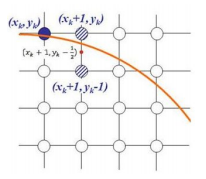
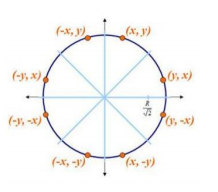
**Practical No 3**

**B) Implementation of Midpoint circle drawing algorithm.**

**Aim: Write a program to implementation of Midpoint circle drawing algorithm.**

**Theory:**

This algorithm draws all eight octants simultaneously, starting from each cardinal direction (0°, 90°, 180°, 270°) and extends both ways to reach the nearest multiple of 45° (45°, 135°, 225°, 315°). You can determine where to stop because when y = x, you have reached 45°. The reason for using these angles is shown in the above picture: As you increase y, you do not skip nor repeat any y value until you get to 45°. So, during the while loop, y increments by 1 each iteration, and x decrements by 1 on occasion, never exceeding 1 in one iteration. This changes at 45° because that is the point where the tangent is rise=run. Whereas rise>run before and rise<run after.



The objective of the algorithm is to find a path through the pixel grid using pixels which are as close as possible to solutions of x2+y2=r2. At each step, the path is extended by choosing the adjacent pixel which satisfies x2+y2<=r2 but maximizes x2+y2. Since the candidate pixels are adjacent, the arithmetic to calculate the latter expression is simplified, requiring only bit shifts and additions. This algorithm starts with the circle equation. For simplicity, assume the center of (0,0). We consider first only the first octant and draw a curve which starts at point (r,0) and proceeds counter clockwise, reaching the angle of 45. The "fast" direction here (the basis vector with the greater increase in value) is the y direction. The algorithm always takes a step in the positive y direction (upwards), and occasionally takes a step in the "slow" direction (the negative x direction). From the circle equation we obtain the transformed equation x2+y2-r2=0, where r2 is computed only a single time during initialization.

**Algorithm:**

* Get the value of radius (r) and set the initial values x0 and y0 to (0,r).
* Set the initial decision parameter p0 to p0 = Fcircle(1,r-1/2) = 1+(r-1/2)-r2 = 5/4-r.
* At each xi position starting at i=0 perform the following test:

1. If pi<0 the next point is (xi+1,yi) and pi+1=pi+2xi+1+1.

2. Else if pi>= 0 the next point is (xi+1,yi-1) and pi=pi+2xi+1+1-2yi+1.

* Determine the symmetry points in other octants.
* Move pixel positions (x,y) onto the circular path centred on (xc,yc) and plot the co-ordinates x=x+xc, y=y+yc.
* Repeat the steps 3-5 until x>=y.

**Conclusion: We have implemented Midpoint circle drawing algorithm.**

**Code:**

#include<iostream.h>

#include<graphics.h>

#include<conio.h>

#include<dos.h>

void drawcircle(int x0, int y0, int radius)

{

int x = radius;

int y = 0;

int err = 0;

while (x >= y)

{

putpixel(x0 + x, y0 + y, 15);

putpixel(x0 + y, y0 + x, 15);

putpixel(x0 - y, y0 + x, 15);

putpixel(x0 - x, y0 + y, 15);

putpixel(x0 - x, y0 - y, 15);

putpixel(x0 - y, y0 - x, 15);

putpixel(x0 + y, y0 - x, 15);

putpixel(x0 + x, y0 - y, 15);

if (err <= 0)

{

y += 1;

err += 2\*y + 1;

}

if (err > 0)

{

x -= 1;

err -= 2\*x + 1;

}

delay(100);

}

}

int main()

{

int gdriver=DETECT, gmode, error, x, y, r;

initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");

cout<<"\nMidpoint circle drawing algorithm \n";

cout<<"Enter radius of circle: ";

cin>>r;

cout<<"\nEnter co-ordinates of center(x and y): ";

cin>>x>>y;

drawcircle(x, y, r);

getch();

return 0;

}

**Output:**

