**PRACTICAL 1**

• **Mid-Point Line Drawing Algorithm:** -

Step 1 − Get the input of two end points (X0,Y0)(X0,Y0) and (X1,Y1)(X1,Y1).

Step 2 − Calculate the difference between two end points.

dx = X1 - X0

dy = Y1 - Y0

Step 3 − Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If dx > dy, then you need more steps in x coordinate; otherwise in y coordinate.

if (absolute(dx) > absolute(dy))

Steps = absolute(dx);

else Steps = absolute(dy);

Step 4 − Calculate the increment in x coordinate and y coordinate.

Xincrement = dx / (float) steps;

Yincrement = dy / (float) steps;

Step 5 − Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

**Code:**

#include<iostream>

#include<conio.h>

#include<graphics.h>

#include<math.h>

using namespace std;

int main(){

int gd = DETECT, gm;

char data[] = "C:\\MinGW\\lib\\libbgi.a";

int x1,y1,x2,y2;

cout<<"Enter the two points to draw line:-\t";

cin>>x1>>y1>>x2>>y2;

int dx=x2-x1;

int dy=y2-y1;

int steps;

if(abs(dx)>abs(dy)){

steps=abs(dx);

}

else{

steps=abs(dy);

}

float Ix=dx/(float)steps;

float Iy=dy/(float)steps;

float x=x1;

float y=y1;

initgraph(&gd, &gm, data);

putpixel(round(x),round(y),WHITE);

for(int i=0;i<steps;i++){

x=x+Ix;

y=y+Iy;

putpixel(round(x),round(y),WHITE);

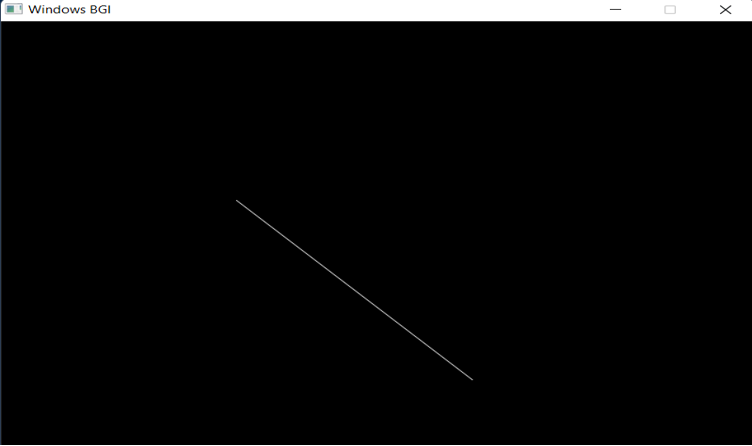
}

getch();

closegraph();

}

**OUTPUT**



PRACTICAL 2

Bresenham’s Algorithm: -

Step 1 − Input the two end-points of line, storing the left end-point in (x0,y0)(x0,y0).

Step 2 − Plot the point (x0,y0)(x0,y0).

Step 3 − Calculate the constants dx, dy, 2dy, and 2dy–2dx2dy–2dx and get the first value for the decision parameter as –

p0=2dy−dxp0=2dy−dx

Step 4 − At each XkXk along the line, starting at k = 0, perform the following test –

If pk < 0, the next point to plot is (xk+1,yk)(xk+1,yk) and pk+1=pk+2dypk+1=pk+2dy

Otherwise, (xk,yk+1)(xk,yk+1)

pk+1=pk+2dy−2dxpk+1

=pk+2dy−2dx

Step 5 − Repeat step 4 dx–1dx–1 times.

CODE

#include<iostream>

#include<conio.h>

#include<graphics.h>

#include<math.h>

using namespace std;

int main(){

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    int x1,y1,x2,y2;

    cout<<"Enter the two points to draw line:-\t";

    cin>>x1>>y1>>x2>>y2;

    int dx=x2-x1;

    int dy=y2-y1;

    int p=(2\*dy)-dx;

    int x,y,Xen;

    if(x1>x2){

        x=x2;

        y=y2;

        Xen=x1;

    }

    else{

        x=x1;

        y=y1;

        Xen=x2;

    }

    initgraph(&gd, &gm, data);

    putpixel(round(x),round(y),WHITE);

    while(x<Xen){

        x=x+1;

        if(p<0){

            p=p+2\*dy;

        }

        else{

            y=y+1;

            p=p+2\*dy-2\*dx;

        }

        putpixel(round(x),round(y),WHITE);

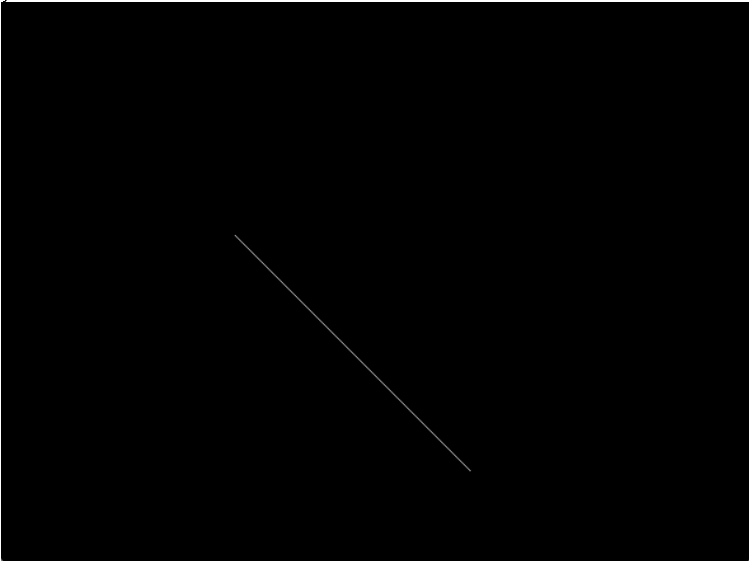
    }

    getch();

    closegraph();

}

OUTPUT:



PRACTICAL 3

Mid-Point Circle Algorithm: -

Step 1: Start.

Step 2: Declare x, y, r, xc , yc , P as variables, where (xc , yc) are coordinates of the center.

Step 3: Put x = 0 and y = r

Step 4: Repeat the steps while x ≤ y;

Step 5: Plot (x, y).

Step 6: if (P < 0):

Set P = P + 2x + 3

else if (P >= 0):

Set P = P + 2(x-y) + 5 y = y – 1

Step 7: Do x = x + 1

Step 8: End

CODE:  
#include<iostream>

#include<conio.h>

#include<graphics.h>

#include<math.h>

using namespace std;

void plotpixel(int x,int y,int Xc,int Yc){

    putpixel(Xc+x,Yc+y,WHITE);

    putpixel(Xc+y,Yc+x,WHITE);

    putpixel(Xc+y,Yc-x,WHITE);

    putpixel(Xc+x,Yc-y,WHITE);

    putpixel(Xc-x,Yc-y,WHITE);

    putpixel(Xc-y,Yc-x,WHITE);

    putpixel(Xc-y,Yc+x,WHITE);

    putpixel(Xc-x,Yc+y,WHITE);

}

int main(){

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    int Xc,Yc,r;

    cout<<"Enter the center of the circle:-\t";

    cin>>Xc>>Yc;

    cout<<"Enter the radius of the circle:-\t";

    cin>>r;

    float x=0,y=r;

    float p=5/4-r;

    initgraph(&gd, &gm, data);

    plotpixel(x,y,Xc,Yc);

    while(x<y){

        x=x+1;

        if(p<0){

            p=p+2\*x+1;

        }

        else{

            y=y-1;

            p=p+2\*x-2\*y+1;

        }

        plotpixel(x,y,Xc,Yc);

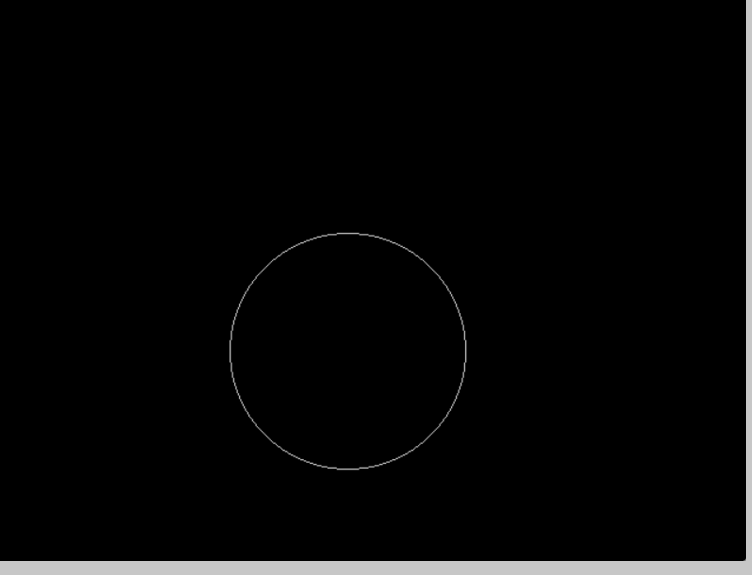
    }

    getch();

    closegraph();

}

OUTPUT



**PRACTICAL 4**

**Bresenham’s circle algorithm: -**

Step 1 − Get the coordinates of the center of the circle and radius, and store them in x, y, and R respectively. Set P=0 and Q=R.

Step 2 − Set decision parameter D = 3 – 2R.

Step 3 − Repeat through step-8 while P ≤ Q.

Step 4 − Call Draw Circle X,Y,P,QX,Y,P,Q.

Step 5 − Increment the value of P.

Step 6 − If D < 0 then D = D + 4P + 6.

Step 7 − Else Set R = R - 1, D = D + 4P−QP−Q + 10.

Step 8 − Call Draw Circle X,Y,P,QX,Y,P,Q.

Draw Circle Method(X, Y, P, Q).

Call Putpixel (X + P, Y + Q).

Call Putpixel (X - P, Y + Q).

Call Putpixel (X + P, Y - Q).

Call Putpixel (X - P, Y - Q).

Call Putpixel (X + Q, Y + P).

Call Putpixel (X - Q, Y + P).

Call Putpixel (X + Q, Y - P).

Call Putpixel (X - Q, Y - P).

CODE:

#include<iostream>

#include<conio.h>

#include<graphics.h>

#include<math.h>

using namespace std;

void draw\_Circle(int xc, int yc, int x, int y)

{

    putpixel(xc+x, yc+y, YELLOW);

    putpixel(xc-x, yc+y, YELLOW);

    putpixel(xc+x, yc-y, YELLOW);

    putpixel(xc-x, yc-y, YELLOW);

    putpixel(xc+y, yc+x, YELLOW);

    putpixel(xc-y, yc+x, YELLOW);

    putpixel(xc+y, yc-x, YELLOW);

    putpixel(xc-y, yc-x, YELLOW);

}

void circleBres(int xc, int yc, int r)

{

    int x = 0, y = r;

    int d = 3 - 2 \* r;

    draw\_Circle(xc, yc, x, y);

    while (y >= x)

    {

        x++;

        if (d > 0)

        {

            y--;

            d = d + 4 \* (x - y) + 10;

        }

        else

            d = d + 4 \* x + 6;

        draw\_Circle(xc, yc, x, y);

        }

}

int main()

{

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    int Xc,Yc,r;

    cout<<"Enter the center of the circle:-\t";

    cin>>Xc>>Yc;

    cout<<"Enter the radius of the circle:-\t";

    cin>>r;

    initgraph(&gd, &gm, data);

    circleBres(Xc, Yc, r);

   getch();

    closegraph();

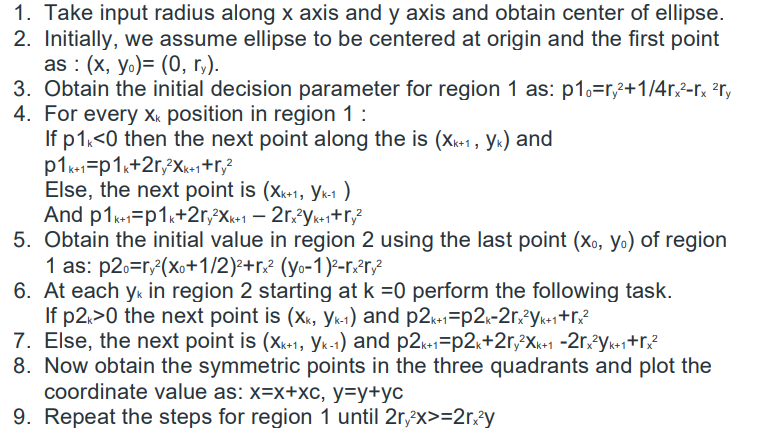
}

OUTPUT:



**PRACTICAL 5**

**Mid-Point Ellipse Algorithm: -**



CODE:  
#include<iostream>

#include<conio.h>

#include<graphics.h>

#include<math.h>

using namespace std;

int main(){

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    int xc,yc;

    cout<<"Enter the center xc and yc :";

    cin>>xc>>yc;

    int ry,rx;

    cout<<"Enter the value of minor(b) and major(a) axis :";

    cin>>ry>>rx;

    int p1=ry\*ry-rx\*rx\*ry+rx\*rx/4;

    int x,y;

    x=0,y=ry;

    initgraph(&gd, &gm, data);

        putpixel(x+xc,y+yc,WHITE);

        putpixel(-x+xc,y+yc,WHITE);

        putpixel(x+xc,-y+yc,WHITE);

        putpixel(-x+xc,-y+yc,WHITE);

    while(2\*ry\*ry\*x<2\*rx\*rx\*y){

        x=x+1;

        if(p1<0){

            p1=p1+2\*ry\*ry\*x+ry\*ry;

        }

        else{

            y=y-1;

            p1=p1+2\*ry\*ry\*x-2\*rx\*rx\*y+ry\*ry;

        }

      putpixel(x+xc,y+yc,WHITE);

        putpixel(-x+xc,y+yc,WHITE);

        putpixel(x+xc,-y+yc,WHITE);

        putpixel(-x+xc,-y+yc,WHITE);

    }

    int p2=ry\*ry\*(x+0.5)\*(x+0.5) +rx\*rx\*(y-1)\*(y-1)-rx\*rx\*ry\*ry;

    while(y>0){

        y=y-1;

        if(p2<0){

            x=x+1;

            p2=p2+2\*ry\*ry\*x-2\*rx\*rx\*y+rx\*rx;

        }

        else{

            p2=p2-2\*rx\*rx\*y+rx\*rx;

        }

      putpixel(x+xc,y+yc,WHITE);

        putpixel(-x+xc,y+yc,WHITE);

        putpixel(x+xc,-y+yc,WHITE);

        putpixel(-x+xc,-y+yc,WHITE);

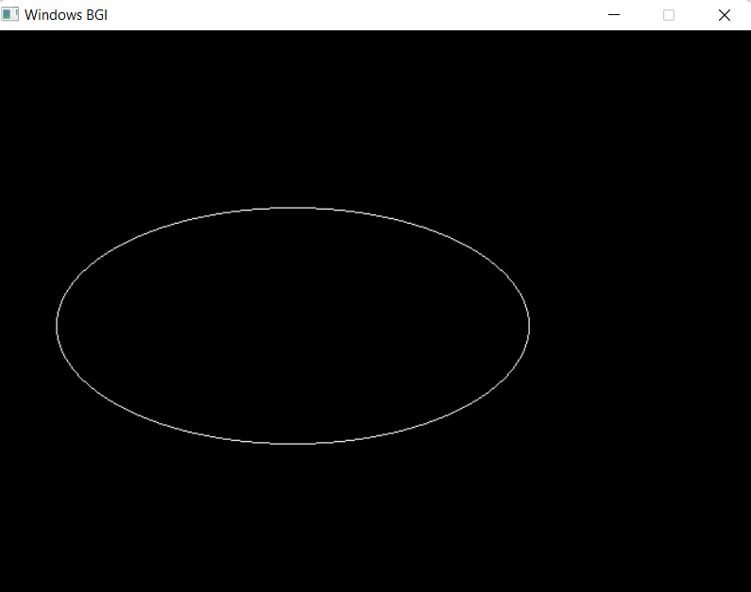
    }

     getch();

    closegraph();

}

CODE:

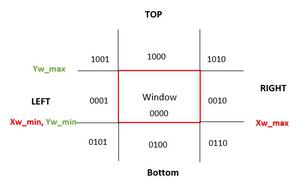


**PRACTICAL 6:**

**Cohen-Sutherland Algorithm: -**

In this algorithm, we are given 9 regions on the screen. Out of which one region is of the window and the rest 8 regions are around it given by 4 digit binary.  The division of the regions are based on (x\_max, y\_max) and (x\_min, y\_min).

The central part is the viewing region or window, all the lines which lie within this region are completely visible. A region code is always assigned to the endpoints of the given line.



CODE:

#include <iostream>

#include <graphics.h>

#include <conio.h>

#include <stdio.h>

#include <math.h>

using namespace std;

int main()

{

    int rcode\_begin[4] = {0, 0, 0, 0}, rcode\_end[4] = {0, 0, 0, 0}, region\_code[4];

    int W\_xmax, W\_ymax, W\_xmin, W\_ymin, flag = 0;

    float slope;

    int x, y, x1, y1, i, xc, yc;

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    initgraph(&gd, &gm, data);

    cout << " Cohen Sutherlsnd Line Clipping algorithm " << endl;

    cout << "Now, enter XMin, YMin =" << endl;

    cin >> W\_xmin, W\_ymin;

    cout << " First enter XMax, YMax =";

    cin >> W\_xmax, &W\_ymax;

    cout << " Please enter intial point x and y= ";

    cin >> x, y;

    cout << " Now, enter final point x1 and y1= ";

    cin >> x1, y1;

    cleardevice();

    rectangle(W\_xmin, W\_ymin, W\_xmax, W\_ymax);

    line(x, y, x1, y1);

    line(0, 0, 600, 0);

    line(0, 0, 0, 600);

    if (y > W\_ymax)

    {

        rcode\_begin[0] = 1; // Top

        flag = 1;

    }

    if (y < W\_ymin)

    {

        rcode\_begin[1] = 1; // Bottom

        flag = 1;

    }

    if (x > W\_xmax)

    {

        rcode\_begin[2] = 1; // Right

        flag = 1;

    }

    if (x < W\_xmin)

    {

        rcode\_begin[3] = 1; // Left

        flag = 1;

    }

    // end point of Line

    if (y1 > W\_ymax)

    {

        rcode\_end[0] = 1; // Top

        flag = 1;

    }

    if (y1 < W\_ymin)

    {

        rcode\_end[1] = 1; // Bottom

        flag = 1;

    }

    if (x1 > W\_xmax)

    {

        rcode\_end[2] = 1; // Right

        flag = 1;

    }

    if (x1 < W\_xmin)

    {

        rcode\_end[3] = 1; // Left

        flag = 1;

    }

    if (flag == 0)

    {

        cout << "No need of clipping as it is already in window";

    }

    flag = 1;

    for (i = 0; i < 4; i++)

    {

        region\_code[i] = rcode\_begin[i] && rcode\_end[i];

        if (region\_code[i] == 1)

            flag = 0;

    }

    if (flag == 0)

    {

        cout << " Line is completely outside the window" << endl;

    }

    else

    {

        slope = (float)(y1 - y) / (x1 - x);

        if (rcode\_begin[2] == 0 && rcode\_begin[3] == 1) // left

        {

            y = y + (float)(W\_xmin - x) \* slope;

            x = W\_xmin;

        }

        if (rcode\_begin[2] == 1 && rcode\_begin[3] == 0) // right

        {

            y = y + (float)(W\_xmax - x) \* slope;

            x = W\_xmax;

        }

        if (rcode\_begin[0] == 1 && rcode\_begin[1] == 0) // top

        {

            x = x + (float)(W\_ymax - y) / slope;

            y = W\_ymax;

        }

        if (rcode\_begin[0] == 0 && rcode\_begin[1] == 1) // bottom

        {

            x = x + (float)(W\_ymin - y) / slope;

            y = W\_ymin;

        }

        // end points

        if (rcode\_end[2] == 0 && rcode\_end[3] == 1) // left

        {

            y1 = y1 + (float)(W\_xmin - x1) \* slope;

            x1 = W\_xmin;

        }

        if (rcode\_end[2] == 1 && rcode\_end[3] == 0) // right

        {

            y1 = y1 + (float)(W\_xmax - x1) \* slope;

            x1 = W\_xmax;

        }

        if (rcode\_end[0] == 1 && rcode\_end[1] == 0) // top

        {

            x1 = x1 + (float)(W\_ymax - y1) / slope;

            y1 = W\_ymax;

        }

        if (rcode\_end[0] == 0 && rcode\_end[1] == 1) // bottom

        {

            x1 = x1 + (float)(W\_ymin - y1) / slope;

            y1 = W\_ymin;

        }

    }

    delay(1000);

    clearviewport();

    rectangle(W\_xmin, W\_ymin, W\_xmax, W\_ymax);

    line(0, 0, 600, 0);

    line(0, 0, 0, 600);

    setcolor(RED);

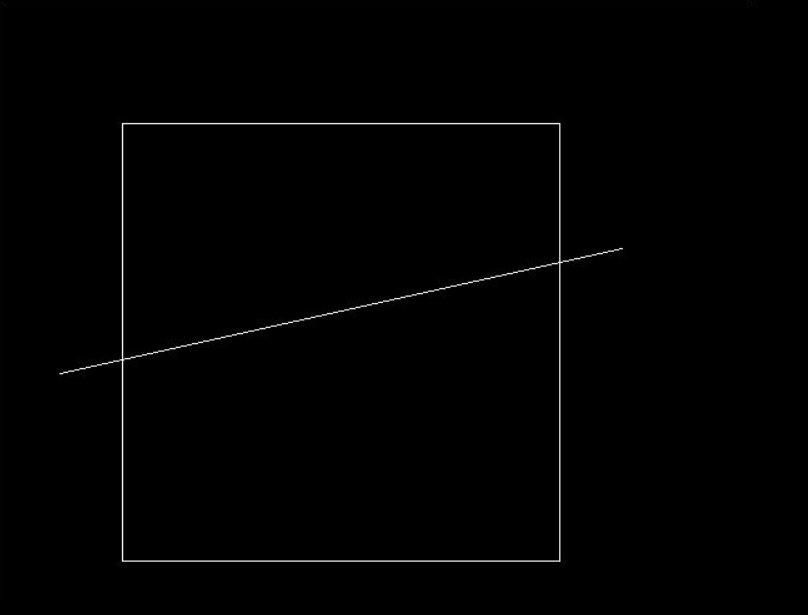
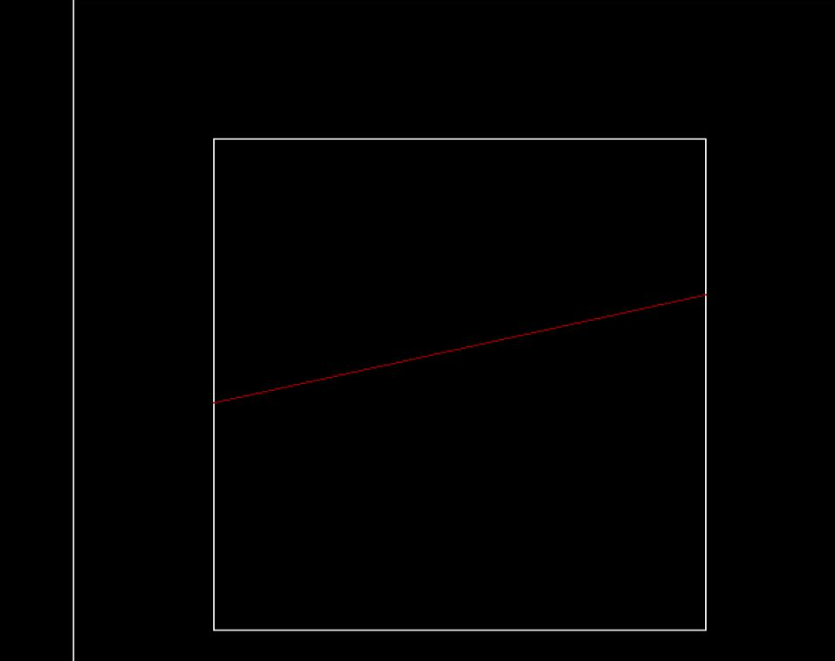
    line(x, y, x1, y1);

    getch();

    closegraph();

}

**OUTPUT**

** **

**PRACTICAL 7:**

***Program to rotate line about origin.***

* Xnew = Xold x cosθ – Yold x sinθ
* Ynew = Xold x sinθ + Yold x cosθ

**CODE:**

#include<iostream>

#include<graphics.h>

#include<math.h>

using namespace std;

int main(){

    int x,y,a;

    cout<<"Enter the point you want to rotate:- ";

    cin>>x>>y;

    cout<<"\nEnter the angle by which you want to rotate:- ";

    cin>>a;

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    initgraph(&gd, &gm, data);

    line(0,0,x,y);

    delay(1000);

    float x\_n=x\*(cos((a\*3.14)/180))-y\*(sin((a\*3.14)/180));

    float y\_n=y\*(cos((a\*3.14)/180))+x\*(sin((a\*3.14)/180));

    setcolor(RED);

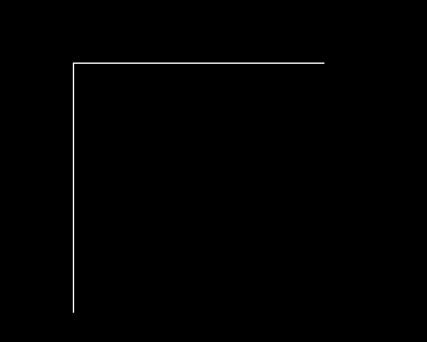
    line(0,0,round(x\_n),round(y\_n));

    getch();

    closegraph();

}

**OUTPUT:**



**PRACTICAL 8:**

***PROGRAM TO ROTATE A GIVEN SQUARE BY AN ANGLE***

Every coordinate of the square should be rotated by given angle theta.

* Xnew = Xold x cosθ – Yold x sinθ
* Ynew = Xold x sinθ + Yold x cosθ

**CODE:**

#include<graphics.h>

#include<bits/stdc++.h>

using namespace std;

int main(){

float x1,y1,x2,y2,a;

cout<<"Enter the end points of a diagonal of a square:- ";

cin>>x1>>y1>>x2>>y2;

cout<<"Enter the angle:- ";

cin>>a;

float x3=x1,y3=y2,x4=x2,y4=y1;

float n=(x2+x1)/2;

float m=(y2+y1)/2;

int gd = DETECT, gm;

char data[] = "C:\\MinGW\\lib\\libbgi.a";

initgraph(&gd, &gm, data);

line(x1,y1,x3,y3);

line(x1,y1,x4,y4);

line(x2,y2,x3,y3);

line(x2,y2,x4,y4);

delay(1000);

float x5=n+(x1-n)\*(cos((a\*3.14)/180))-(y1-m)\*(sin((a\*3.14)/180));

float y5=m+(y1-m)\*(cos((a\*3.14)/180))+(x1-n)\*(sin((a\*3.14)/180));

float x6=n+(x2-n)\*(cos((a\*3.14)/180))-(y2-m)\*(sin((a\*3.14)/180));

float y6=m+(y2-m)\*(cos((a\*3.14)/180))+(x2-n)\*(sin((a\*3.14)/180));

float x7=n+(x3-n)\*(cos((a\*3.14)/180))-(y3-m)\*(sin((a\*3.14)/180));

float y7=m+(y3-m)\*(cos((a\*3.14)/180))+(x3-n)\*(sin((a\*3.14)/180));

float x8=n+(x4-n)\*(cos((a\*3.14)/180))-(y4-m)\*(sin((a\*3.14)/180));

float y8=m+(y4-m)\*(cos((a\*3.14)/180))+(x4-n)\*(sin((a\*3.14)/180));

setcolor(RED);

line(round(x5),round(y5),round(x7),round(y7));

line(round(x5),round(y5),round(x8),round(y8));

line(round(x6),round(y6),round(x7),round(y7));

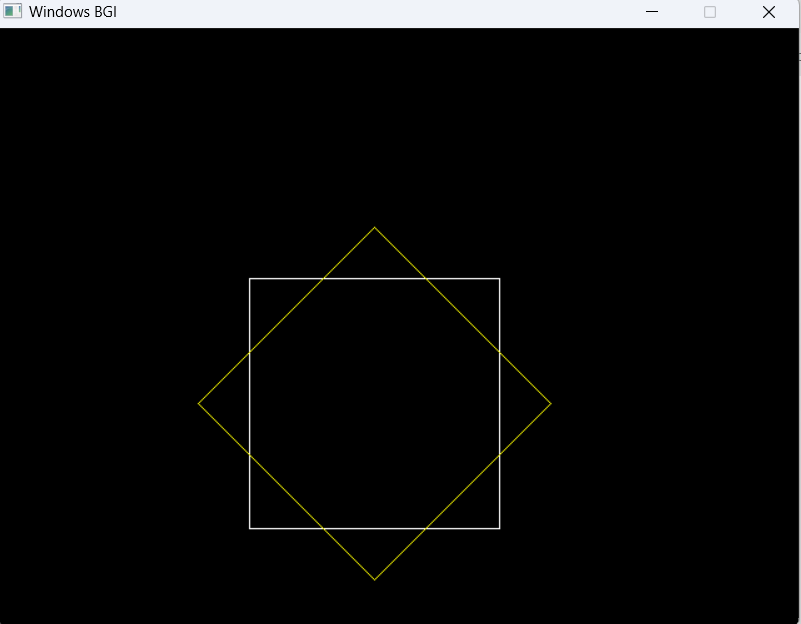
line(round(x6),round(y6),round(x8),round(y8));

getch();

closegraph();

}

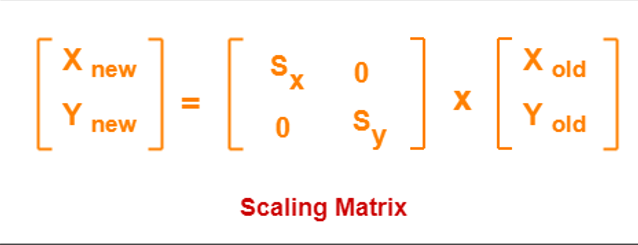
**OUTPUT:**



**PRACTICAL 9:**

***PROGRAM TO SCALE A GIVEN SQUARE BY GIVIEN SCALING FACTOR***

* Scaling may be used to increase or reduce the size of object.
* Scaling subject the coordinate points of the original object to change.
* Scaling factor determines whether the object size is to be increased or reduced.
* If scaling factor > 1, then the object size is increased.
* If scaling factor < 1, then the object size is reduced.

******

***CODE:***

#include <iostream>

#include <conio.h>

#include <graphics.h>

#include <math.h>

using namespace std;

int main()

{

    int x1, y1, x2,y2;

    int scaling\_x, scaling\_y;

    cout << "Enter the end points of a diagonal of a square:- ";

    cin >> x1 >> y1 >> x2 >> y2;

    cout << "Enter the scaling factor of X and Y:- ";

    cin >> scaling\_x>> scaling\_y;

    float x3 = x1, y3 = y2, x4 = x2, y4 = y1;

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    initgraph(&gd, &gm, data);

    line(x1,y1,x3,y3);

    line(x1,y1,x4,y4);

    line(x2,y2,x3,y3);

    line(x2,y2,x4,y4);

    delay(1000);

    float x5 = x1 \* scaling\_x;

    float y5 = y1 \* scaling\_y;

    float x6 = x2 \* scaling\_x;

    float y6 = y2 \* scaling\_y;

    float x7 = x3 \* scaling\_x;

    float y7 = y3 \* scaling\_y;

    float x8 = x4 \* scaling\_x;

    float y8 = y4 \* scaling\_y;

    setcolor(YELLOW);

    line(round(x5), round(y5), round(x7), round(y7));

    line(round(x5), round(y5), round(x8), round(y8));

    line(round(x6), round(y6), round(x7), round(y7));

    line(round(x6), round(y6), round(x8), round(y8));

    getch();

    closegraph();

}

OUTPUT:

