NATIONAL UNIVERITY OF MODERN LANGUAGES **Faculty of Engineering & IT** **BS(Software Engineering)**	

Date:_______________

To be filled by student

Subject: **COMPUTER GRAPHICS**

Assignment title: **LAB MANUAL**

Submitted to: **ZAINAB MALIK**

Submitted by

IFTIKHAR AHMAD Roll# **10441**

Program_SemesterBS(SOFTWARE ENGINEERING)-7TH EVENING

To be filled by Teacher:

Total Marks______________________MarksObtained_____________________________________

Remarks____
____
____

Signature____________________

Note: Assignment will not be accepted after due date. Copied and irrelevant assignments will be marked zero. Half marks will be detected for poor formatted assignments.

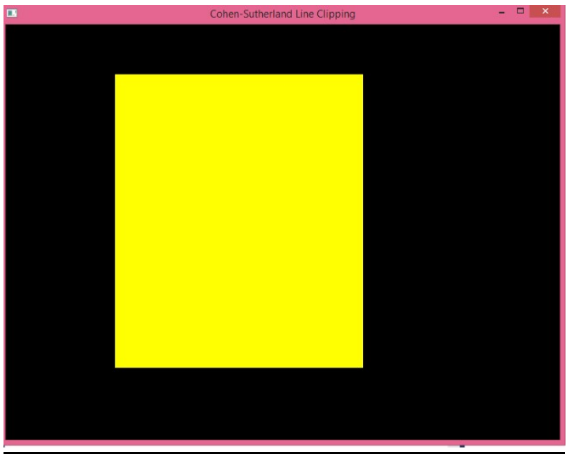
Tasks:

17) Line Clipping

18) Digital Differential algorithm

19) Tile Pattern

Lab Task 17:



Code:

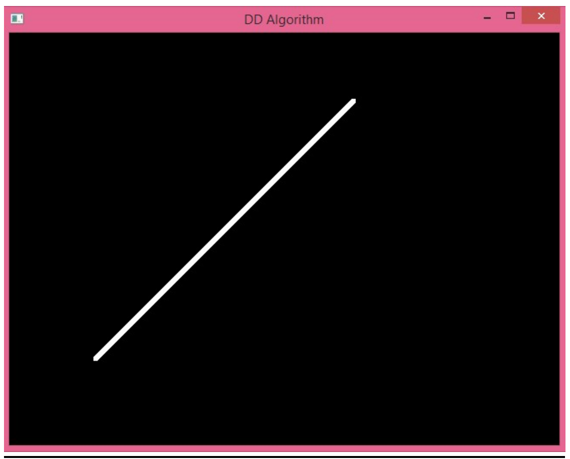
```
#include<windows.h>
#include<gl/Gl.h>
#include<glut.h>

intscreenheight = 600;
intscreenwidth = 800;
bool flag = true;

int x0 = 0, y0 = 0, x1 = 0, y1 = 0;
intxmin; intymin; intxmax; intymax;

voidDrawRect(intx0, inty0, intx1, inty1)
{
        glClear(GL_COLOR_BUFFER_BIT);
        glRecti(x0, y0, x1, y1);
        glFlush();
}

voidDrawLineSegment(intx0, inty0, intx1,
inty1){
        glColor3d(1, 0, 0);
        glBegin(GL_LINES);
        glVertex2i(x0, y0);
        glVertex2i(x1, y1);
        glEnd();
        glFlush();
}

typedefintOutCode;

constint INSIDE = 0; // 0000
constint BOTTOM = 1;  // 0001
constint RIGHT = 2; // 0010
constint TOP = 4; // 0100
constint LEFT = 8;  // 1000

OutCodeComputeOutCode(intx, inty)
{
        OutCode code;

        code = INSIDE;     // initialised
as being inside of clip window

        if (x<xmin)      // to the left of
clip window
               code = LEFT;
        elseif (x>xmax)      // to the
left of clip window
               code = RIGHT;
        elseif (y<ymin)      // to the
left of clip window
               code = BOTTOM;// Complete
the code segment here
        elseif (y>ymax)      // to the
left of clip window
               code = TOP;
        return(code);
}

voidCohenSutherlandLineClip(intx0, inty0,
intx1, inty1)
{
        // Complete the code segment here
        OutCode outcode0 =
ComputeOutCode(x0, y0);
        OutCode outcode1 =
ComputeOutCode(x1, y1);

        bool accept = false;

        while (true) {
               if (!(outcode0 | outcode1))
{//Bitwise OR is 0. Trivially accept
                      accept = true;
                      break;
               }
               elseif (outcode0 &
outcode1) {
                      // Bitwise AND is
not 0. Trivially reject
                      break;
               }
               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.
                      OutCodeoutcodeOut =
outcode0 ?outcode0 : outcode1;
                      // Now find the
intersection point;
```

```
                    if (outcodeOut& TOP)
{// point is above the window
                           x = x0 + (x1
- x0) * (ymax - y0) / (y1 - y0);
                           y = ymax;
                    }
                    elseif (outcodeOut&
BOTTOM) {// point is below the window
                           x = x0 + (x1
- x0) * (ymin - y0) / (y1 - y0);
                           y = ymin;
                    }
                    elseif (outcodeOut&
RIGHT) { // point is to the right of
window
                           y = y0 + (y1
- y0) * (xmax - x0) / (x1 - x0);
                           x = xmax;
                    }
                    else { // point is
to the left of window
                           y = y0 + (y1
- y0) * (xmin - x0) / (x1 - x0);
                           x = xmin;
                    }
                    //Now move outside
point to intersection point 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);
                    }
             }


      }
      if (accept)
             DrawLineSegment(x0, y0, x1,
y1);
}
voidmyMouse(intbutton, intstate, intx,
inty) {
      glPointSize(5.0);
      if (state == GLUT_DOWN)
      {
             if (button ==
GLUT_LEFT_BUTTON)
             {
                    if (flag){
                           x0 = x;
                           y0 =
screenheight - y;
                           flag = false;
                    }
                    else {
                           x1 = x;
                           y1 =
screenheight - y;

      CohenSutherlandLineClip(x0, y0,
x1, y1);
                           flag = true;
                    }
             }
             elseif (button ==
GLUT_RIGHT_BUTTON)
             {
                    if (flag){
                           x0 = x;
                           y0 =
screenheight - y;
                           flag = false;
                    }
                    else {
                           x1 = x;
                           y1 =
screenheight - y;
                           glColor3d(1,
1, 0);
                           DrawRect(x0,
y0, x1, y1);
                           xmin = x0;
                           xmax = x1;
                           ymin = y0;
                           ymax = y1;
                           flag = true;
                    }
             }
      }
}


voidmyDisplay()
{
      glClearColor(0.0f, 0.0f, 0.0f,
0.0f); //black

      glFlush();
}

int main(intargc, char ** argv) {

      glutInit(&argc, argv);
      glutInitWindowPosition(0, 0);
      glutInitWindowSize(800, 600);

      // create window
```

```
	glutCreateWindow("Cohen-Sutherland 
Line Clipping");

	// set the view frustum
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0, 800, 0, 600);
	glClear(GL_COLOR_BUFFER_BIT);
	// clear rendering surface
	glViewport(0, 0, 800, 600);

	glutMouseFunc(myMouse);
	glutDisplayFunc(myDisplay);
	glutMainLoop();

	return(0);
}
```

```
int main(intargc, char **argv)
{
	glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE);
	glutInitWindowSize(640, 480);
glutInitWindowPosition(300, 300);
glutCreateWindow("tittle");
	glutDisplayFunc(display);
	myInit();
	glutMainLoop();
}
```

Lab Task 18:

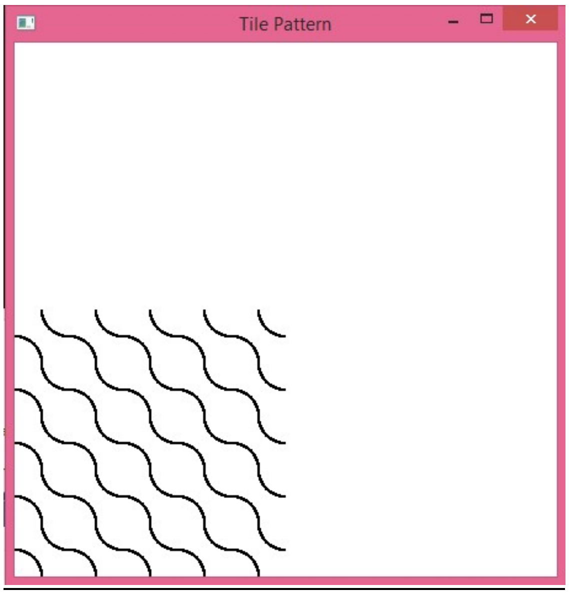


Code:

```
#include<Windows.h>
#include<gl/GL.h>
#include<glut.h>
#include<math.h>
voidmyInit()
{
	glClearColor(0.0, 0.0, 0.0, 0.0);
	glColor3f(1.0f, 1.0f, 1.0f);
	glPointSize(5.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0.0, 640.0, 0.0, 
480.0);
}
voidDDAline(intx1,inty1, intx2, inty2)
{
	int dx, dy, steps;
	floatxinc, yinc, x, y;
	dx = x2 - x1;
	dy = y2 - y1;
	if (abs(dx) > abs(dy))
	{
		steps = dx;
	}
	else
	{
		steps = dy;
	}
	xinc = dx / float(steps);
	yinc = dy / float(steps);
	x = x1;
	y = y1;
	glBegin(GL_POINTS);
	glVertex2f(x, y);
	for (int k = 0; k < steps; k++)
	{
		x = x + xinc;
		y = y + yinc;

	glVertex2f(round(x),round(y));
	}
	glEnd();
	glFlush();

}
void display()
{
	glClear(GL_COLOR_BUFFER_BIT);
	DDAline(100,100,400,400);
	glFlush();
}
int main(intargc, char **argv)
{
	glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE);
	glutInitWindowSize(640, 480);
glutInitWindowPosition(300, 300);
glutCreateWindow("DD Algorithm");
```

```
	glutDisplayFunc(display);
	myInit();
	glutMainLoop();
}
```

Lab Task 19:



Code:

```
#include<windows.h>// use as needed for
your system
#include<glut.h>
#include<math.h>

//<<<<<<<<<<<<<<<<<<<<<<<myInit>>>>>>>>>>
>>>>>>>>>>
voidmyInit(void)
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
// set the bg color to a bright white
	glColor3f(0.0f, 0.0f, 0.0f); //
set the drawing color to black
	glPointSize(2.0); //set the point
size to 4 by 4 pixels
	glMatrixMode(GL_PROJECTION);// set
up appropriate matrices- to be explained
	glLoadIdentity();// to be
explained
	gluOrtho2D(0.0, 440.0, 0.0,
440.0);// to be explained
}

voidsetWindow(GLdoubleleft,
GLdoubleright, GLdoublebottom,
GLdoubletop)
{
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(left, right, bottom,
top);
}

voidsetViewPort(GLintleft, GLintright,
GLintbottom, GLinttop)
{
	glViewport(left, right, right -
left, top - bottom);
}

voiddrawTile()
{
	glBegin(GL_POINTS);
	for (floati = 0.01745329; i<=
0.01745329 * 90; i += 0.01745329)
	{
		glVertex2f(220 * cos(i),
220 * sin(i));
	}
	glEnd();
	glFlush();
	glBegin(GL_POINTS);
	for (floati = 0.01745329 * 180;
i<= 0.01745329 * 270; i += 0.01745329)
	{
		glVertex2f(220 * cos(i) +
440, 220 * sin(i) + 440);
	}
	glEnd();
	glFlush();
}
//<<<<<<<<<<<<<<<<<<<<<<<<myDisplay>>>>>>
>>>>>>>>>>>
// the redraw function
voidmyDisplay(void)
{
	glClear(GL_COLOR_BUFFER_BIT); //
clear the screen
	for (inti = 0; i<= 4; i++)
	{
		for (int j = 0; j <= 4;
j++)
		{
			glViewport(i * 44, j
* 44, 44, 44);
			drawTile();
		}
	}

	glFlush(); // send all output to
display
}
//<<<<<<<<<<<<<<<<<<<<<<<< main
>>>>>>>>>>>>>>>>>>>>>>
```

```
void main(intargc, char **argv)
{
	glutInit(&argc, argv); //
initialize the toolkit
	glutInitDisplayMode(GLUT_SINGLE |
GLUT_RGB); // set the display mode
	glutInitWindowSize(440, 440); //
set the window size
	glutInitWindowPosition(100, 150);
// set the window position on the screen
	glutCreateWindow("Tile Pattern");
// open the screen window(with its
exciting title)
	glutDisplayFunc(myDisplay); //
register the redraw function
	myInit();
	glutMainLoop(); // go into a
perpetual loop
}
```