ASSIGNMENT NO. 2

SNEHA KADAM
C21128

```
// C++ program to implement Cohen Sutherland algorithm
// for line clipping.
// including libraries
#include <bits/stdc++.h>
#include <graphics.h>
using namespace std;

// Global Variables
int xmin, xmax, ymin, ymax;

// Lines where co-ordinates are (x1, y1) and (x2, y2)
struct lines {
	int x1, y1, x2, y2;
};

// This will return the sign required.
int sign(int x)
{
	if (x > 0)
		return 1;
	else
		return 0;
}

// CohenSutherLand LineClipping Algorithm As Described in theory.
// This will clip the lines as per window boundaries.
void clip(struct lines mylines)
{
	// arrays will store bits
	// Here bits implies initial Point whereas byte implies end points
	int bits[4], byte[4], i, var;
	// setting color of graphics to be RED
	setcolor(RED);

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

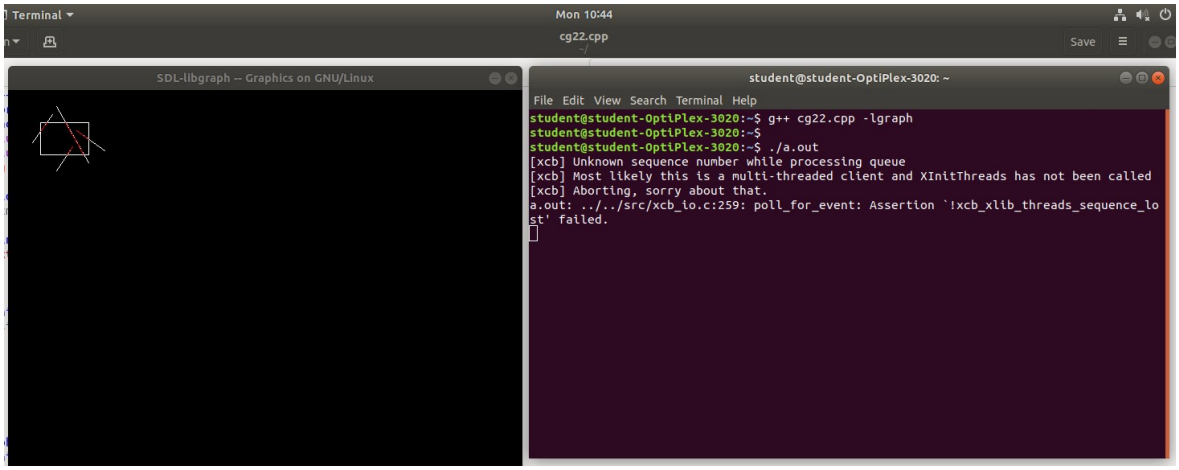
```
            if (bits[i] == 0)
                initial += '0';
            else
                initial += '1';
        }
        for (i = 0; i < 4; i++) {
            if (byte[i] == 0)
                end += '0';
            else
                end += '1';
        }

        // finding slope of line y=mx+c as (y-y1)=m(x-x1)+c
        // where m is slope m=dy/dx;

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

        // if both points are inside the Accept the line and draw
        if (initial == end && end == "0000") {
            // inbuilt 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] & byte[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 both bit are same hence we cannot find any intersection with boundary so
continue
                if (bits[i] == byte[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 && byte[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 && byte[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 && byte[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 && byte[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);
byte[0] = sign(xmin - mylines.x2);
bits[1] = sign(mylines.x1 - xmax);
byte[1] = sign(mylines.x2 - xmax);
bits[2] = sign(ymin - mylines.y1);
```

```
                byte[2] = sign(ymin - mylines.y2);
                bits[3] = sign(mylines.y1 - ymax);
                byte[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 (byte[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;

        // Setting values of Clipping window
        xmin = 40;
        xmax = 100;
        ymin = 40;
        ymax = 80;

        // initialize the graph
        initgraph(&gd, &gm, NULL);

        // Drawing Window using Lines
        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 = 30;
        mylines[0].y1 = 65;
        mylines[0].x2 = 55;
        mylines[0].y2 = 30;

        mylines[1].x1 = 60;
        mylines[1].y1 = 20;
        mylines[1].x2 = 100;
        mylines[1].y2 = 90;

        mylines[2].x1 = 60;
        mylines[2].y1 = 100;
        mylines[2].x2 = 80;
        mylines[2].y2 = 70;

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

        // Drawing Initial Lines without clipping
        for (int i = 0; i < 4; i++) {
                line(mylines[i].x1, mylines[i].y1,
                        mylines[i].x2, mylines[i].y2);
                delay(1000);
        }

        // Drawing clipped Line
        for (int i = 0; i < 4; i++) {
                // Calling clip() which in term clip the line as per window and draw it
                clip(mylines[i]);
                delay(1000);
        }
        delay(4000);
        getch();
        // For Closing the graph.
        closegraph();
        return 0;
}
```

//OUTPUT


Terminal
Mon 10:44
cg22.cpp
Save
SDL-libgraph -- Graphics on GNU/Linux
student@student-OptiPlex-3020: ~
File Edit View Search Terminal Help
student@student-OptiPlex-3020:~$ g++ cg22.cpp -lgraph
student@student-OptiPlex-3020:~$
student@student-OptiPlex-3020:~$ ./a.out
[xcb] Unknown sequence number while processing queue
[xcb] Most likely this is a multi-threaded client and XInitThreads has not been called
[xcb] Aborting, sorry about that.
a.out: ../../src/xcb_io.c:259: poll_for_event: Assertion `!xcb_xlib_threads_sequence_lo
st' failed.