Experiment Name: Write a program to implement DDA line algorithm.

Objective: This objective of the program is to implement DDA line algorithm.

Theory: The digital differential analyzer (DDA) is an incremental scan-conversation method. Such an approach is characterized by performing calculations at each step using results from the preceding step. Suppose at step i we have calculated (x_i, y_i) to be a point on the line. Since the next point (xi+1, yi+1) should satisfy $\Delta y/\Delta x=m$ where $\Delta y=y_{i+1}-y_i$ and $\Delta x=x_{i+1}-x_i$ we have

$$y_{i+1}=y_i + m\Delta x$$

or

$$x_{i+1}=x_i + \Delta y/m$$

Algorithm:

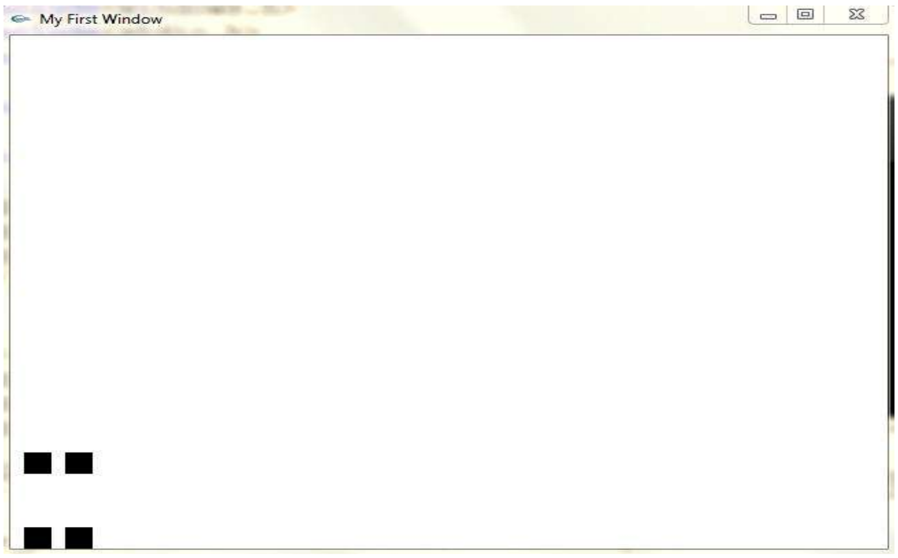
```
int x=x'_1, y=y'_1;
int dx= x'_2-x'_1, dy= y'_2-y'_1, dT=2(dy-dx), dS=2dy;
int d=2dy-dx; setPixel(x,y);
while(x<x'_2)  {
        x++;
        if(d<0)
            d=d+ds;
        else {
            y++;
            d=d+dT;
        }
        setPixel(x,y);
    }
```

Source Code:

```
#include<windows.h>
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
void myInit(void) {
glClearColor(1.0, 1.0, 1.0, 0.0);
glColor3f(0.0f, 0.0f, 0.0f);
glPointSize(4.0);
glLineWidth(4.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1.0,1.0,-1.0, 1.0, -1.0, 1.0);
gluOrtho2D(0.0, 640.0, 0.0, 480.0); }
void myDisplay(void) {
```

```
glClear(GL_COLOR_BUFFER_BIT);
glPointSize(20.0);
glBegin(GL_POINTS);
            {
glVertex2i(20,10);
glVertex2i(50,10);
glVertex2i(20,80);
glVertex2i(50,80);
}
glEnd();
glFlush();  }
int  main(int argc, char** argv) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(640,480);
glutInitWindowPosition(100,150);
glutCreateWindow("My First Window");
glutDisplayFunc(myDisplay);
myInit();
glutMainLoop();

}
```

Sample output:



Discussion:

This program is to implement DDA line algorithm.

Experiment Name: Write a program to implement Bresenham's line algorithm.

Objective: This objective of the program is to implement Bresenham's line algorithm.

Theory: Bresenham's line algorithm is a highly efficient incremental method for scan-converting lines. We want to scan-convert the line where $0<m<1$. We start with pixel $P'_1(x'_1, y'_1)$, then select subsequent pixels as we work our way to the right, one pixel position at a time in the horizontal direction towards $P'_2(x'_2, y'_2)$.

Algorithm:

```
int x=x'1, y=y'1;
int dx= x'2-x'1, dy= y'2-y'1, dT=2(dy-dx), dS=2dy;
int d=2dy-dx; setPixel(x,y);
while(x<x'2)  {
            x++;
            if(d<0)
                  d=d+ds;
            else {
                  y++;
                  d=d+dT;
            }
            setPixel(x,y);
      }
```

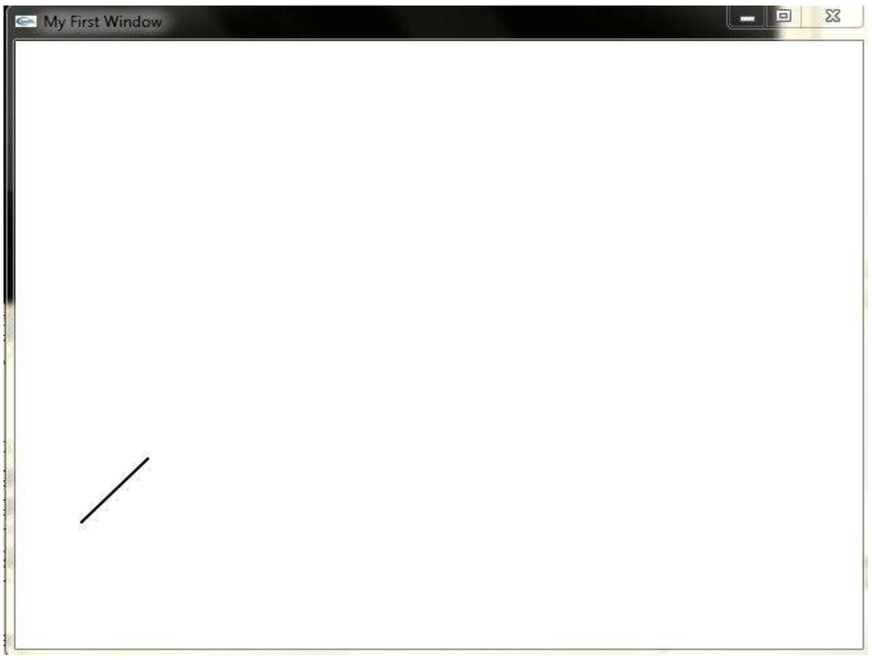
Source Code:

```
#include<windows.h>
#include<GL/glut.h>
void myInit(void)
{
glClearColor(1.0, 1.0, 1.0, 0.0);
glColor3f(0.0f, 0.0f, 0.0f);
glPointSize(4.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1.0,1.0,-1.0, 1.0, -1.0, 1.0);
gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void myDisplay(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glPointSize(2.0);
```

```
GLint x,y, x1=50, x2=100, y1=100, y2=200;
GLint dx, dy, inc1, inc2, d;

x = x1; y = y1;
dx = x2-x1; dy = y2-y1;
inc1 = 2 * dy; inc2 = 2 * (dy-dx);
d = inc1 - dx;
while(x <= x2) {
glBegin(GL_POINTS);
{
        glVertex2i(x,y);
}
glEnd();

x++;

if(d < 0)
        d = d + inc1;

 else {d = d + inc2;

                y++;

           }

      }

glFlush();

}

int  main(int argc, char** argv)

{

  glutInit(&argc, argv);

  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

  glutInitWindowSize(640,480);

  glutInitWindowPosition(100,150);

  glutCreateWindow("My First Window");

glutDisplayFunc(myDisplay);

 myInit();

  glutMainLoop();
```

```
}
```

Sample Output:



Discussion:

This program is to implement Bresenham’s line algorithm.

Experiment Name: Write a program to implement Bresenham's circle algorithm.

Objective: This objective of the program is to implement Bresenham's circle algorithm.

Theory: Bresenham's circle algorithm allows scan-converting a circle using Bresenham's algorithm works as follows. If the eight-way symmetry of a circle is used to generate a circle, points will only have to be generated through a 45° angle. And, if points are generated from 90°t 45°, moves will be made only in the +x and –y directions.

Algorithm:

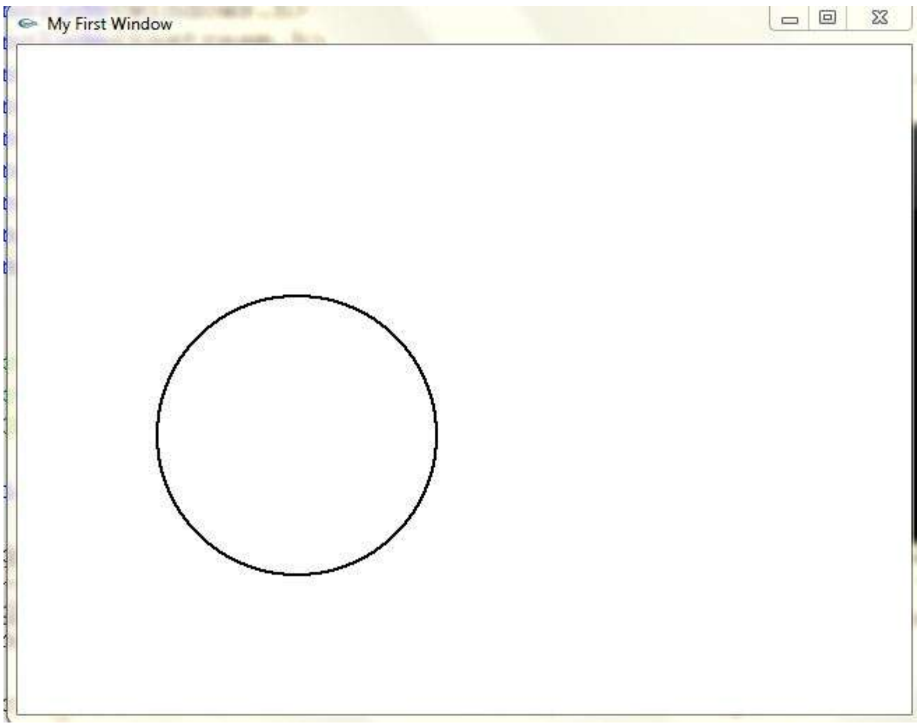
```
int x=0, y=r, d=3-2r;
while(x<=y) {
        setPixel(x,y);
        if(d<0)
                d=d+4x+6;
        else {
                d=d+4(x-y)+10;
                y--;
        }
        x++;
}
```

Source Code:

```
#include<windows.h>
#include<iostream.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<GL/gl.h>
#include<GL/glu.h>
#include<GL/glut.h>
void myInit(void)
{
glClearColor(1.0, 1.0, 1.0, 0.0);
glColor3f(0.0f, 0.0f, 0.0f);
glPointSize(4.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
```

```
void myDisplay(void) {
glClear(GL_COLOR_BUFFER_BIT);

glPointSize(2.0);
GLint h = 200,k = 200, x, y, r, d;
y = r = 100;
d = 3 - 2*r;
for(x=0; x<=y; x++)
{
        glBegin(GL_POINTS);
{
        glVertex2i(x+h,y+k);
        glVertex2i(x+h,k-y);
        glVertex2i(y+h,x+k);
        glVertex2i(y+h,k-x);
        glVertex2i(-y+h,x+k);
        glVertex2i(-y+h,k-x);
        glVertex2i(-x+h,y+k);
        glVertex2i(-x+h,k-y);
}
        glEnd();
if(d<0)
d = d + 4*x + 6;
else{
d = d + 4*(x-y) + 10;
y--;
}
}
glFlush();
}


int  main(int argc, char** argv)
{

  glutInit(&argc, argv);

  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

  glutInitWindowSize(640,480);

  glutInitWindowPosition(100,150);

  glutCreateWindow("My First Window");

glutDisplayFunc(myDisplay);

myInit();

  glutMainLoop();
}
```

Sample Output:



Discussion:

This program is to implement Bresenham’s circle algorithm.

Experiment Name: Write a program to implement midpoint ellipse drawing algorithm.

Objective: This objective of the program is to implement midpoint ellipse drawing algorithm.

Theory: The ellipse, like the circle, shows symmetry. In the case of an ellipse, however, symmetry is four-rather than eight-way. There are two methods of mathematically defining a ellipse. The polynomial method of defining an ellipse is given by the expression

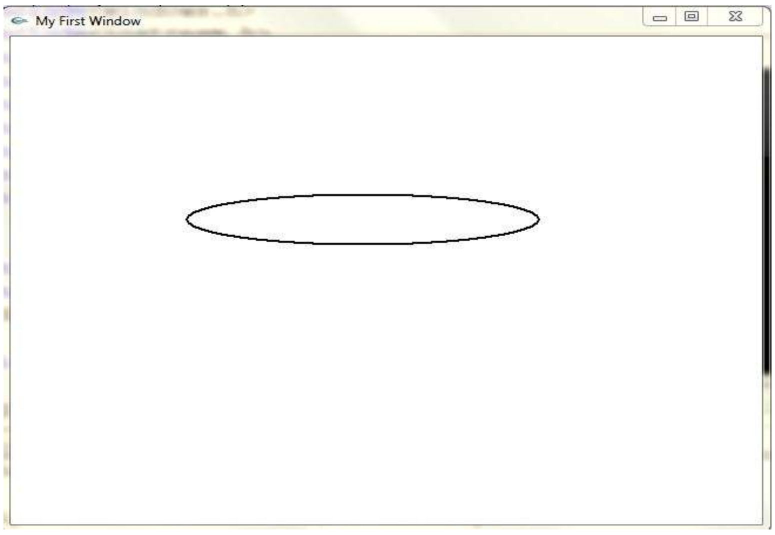
$$(x-h)^2/a^2 + (y-k)^2/b^2 = 1$$

Source Code:

```
#include<windows.h>
#include<iostream.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<GL/gl.h>
#include<GL/glu.h>
#include<GL/glut.h>
const int screenWidth = 640*2;
const int screenHeight = 480*2;
GLdouble A,B,C,D;
void myInit(void)
{
glClearColor(1.0, 1.0, 1.0, 0.0);
glColor3f(0.0f, 0.0f, 0.0f);
glPointSize(4.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1.0,1.0,-1.0, 1.0, -1.0, 1.0);
A = screenWidth/4.0;
B= 0.0;
C=D=screenHeight/2.0;
gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void myDisplay(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glPointSize(2.0);
GLint h = 300, k = 300, x, x2, y, i = 1, a = 150, b = 25; x2 = a;
for(x=1;x<=x2;x++)  {
```

```
y = b * sqrt(1 - (pow(x,2)/pow(a,2)));
glBegin(GL_POINTS);

{
          glVertex2i(x+h,y+k);
          glVertex2i(-x+h,-y+k);
          glVertex2i(-x+h,y+k);
           glVertex2i(x+h,-y+k);

}
glEnd();

}

glFlush();

}

int  main(int argc, char** argv)

{

  glutInit(&argc, argv);

  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

  glutInitWindowSize(640,480);

  glutInitWindowPosition(100,150);

  glutCreateWindow("My First Window");

  glutDisplayFunc(myDisplay);

  myInit();

  glutMainLoop();

}
```

Sample Output:



Discussion:

This program is to implement ellipse drawing algorithm.

Experiment Name: Write a program to implement Cohen-Suntherland line clipping algorithm.

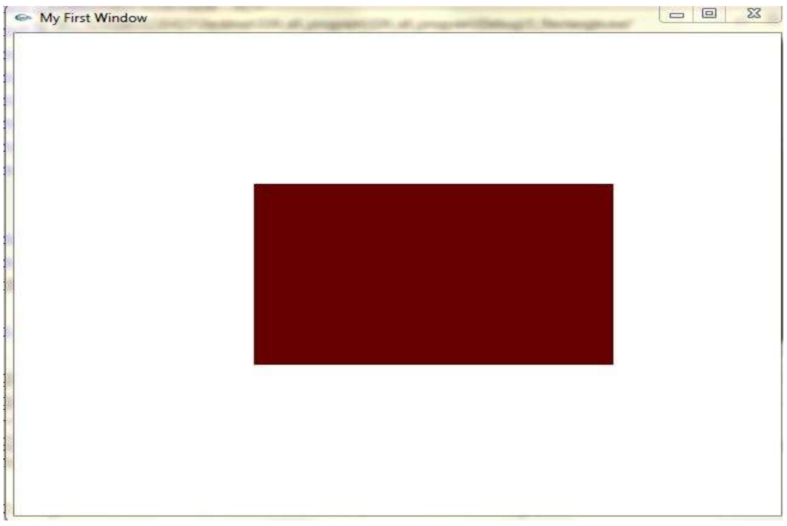
Objective: This objective of the program is to implement Cohen-Suntherland line clipping algorithm.

Theory: A rectangle whose sides are parallel to the coordinate axes may be constructed if the locations of two vertices. The remaining corner points are then derived. Once the vertices are known, the 4 sets of coordinates are sent to the line routine & the rectangle is scan-converted. In the case of the rectangle, lines would be drawn as follows: line (x_1,y_1) to (x_1,y_1); line (x_1,y_2) to (x_2,y_2); line (x_2,y_2) to (x_2,y_1); line (x_2,y_1) to (x_1,y_1).

Source Code:

```
#include<windows.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<GL/gl.h>
#include<GL/glu.h>
#include<GL/glut.h>
const int screenWidth = 640;
const int screenHeight = 480;
GLdouble A,B,C,D;
void myInit(void)
{
glClearColor(1.0, 1.0, 1.0, 0.0);
glColor3f(0.0f, 0.0f, 0.0f);
glPointSize(4.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1.0,1.0,-1.0, 1.0, -1.0, 1.0);
A = screenWidth/4.0;
B= 0.0;
C=D=screenHeight/2.0;
gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void myDisplay(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.4,0.0,0.0);
glRecti(200,150,500,330);
glFlush();
}
int  main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(640,480);
  glutInitWindowPosition(100,150);
  glutCreateWindow("My First Window");
  glutDisplayFunc(myDisplay);
  myInit();
  glutMainLoop();
  return 1;
}
```

Sample Output:



Discussion:

This program is to implement polygon (rectangle) filling algorithm.

Experiment Name: Write a program to implement Cohen-Suntherland line clipping algorithm.

Objective: This objective of the program is to implement Cohen-Suntherland line clipping algorithm.

Theory: In this algorithm we divide the line clipping process two phases: (1) identify those lines which intersect the clipping window & so need to be clipped & (2) perform the clipping. All lines fall into one of the following clipping categories:
1. Visible- both endpoints of the line lie within the window.
2. Not visible- the line definitely lies outside the window. This will occur if the line from (x_1,y_1) to (x_2,y_2) satisfies any one of the following four inequalities:

$$x_1,x_2 > x_{max} \quad y_1,y_2 > y_{max}$$
$$x_1,x_2 < x_{min} \quad y_1,y_2 < y_{min}$$

3. Clipping candidate- the line is in neither category 1 nor 2.

Source Code:

```
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<cctype>
#include<cmath>
#include<iostream>
#include<iterator>
#include <windows.h>
#include <gl/glut.h>
using namespace std;
#define REP(i,n) for(i=0; i<(n); i++)
#define FOR(i,a,b) for(i=(a); i<=(b); i++)
#define WIDTH 640
#define HEIGHT 480
#define Wxmin 0
#define Wxmax 200
#define Wymin 0
#define Wymax 200
struct wind  {
        double x[4],y[4];
}W;
void reshape(int width, int height)  {
        glViewport(0,0,width, height);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-WIDTH/2, WIDTH/2+1, -HEIGHT/2, HEIGHT/2+1,-1000,1000);
```

```
}
void drawPixel(int x, int y, int op)  {
        if(op == 0)              glVertex2i(x,y);
        else if(op == 3)         glVertex2i(-x,y);
        else if(op == 4)         glVertex2i(-x,-y);
        else if(op == 7)         glVertex2i(x,-y);
        else if(op == 1)         glVertex2i(y,x);
        else if(op == 2)         glVertex2i(-y,x);
        else if(op == 5)         glVertex2i(-y, -x);
        else if(op == 6)         glVertex2i(y, -x);
}
void drawLine(int x0,int y0,int x1,int y1,int value)     {
        int dx = x1 - x0;
        int dy = y1 - y0;
        int dinitial = 2*dy - dx;
        int dE = 2*dy;
        int dNE = 2*(dy - dx);
        int x = x0;
        int y = y0;
        drawPixel(x,y,value);
        while ( x < x1 )    {
                if( dinitial < 0 )  {
                        x ++;
                        dinitial += dE;
                }
                else  {
                        x++;
                        y++;
                        dinitial += dNE;
                }
        drawPixel(x,y,value);
        } }
void drawSlop(int x0, int y0, int x1, int y1)   {
        int dx = x1 - x0;
        int dy = y1 - y0;
        if(abs(dx)>=abs(dy))  {
                if((x1>=x0)&&(y1>=y0))
                        drawLine(x0,y0,x1,y1,0);
                else if((x1<x0)&&(y1>=y0))
                        drawLine(-x0,y0,-x1,y1,3);
                else if((x1<x0) && (y1<y0))
                        drawLine(-x0,-y0,-x1,-y1,4);
                else
                        drawLine(x0,-y0,x1,-y1,7);    }

        else    {
        if((x1>=x0)&&(y1>=y0))
                drawLine(y0,x0,y1,x1,1);
        else if((x1<x0)&&(y1>=y0))
                drawLine(y0,-x0,y1,-x1,2);
        else if((x1<x0)&&(y1<y0))
                drawLine(-y0,-x0,-y1,-x1,5);
        else if((x1>=x0)&&(y1<=y0))
                drawLine(-y0,x0,-y1,x1,6);    }
        }
void CohenSutherlandLineClipAndDraw(double x0,double y0,double x1,double y1,double
xmin,double xmax,double ymin,double ymax)
        {
        outcode outcode0,outcode1,outcodeOut;
        bool accept = false,done = false;
        outcode0 = CompOutCode(x0,y0,xmin,xmax,ymin,ymax);
        outcode1 = CompOutCode(x1,y1,xmin,xmax,ymin,ymax);
        do{
                if(!(outcode0|outcode1))
                        accept = true; done = true;
                else if(outcode0&outcode1)
                        done = true;
                else  {
                        double x,y;
                        outcodeOut = outcode0?outcode0:outcode1;
                        if(outcodeOut&TOP){
                                x = x0 + (x1-x0)*(ymax-y0)/(y1-y0);
                                y = ymax;
                        }
                else if(outcodeOut&BOTTOM){
                        x = x0 + (x1-x0)*(ymin-y0)/(y1-y0);
                        y = ymin;
                        }
                else if(outcodeOut&RIGHT){
                        y = y0 + (y1-y0)*(xmax-x0)/(x1-x0);
                        x = xmax;
                        }
                else  {
                        y = y0 + (y1-y0)*(xmin-x0)/(x1-x0);
                        x = xmin;
```

```
                    }
                if(outcodeOut==outcode0) {
                    x0 = x;  y0= y;
                    outcode0 = CompOutCode(x0,y0,xmin,xmax,ymin,ymax);  }

                else {
                    x1 = x;  y1 = y;
                    outcode1 = CompOutCode(x1,y1,xmin,xmax,ymin,ymax); }  }
                }while(done==false);
                if(accept) {
                    drawSlop(x0,y0,x1,y1);

        }

}

CohenSutherlandLineClipAndDraw(P[i].x,P[i].y,P[j].x,P[j].y,W.x[0],W.x[1],W.y[0],W.y[2]);

}

REP(i,2)

        {

                j = (i + 2)%4;

CohenSutherlandLineClipAndDraw(P[i].x,P[i].y,P[j].x,P[j].y,W.x[0],W.x[1],W.y[0],W.y[2]);

}

}

void display(void)

{

glColor4f(1.0,0.0,0.0,1.0);

        angle = 0;

        getPoint();

        getPos();

        setW();

        while(true)

        {
                glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
                glBegin(GL_POINTS);
                resetW();
                glColor4f(3.0,5.0,1.0,1.0);
                drawW();
                glColor4f(0.0,0.0,0.0,0.0);
                drawTetra();
                glColor4f(1.0,0.0,0.0,1.0);
                drawTetraC();
                glEnd();
                glutSwapBuffers();
        }
}
int main(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE);
        glutInitWindowPosition(-1,-1);
        glutInitWindowSize(WIDTH, HEIGHT);
        glutCreateWindow("My Window");
        glutReshapeFunc(reshape);
        glutDisplayFunc(display);
        glutMainLoop();
}
```

Discussion:

This program is to implement polygon (rectangle) filling algorithm.

Experiment Name: Write a program to implement a scaling & shifting polygon by mouse clipping.

Objective: This objective of the program is to implement a scaling & shifting polygon by mouse clipping.

Theory: In this algorithm we divide the line clipping process two phases: (1) identify those lines which intersect the clipping window & so need to be clipped & (2) perform the clipping. All lines fall into one of the following clipping categories:
1. Visible- both endpoints of the line lie within the window.
2. Not visible- the line definitely lies outside the window. This will occur if the line from (x_1,y_1) to (x_2,y_2) satisfies any one of the following four inequalities:

$$x_1,x_2 > x_{max} \quad y_1,y_2 > y_{max}$$
$$x_1,x_2 < x_{min} \quad y_1,y_2 < y_{min}$$

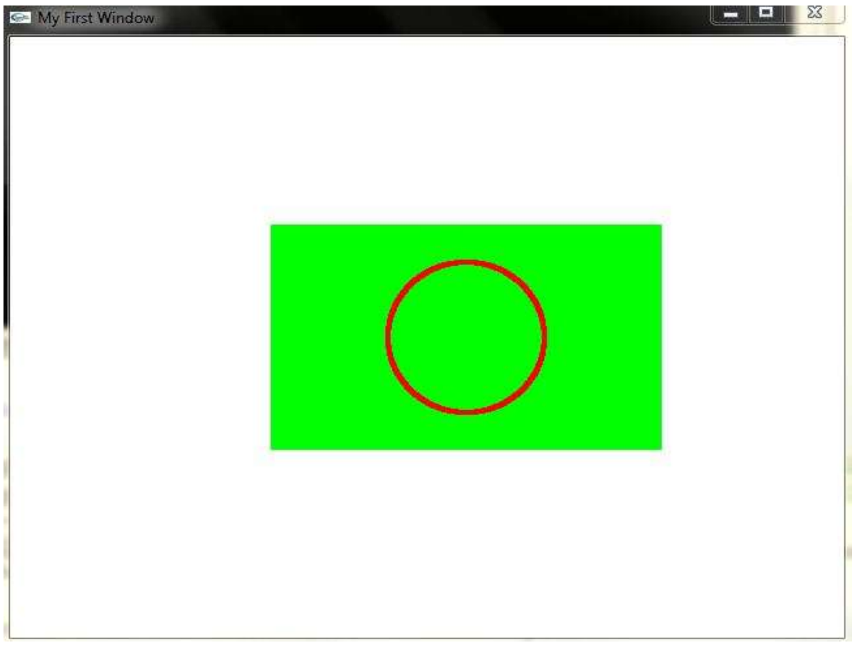
3. Clipping candidate- the line is in neither category 1 nor 2.

Source Code:

```
#include<windows.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<GL/gl.h>
#include<GL/glu.h>
#include<GL/glut.h>
const int screenWidth = 640;
const int screenHeight = 480;
GLdouble A,B,C,D;
void myInit(void) {
        glClearColor(1.0, 1.0, 1.0, 0.0);
        glColor3f(0.0f, 0.0f, 0.0f);
        glPointSize(4.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
         glOrtho(-1.0,1.0,-1.0, 1.0, -1.0, 1.0);
        A = screenWidth/4.0;
        B= 0.0;    C=D=screenHeight/2.0;
        gluOrtho2D(0.0, 640.0, 0.0, 480.0);  }
void myMouse(int button, int state, int x ,int y)  {
if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        glColor4f(0.0,1.0,0.0,1.0);
        glRecti(200,150,500,330);
```

```
    }
else if(button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)  {
        glColor3f(1.0,0.0,0.0);
        GLint h = 350,k = 240, x, y, r, d;
        y = r = 60;      d = 3 - 2*r;
for(x=0; x<=y; x++)   {
        glBegin(GL_POINTS);
{
        glVertex2i(x+h,y+k);
        glVertex2i(x+h,k-y);
        glVertex2i(y+h,x+k);
        glVertex2i(y+h,k-x);
        glVertex2i(-y+h,x+k);
        glVertex2i(-y+h,k-x);
        glVertex2i(-x+h,y+k);
        glVertex2i(-x+h,k-y);
}
        if(d<0)
                 d = d + 4*x + 6;
        else{
                d = d + 4*(x-y) + 10;
                 y--;
                }        }
        }
}
void myDisplay(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
}
int  main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(640,480);
        glutInitWindowPosition(100,150);

        glutCreateWindow("My First Window");

        glutDisplayFunc(myDisplay);

        glutMouseFunc(myMouse);

         myInit();
        glutMainLoop();
}
```

Sample Output:



Discussion:

This program is to implement a scaling & shifting polygon by mouse clipping.

Experiment Name: Write a program to implement a scaling line triangle transformation.

Objective: This objective of the program is to implement a scaling line triangle transformation.

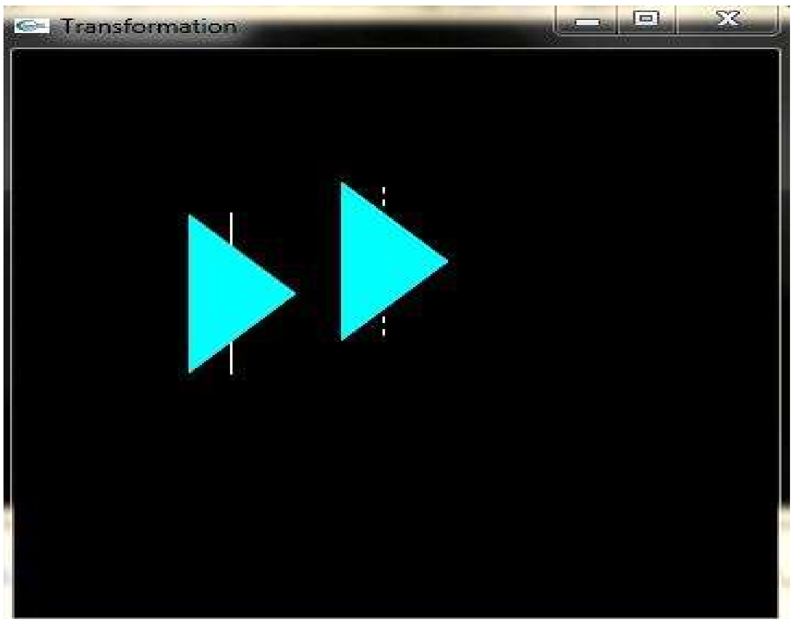
Theory: Scaling is the process of expanding or compressing the dimensions of an object. Positive scaling constants s_x and s_y are used to describe changes in length with respect to the x direction and y direction, respectively. A scaling constant greater than one indicates an expansion of length, and less than one, compressing of length. The scaling transformation $Ss_x.s_y$ is given by $P'=Ss_x.s_y(P)$ where $x'=s_xx$ and y' s_yy. Notice that, after a scaling transformation is performed, the new object is located at a different position relative to the origin.

Source Code:

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>
void triangle()
{
        glColor3f(0.0, 1.0, 1.0);
        glBegin(GL_TRIANGLES);
        glVertex2i (80, 150);
        glVertex2i (80, 250);
        glVertex2i (130,200);

}
void line(int sx,int sy,int dx,int dy)
{
        glColor3f(1.0, 1.0, 1.0);
        glBegin(GL_LINES);
        glVertex2i (sx, sy);
        glVertex2i (dx, dy);
}
void display(void)
{
        glClear (GL_COLOR_BUFFER_BIT);
        line(100,150,100,250);
        triangle();
        glTranslatef(70.0, 20.0,0.0);
        glEnable(GL_LINE_STIPPLE);
        glLineStipple(1, 0xF0F0);
        line(100,150,100,250);
        triangle();
}
```

```
void init (void)
{

        glClearColor (0.0, 0.0, 0.0, 0.0);
        gluOrtho2D(0.0, 350.0, 0.0, 350.0);
}
int main(int argc, char** argv)
{

        glutInit(&argc, argv);
        glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize (350, 350);
        glutInitWindowPosition (100, 100);
        glutCreateWindow ("Transformation");
        init();
        glutDisplayFunc(display);
        glutMainLoop();

}
```

Sample Output:



Discussion:

This program is to implement a scaling line triangle transformation.

Experiment Name: Write a program to implement composite (transition+scaling) transformation.

Objective: This objective of the program is to implement composite (transition+scaling) transformation.

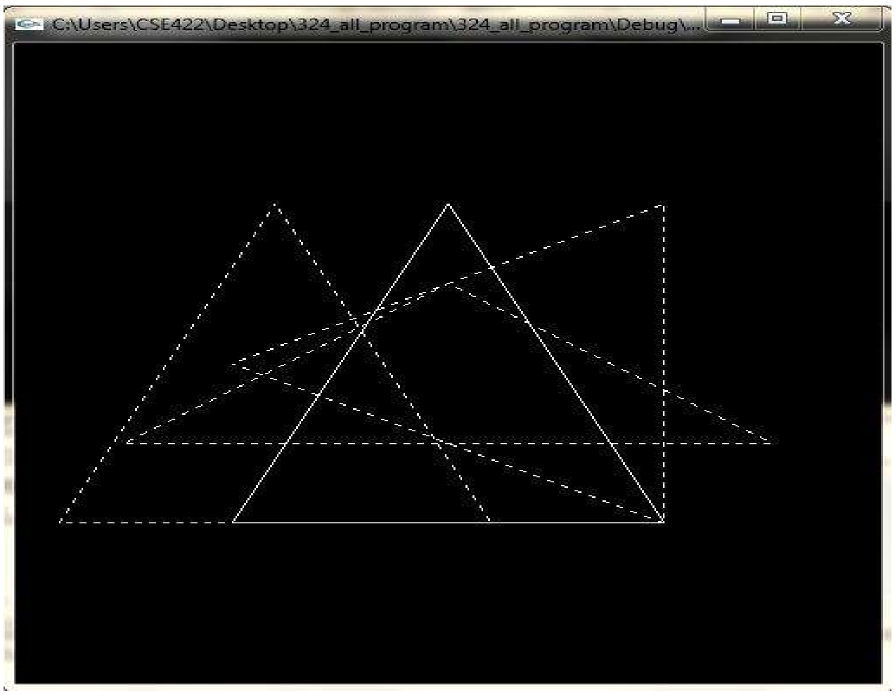
Theory: More complex geometric and coordinate transformations can be built from the basic transformations described by using the process of composition of functions. For example, such operations as rotation about a point other than the origin or reflection about lines other than the axes can be constructed from the basic transformations. If the xy coordinate system is displaced to a new position, where the direction and distance of the displacement is given by the vector $v=t_xI+t_yJ$, the coordinates of a point systems are related by the translation transformation T_v, where $x'=x-t_x$ and $y'=y-t_y$.

Source Code:

```
#include <GL/glut.h>

#include <stdlib.h>

void init(void)

{

glClearColor (0.0, 0.0, 0.0, 0.0);

glMatrixMode(GL_PROJECTION);

glLoadIdentity();

glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);

glMatrixMode(GL_MODELVIEW);

glLoadIdentity();

}

void draw_triangle(void)

{

  glBegin (GL_LINE_LOOP);

  glVertex2f(0.0, 25.0);

  glVertex2f(25.0, -25.0);

  glVertex2f(-25.0, -25.0);

}
```

```
void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (1.0, 1.0, 1.0);
  glLoadIdentity ();
  glColor3f (1.0, 1.0, 1.0);
  draw_triangle ();
  glEnable (GL_LINE_STIPPLE);
  glLineStipple (1, 0xF0F0);
  glLoadIdentity ();
  glTranslatef (-20.0, 0.0, 0.0);
  draw_triangle ();
  glLineStipple (1, 0xF00F);
  glLoadIdentity ();
  glScalef (1.5, 0.5, 1.0);
  draw_triangle ();
 glLineStipple (1, 0x8888);
  glLoadIdentity ();
  glRotatef (90.0, 0.0, 0.0, 1.0);
  draw_triangle ();
  glDisable (GL_LINE_STIPPLE);
}
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (500, 500);

  glutInitWindowPosition (100, 100);

  glutCreateWindow (argv[0]);

  init ();

  glutDisplayFunc(display);

  glutMainLoop();

}
```

Sample Output:


C:\Users\CSE422\Desktop\324_all_program\324_all_program\Debug\...

Discussion:

This program is to implement composite (transition+scaling) transformation.