Program 5

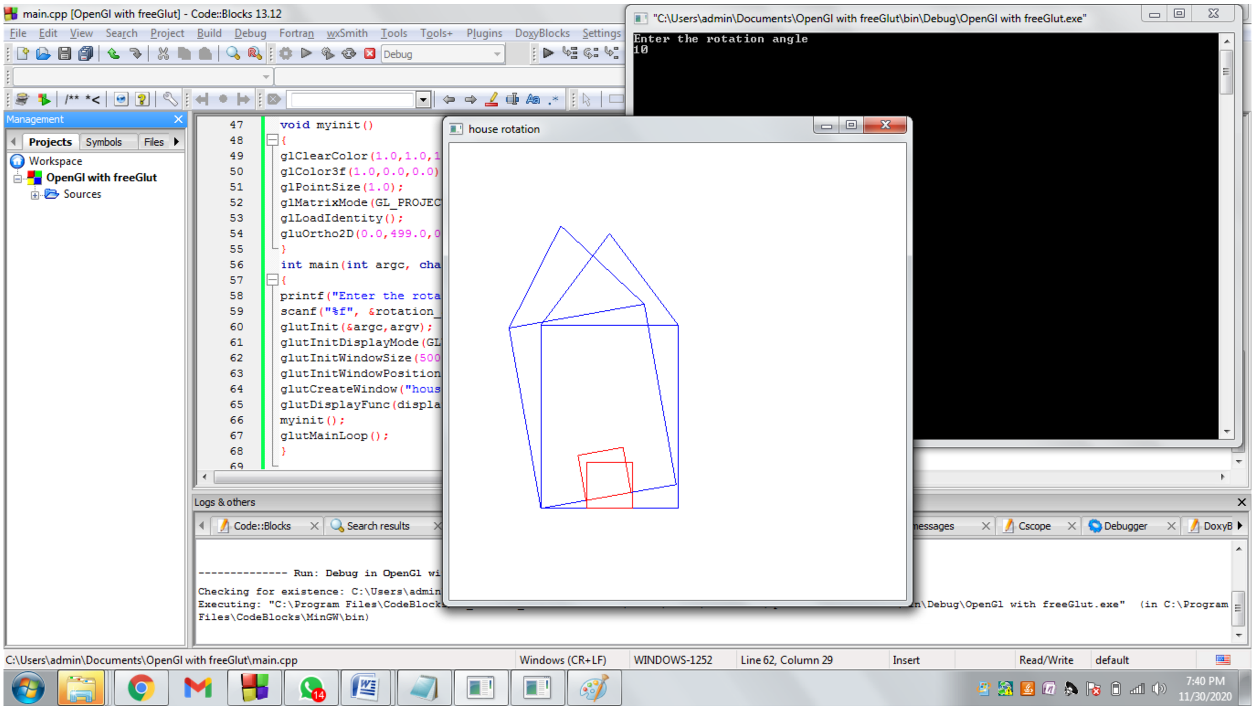
5. Write a program to create a house like figure and perform the following operations.

I). Rotate it about a given fixed point using OpenGL transformation functions.

```
#define BLACK 0

#include <stdio.h>

#include <math.h>

#include <GL/glut.h>

GLfloat house[3][9]={{100.0,100.0,175.0,250.0,250.0,150.0,150.0,200.0,200.0},

{100.0,300.0,400.0,300.0,100.0,100.0,150.0,150.0,100.0},

{1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0}

};

GLfloat arbitrary_x=100.0;

GLfloat arbitrary_y=100.0;

GLfloat rotation_angle;

void drawhouse()

{

glColor3f(0.0, 0.0, 1.0);

glBegin(GL_LINE_LOOP);

 glVertex2f(house[0][0],house[1][0]);

 glVertex2f(house[0][1],house[1][1]);

 glVertex2f(house[0][3],house[1][3]);

 glVertex2f(house[0][4],house[1][4]);

 glEnd();

glColor3f(1.0,0.0,0.0);

 glBegin(GL_LINE_LOOP);
 glVertex2f(house[0][5],house[1][5]);
 glVertex2f(house[0][6],house[1][6]);
 glVertex2f(house[0][7],house[1][7]);
 glVertex2f(house[0][8],house[1][8]);
 glEnd();
glColor3f(0.0, 0.0, 1.0);
 glBegin(GL_LINE_LOOP);
 glVertex2f(house[0][1],house[1][1]);
 glVertex2f(house[0][2],house[1][2]);
 glVertex2f(house[0][3],house[1][3]);
 glEnd();
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
drawhouse();
glTranslatef(arbitrary_x,arbitrary_y,0.0);
glRotatef(rotation_angle,0.0,0.0,1.0);
glTranslatef(-(arbitrary_x),-(arbitrary_y),0.0);
drawhouse();
glFlush();
}
void myinit()
```

```
{

glClearColor(1.0,1.0,1.0,1.0);

glColor3f(1.0,0.0,0.0);

glPointSize(1.0);

glMatrixMode(GL_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,499.0,0.0,499.0);

}

int main(int argc, char** argv)

{

printf("Enter the rotation angle\n");

scanf("%f", &rotation_angle);

glutInit(&argc,argv);

glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

glutInitWindowSize(500,500);

glutInitWindowPosition(0,0);

glutCreateWindow("house rotation");

glutDisplayFunc(display);

myinit();

glutMainLoop();

}
```

OUTPUT:



main.cpp [OpenGl with freeGlut] - Code::Blocks 13.12

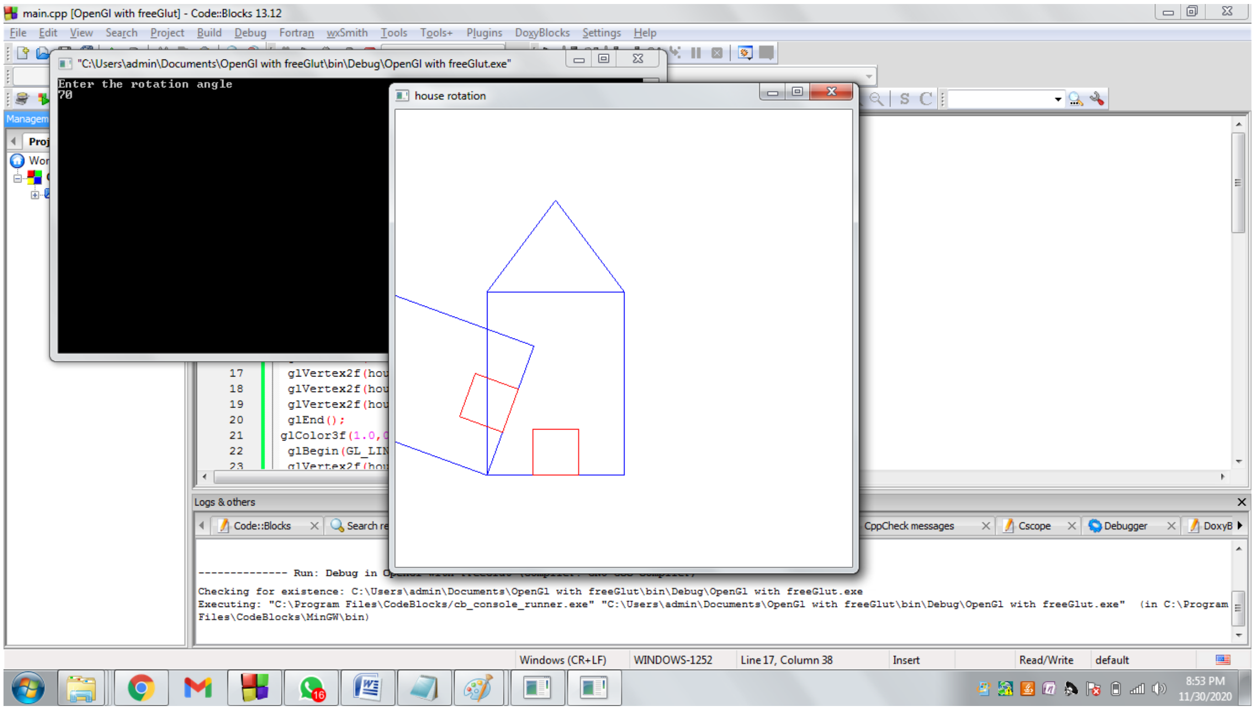
"C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe"

```
Enter the rotation angle
10
```

```
void myinit()
{
glClearColor(1.0,1.0,1
glColor3f(1.0,0.0,0.0)
glPointSize(1.0);
glMatrixMode(GL_PROJEC
glLoadIdentity();
gluOrtho2D(0.0,499.0,0
}
int main(int argc, cha
{
printf("Enter the rota
scanf("%f", &rotation_
glutInit(&argc,argv);
glutInitDisplayMode(GL
glutInitWindowSize(500
glutInitWindowPosition
glutCreateWindow("hous
glutDisplayFunc(displa
myinit();
glutMainLoop();
}
```

```
-------------- Run: Debug in OpenGl wi
Checking for existence: C:\Users\admin
Executing: "C:\Program Files\CodeBlock                                   n\Debug\OpenGl with freeGlut.exe"  (in C:\Program
Files\CodeBlocks\MinGW\bin)
```

C:\Users\admin\Documents\OpenGl with freeGlut\main.cpp | Windows (CR+LF) | WINDOWS-1252 | Line 62, Column 29 | Insert | Read/Write | default



main.cpp [OpenGl with freeGlut] - Code::Blocks 13.12

"C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe"

```
Enter the rotation angle
70
```

```
 glVertex2f(hou
 glVertex2f(hou
 glVertex2f(hou
 glEnd();
glColor3f(1.0,0
 glBegin(GL_LIN
 glVertex2f(hou
```

```
-------------- Run: Debug in O
Checking for existence: C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe"  (in C:\Program
Files\CodeBlocks\MinGW\bin)
```

Windows (CR+LF) | WINDOWS-1252 | Line 17, Column 38 | Insert | Read/Write | default

ii). Reflect it about an axis y=mx+c using OpenGL transformation functions.

```
#include<gl/glut.h>
#include <math.h>
#include<stdlib.h>
#include<stdio.h>
//RIGHT CLICK TO SHOW REFLECTED HOUSE
float house[11][2] = { { 100,200 },{ 200,250 },{ 300,200 },{ 100,200 },{ 100,100 },{
175,100 },{ 175,150 },{ 225,150 },{ 225,100 },{ 300,100 },{ 300,200 } };
int angle;
float m, c, theta;
void display()
{
glClearColor(1, 1, 1, 0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-450, 450, -450, 450);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
//NORMAL HOUSE
glColor3f(1, 0, 0);
glBegin(GL_LINE_LOOP);
for (int i = 0; i < 11; i++)
glVertex2fv(house[i]);
```

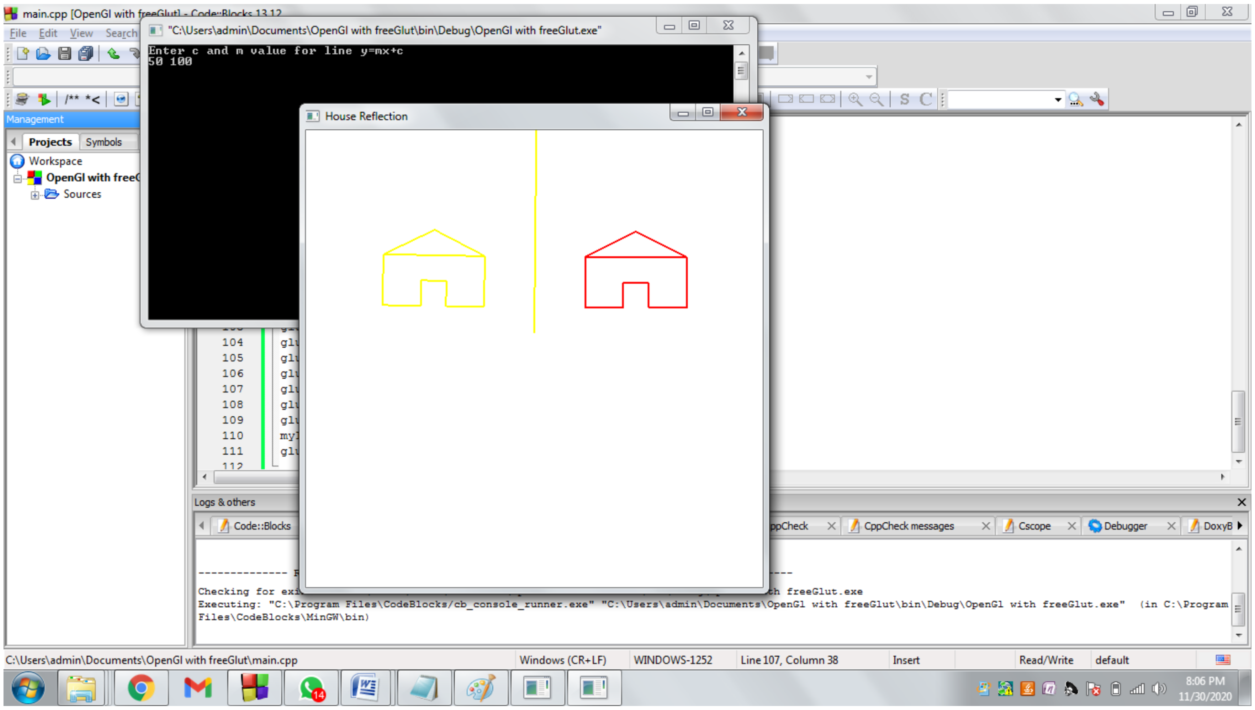
```
glEnd();

glFlush();

//ROTATED HOUSE

glPushMatrix();

glTranslatef(100, 100, 0);

glRotatef(angle, 0, 0, 1);

glTranslatef(-100, -100, 0);

glColor3f(1, 1, 0);

glBegin(GL_LINE_LOOP);

for (int i = 0; i < 11; i++)

glVertex2fv(house[i]);

glEnd();

glPopMatrix();

glFlush();

}

void display2()

{

glClearColor(1, 1, 1, 0);

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glMatrixMode(GL_PROJECTION);

glLoadIdentity();

gluOrtho2D(-450, 450, -450, 450);

glMatrixMode(GL_MODELVIEW);

glLoadIdentity();

//normal house
```

```
glColor3f(1, 0, 0);
glBegin(GL_LINE_LOOP);
for (int i = 0; i < 11; i++)
glVertex2fv(house[i]);
glEnd();
glFlush();
// line
float x1 = 0, x2 = 500;
float y1 = m * x1 + c;
float y2 = m * x2 + c;
glColor3f(1, 1, 0);
glBegin(GL_LINES);
glVertex2f(x1, y1);
glVertex2f(x2, y2);
glEnd();
glFlush();
//Reflected
glPushMatrix();
glTranslatef(0, c, 0);
theta = atan(m);
theta = theta * 180 / 3.14;
glRotatef(theta, 0, 0, 1);
glScalef(1, -1, 1);
glRotatef(-theta, 0, 0, 1);
glTranslatef(0, -c, 0);
```

```
glBegin(GL_LINE_LOOP);

for (int i = 0; i < 11; i++)

glVertex2fv(house[i]);

glEnd();

glPopMatrix();

glFlush();

}

void myInit() {

glClearColor(1.0, 1.0, 1.0, 1.0);

glColor3f(1.0, 0.0, 0.0);

glLineWidth(2.0);

glMatrixMode(GL_PROJECTION);

glLoadIdentity();

gluOrtho2D(-450, 450, -450, 450);

}

void mouse(int btn, int state, int x, int y) {

if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {

display();

}

else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {

display2();

}

}

int main(int argc, char** argv)

{
```

```
printf("Enter c and m value for line y=mx+c\n");

scanf("%f %f", &c, &m);

glutInit(&argc, argv);

glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("House Reflection");

glutDisplayFunc(display);

glutMouseFunc(mouse);

myInit();

glutMainLoop(); }
```

OUTPUT:



```
Enter c and m value for line y=mx+c
50 100
```

```
-------------- F                                              ----
Checking for exi                                              th freeGlut.exe
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe"  (in C:\Program
Files\CodeBlocks\MinGW\bin)
```

Program 6

6. Write a program to implement the Cohen-Sutherland line clipping algorithm. Make provision to specify the input for multiple lines, window for clipping and viewport for displaying the clipped image.

```
#include <stdio.h>
#include <GL/glut.h>
#define outcode int
double xmin=50,ymin=50, xmax=100,ymax=100; // Window boundaries
double xvmin=200,yvmin=200,xvmax=300,yvmax=300; // Viewport boundaries
//bit codes for the right, left, top, & bottom
const int RIGHT = 8;
const int LEFT = 2;
const int TOP = 4;
const int BOTTOM = 1;
//used to compute bit codes of a point
outcode ComputeOutCode (double x, double y);
//Cohen-Sutherland clipping algorithm clips a line from
//P0 = (x0, y0) to P1 = (x1, y1) against a rectangle with
//diagonal from (xmin, ymin) to (xmax, ymax).
void CohenSutherlandLineClipAndDraw (double x0, double y0,double x1, double y1)
{
//Outcodes for P0, P1, and whatever point lies outside the clip rectangle
outcode outcode0, outcode1, outcodeOut;
bool accept = false, done = false;
//compute outcodes
outcode0 = ComputeOutCode (x0, y0);
```

```
outcode1 = ComputeOutCode (x1, y1);
do{
if (!(outcode0 | outcode1)) //logical or is 0 Trivially accept & exit
{
accept = true;
done = true;
}
else if (outcode0 & outcode1) //logical and is not 0. Trivially reject and exit
done = true;
else
{
   //failed both tests, so calculate the line segment to clip
//from an outside point to an intersection with clip edge
double x, y;
//At least one endpoint is outside the clip rectangle; pick it.
outcodeOut = outcode0? outcode0: outcode1;
//Now find the intersection point;
//use formulas y = y0 + slope * (x - x0), x = x0 + (1/slope)* (y - y0)
if (outcodeOut & TOP) //point is above the clip rectangle
{
x = x0 + (x1 - x0) * (ymax - y0)/(y1 - y0);
y = ymax;
}
else if (outcodeOut & BOTTOM) //point is below the clip rectangle
{
```

```
x = x0 + (x1 - x0) * (ymin - y0)/(y1 - y0);
y = ymin;
}
else if (outcodeOut & RIGHT) //point is to the right of clip rectangle
{
y = y0 + (y1 - y0) * (xmax - x0)/(x1 - x0);
x = xmax;
}
else //point is to the left of clip rectangle
{
y = y0 + (y1 - y0) * (xmin - x0)/(x1 - x0);
x = xmin;
}
//Now we move outside point to intersection point to clip
//and get ready for next pass.
if (outcodeOut == outcode0)
{
x0 = x;
y0 = y;
outcode0 = ComputeOutCode (x0, y0);
}
else
{
x1 = x;
y1 = y;
```

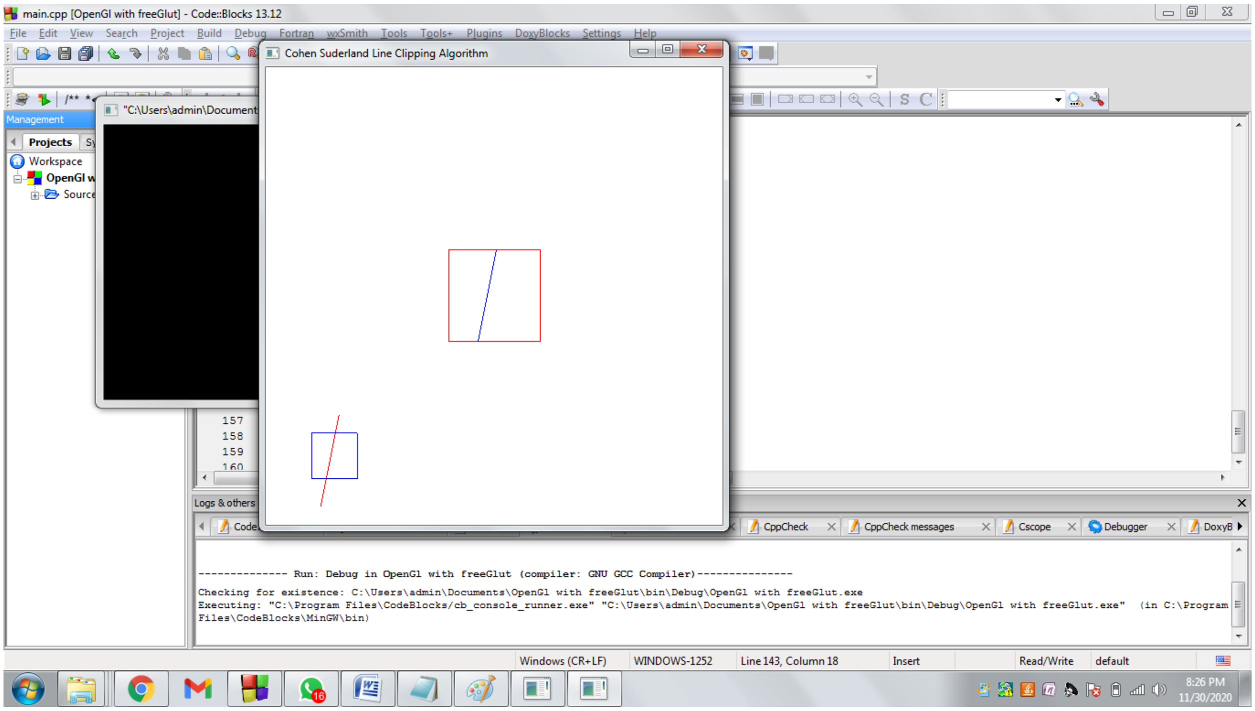
```
outcode1 = ComputeOutCode (x1, y1);

}

}

}while (!done);

if (accept)

{ // Window to viewport mappings

double sx=(xvmax-xvmin)/(xmax-xmin); // Scale parameters

double sy=(yvmax-yvmin)/(ymax-ymin);

double vx0=xvmin+(x0-xmin)*sx;

double vy0=yvmin+(y0-ymin)*sy;

double vx1=xvmin+(x1-xmin)*sx;

double vy1=yvmin+(y1-ymin)*sy;

//draw a red colored viewport

glColor3f(1.0, 0.0, 0.0);

glBegin(GL_LINE_LOOP);

glVertex2f(xvmin, yvmin);

glVertex2f(xvmax, yvmin);

glVertex2f(xvmax, yvmax);

glVertex2f(xvmin, yvmax);

glEnd();

glColor3f(0.0,0.0,1.0); // draw blue colored clipped line

glBegin(GL_LINES);

glVertex2d (vx0, vy0);

glVertex2d (vx1, vy1);

glEnd();
```

```
}

}

//Compute the bit code for a point (x, y) using the clip rectangle

//bounded diagonally by (xmin, ymin), and (xmax, ymax)

outcode ComputeOutCode (double x, double y)

{

outcode code = 0;

if (y > ymax) //above the clip window

code |= TOP;

else if (y < ymin) //below the clip window

code |= BOTTOM;

if (x > xmax) //to the right of clip window

code |= RIGHT;

else if (x < xmin) //to the left of clip window

code |= LEFT;

return code;

}

void display()

{

double x0=60,y0=20,x1=80,y1=120;

glClear(GL_COLOR_BUFFER_BIT);

//draw the line with red color

glColor3f(1.0,0.0,0.0);

//bres(120,20,340,250);

glBegin(GL_LINES);
```

```
glVertex2d (x0, y0);
glVertex2d (x1, y1);
glEnd();
//draw a blue colored window
glColor3f(0.0, 0.0, 1.0);
glBegin(GL_LINE_LOOP);
 glVertex2f(xmin, ymin);
 glVertex2f(xmax, ymin);
 glVertex2f(xmax, ymax);
 glVertex2f(xmin, ymax);
glEnd();
CohenSutherlandLineClipAndDraw(x0,y0,x1,y1);
glFlush();
}
void myinit()
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(1.0,0.0,0.0);
glPointSize(1.0);
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);

glutInitWindowPosition(0,0);

glutCreateWindow("Cohen Suderland Line Clipping Algorithm");

glutDisplayFunc(display);

myinit();

glutMainLoop();

}
```

OUTPUT:



```
-------------- Run: Debug in OpenGl with freeGlut (compiler: GNU GCC Compiler)---------------
Checking for existence: C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe"  (in C:\Program
Files\CodeBlocks\MinGW\bin)
```

Program 7

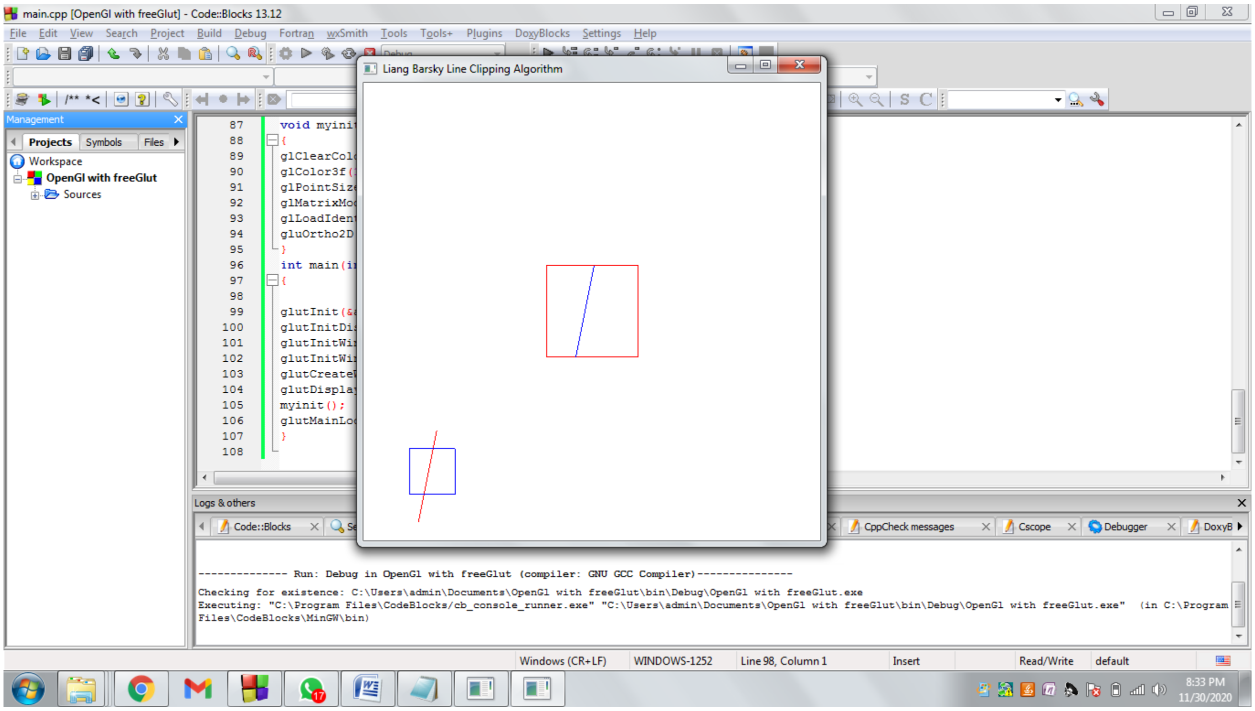
7. Write a program to implement the Liang-Barsky line clipping algorithm. Make provision to specify the input for multiple lines, window for clipping and viewport for displaying the clipped image.

```
// Liang-Barsky Line Clipping Algorithm with Window to viewport Mapping */

#include <stdio.h>

#include <GL/glut.h>

double xmin=50,ymin=50, xmax=100,ymax=100; // Window boundaries

double xvmin=200,yvmin=200,xvmax=300,yvmax=300; // Viewport boundaries

int cliptest(double p, double q, double *t1, double *t2)

{ double t=q/p;

 if(p < 0.0) // potentially enry point, update te

 {

 if( t > *t1) *t1=t;

 if( t > *t2) return(false); // line portion is outside

 }

 else

 if(p > 0.0) // Potentially leaving point, update tl

 {

 if( t < *t2) *t2=t;

 if( t < *t1) return(false); // line portion is outside

 }

 else

 if(p == 0.0)

 {

 if( q < 0.0) return(false); // line parallel to edge but outside
 }
return(true);
}
void LiangBarskyLineClipAndDraw (double x0, double y0,double x1, double y1)
{
double dx=x1-x0, dy=y1-y0, te=0.0, tl=1.0;
if(cliptest(-dx,x0-xmin,&te,&tl)) // inside test wrt left edge
if(cliptest(dx,xmax-x0,&te,&tl)) // inside test wrt right edge
if(cliptest(-dy,y0-ymin,&te,&tl)) // inside test wrt bottom edge
if(cliptest(dy,ymax-y0,&te,&tl)) // inside test wrt top edge
{
if( tl < 1.0 )
{
x1 = x0 + tl*dx;
y1 = y0 + tl*dy;
}
if( te > 0.0 )
{ x0 = x0 + te*dx;
y0 = y0 + te*dy;
}
 // Window to viewport mappings
double sx=(xvmax-xvmin)/(xmax-xmin); // Scale parameters
double sy=(yvmax-yvmin)/(ymax-ymin);
double vx0=xvmin+(x0-xmin)*sx;
double vy0=yvmin+(y0-ymin)*sy;
double vx1=xvmin+(x1-xmin)*sx;

double vy1=yvmin+(y1-ymin)*sy;

//draw a red colored viewport

glColor3f(1.0, 0.0, 0.0);

glBegin(GL_LINE_LOOP);

glVertex2f(xvmin, yvmin);

glVertex2f(xvmax, yvmin);

glVertex2f(xvmax, yvmax);

glVertex2f(xvmin, yvmax);

glEnd();

glColor3f(0.0,0.0,1.0); // draw blue colored clipped line

glBegin(GL_LINES);

glVertex2d (vx0, vy0);

glVertex2d (vx1, vy1);

glEnd();

}

}// end of line clipping

void display()

{

double x0=60,y0=20,x1=80,y1=120;

glClear(GL_COLOR_BUFFER_BIT);

//draw the line with red color

glColor3f(1.0,0.0,0.0);

//bres(120,20,340,250);

glBegin(GL_LINES);
```

```
glVertex2d (x0, y0);
glVertex2d (x1, y1);
glEnd();
//draw a blue colored window
glColor3f(0.0, 0.0, 1.0);
glBegin(GL_LINE_LOOP);
 glVertex2f(xmin, ymin);
 glVertex2f(xmax, ymin);
 glVertex2f(xmax, ymax);
 glVertex2f(xmin, ymax);
glEnd();
LiangBarskyLineClipAndDraw(x0,y0,x1,y1);
glFlush();
}
void myinit()
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(1.0,0.0,0.0);
glPointSize(1.0);
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);

glutInitWindowPosition(0,0);

glutCreateWindow("Liang Barsky Line Clipping Algorithm");

glutDisplayFunc(display);

myinit();

glutMainLoop();

}
```

OUTPUT:



main.cpp [OpenGl with freeGlut] - Code::Blocks 13.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects Symbols Files
Workspace
OpenGl with freeGlut
Sources
Liang Barsky Line Clipping Algorithm
Logs & others
Code::Blocks
CppCheck messages
Cscope
Debugger
DoxyB
Windows (CR+LF) WINDOWS-1252 Line 98, Column 1 Insert Read/Write default
8:33 PM
11/30/2020

```
void myini
{
glClearCol
glColor3f(
glPointSiz
glMatrixMo
glLoadIden
gluOrtho2D
}
int main(i
{

glutInit(&
glutInitDi
glutInitWi
glutInitWi
glutCreate
glutDispla
myinit();
glutMainLo
}
```

```
-------------- Run: Debug in OpenGl with freeGlut (compiler: GNU GCC Compiler)---------------
Checking for existence: C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe"  (in C:\Program
Files\CodeBlocks\MinGW\bin)
```

PROGRAM 8

8. Write a program to implement the Cohen-Hodgeman polygon clipping algorithm. Make provision to specify the input polygon and window for clipping.

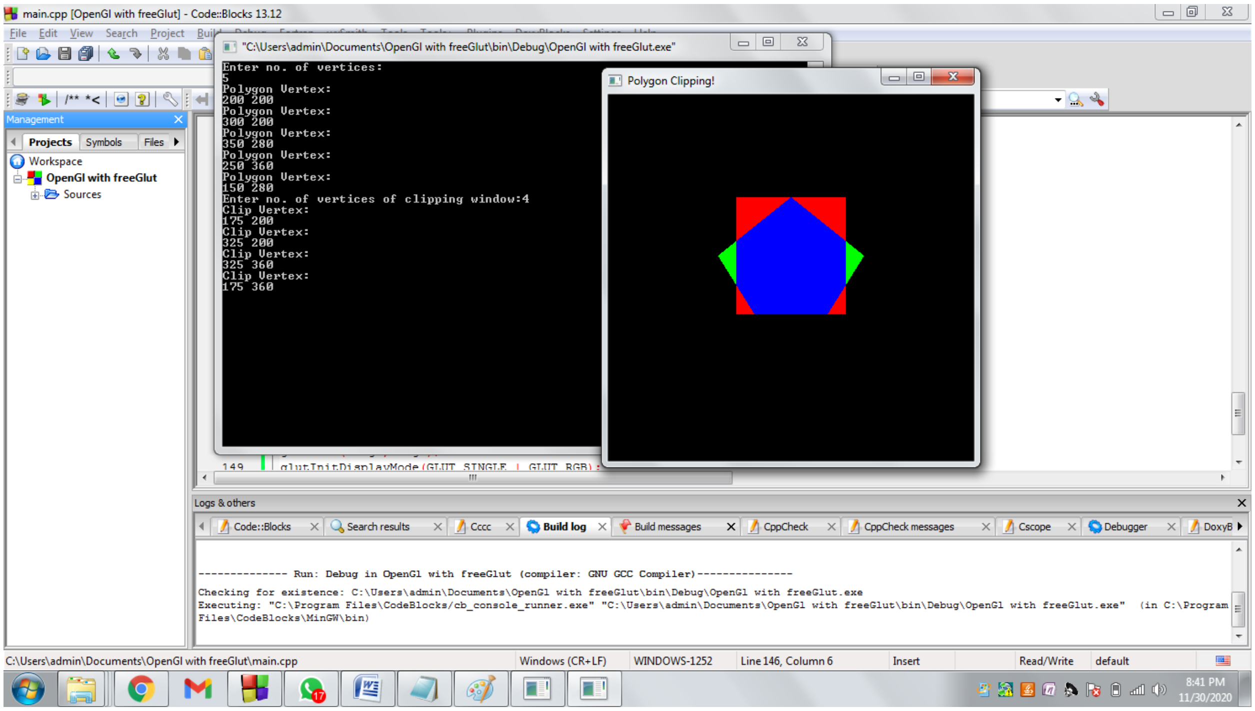
```
#include<iostream>

#include<GL/glut.h>

#include<stdio.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("%d", &poly_size);

org_poly_size = poly_size;

for (int i = 0; i < poly_size; i++)

{

printf("Polygon Vertex:\n");

scanf("%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("%d", &clipper_size);

for (int i = 0; i < clipper_size; i++)

{

printf("Clip Vertex:\n");

scanf("%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;

}
```

OUTPUT:



```
Enter no. of vertices:
5
Polygon Vertex:
200 200
Polygon Vertex:
300 200
Polygon Vertex:
350 280
Polygon Vertex:
250 360
Polygon Vertex:
150 280
Enter no. of vertices of clipping window:4
Clip Vertex:
175 200
Clip Vertex:
325 200
Clip Vertex:
325 360
Clip Vertex:
175 360
```

```
149    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
```

```
-------------- Run: Debug in OpenGl with freeGlut (compiler: GNU GCC Compiler)---------------
Checking for existence: C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe"  (in C:\Program Files\CodeBlocks\MinGW\bin)
```

C:\Users\admin\Documents\OpenGl with freeGlut\main.cpp | Windows (CR+LF) | WINDOWS-1252 | Line 146, Column 6 | Insert | Read/Write | default

8:41 PM
11/30/2020