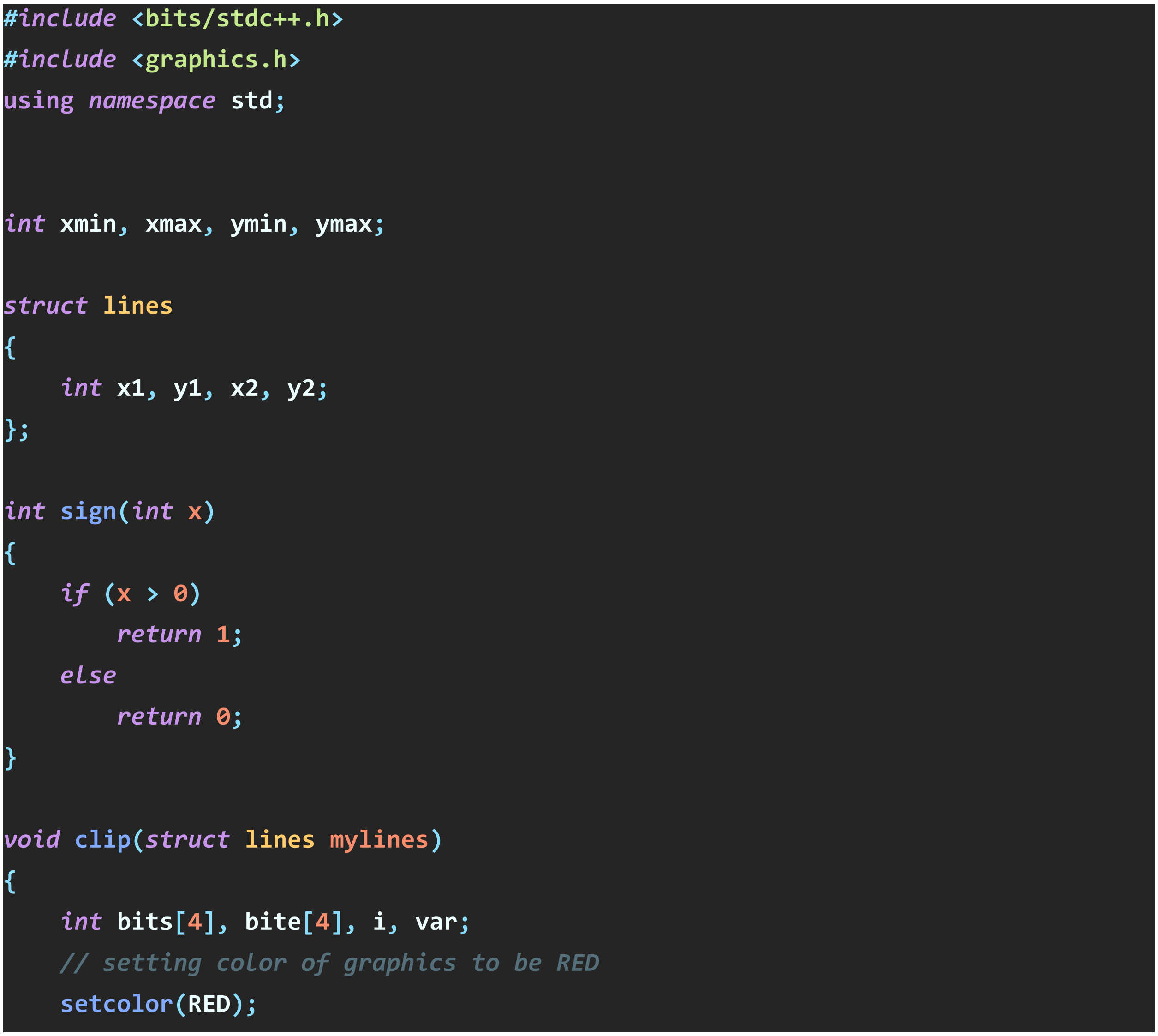
Assignment - 5

Name : Emmadi Sumith Kumar
Branch : CSE - B1
Subject : Computer Graphics
Roll No : UI20CS21

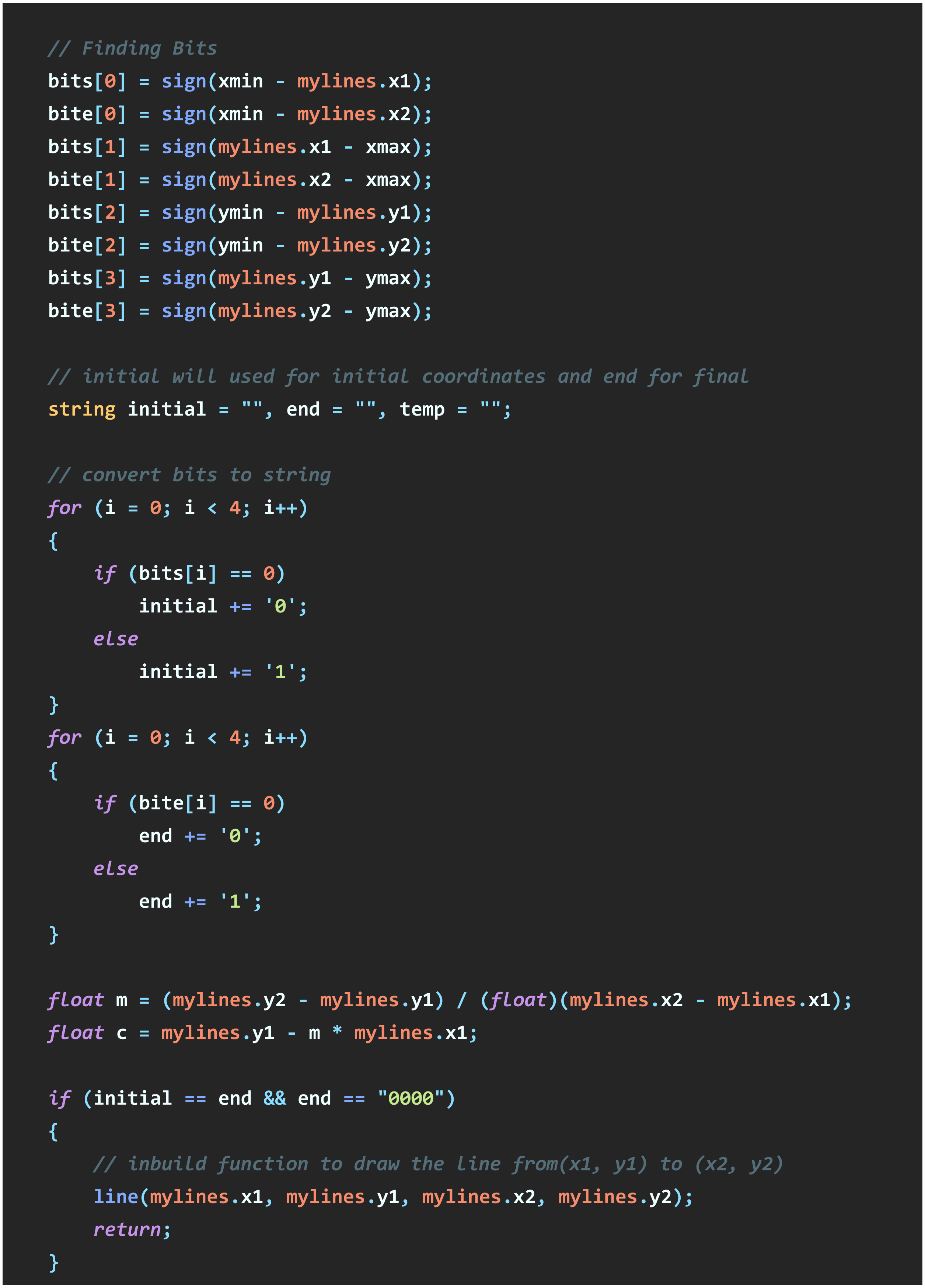
Write a program for line clipping using switch case

1. Cohen Sutherland Algorithm
2. Mid Point Subdivision

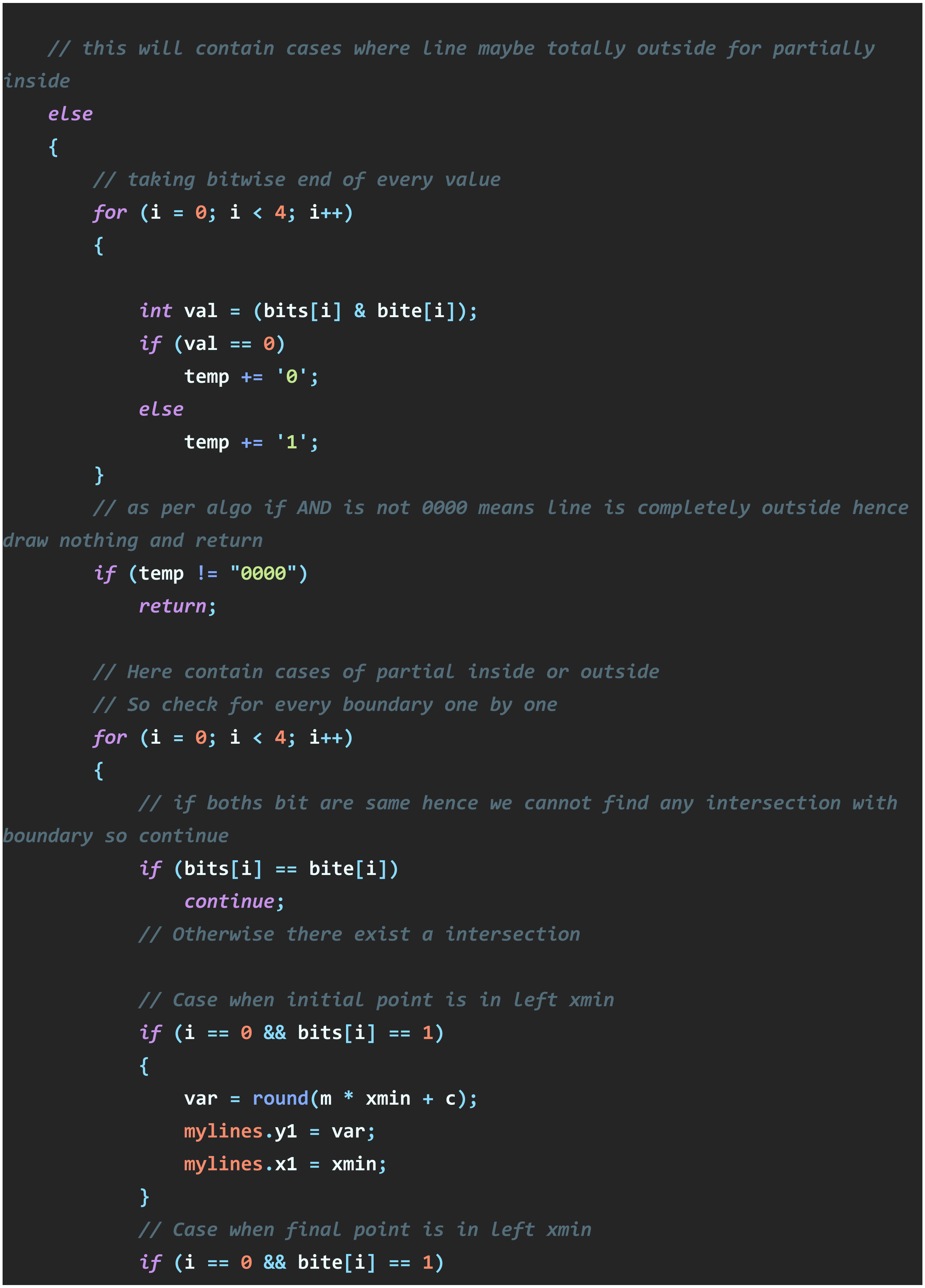
1. Cohen Sutherland Algorithm



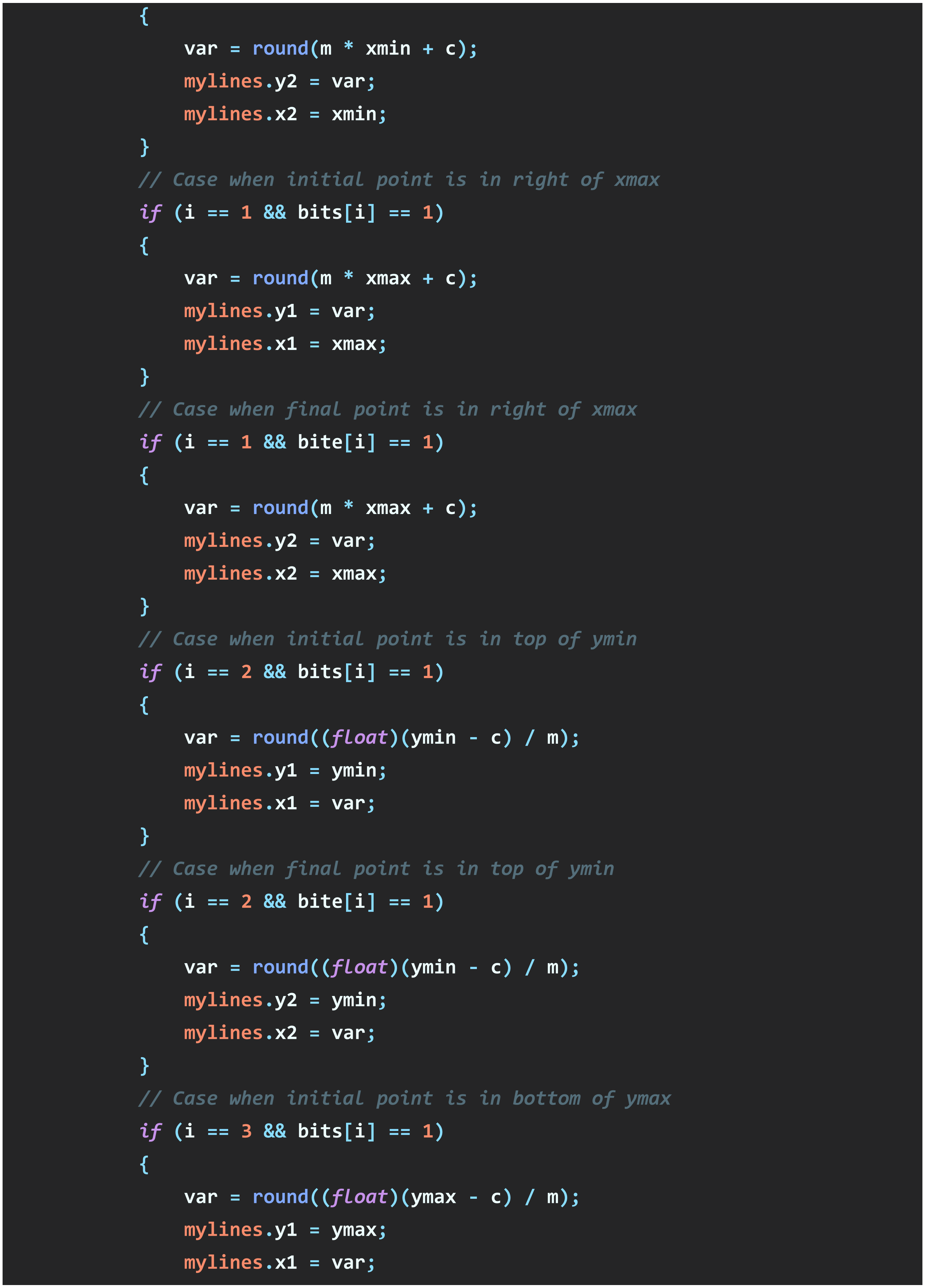
```
#include <bits/stdc++.h>
#include <graphics.h>
using namespace std;


int xmin, xmax, ymin, ymax;

struct lines
{
    int x1, y1, x2, y2;
};

int sign(int x)
{
    if (x > 0)
        return 1;
    else
        return 0;
}

void clip(struct lines mylines)
{
    int bits[4], bite[4], i, var;
    // setting color of graphics to be RED
    setcolor(RED);
```



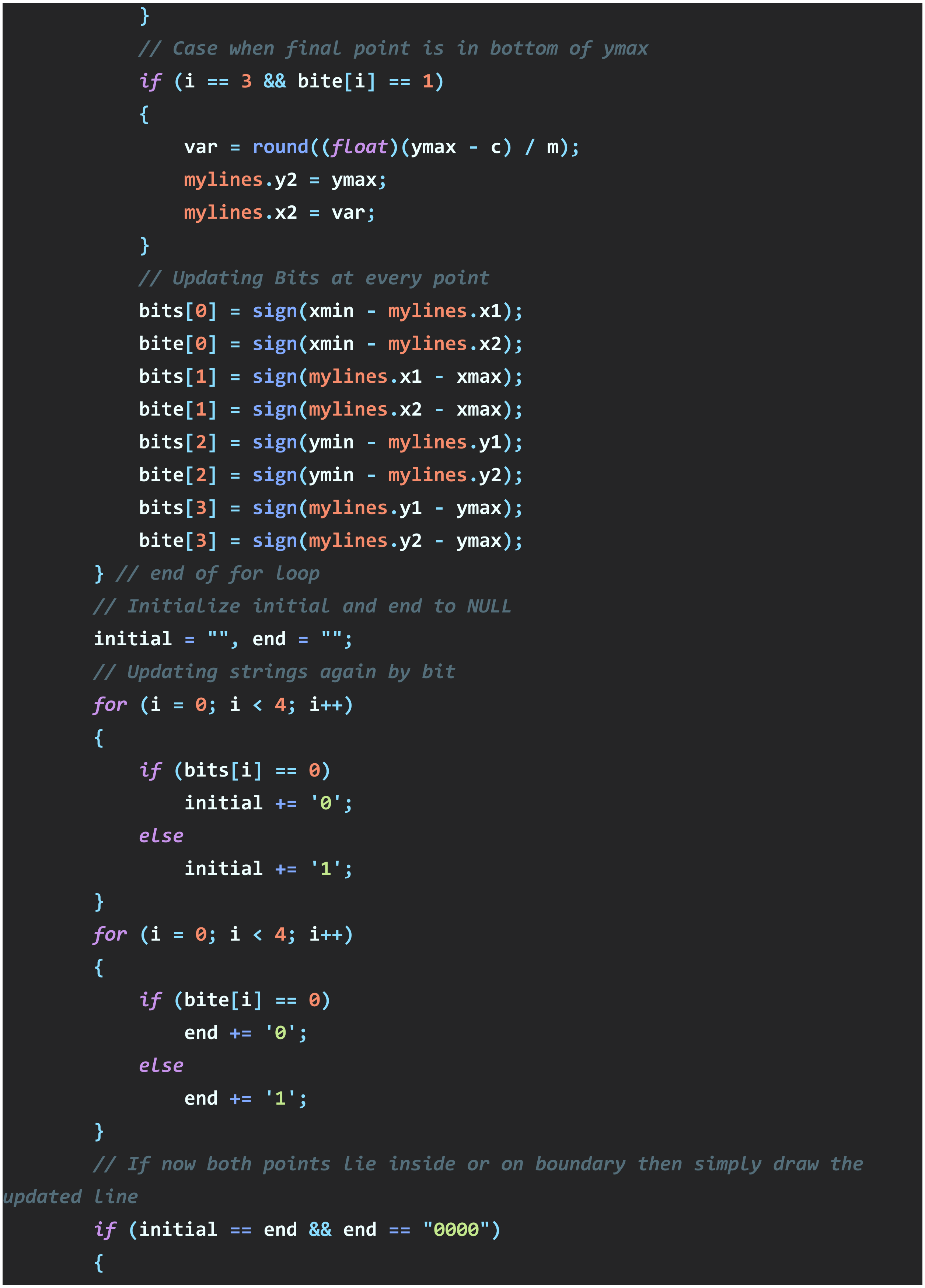
```
// Finding Bits
bits[0] = sign(xmin - mylines.x1);
bite[0] = sign(xmin - mylines.x2);
bits[1] = sign(mylines.x1 - xmax);
bite[1] = sign(mylines.x2 - xmax);
bits[2] = sign(ymin - mylines.y1);
bite[2] = sign(ymin - mylines.y2);
bits[3] = sign(mylines.y1 - ymax);
bite[3] = sign(mylines.y2 - ymax);

// initial will used for initial coordinates and end for final
string initial = "", end = "", temp = "";

// convert bits to string
for (i = 0; i < 4; i++)
{
    if (bits[i] == 0)
        initial += '0';
    else
        initial += '1';
}
for (i = 0; i < 4; i++)
{
    if (bite[i] == 0)
        end += '0';
    else
        end += '1';
}

float m = (mylines.y2 - mylines.y1) / (float)(mylines.x2 - mylines.x1);
float c = mylines.y1 - m * mylines.x1;

if (initial == end && end == "0000")
{
    // inbuild function to draw the line from(x1, y1) to (x2, y2)
    line(mylines.x1, mylines.y1, mylines.x2, mylines.y2);
    return;
}
```



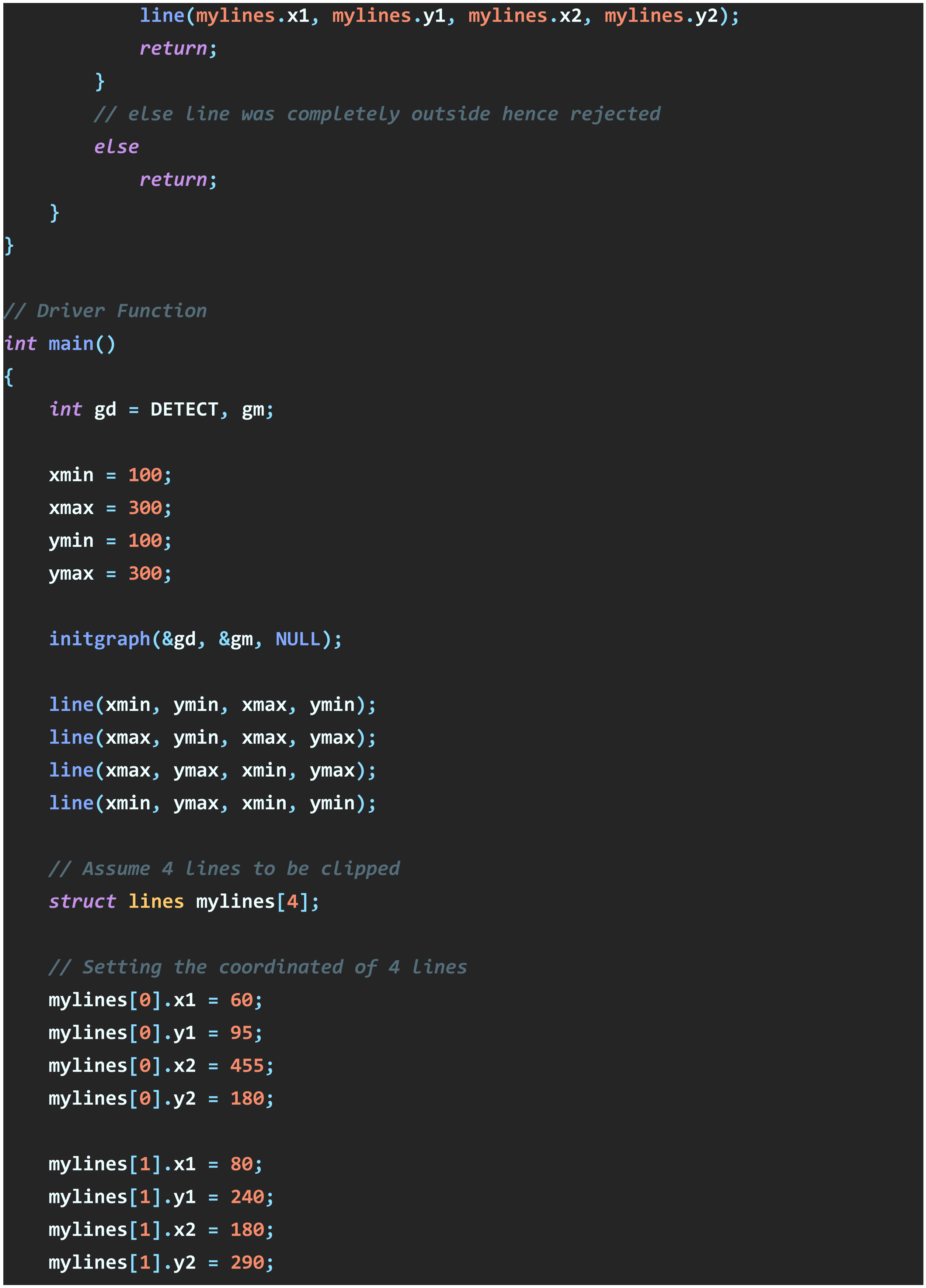
```
    // this will contain cases where line maybe totally outside for partially
inside
    else
    {
        // taking bitwise end of every value
        for (i = 0; i < 4; i++)
        {

            int val = (bits[i] & bite[i]);
            if (val == 0)
                temp += '0';
            else
                temp += '1';
        }
        // as per algo if AND is not 0000 means line is completely outside hence
draw nothing and return
        if (temp != "0000")
            return;

        // Here contain cases of partial inside or outside
        // So check for every boundary one by one
        for (i = 0; i < 4; i++)
        {
            // if boths bit are same hence we cannot find any intersection with
boundary so continue
            if (bits[i] == bite[i])
                continue;
            // Otherwise there exist a intersection

            // Case when initial point is in left xmin
            if (i == 0 && bits[i] == 1)
            {
                var = round(m * xmin + c);
                mylines.y1 = var;
                mylines.x1 = xmin;
            }
            // Case when final point is in left xmin
            if (i == 0 && bite[i] == 1)
```



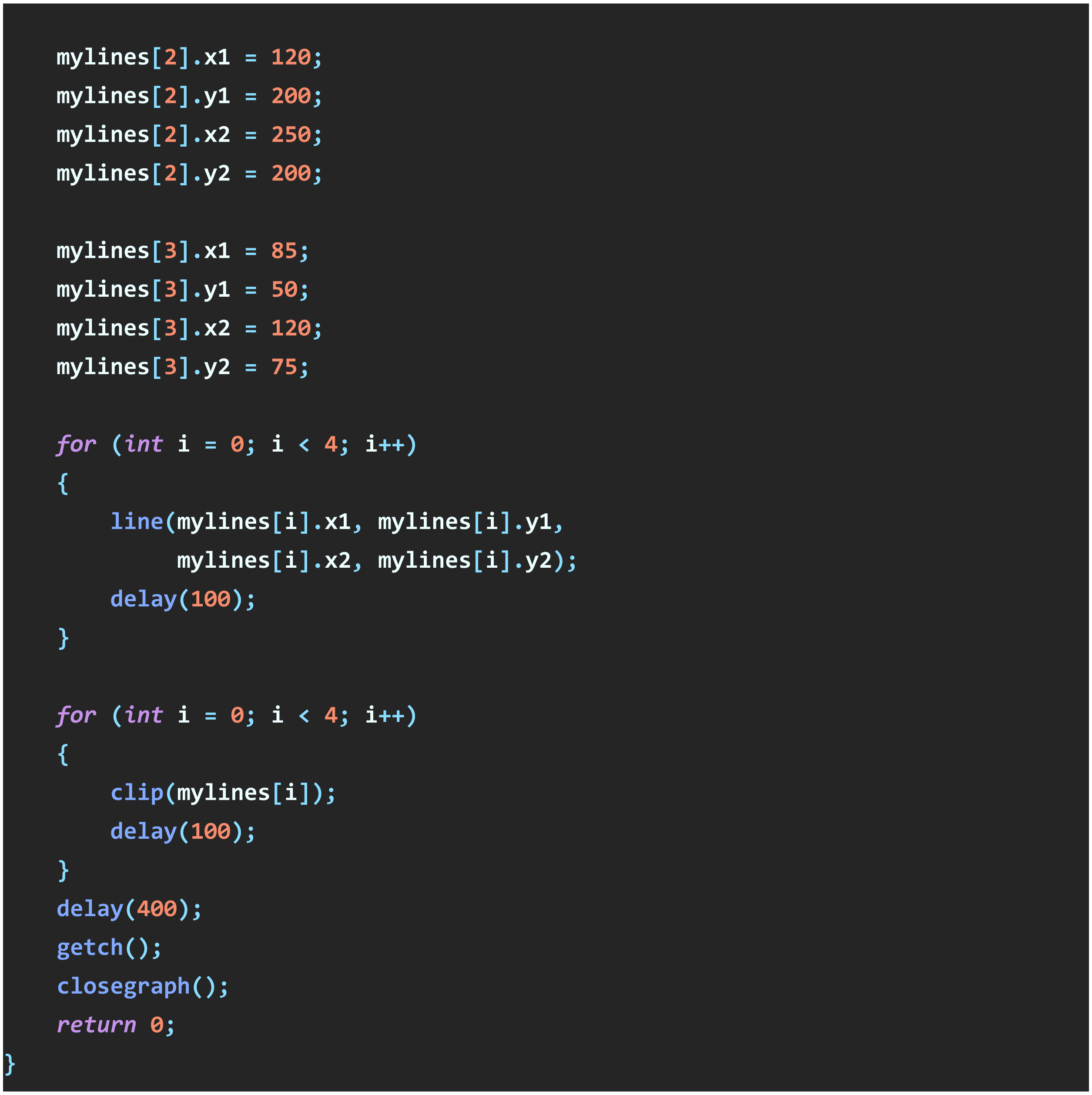
```
        {
            var = round(m * xmin + c);
            mylines.y2 = var;
            mylines.x2 = xmin;
        }
        // Case when initial point is in right of xmax
        if (i == 1 && bits[i] == 1)
        {
            var = round(m * xmax + c);
            mylines.y1 = var;
            mylines.x1 = xmax;
        }
        // Case when final point is in right of xmax
        if (i == 1 && bite[i] == 1)
        {
            var = round(m * xmax + c);
            mylines.y2 = var;
            mylines.x2 = xmax;
        }
        // Case when initial point is in top of ymin
        if (i == 2 && bits[i] == 1)
        {
            var = round((float)(ymin - c) / m);
            mylines.y1 = ymin;
            mylines.x1 = var;
        }
        // Case when final point is in top of ymin
        if (i == 2 && bite[i] == 1)
        {
            var = round((float)(ymin - c) / m);
            mylines.y2 = ymin;
            mylines.x2 = var;
        }
        // Case when initial point is in bottom of ymax
        if (i == 3 && bits[i] == 1)
        {
            var = round((float)(ymax - c) / m);
            mylines.y1 = ymax;
            mylines.x1 = var;
```



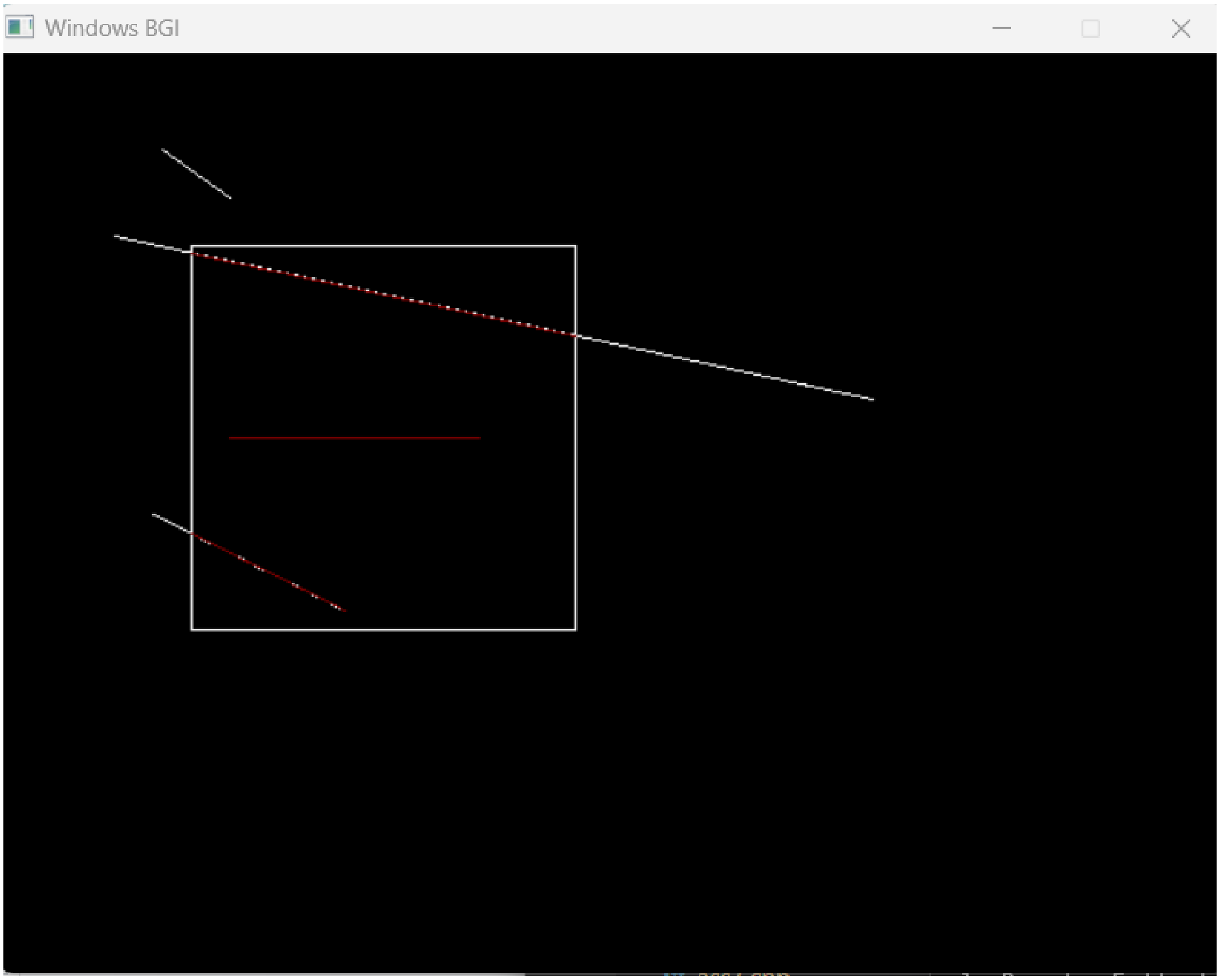
```
            }
            // Case when final point is in bottom of ymax
            if (i == 3 && bite[i] == 1)
            {
                var = round((float)(ymax - c) / m);
                mylines.y2 = ymax;
                mylines.x2 = var;
            }
            // Updating Bits at every point
            bits[0] = sign(xmin - mylines.x1);
            bite[0] = sign(xmin - mylines.x2);
            bits[1] = sign(mylines.x1 - xmax);
            bite[1] = sign(mylines.x2 - xmax);
            bits[2] = sign(ymin - mylines.y1);
            bite[2] = sign(ymin - mylines.y2);
            bits[3] = sign(mylines.y1 - ymax);
            bite[3] = sign(mylines.y2 - ymax);
        } // end of for loop
        // Initialize initial and end to NULL
        initial = "", end = "";
        // Updating strings again by bit
        for (i = 0; i < 4; i++)
        {
            if (bits[i] == 0)
                initial += '0';
            else
                initial += '1';
        }
        for (i = 0; i < 4; i++)
        {
            if (bite[i] == 0)
                end += '0';
            else
                end += '1';
        }
        // If now both points lie inside or on boundary then simply draw the
updated line
        if (initial == end && end == "0000")
        {
```



```
            line(mylines.x1, mylines.y1, mylines.x2, mylines.y2);
            return;
        }
        // else line was completely outside hence rejected
        else
            return;
    }
}

// Driver Function
int main()
{
    int gd = DETECT, gm;

    xmin = 100;
    xmax = 300;
    ymin = 100;
    ymax = 300;

    initgraph(&gd, &gm, NULL);

    line(xmin, ymin, xmax, ymin);
    line(xmax, ymin, xmax, ymax);
    line(xmax, ymax, xmin, ymax);
    line(xmin, ymax, xmin, ymin);

    // Assume 4 lines to be clipped
    struct lines mylines[4];

    // Setting the coordinated of 4 lines
    mylines[0].x1 = 60;
    mylines[0].y1 = 95;
    mylines[0].x2 = 455;
    mylines[0].y2 = 180;

    mylines[1].x1 = 80;
    mylines[1].y1 = 240;
    mylines[1].x2 = 180;
    mylines[1].y2 = 290;
```

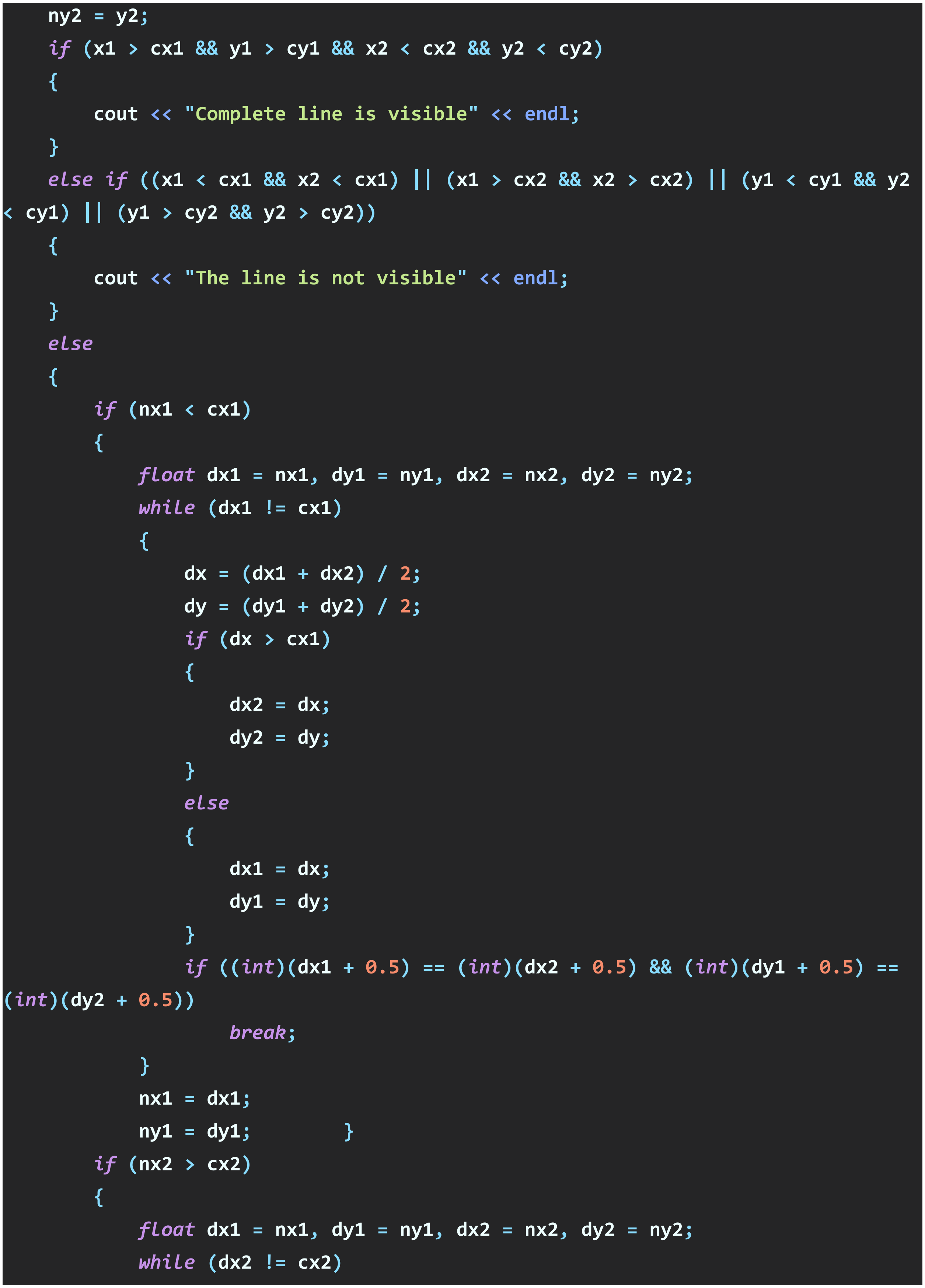


```
    mylines[2].x1 = 120;
    mylines[2].y1 = 200;
    mylines[2].x2 = 250;
    mylines[2].y2 = 200;

    mylines[3].x1 = 85;
    mylines[3].y1 = 50;
    mylines[3].x2 = 120;
    mylines[3].y2 = 75;

    for (int i = 0; i < 4; i++)
    {
        line(mylines[i].x1, mylines[i].y1,
             mylines[i].x2, mylines[i].y2);
        delay(100);
    }

    for (int i = 0; i < 4; i++)
    {
        clip(mylines[i]);
        delay(100);
    }
    delay(400);
    getch();
    closegraph();
    return 0;
}
```


Windows BGI

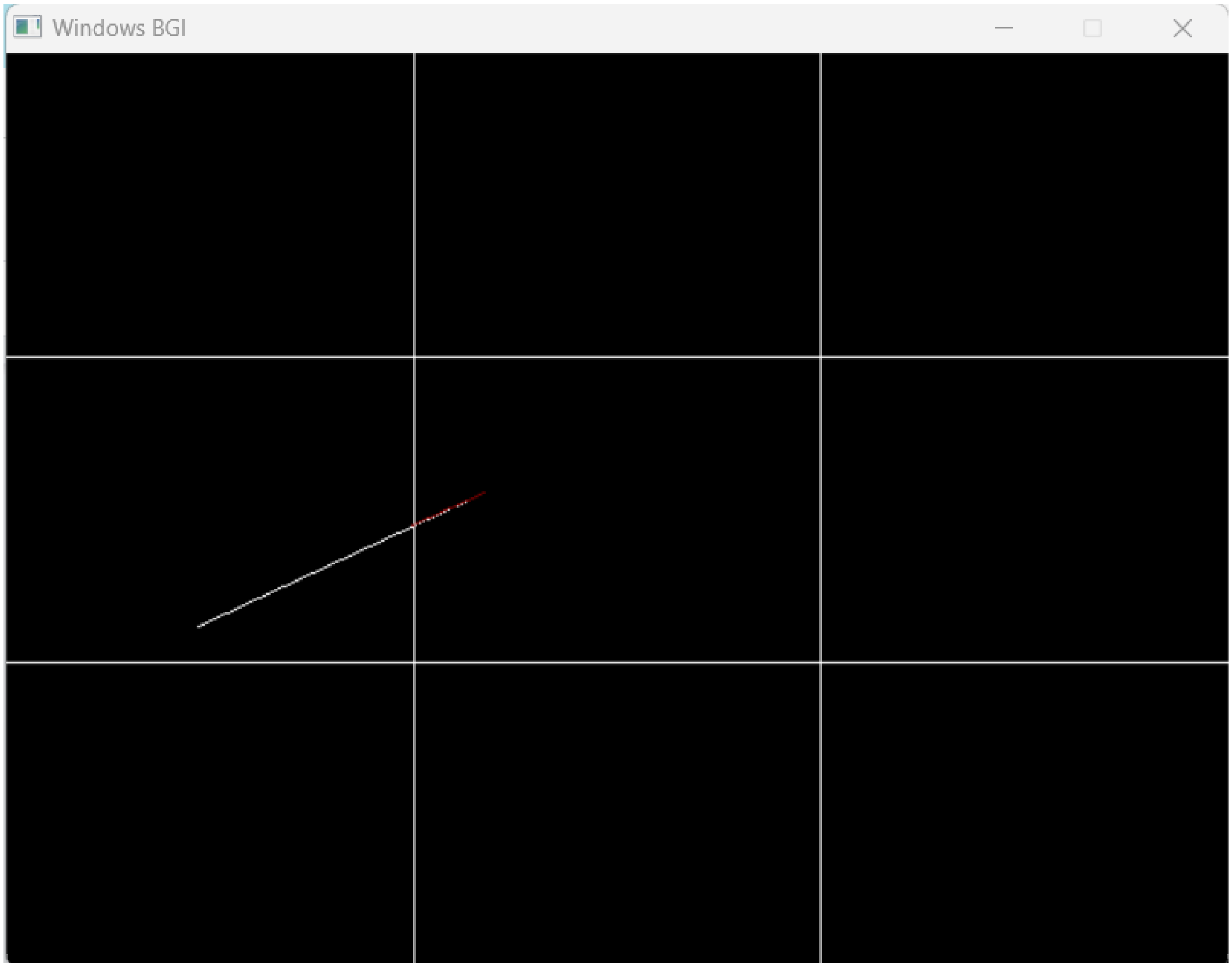
2. Mid Point Subdivision

```
#include <graphics.h>
#include <iostream>
#include <math.h>
#include <conio.h>
using namespace std;
int main()
{
    int gd = DETECT, gm;
    float x1, y1, y2, x2;
    initgraph(&gd, &gm, NULL);
    float X = getmaxx();
    float Y = getmaxy();
    float cx1 = X / 3;
    float cy1 = Y / 3;
    line(cx1, 0, cx1, Y);
    line(0, cy1, X, cy1);
    float cx2 = (2 * X) / 3;
    float cy2 = (2 * Y) / 3;
    line(cx2, 0, cx2, Y);
    line(0, cy2, X, cy2);
    cout << "X min: 0 X max: " << X << "Y min:0 Y max: " << Y << endl;
    cout << "Display Window size: X min: " << cx1 << " Y min: " << cy1 << " X max: " << cx2 << " Y max: " << cy2 << endl;
    cout << "Enter values of x1 y1 = ";
    cin >> x1 >> y1;
    cout << "Enter values of x2 y2 = ";
    cin >> x2 >> y2;
    line(x1, y1, x2, y2);
    float dx = (x2 - x1);
    float dy = (y2 - y1);
    float m = dy / dx;
    int ch;
    float nx1, ny1, nx2, ny2;
    nx1 = x1;
    ny1 = y1;
    nx2 = x2;
```



```
    ny2 = y2;
    if (x1 > cx1 && y1 > cy1 && x2 < cx2 && y2 < cy2)
    {
        cout << "Complete line is visible" << endl;
    }
    else if ((x1 < cx1 && x2 < cx1) || (x1 > cx2 && x2 > cx2) || (y1 < cy1 && y2
< cy1) || (y1 > cy2 && y2 > cy2))
    {
        cout << "The line is not visible" << endl;
    }
    else
    {
        if (nx1 < cx1)
        {
            float dx1 = nx1, dy1 = ny1, dx2 = nx2, dy2 = ny2;
            while (dx1 != cx1)
            {
                dx = (dx1 + dx2) / 2;
                dy = (dy1 + dy2) / 2;
                if (dx > cx1)
                {
                    dx2 = dx;
                    dy2 = dy;
                }
                else
                {
                    dx1 = dx;
                    dy1 = dy;
                }
                if ((int)(dx1 + 0.5) == (int)(dx2 + 0.5) && (int)(dy1 + 0.5) ==
(int)(dy2 + 0.5))
                    break;
            }
            nx1 = dx1;
            ny1 = dy1;        }
        if (nx2 > cx2)
        {
            float dx1 = nx1, dy1 = ny1, dx2 = nx2, dy2 = ny2;
            while (dx2 != cx2)
            {
                dx = (dx1 + dx2) / 2;
                dy = (dy1 + dy2) / 2;
                if (dx > cx2)
                {
                    dx2 = dx;
                    dy2 = dy;
                }
                else
                {
                    dx1 = dx;
                    dy1 = dy;
                }
                if ((int)(dx1 + 0.5) == (int)(dx2 + 0.5) && (int)(dy1 + 0.5) ==
(int)(dy2 + 0.5))
                    break;
            }
            nx2 = dx2;
            ny2 = dy2;
        }
        if (ny1 < cy1)
        {
            float dx1 = nx1, dy1 = ny1, dx2 = nx2, dy2 = ny2;
            while (dy1 != cy1)
            {
                dx = (dx1 + dx2) / 2;
                dy = (dy1 + dy2) / 2;
                if (dy > cy1)
                {
                    dx2 = dx;
                    dy2 = dy;
                }
                else
                {
                    dx1 = dx;
                    dy1 = dy;
                }
                if ((int)(dx1 + 0.5) == (int)(dx2 + 0.5) && (int)(dy1 + 0.5) ==
(int)(dy2 + 0.5))
```

```
                    break;
            }
            nx1 = dx1;
            ny2 = dy1;
        }
        if (ny2 > cy2)
        {
            float dx1 = nx1, dy1 = ny1, dx2 = nx2, dy2 = ny2;
            while (dy2 != cy2)
            {
                dx = (dx1 + dx2) / 2;
                dy = (dy1 + dy2) / 2;
                if (dy > cy2)
                {
                    dx2 = dx;
                    dy2 = dy;
                }
                else
                {
                    dx1 = dx;
                    dy1 = dy;
                }
                if ((int)(dx1 + 0.5) == (int)(dx2 + 0.5) && (int)(dy1 + 0.5) == (int)(dy2 + 0.5))
                    break;
            }
            nx2 = dx2;
            ny2 = dy2;
            // cout<<"x2: "<<nx2<<" y2: "<<ny2<<endl;
        }
    }
    setcolor(RED);
    line(nx1, ny1, nx2, ny2);
    getch();
    closegraph();
    return 0;
}
```


Windows BGI