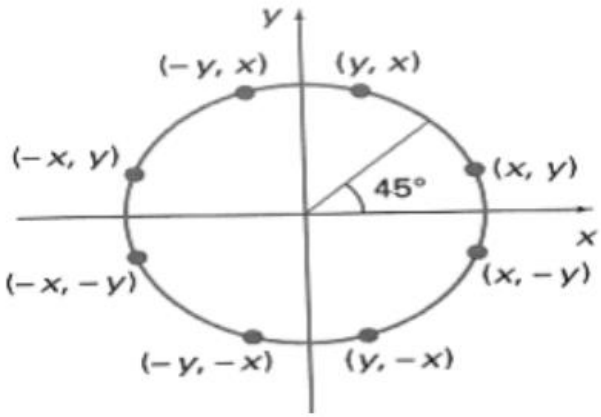
Aim: To implement midpoint circle algorithm.

Objective:

Draw a circle using midpoint circle drawing algorithm by determining the points needed for rasterizing a circle. The mid-point algorithm to calculate all the perimeter points of the circle in the first octant and then print them along with their mirror points in the other octants.

Theory:

The shape of the circle is similar in each quadrant. We can generate the points in one section and the points in other sections can be obtained by considering the symmetry about x-axis and y-axis.



The equation of circle with center at origin is x 2 + y 2 = r 2

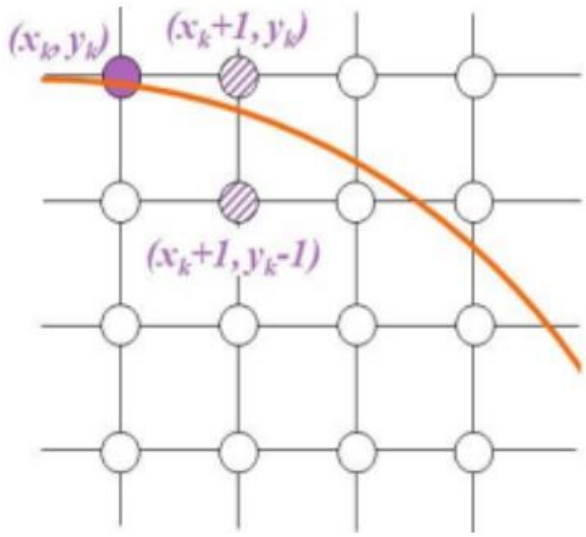
Let the circle function is f circle (x, y) -

 is < 0, if (x, y) is inside circle boundary,

 is = 0, if (x, y) is on circle boundary,

 is > 0, if (x, y) is outside circle boundary.

Consider the pixel at (xk, yk) is plotted,



Now the next pixel along the circumference of the circle will be either (xk + 1, yk) or (xk + 1, yk – 1) whichever is closer to the circle boundary.

Let the decision parameter pk is equal to the circle function evaluate at the mid-point between two pixels.

If pk < 0, the midpoint is inside the circle and the pixel at yk is closer to the circle boundary.

Otherwise, the midpoint is outside or on the circle boundary and the pixel at yk – 1 is closer to the circle boundary.

Algorithm –

```
{
    x =0 , y = ry;
    p = ry*ry-ry*rx*rx+rx*rx/4;
    dx = 2*x*ry*ry;
    dy = 2*y*rx*rx;

    while(dx<dy)
    {
        putpixel(x,y);
```

```
        if(p<0)
        {
            x=x+1;
            p=p+2*ry*ry*xk+1 + ry*ry;
        }
        else
        {
            x=x+1;
            y=y-1;
            p=p+2*ry*ry*xk+1 + 2ry*ry -2*x*x;
        }
    }

    while(dx<dy)
    {
        putpixel(x,y);
        if(p<0)
        {
            p=p-2*rx*rx*yk+1 + rx*rx;
            y=y-1;
        }
        else
        {
            x=x+1;
            y=y-1;
```

```
            p=p+2*ry*ry*xk+1;

        }

    }

}
```

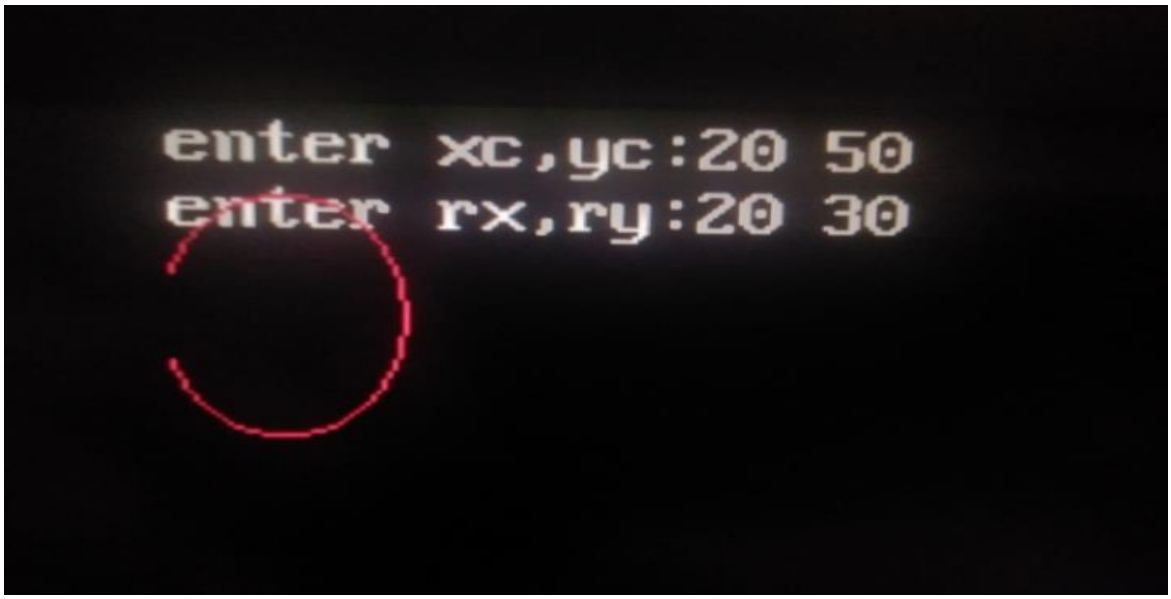
Program –

```
#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int p,rx,ry,xc,yc,x,y;

int gd=DETECT,gm;

initgraph(&gd,&gm,"c:\\TURBOC3\\BGI");

printf("enter xc,yc:");

scanf("%d%d",&xc,&yc);

printf("enter rx,ry:");

scanf("%d%d",&rx,&ry);

x=0;

y=ry;

p=(ry*ry)-(ry*rx*rx)+((rx*rx)/4);

while((2*x*ry*ry)<(2*y*rx*rx))

{putpixel(xc+x,yc+y,RED);

putpixel(xc-x,yc+y,RED);

putpixel(xc+x,yc-y,RED);
```

```
putpixel(xc-x,yc-y,RED);

if(p<0)

{

p=p+(2*ry*ry*x)+(ry*ry);

x=x+1;

}

else

{

p=p+(2*ry*ry*x)+(ry*ry)-(2*rx*rx*y);

x=x+1;

y=y-1;

}

}

p=(x+0.5)*(x+0.5)*(ry*ry)+(rx*rx)*(y-1)*(y-1)-(rx*rx*ry*ry);

while(y>=0)

{

putpixel(xc+x,yc+y,RED);

putpixel(xc-x,yc+y,RED);

putpixel(xc+x,yc-y,RED);

putpixel(xc-x,yc-y,RED);

if(p<0)

{

p=p+(rx*rx)-(2*rx*rx*y);

y=y-1;

}
```

```
else

{

p=p+(rx*rx)-(2*rx*rx*y)+(2*ry*ry*x);

y=y-1;

x=x+1;

}

}

getch();

closegraph();

}
```

output –



Conclusion: Comment on

1. Fast or slow : - Fast

2. Draw one arc only and repeat the process in 8 quadrants : - Yes, the algorithm draws only a single octant of the circle (usually the top-right quadrant) and then replicates that pattern in all eight octants by changing the signs of coordinates.

3. Difference with line drawing method : - The midpoint circle algorithm is specialized for drawing circles. It adjusts the decision parameter based on the location of the pixel on the circle relative to its midpoint. This is different from line drawing algorithms like Bresenham's, which focus on incremental choices for the next pixel on a line. The algorithms share similarities, but the circle algorithm is tailored to the unique properties of circular shapes.