSIDE 0  // 0000
#define LEFT 1    // 0001
#define RIGHT 2   // 0010
#define BOTTOM 4  // 0100
#define TOP 8     // 1000

// Function prototypes
void ddaLine(int x0, int y0, int x1, int y1);
int computeRegionCode(int x, int y);
void cohenSutherlandClip(int x0, int y0, int x1, int y1);

// Function to compute the region code for a point (x, y)
int computeRegionCode(int x, int y) {
   int code = INSIDE;

   if (x < X_MIN)       // to the left of the window
```

```
        code |= LEFT;
    else if (x > X_MAX)   // to the right of the window
        code |= RIGHT;
    if (y < Y_MIN)       // below the window
        code |= BOTTOM;
    else if (y > Y_MAX)   // above the window
        code |= TOP;

    return code;
}

// Implementing the Cohen-Sutherland Line Clipping Algorithm
void cohenSutherlandClip(int x0, int y0, int x1, int y1) {
    int code0 = computeRegionCode(x0, y0);
    int code1 = computeRegionCode(x1, y1);
    bool accept = false;

    while (true) {
        if ((code0 == 0) && (code1 == 0)) {
            // Both endpoints are inside the rectangle
            accept = true;
            break;
        } else if ((code0 & code1) != 0) {
            // Logical AND is not 0: Both points are outside the rectangle in the same region
            break;
        } else {
            // Calculate the intersection point
            int codeOut;
            int x, y;

            // Determine which point to clip
```

```
if (code0 != 0) {
   codeOut = code0;
} else {
   codeOut = code1;
}

// Now find the intersection point using the codeOut
if (codeOut & TOP) {  // point is above the window
   x = x0 + (x1 - x0) * (Y_MAX - y0) / (y1 - y0);
   y = Y_MAX;
} else if (codeOut & BOTTOM) {  // point is below the window
   x = x0 + (x1 - x0) * (Y_MIN - y0) / (y1 - y0);
   y = Y_MIN;
} else if (codeOut & RIGHT) {  // point is to the right of the window
   y = y0 + (y1 - y0) * (X_MAX - x0) / (x1 - x0);
   x = X_MAX;
} else if (codeOut & LEFT) {  // point is to the left of the window
   y = y0 + (y1 - y0) * (X_MIN - x0) / (x1 - x0);
   x = X_MIN;
}

// Update the point outside the rectangle
if (codeOut == code0) {
   x0 = x;
   y0 = y;
   code0 = computeRegionCode(x0, y0);
} else {
   x1 = x;
   y1 = y;
   code1 = computeRegionCode(x1, y1);
}
```

```
        }
    }

    // If the line is accepted, draw it using DDA
    if (accept) {
        ddaLine(x0, y0, x1, y1);
    } else {
        cout << "Line is outside the clipping window" << endl;
    }
}

// DDA Line Drawing Algorithm to draw a line
void ddaLine(int x0, int y0, int x1, int y1) {
    int dx = x1 - x0;
    int dy = y1 - y0;
    int steps;
    float xIncrement, yIncrement;

    // Calculate the number of steps required
    if (abs(dx) > abs(dy)) {
        steps = abs(dx);
    } else {
        steps = abs(dy);
    }

    // Calculate the increment values for x and y
    xIncrement = dx / (float)steps;
    yIncrement = dy / (float)steps;

    // Draw the line
    float x = x0;
```

```
    float y = y0;
    for (int i = 0; i <= steps; i++) {
        putpixel(round(x), round(y), WHITE);  // Plot the pixel
        x += xIncrement;
        y += yIncrement;
    }
}

int main() {
    int x0, y0, x1, y1;

    // Initialize the graphics mode
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    // Set the clipping window (draw the clipping rectangle)
    rectangle(X_MIN, Y_MIN, X_MAX, Y_MAX);

    // Input the coordinates of the line
    cout << "Enter the coordinates of the line (x0, y0, x1, y1): ";
    cin >> x0 >> y0 >> x1 >> y1;

    // Call the Cohen-Sutherland algorithm to clip the line
    cohenSutherlandClip(x0, y0, x1, y1);

    // Wait for the user to close the graphics window
    getch();

    // Close the graphics mode
    closegraph();
```

```
    return 0;
}
```

2)

```
#include <graphics.h>
#include <iostream>
#include <cmath>

using namespace std;

// Clipping window boundaries
const int X_MIN = 100;
const int Y_MIN = 100;
const int X_MAX = 400;
const int Y_MAX = 400;

// Region codes for Cohen-Sutherland algorithm
const int INSIDE = 0;  // 0000
const int LEFT = 1;    // 0001
const int RIGHT = 2;   // 0010
const int BOTTOM = 4;  // 0100
const int TOP = 8;     // 1000

// Function prototypes
void ddaLine(int x0, int y0, int x1, int y1);
int computeRegionCode(int x, int y);
void cohenSutherlandClip(int x0, int y0, int x1, int y1);
```

```
// Function to compute the region code for a point (x, y)
int computeRegionCode(int x, int y) {
    int code = INSIDE;

    if (x < X_MIN)       // to the left of the window
        code |= LEFT;
    else if (x > X_MAX)   // to the right of the window
        code |= RIGHT;
    if (y < Y_MIN)       // below the window
        code |= BOTTOM;
    else if (y > Y_MAX)   // above the window
        code |= TOP;

    return code;
}

// Implementing the Cohen-Sutherland Line Clipping Algorithm
void cohenSutherlandClip(int x0, int y0, int x1, int y1) {
    int code0 = computeRegionCode(x0, y0);
    int code1 = computeRegionCode(x1, y1);
    bool accept = false;

    while (true) {
        if ((code0 == INSIDE) && (code1 == INSIDE)) {
            // Both endpoints are inside the rectangle
            accept = true;
            break;
        } else if ((code0 & code1) != 0) {
            // Logical AND is not 0: Both points are outside the rectangle in the same region
            break;
        } else {
```

```
// Calculate the intersection point
int codeOut;
int x, y;

// Determine which point to clip
if (code0 != INSIDE) {
   codeOut = code0;
} else {
   codeOut = code1;
}

// Now find the intersection point using the codeOut
if (codeOut & TOP) {  // point is above the window
   x = x0 + (x1 - x0) * (Y_MAX - y0) / (y1 - y0);
   y = Y_MAX;
} else if (codeOut & BOTTOM) {  // point is below the window
   x = x0 + (x1 - x0) * (Y_MIN - y0) / (y1 - y0);
   y = Y_MIN;
} else if (codeOut & RIGHT) {  // point is to the right of the window
   y = y0 + (y1 - y0) * (X_MAX - x0) / (x1 - x0);
   x = X_MAX;
} else if (codeOut & LEFT) {  // point is to the left of the window
   y = y0 + (y1 - y0) * (X_MIN - x0) / (x1 - x0);
   x = X_MIN;
}

// Update the point outside the rectangle
if (codeOut == code0) {
   x0 = x;
   y0 = y;
   code0 = computeRegionCode(x0, y0);
```

```
            } else {
                x1 = x;
                y1 = y;
                code1 = computeRegionCode(x1, y1);
            }
        }
    }

    // If the line is accepted, draw it using DDA
    if (accept) {
        ddaLine(x0, y0, x1, y1);
    } else {
        cout << "Line is outside the clipping window" << endl;
    }
}

// DDA Line Drawing Algorithm to draw a line
void ddaLine(int x0, int y0, int x1, int y1) {
    int dx = x1 - x0;
    int dy = y1 - y0;
    int steps;
    float xIncrement, yIncrement;

    // Calculate the number of steps required
    if (abs(dx) > abs(dy)) {
        steps = abs(dx);
    } else {
        steps = abs(dy);
    }

    // Calculate the increment values for x and y
```

```
    xIncrement = dx / (float)steps;
    yIncrement = dy / (float)steps;

    // Draw the line
    float x = x0;
    float y = y0;
    for (int i = 0; i <= steps; i++) {
        putpixel(round(x), round(y), WHITE);  // Plot the pixel
        x += xIncrement;
        y += yIncrement;
    }
}

int main() {
    int x0, y0, x1, y1;

    // Initialize the graphics mode
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    // Set the clipping window (draw the clipping rectangle)
    rectangle(X_MIN, Y_MIN, X_MAX, Y_MAX);

    // Input the coordinates of the line
    cout << "Enter the coordinates of the line (x0, y0, x1, y1): ";
    cin >> x0 >> y0 >> x1 >> y1;

    // Call the Cohen-Sutherland algorithm to clip the line
    cohenSutherlandClip(x0, y0, x1, y1);

    // Wait for the user to close the graphics window
```

```
    getch();

    // Close the graphics mode
    closegraph();

    return 0;
}
```