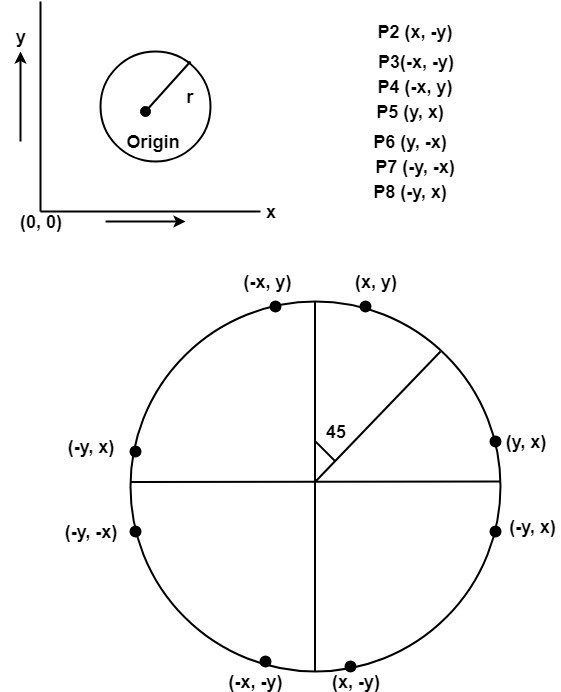
**Defining a Circle:**

Circle is an eight-way symmetric figure. The shape of circle is the same in all quadrants. In each quadrant, there are two octants. If the calculation of the point of one octant is done, then the other seven points can be calculated easily by using the concept of eight-way symmetry.

For drawing, circle considers it at the origin. If a point is P1(x, y), then the other seven points will be



So we will calculate only 45°arc. From which the whole circle can be determined easily.

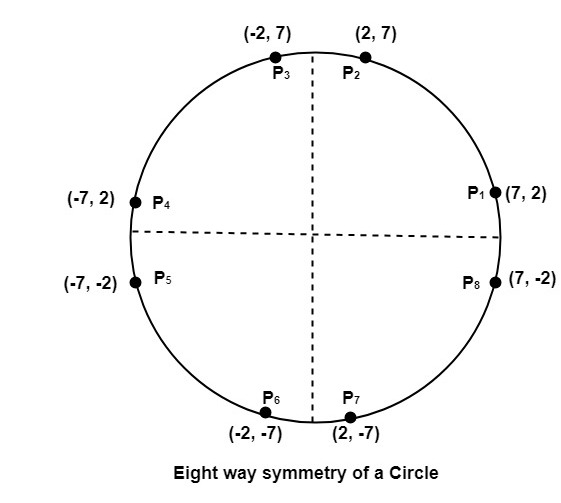
If we want to display circle on screen then the putpixel function is used for eight points as shown below:

          putpixel (x, y, color)  
          putpixel (x, -y, color)  
          putpixel (-x, y, color)  
          putpixel (-x, -y, color)  
          putpixel (y, x, color)  
          putpixel (y, -x, color)  
          putpixel (-y, x, color)  
          putpixel (-y, -x, color)

**Example:** Let we determine a point (2, 7) of the circle then other points will be (2, -7), (-2, -7), (-2, 7), (7, 2), (-7, 2), (-7, -2), (7, -2)

These seven points are calculated by using the property of reflection. The reflection is accomplished in the following way:

The reflection is accomplished by reversing x, y co-ordinates.



There are two standards methods of mathematically defining a circle centered at the origin.

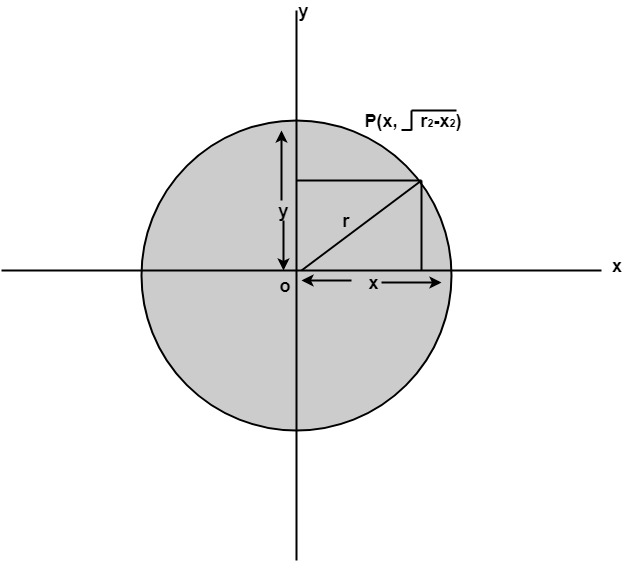
1. Defining a circle using Polynomial Method
2. Defining a circle using Polar Co-ordinates

# Defining a circle using Polynomial Method:

The first method defines a circle with the second-order polynomial equation as shown in fig:

                    y2=r2-x2  
Where x = the x coordinate  
          y = the y coordinate  
          r = the circle radius

With the method, each x coordinate in the sector, from 90° to 45°, is found by stepping x from 0 to Defining a circle using Polynomial Method& each y coordinate is found by evaluating Defining a circle using Polynomial Methodfor each step of x.



## Algorithm:

**Step1:** Set the initial variables  
          r = circle radius  
          (h, k) = coordinates of circle center  
                x=o  
                I = step size  
                xend= Defining a circle using Polynomial Method

**Step2:** Test to determine whether the entire circle has been scan-converted.

If x > xend then stop.

**Step3:** Compute y = Defining a circle using Polynomial Method

**Step4:** Plot the eight points found by symmetry concerning the center (h, k) at the current (x, y) coordinates.

                Plot (x + h, y +k)          Plot (-x + h, -y + k)  
                Plot (y + h, x + k)          Plot (-y + h, -x + k)  
                Plot (-y + h, x + k)          Plot (y + h, -x + k)  
                Plot (-x + h, y + k)          Plot (x + h, -y + k)

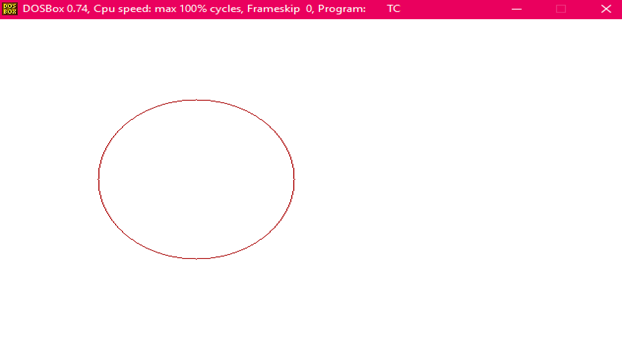
**Step5:** Increment x = x + i

**Step6:** Go to step (ii).

### Program to draw a circle using Polynomial Method:

1. #include<graphics.h>
2. #include<conio.h>
3. #include<math.h>
4. voidsetPixel(int x, int y, int h, int k)
5. {
6. putpixel(x+h, y+k, RED);
7. putpixel(x+h, -y+k, RED);
8. putpixel(-x+h, -y+k, RED);
9. putpixel(-x+h, y+k, RED);
10. putpixel(y+h, x+k, RED);
11. putpixel(y+h, -x+k, RED);
12. putpixel(-y+h, -x+k, RED);
13. putpixel(-y+h, x+k, RED);
14. }
15. main()
16. {
17. intgd=0, gm,h,k,r;
18. double x,y,x2;
19. h=200, k=200, r=100;
20. initgraph(&gd, &gm, "C:\\TC\\BGI ");
21. setbkcolor(WHITE);
22. x=0,y=r;
23. x2 = r/sqrt(2);
24. while(x<=x2)
25. {
26. y = sqrt(r\*r - x\*x);
27. setPixel(floor(x), floor(y), h,k);
28. x += 1;
29. }
30. getch();
31. closegraph();
32. return 0;
33. }

**Output:**

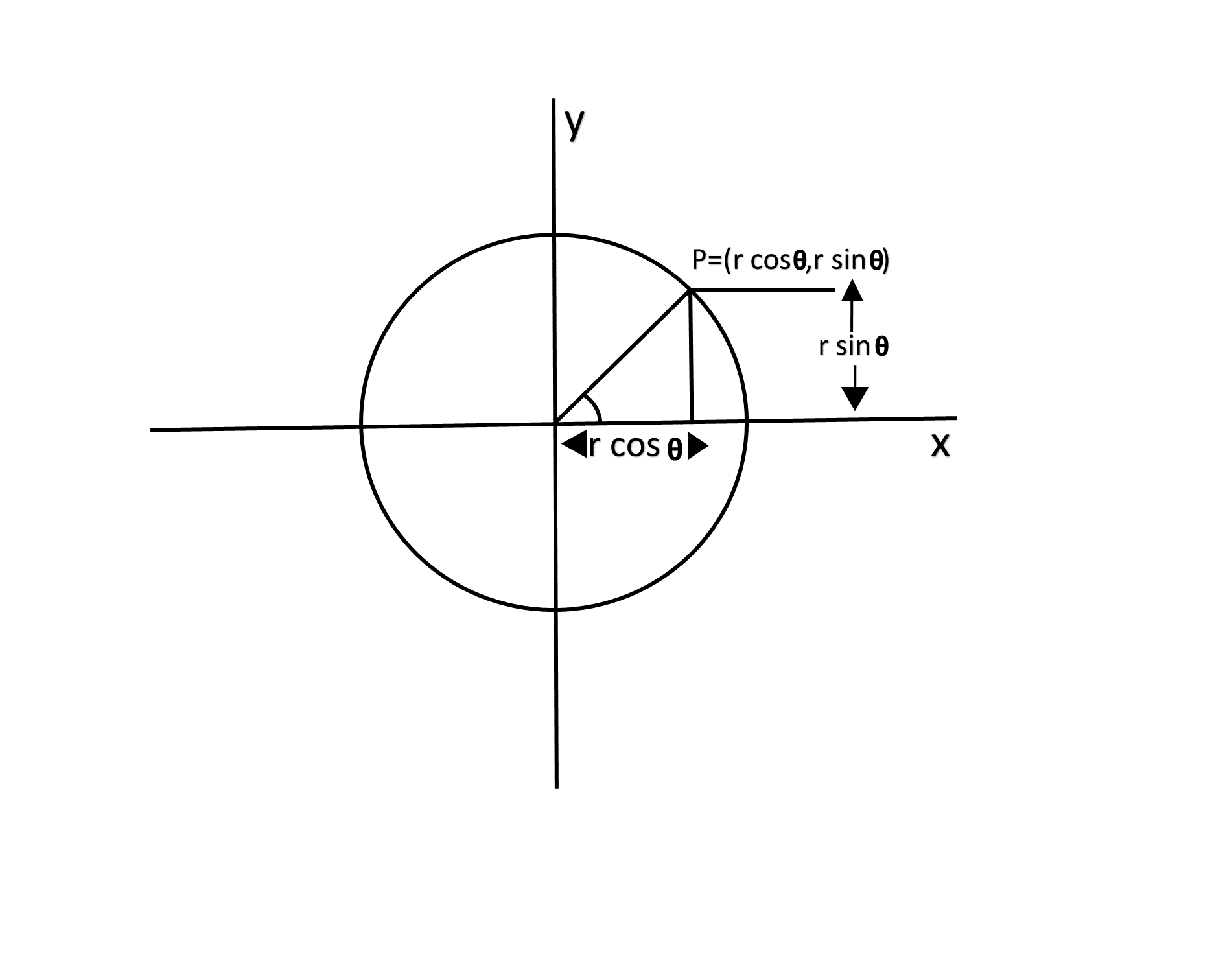


# Defining a circle using Polar Co-ordinates :

The second method of defining a circle makes use of polar coordinates as shown in fig:

            x=r cos θ             y = r sin θ  
Where θ=current angle  
r = circle radius  
x = x coordinate  
y = y coordinate

By this method, θ is stepped from 0 to Defining a circle using Polar Co-ordinates& each value of x & y is calculated.



## Algorithm:

**Step1:** Set the initial variables:

            r = circle radius  
            (h, k) = coordinates of the circle center  
                i = step size  
            θ\_end=Defining a circle using Polar Co-ordinates  
            θ=0

**Step2:** If θ>θendthen stop.

**Step3:** Compute

            x = r \* cos θ            y=r\*sin?θ

**Step4:** Plot the eight points, found by symmetry i.e., the center (h, k), at the current (x, y) coordinates.

Plot (x + h, y +k)             Plot (-x + h, -y + k)  
Plot (y + h, x + k)             Plot (-y + h, -x + k)  
Plot (-y + h, x + k)             Plot (y + h, -x + k)  
Plot (-x + h, y + k)             Plot (x + h, -y + k)

**Step5:** Increment θ=θ+i

**Step6:** Go to step (ii).

### Program to draw a circle using Polar Coordinates:

1. #include <graphics.h>
2. #include <stdlib.h>
3. #define color 10
4. void eightWaySymmetricPlot(int xc,int yc,int x,int y)
5. {
6. putpixel(x+xc,y+yc,color);
7. putpixel(x+xc,-y+yc,color);
8. putpixel(-x+xc,-y+yc,color);
9. putpixel(-x+xc,y+yc,color);
10. putpixel(y+xc,x+yc,color);
11. putpixel(y+xc,-x+yc,color);
12. putpixel(-y+xc,-x+yc,color);
13. putpixel(-y+xc,x+yc,color);
14. }
15. void PolarCircle(int xc,int yc,int r)
16. {
17. int x,y,d;
18. x=0;
19. y=r;
20. d=3-2\*r;
21. eightWaySymmetricPlot(xc,yc,x,y);
22. while(x<=y)
23. {
24. if(d<=0)
25. {
26. d=d+4\*x+6;
27. }
28. else
29. {
30. d=d+4\*x-4\*y+10;
31. y=y-1;
32. }
33. x=x+1;
34. eightWaySymmetricPlot(xc,yc,x,y);
35. }
36. }
37. int main(void)
38. {
39. int gdriver = DETECT, gmode, errorcode;
40. int xc,yc,r;
41. initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");
42. errorcode = graphresult();
43. if (errorcode != grOk)
44. {
45. printf("Graphics error: %s\n", grapherrormsg(errorcode));
46. printf("Press any key to halt:");
47. getch();
48. exit(1);
49. }
50. printf("Enter the values of xc and yc ,that is center points of circle : ");
51. scanf("%d%d",&xc,&yc);
52. printf("Enter the radius of circle : ");
53. scanf("%d",&r);
54. PolarCircle(xc,yc,r);
55. getch();
56. closegraph();
57. return 0;
58. }

**Output:**

