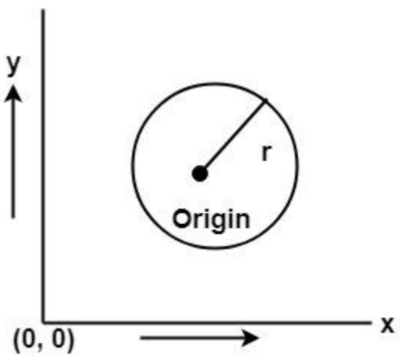
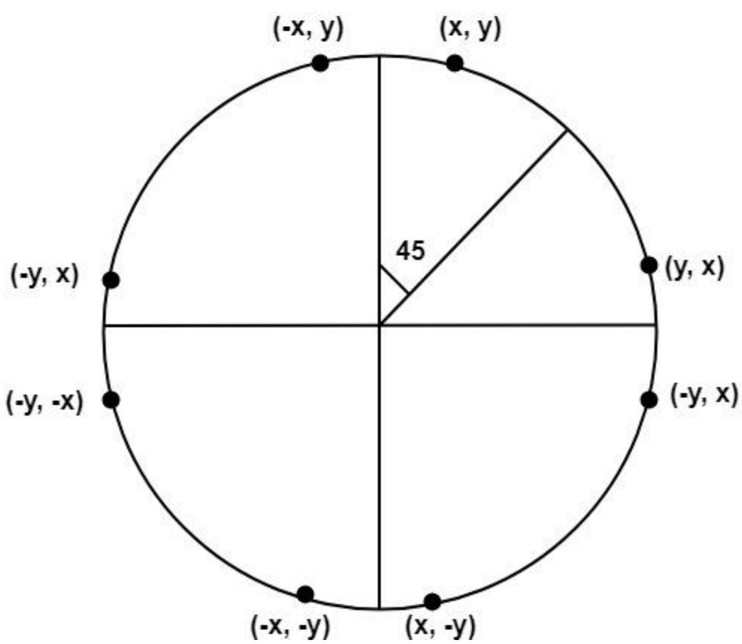
Defining a Circle:

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 $P_1(x, y)$, then the other seven points will be



P2 (x, -y)
P3(-x, -y)
P4 (-x, y)
P5 (y, x)
P6 (y, -x)
P7 (-y, -x)
P8 (-y, x)



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

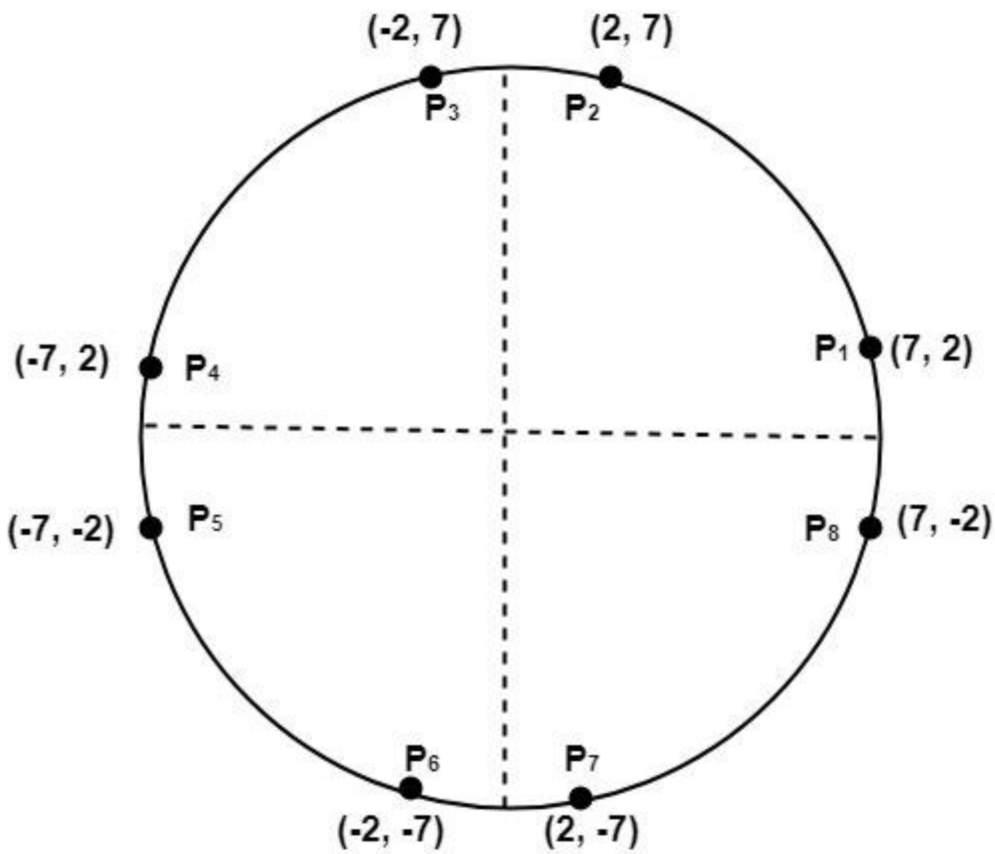
If we want to display circle on screen then the putpixel function is used for eight points as shown below:

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.



Eight way symmetry of a Circle

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:

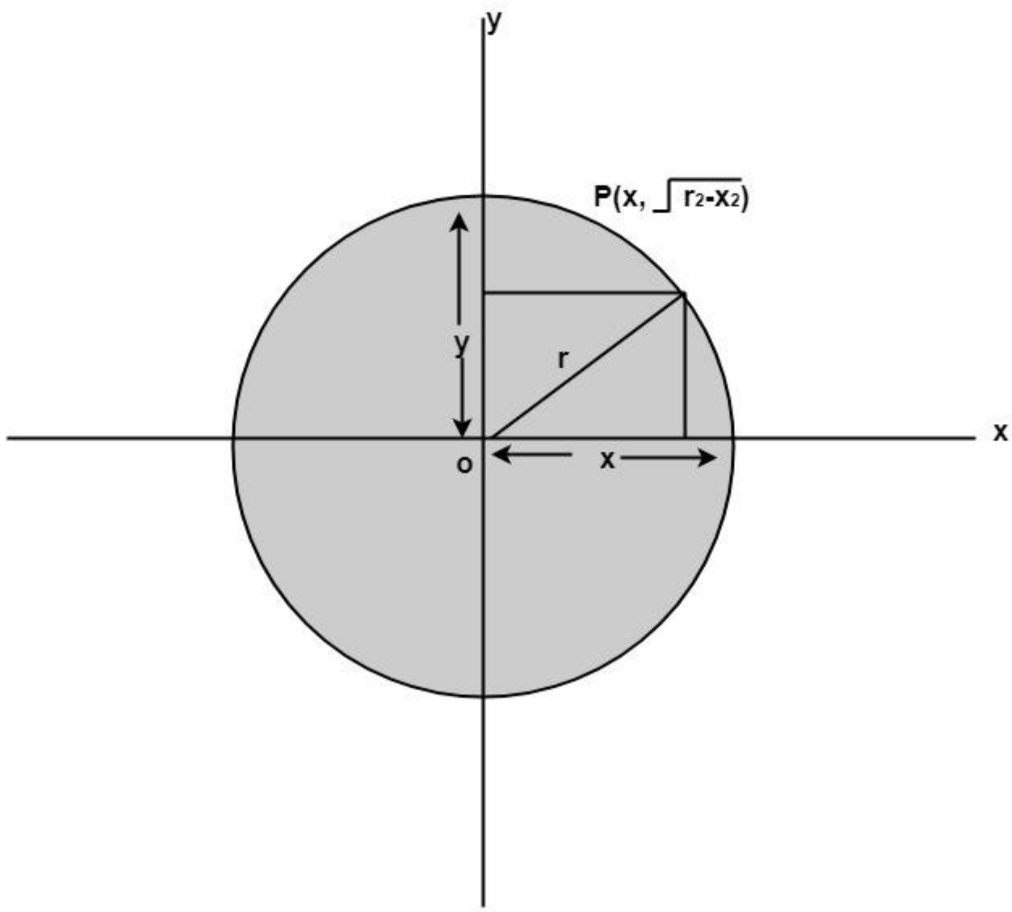
$$y^2=r^2-x^2$$

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 $\frac{r}{\sqrt{2}}$ & each y coordinate is found by evaluating $\sqrt{r^2-x^2}$ for each step of x.



Algorithm:

Step1: Set the initial variables

r = circle radius

(h, k) = coordinates of circle center

x=o

I = step size

$x_{end} = \frac{r}{\sqrt{2}}$

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

If $x > x_{end}$ then stop.

Step3: Compute y = $\sqrt{r^2 - x^2}$

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

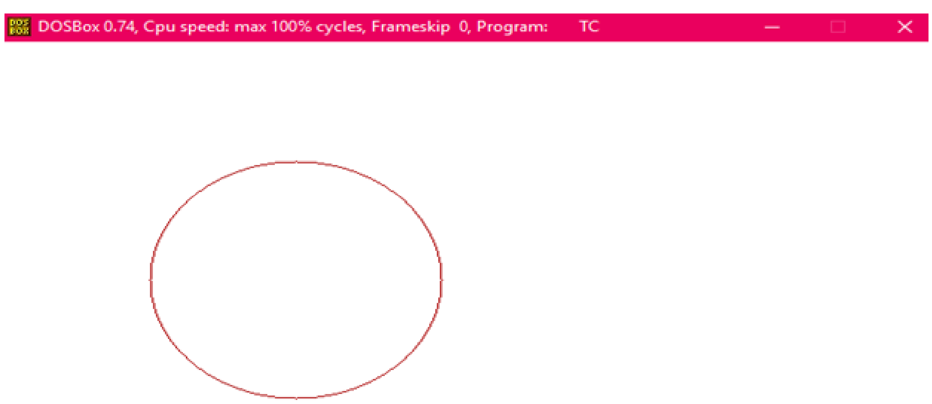
Step6: Go to step (ii).

Program to draw a circle using Polynomial Method:

```
#include<graphics.h>
#include<conio.h>
#include<math.h>
voidsetPixel(int x, int y, int h, int k)
{
    putpixel(x+h, y+k, RED);
    putpixel(x+h, -y+k, RED);
    putpixel(-x+h, -y+k, RED);
    putpixel(-x+h, y+k, RED);
    putpixel(y+h, x+k, RED);
    putpixel(y+h, -x+k, RED);
    putpixel(-y+h, -x+k, RED);
```

```
    putpixel(-y+h, x+k, RED);
}
main()
{
    intgd=0, gm,h,k,r;
    double x,y,x2;
    h=200, k=200, r=100;
    initgraph(&gd, &gm, "C:\\TC\\BGI ");
    setbkcolor(WHITE);
    x=0,y=r;
    x2 = r/sqrt(2);
    while(x<=x2)
    {
        y = sqrt(r*r - x*x);
        setPixel(floor(x), floor(y), h,k);
        x += 1;
    }
    getch();
    closegraph();
    return 0;
}
```

Output:



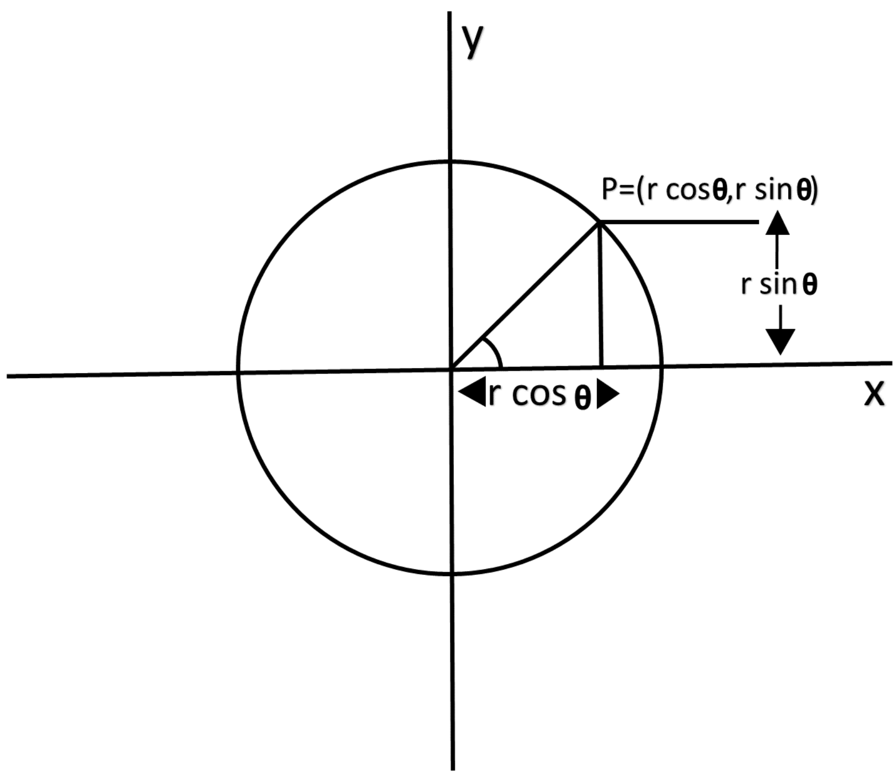
Defining a circle using Polar Co-ordinates :

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

$x = r \cos \theta \qquad y = r \sin \theta$

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

By this method, θ is stepped from 0 to $\frac{\pi}{4}$ & 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= $(\frac{22}{7})/4$
θ=0

Step2: If $\theta>\theta_{end}$then 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).

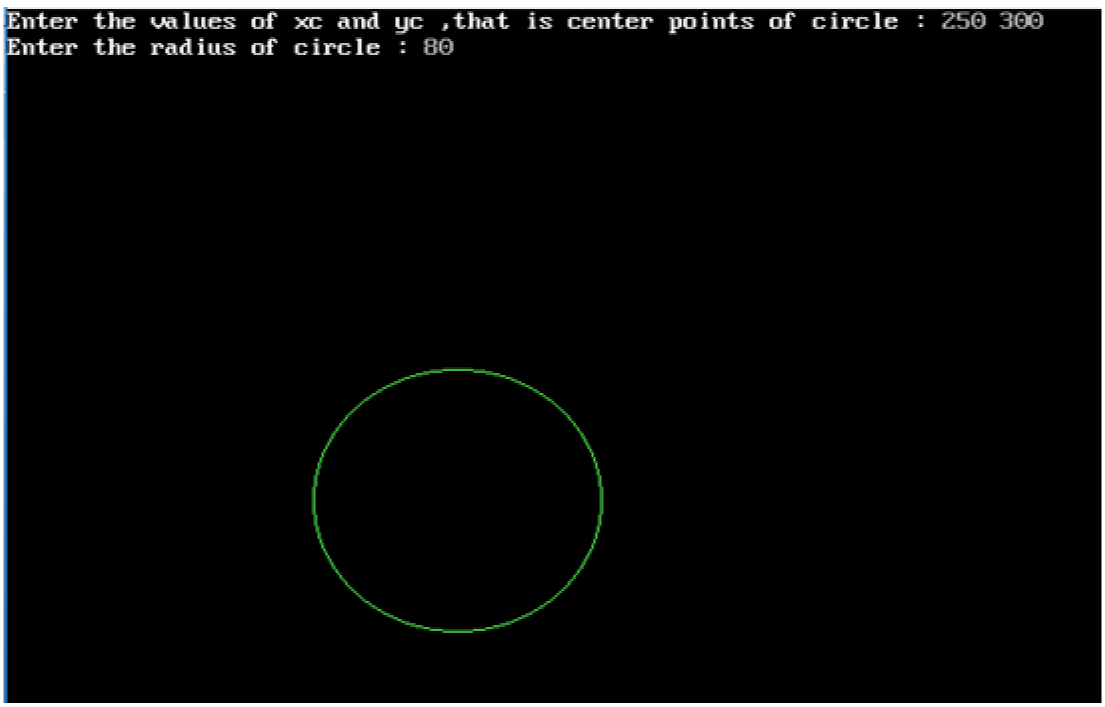
Program to draw a circle using Polar Coordinates:

```
#include <graphics.h>
#include <stdlib.h>
#define color 10
void eightWaySymmetricPlot(int xc,int yc,int x,int y)
{
    putpixel(x+xc,y+yc,color);
    putpixel(x+xc,-y+yc,color);
    putpixel(-x+xc,-y+yc,color);
    putpixel(-x+xc,y+yc,color);
    putpixel(y+xc,x+yc,color);
    putpixel(y+xc,-x+yc,color);
    putpixel(-y+xc,-x+yc,color);
```

```
    putpixel(-y+xc,x+yc,color);
}
void PolarCircle(int xc,int yc,int r)
{
    int x,y,d;
    x=0;
    y=r;
    d=3-2*r;
    eightWaySymmetricPlot(xc,yc,x,y);
    while(x<=y)
    {
        if(d<=0)
        {
            d=d+4*x+6;
        }
        else
        {
            d=d+4*x-4*y+10;
            y=y-1;
        }
        x=x+1;
        eightWaySymmetricPlot(xc,yc,x,y);
    }
}
int main(void)
{
    int gdriver = DETECT, gmode, errorcode;
    int xc,yc,r;
    initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");
errorcode = graphresult();
if (errorcode != grOk)
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);
    }
printf("Enter the values of xc and yc ,that is center points of circle : ");
```

```
scanf("%d%d",&xc,&yc);
printf("Enter the radius of circle : ");
scanf("%d",&r);
PolarCircle(xc,yc,r);
getch();
closegraph();
return 0;
}
```

Output:



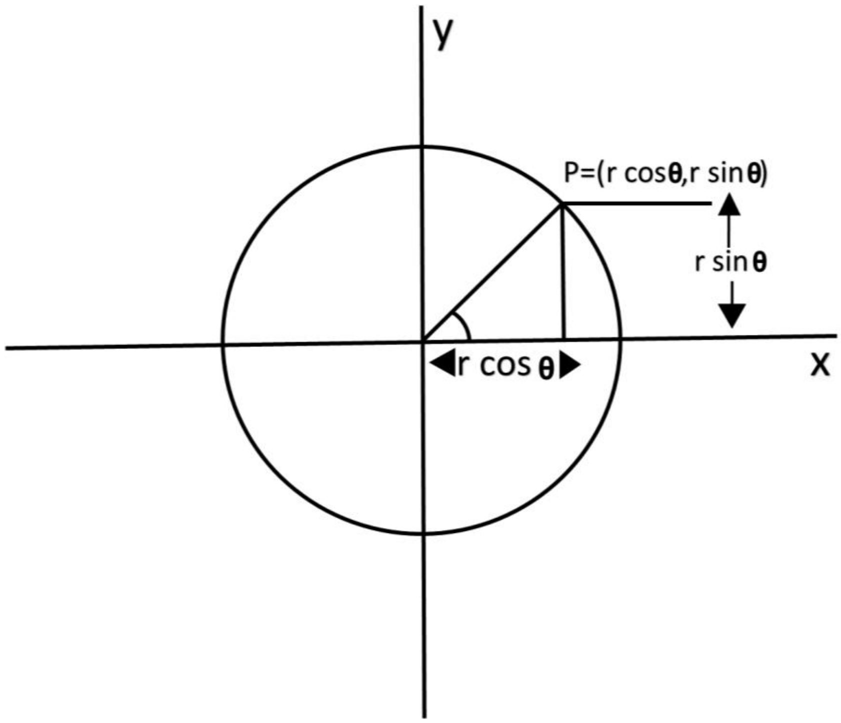
Defining a circle using Polar Co-ordinates :

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

$x = r \cos \theta$ $y = r \sin \theta$

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

By this method, θ is stepped from 0 to $\frac{\pi}{4}$ & 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= $(\frac{22}{7})/4$
θ=0

Step2: If $\theta > \theta_{end}$ then 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 (-x + h, y + k)
Plot (y + h, -x + k)
Plot (x + h, -y + k)

Step5: Increment θ=θ+i

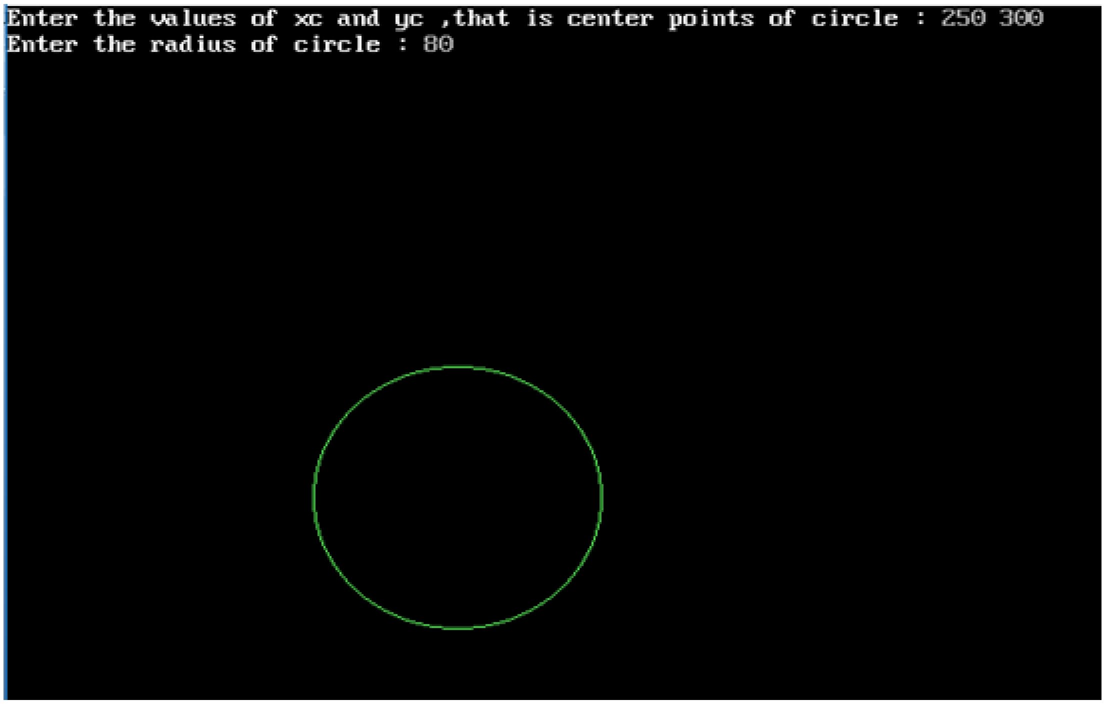
Step6: Go to step (ii).

Program to draw a circle using Polar Coordinates:

```
#include <graphics.h>
#include <stdlib.h>
#define color 10
void eightWaySymmetricPlot(int xc,int yc,int x,int y)
{
    putpixel(x+xc,y+yc,color);
    putpixel(x+xc,-y+yc,color);
    putpixel(-x+xc,-y+yc,color);
    putpixel(-x+xc,y+yc,color);
    putpixel(y+xc,x+yc,color);
    putpixel(y+xc,-x+yc,color);
    putpixel(-y+xc,-x+yc,color);
    putpixel(-y+xc,x+yc,color);
}
void PolarCircle(int xc,int yc,int r)
{
    int x,y,d;
    x=0;
    y=r;
    d=3-2*r;
    eightWaySymmetricPlot(xc,yc,x,y);
    while(x<=y)
    {
        if(d<=0)
        {
            d=d+4*x+6;
        }
        else
        {
            d=d+4*x-4*y+10;
            y=y-1;
```

```
        }
        x=x+1;
        eightWaySymmetricPlot(xc,yc,x,y);
    }
}
int main(void)
{
    int gdriver = DETECT, gmode, errorcode;
    int xc,yc,r;
    initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");
errorcode = graphresult();
if (errorcode != grOk)
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);
    }
printf("Enter the values of xc and yc ,that is center points of circle : ");
    scanf("%d%d",&xc,&yc);
    printf("Enter the radius of circle : ");
    scanf("%d",&r);
    PolarCircle(xc,yc,r);
    getch();
    closegraph();
    return 0;
}
```

Output:



MidPoint Circle Algorithm

It is based on the following function for testing the spatial relationship between the arbitrary point (x, y) and a circle of radius r centered at the origin:

Now, consider the coordinates of the point halfway between pixel T and pixel S

This is called midpoint (x_{i+1},y_i-) and we use it to define a decision parameter:

P_i=f (x_{i+1},y_i-) = $(x_{i+1})^2$+(y_i- $)^2$-r^2 ....equation 2

If P_i is -ve ⟹midpoint is inside the circle and we choose pixel T

If P_i is+ve ⟹midpoint is outside the circle (or on the circle)and we choose pixel S.

The decision parameter for the next step is:

P_{i+1}=$(x_{i+1}+1)^2$+(y_{i+1}- $)^2$- r^2............equation 3

Since x_{i+1}=x_{i+1}, we have

If pixel T is choosen $\Longrightarrow P_i<0$

We have $y_{i+1}=y_i$

If pixel S is choosen $\Longrightarrow P_i \geq 0$

We have $y_{i+1}=y_i-1$

We can continue to simplify this in n terms of (x_i,y_i) and get

Now, initial value of P_i (0,r)from equation 2

We can put $\cong 1$
$\therefore$r is an integer
So, $P_1=1-r$

Algorithm:

Step1: Put x =0, y =r in equation 2
We have p=1-r

Step2: Repeat steps while x ≤ y
Plot (x, y)
If (p<0)
Then set p = p + 2x + 3
Else
p = p + 2(x-y)+5
y =y - 1 (end if)
x =x+1 (end loop)

Step3: End

Program to draw a circle using Midpoint Algorithm:

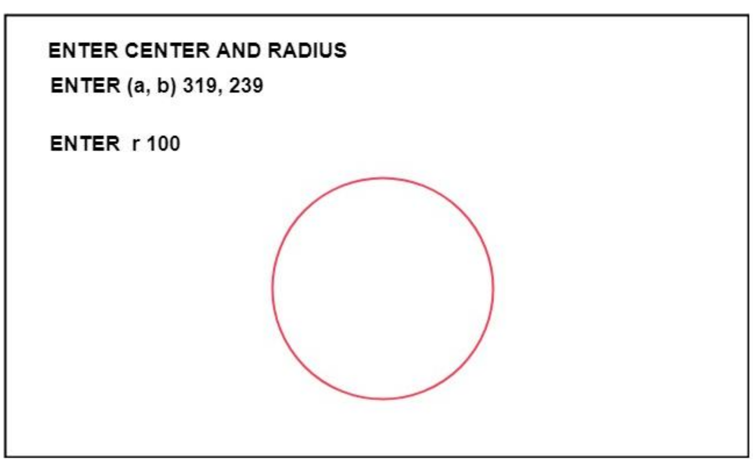
```
#include <graphics.h>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>

class bresen
{
```

```
    float x, y,a, b, r, p;
    public:
    void get ();
    void cal ();
};
    void main ()
    {
    bresen b;
    b.get ();
    b.cal ();
    getch ();
   }
    Void bresen :: get ()
   {
    cout<<"ENTER CENTER AND RADIUS";
     cout<< "ENTER (a, b)";
    cin>>a>>b;
    cout<<"ENTER r";
    cin>>r;
}
void bresen ::cal ()
{
    /* request auto detection */
    int gdriver = DETECT,gmode, errorcode;
    int midx, midy, i;
    /* initialize graphics and local variables */
    initgraph (&gdriver, &gmode, " ");
    /* read result of initialization */
    errorcode = graphresult ();
    if (errorcode ! = grOK)    /*an error occurred */
    {
        printf("Graphics error: %s \n", grapherrormsg (errorcode);
        printf ("Press any key to halt:");
        getch ();
        exit (1); /* terminate with an error code */
    }
    x=0;
    y=r;
```

```
putpixel (a, b+r, RED);
putpixel (a, b-r, RED);
putpixel (a-r, b, RED);
putpixel (a+r, b, RED);
p=5/4)-r;
while (x<=y)
{
   If (p<0)
   p+= (4*x)+6;
   else
   {
      p+=(2*(x-y))+5;
      y--;
   }
   x++;
   putpixel (a+x, b+y, RED);
   putpixel (a-x, b+y, RED);
   putpixel (a+x, b-y, RED);
   putpixel (a+x, b-y, RED);
   putpixel (a+x, b+y, RED);
   putpixel (a+x, b-y, RED);
   putpixel (a-x, b+y, RED);
   putpixel (a-x, b-y, RED);
}
}
```

Output:



Scan Converting a Ellipse:

The ellipse is also a symmetric figure like a circle but is four-way symmetry rather than eight-way.

Program to Implement Ellipse Drawing Algorithm:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void disp();
float x,y;
intxc,yc;
void main()
{
        intgd=DETECT,gm,a,b;
        float p1,p2;
        clrscr();
        initgraph(&gd,&gm,"c:\\turboc3\\bgi");
        printf("*** Ellipse Generating Algorithm ***\n");
        printf("Enter the value of Xc\t");
        scanf("%d",&xc);
        printf("Enter the value of yc\t");
        scanf("%d",&yc);
        printf("Enter X axis length\t");
        scanf("%d",&a);
        printf("Enter Y axis length\t");
        scanf("%d",&b);
        x=0;y=b;
        disp();
        p1=(b*b)-(a*a*b)+(a*a)/4;
        while((2.0*b*b*x)<=(2.0*a*a*y))
        {
                x++;
                if(p1<=0)
                p1=p1+(2.0*b*b*x)+(b*b);
```

```
                else
    {
                        y--;
                        p1=p1+(2.0*b*b*x)+(b*b)-(2.0*a*a*y);
     }
                disp();
                x=-x;
                disp();
                x=-x;
                delay(50);
        }
        x=a;
        y=0;
        disp();
        p2=(a*a)+2.0*(b*b*a)+(b*b)/4;
        while((2.0*b*b*x)>(2.0*a*a*y))
        {
                y++;
                if(p2>0)
                p2=p2+(a*a)-(2.0*a*a*y);
                else
    {
                        x--;
                        p2=p2+(2.0*b*b*x)-(2.0*a*a*y)+(a*a);
     }
                disp();
                y=-y;
                disp();
                y=-y;
                delay(50);
      }
        getch();
        closegraph();
}
void disp()
{
       putpixel(xc+x,yc+y,7);
        putpixel(xc-x,yc+y,7);
```

```
            putpixel(xc+x,yc-y,7);
        putpixel(xc+x,yc-y,7);
}
```

Output:

There two methods of defining an Ellipse:

1. Polynomial Method of defining an Ellipse
2. Trigonometric method of defining an Ellipse

Polynomial Method:

The ellipse has a major and minor axis. If a_1 and $b_1$are major and minor axis respectively. The centre of ellipse is (i, j). The value of x will be incremented from i to $a_1$and value of y will be calculated using the following formula

Drawback of Polynomial Method:

1. It requires squaring of values. So floating point calculation is required.
2. Routines developed for such calculations are very complex and slow.

Algorithm:

1. Set the initial variables: a = length of major axis; b = length of minor axis; (h, k) = coordinates of ellipse center; x = 0; i = step; x_{end} = a.

2. Test to determine whether the entire ellipse has been scan-converted. If $x>x_{end}$, stop.

3. Compute the value of the y coordinate:

4. Plot the four points, found by symmetry, 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)

5. Increment x; x = x + i.

6. Go to step 2.

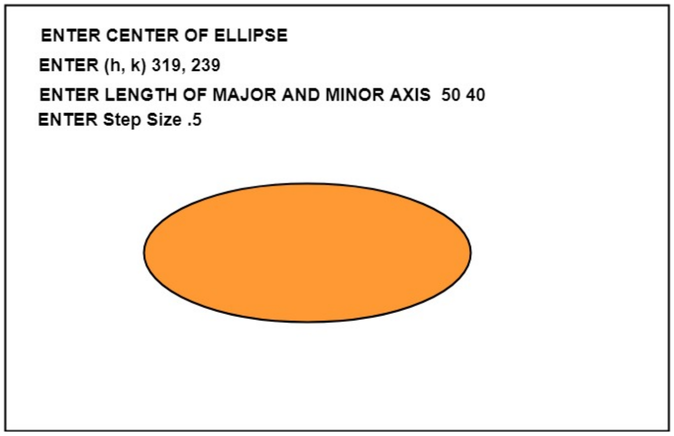
Program to draw an Ellipse using Polynomial Method:

```
#include <graphics.h>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>

class bresen
{
   float x, y, a, b, r, t, te, xend, h, k, step;
   public:
   void get ();
   void cal ();
};
   void main ()
   {
   bresen b;
   b.get ();
   b.cal ();
   getch ();
  }
   void bresen :: get ()
  {
   cout<<"\n ENTER CENTER OF ELLIPSE";
   cout<<"\n enter (h, k) ";
   cin>>h>>k;
   cout<<"\n ENTER LENGTH OF MAJOR AND MINOR AXIS";
   cin>>a>>b;
   cout<<"\n ENTER Step Size";
   cin>> step;
  }
void bresen ::cal ()
{
   /* request auto detection */
   int gdriver = DETECT,gmode, errorcode;
   int midx, midy, i;
   /* initialize graphics and local variables */
```

```
    initgraph (&gdriver, &gmode, " ");
    /* read result of initialization */
    errorcode = graphresult ();
    if (errorcode ! = grOK)    /*an error occurred */
    {
        printf("Graphics error: %s \n", grapherrormsg (errorcode);
        printf ("Press any key to halt:");
        getch ();
        exit (1); /* terminate with an error code */
    }
    x = 0;
    xend=a;
    whilex (x<xend)
    {
        t= (1-((x * x)/ (a * a)));
        if (t<0)
            te=-t;
        else
            te=t;
        y=b * sqrt (te);
        putpixel (h+x, k+y, RED);
        putpixel (h-x, k+y, RED);
        putpixel (h+x, y-y, RED);
        putpixel (h-x, k-y, RED);
        x+=step;
    }
    getch();
}
```

Output:



Trignometric Method:

The following equation defines an ellipse trigonometrically as shown in fig:

x = a * cos (θ) +h and
y = b * sin (θ)+k
where (x, y) = the current coordinates
a = length of major axis
b = length of minor axis
θ= current angle
(h, k) = ellipse center

In this method, the value of θ is varied from 0 to radians. The remaining points are found by symmetry.

Drawback:

1. This is an inefficient method.
2. It is not an interactive method for generating ellipse.
3. The table is required to see the trigonometric value.
4. Memory is required to store the value of θ.

Algorithm:

Step1: Start Algorithm

Step2: Declare variable x_1,y_1,aa_1,bb_1,aa_2,bb_2,fx,fy,p1,a1,b1

Step3: Initialize x_1=0 and y_1=b/* values of starting point of circle */

Step4: Calculate $aa_1=a_1*a_1$
Calculate $bb_1=b_1* b_1$
Calculate $aa_2=aa_1*2$
Calculate $bb_2=bb_1*2$

Step5: Initialize fx = 0

Step6: Initialize fy = aa_2* b_1

Step7: Calculate the value of $p_1$and round if it is integer
$p_1=bb_1-aa_1* b_1+0.25* aa_1/$

Step8:

```
While (fx < fy)
       {
              Set pixel (x1,y1)
                     Increment x i.e., x = x + 1
                     Calculate fx = fx + bb2
                      If (p1 < 0)
                                Calculate p1 = p1 + fx + bb1/
                      else
              {
                     Decrement y i.e., y = y-1
                            Calculate fy = fy - 992;
                             p1=p1 + fx + bb1-fy
                       }
                }
```

Step9: Setpixel (x_1,y_1)

Step10: Calculate p1=bb_1 (x+.5)(x+.5)+aa(y-1)(y-1)-aa_1*bb_1

Step 11:

```
While (y1>0)
                {
                          Decrement y i.e., y = y-1
                           fy=fx-aa2/
                         if (p1>=0)
                 p1=p1 - fx +  aa1/
```

```
                        else
                {
                        Increment x i.e., x = x + 1
                        fx= fx+bb_2
                        p1=p1+fx-fy-aa1
                 }
         }
        Set pixel (x1,y1)
```

Step12: Stop Algorithm

Program to draw a circle using Trigonometric method:

```
#include <graphics.h>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
# define pi 3.14

class bresen
{
    float a, b, h, k, thetaend,step,x,y;
    int i;
    public:
    void get ();
    void cal ();
};
    void main ()
    {
    bresen b;
    b.get ();
    b.cal ();
    getch ();
   }
    void bresen :: get ()
   {
    cout<<"\n ENTER CENTER OF ELLIPSE";
```

```
   cin>>h>>k;
   cout<<"\n ENTER LENGTH OF MAJOR AND MINOR AXIS";
   cin>>a>>b;
   cout<<"\n ENTER STEP SIZE";
   cin>> step;
  }
void bresen ::cal ()
{
   /* request auto detection */
   int gdriver = DETECT,gmode, errorcode;
   int midx, midy, i;
   /* initialize graphics and local variables */
   initgraph (&gdriver, &gmode, " ");
   /* read result of initialization */
   errorcode = graphresult ();
   if (errorcode ! = grOK)    /*an error occurred */
   {
      printf("Graphics error: %s \n", grapherrormsg (errorcode);
      printf ("Press any key to halt:");
      getch ();
      exit (1); /* terminate with an error code */
   }
   theta= 0;
   thetaend=(pi*90)/180;
   whilex (theta<thetaend)
   {
      x = a * cos (theta);
      y = b * sin (theta);
      putpixel (x+h, y+k, RED);
      putpixel (-x+h, y+k, RED);
      putpixel (-x+h, -y+k, RED);
      putpixel (x+h, -y+k, RED);
      theta+=step;
   }
      getch();
}
```

Ellipse Axis Rotation:

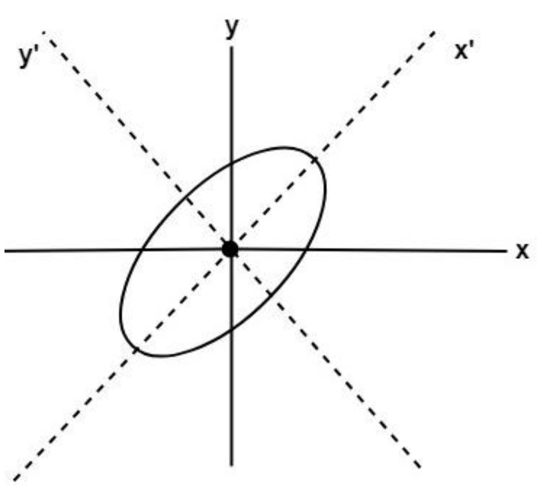
Since the ellipse shows four-way symmetry, it can easily be rotated. The new equation is found by trading a and b, the values which describe the major and minor axes. When the polynomial method is used, the equations used to describe the ellipse become

where (h, k) = ellipse center
a = length of the major axis
b = length of the minor axis
In the trigonometric method, the equations are
$x = b \cos(\theta)+h$ and $y=a \sin(\theta)+k$

Where (x, y) = current coordinates
a = length of the major axis
b = length of the minor axis
θ = current angle
(h, k) = ellipse center

Assume that you would like to rotate the ellipse through an angle other than 90 degrees. The rotation of the ellipse may be accomplished by rotating the x &y axis α degrees.

$x = a \cos(0) - b \sin(0+\infty) + h$ $y= b(\sin 0) + a \cos(0+\infty) + k$



Rotation of an Ellipse

Midpoint Ellipse Algorithm:

This is an incremental method for scan converting an ellipse that is centered at the origin in standard position i.e., with the major and minor axis parallel to coordinate system axis. It is very similar to the midpoint circle algorithm. Because of the four-way symmetry property we need to consider the entire elliptical curve in the first quadrant.

Let's first rewrite the ellipse equation and define the function f that can be used to decide if the midpoint between two candidate pixels is inside or outside the ellipse:

Now divide the elliptical curve from (0, b) to (a, 0) into two parts at point Q where the slope of the curve is -1.

Slope of the curve is defined by the f(x, y) = 0 is where fx & fy are partial derivatives of f(x, y) with respect to x & y.

We have fx = $2b^2$ x, fy=$2a^2$ y & Hence we can monitor the slope value during the scan conversion process to detect Q. Our starting point is (0, b)

Suppose that the coordinates of the last scan converted pixel upon entering step i are (x_i,y_i). We are to select either T (x_{i+1}),y_i) or S (x_{i+1},y_{i-1}) to be the next pixel. The midpoint of T & S is used to define the following decision parameter.

pi = f(x_{i+1}),y_i-)

pi=b^2 $(x_{i+1})^2$+a^2 (y_i- $)^2$-a^2 b^2

If pi<0, the midpoint is inside the curve and we choose pixel T.

If pi>0, the midpoint is outside or on the curve and we choose pixel S.

Decision parameter for the next step is:

$p_{i+1}=f(x_{i+1}+1,y_{i+1}-\quad)$

$= b^2 (x_{i+1}+1)^2+a^2 (y_{i+1}-\quad)^2-a^2 b^2$

Since $x_{i+1}=xi+1$,we have

$p_{i+1}-pi=b^2[((x_{i+1}+1)^2+a^2 (y_{i+1}-\quad)^2-(y_i -\quad)^2]$

$p_{i+1}= pi+2b^2 x_{i+1}+b^2+a^2 [(y_{i+1}-\quad)^2-(y_i -\quad)^2]$

If T is chosen pixel ($pi<0$), we have $y_{i+1}=yi$.

If S is chosen pixel ($pi>0$) we have $y_{i+1}=yi-1$. Thus we can express

p_{i+1}in terms of pi and (x_{i+1},y_{i+1}): $p_{i+1}= pi+2b^2 x_{i+1}+b^2$ if $pi<0$ $=$ $pi+2b^2 x_{i+1}+b^2-2a^2 y_{i+1}$ if $pi>0$

The initial value for the recursive expression can be obtained by the evaluating the original definition of pi with (0, b):

$p1 = (b^2+a^2 (b-\quad)^2-a^2 b^2$
$= b^2-a^2 b+a^2/4$

Suppose the pixel $(x_j\ y_j)$ has just been scan converted upon entering step j. The next pixel is either U (x_j ,y_j-1) or V (x_j+1,y_j-1). The midpoint of the horizontal line connecting U & V is used to define the decision parameter:

$q_j=f(x_j+\quad,y_j-1)$

$q_j=b^2 (x_j+\quad)^2+a^2 (y_j -1)^2-a^2 b^2$

If $q_j<0$, the midpoint is inside the curve and we choose pixel V.

If $q_j\geq 0$, the midpoint is outside the curve and we choose pixel U.Decision parameter for the next step is:

$q_{j+1}=f(x_{j+1}+\quad,y_{j+1}-1)$

$= b^2 (x_{j+1}+\quad)^2+ a^2 (y_{j+1}-1)^2- a^2 b^2$

Since $y_{j+1}=y_j-1$,we have

$q_{j+1}-q_j=b^2$ $[(x_{j+1}+\quad)^2-(x_j+\quad)^2]+a^2(y_{j+1}-1)^2-(y_{j+1})^2]$

$q_{j+1}=q_j+b^2[(x_{j+1}+\quad)^2-(x_j+\quad)^2]-2a^2y_{j+1}+a^2$

If V is chosen pixel (qj<0), we have x_{j+1}=xj.

If U is chosen pixel (pi>0) we have x_{j+1}=xj. Thus we can express

q_{j+1}in terms of q_j and (x_{j+1},y_{j+1}):
$q_{j+1}=q_j+2b^2x_{j+1}-2a^2y_{j+1}+a^2$ if qj < 0
$=q_j-2a^2y_{j+1}+a^2$ if qj>0

The initial value for the recursive expression is computed using the original definition of qj. And the coordinates of $(x_k\ y_k)$ of the last pixel choosen for the part 1 of the curve:

$q1 = f(x_k+\quad,y_k-1)=b^2(x_k+\quad)^2-a^2(y_k-1)^2-a^2b^2$

Algorithm:

```
int x=0, y=b; [starting point]
int fx=0, fy=2a2 b [initial partial derivatives]
int p = b2-a2 b+a2/4
while (fx<="" 1="" {="" set="" pixel="" (x,="" y)="" x++;="" fx="fx" +="" 2b2;
          if (p<0)
          p = p + fx +b2;
          else
          {
                      y--;
                      fy=fy-2a2
                      p = p + fx +b2-fy;
          }
}
Setpixel (x, y);
p=b2(x+0.5)2+ a2 (y-1)2- a2 b2
while (y>0)
{
          y--;
          fy=fy-2a2;
          if (p>=0)
          p=p-fy+a2
           else
```

```
            {
                        x++;
                        fx=fx+2b2
                        p=p+fx-fy+a2;
            }
            Setpixel (x,y);
  }
```

Program to draw an ellipse using Midpoint Ellipse Algorithm:

```
#include <graphics.h>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>

class bresen
{
  float x,y,a, b,r,p,h,k,p1,p2;
  public:
  void get ();
  void cal ();
};
  void main ()
  {
  bresen b;
  b.get ();
  b.cal ();
  getch ();
 }
  void bresen :: get ()
 {
  cout<<"\n ENTER CENTER OF ELLIPSE";
  cout<<"\n ENTER (h, k) ";
      cin>>h>>k;
  cout<<"\n ENTER LENGTH OF MAJOR AND MINOR AXIS";
  cin>>a>>b;
 }
void bresen ::cal ()
```

```
{
    /* request auto detection */
    int gdriver = DETECT,gmode, errorcode;
    int midx, midy, i;
    /* initialize graphics and local variables */
    initgraph (&gdriver, &gmode, " ");
    /* read result of initialization */
    errorcode = graphresult ();
    if (errorcode ! = grOK)    /*an error occurred */
    {
        printf("Graphics error: %s \n", grapherrormsg (errorcode);
        printf ("Press any key to halt:");
        getch ();
        exit (1); /* terminate with an error code */
    }
    x=0;
    y=b;
    // REGION 1
    p1 =(b * b)-(a * a * b) + (a * a)/4);
    {
        putpixel (x+h, y+k, RED);
        putpixel (-x+h, -y+k, RED);
        putpixel (x+h, -y+k, RED);
        putpixel (-x+h, y+k, RED);
        if (p1 < 0)
            p1 += ((2 *b * b) *(x+1))-((2 * a * a)*(y-1)) + (b * b);
        else
        {
            p1+= ((2 *b * b) *(x+1))-((2 * a * a)*(y-1))-(b * b);
            y--;
        }
        x++;
    }
    //REGION 2
    p2 =((b * b)* (x + 0.5))+((a * a)*(y-1) * (y-1))-(a * a *b * b);
    while (y>=0)
    {
        If (p2>0)
```

```
        p2=p2-((2 * a * a)* (y-1))+(a *a);
        else
        {
        p2=p2-((2 * a * a)* (y-1))+((2 * b * b)*(x+1))+(a * a);
        x++;
        }
        y--;
        putpixel (x+h, y+k, RED);
        putpixel (-x+h, -y+k, RED);
        putpixel (x+h, -y+k, RED);
        putpixel (-x+h, y+k, RED);
    }
    getch();
}
```

Output: