# EXPERIMENT NO :3

# OBJECTIVE: TO IMPLEMENTING THE MIDPOINT CIRCLE ALGORITHM

## THEORY:

The Midpoint Circle Algorithm is a method to draw circles in computer graphics. It utilizes the midpoint of the circle's perimeter to determine the pixels to be drawn. The algorithm works by successively determining the points to be plotted as the circle is traversed. It calculates the decision parameter based on the circle's radius and the current pixel position. The algorithm is efficient as it avoids costly trigonometric calculations.

## ALGORTIHM:

*Midpoint Circle Algorithm:*

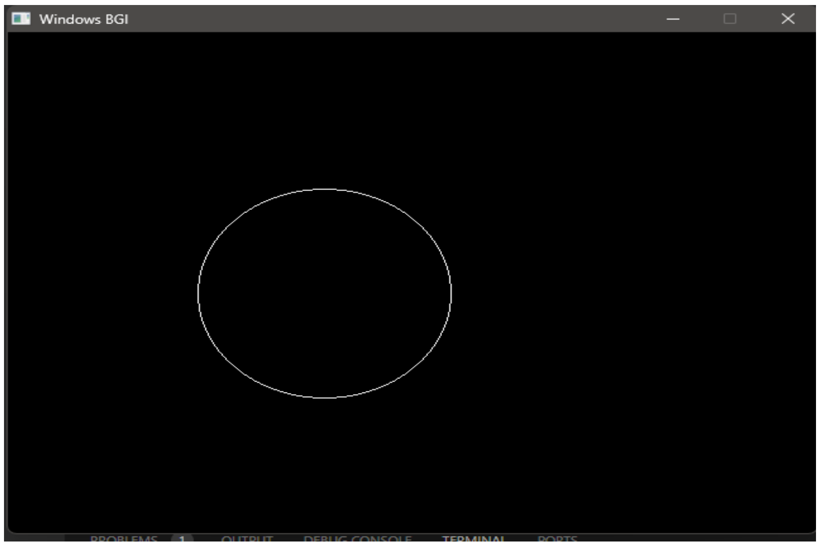
```
1. Input:
  - xc, yc: coordinates of the center of the circle
  - r: radius of the circle
2. Initialization:
  - Set initial values: x = 0, y = r
  - Calculate the initial decision parameter: p = 1 - r
3. Plotting Points:
  - Repeat the following until x <= y:
    - Plot the points symmetrically in all eight octants:
      - (xc + x, yc + y), (xc - x, yc + y)
      - (xc + x, yc - y), (xc - x, yc - y)
      - (xc + y, yc + x), (xc - y, yc + x)
      - (xc + y, yc - x), (xc - y, yc - x)
4. Updating Decision Parameter:
  - If p < 0, update p as p += 2 * x + 3
  - Else, update p as p += 2 * (x - y) + 5 and decrement y by 1
5. Increment x: Increment x by 1
6. Termination:
  - Repeat steps 3-5 until x exceeds y
7. Output:
  - The circle is plotted with the specified center and radius.
```

## Implementation in C:

//Program

```
#include <stdio.h>
#include <graphics.h>
void drawCircle(int xc, int yc, int r) {
    int x = 0, y = r;
    int p = 1 - r; // Initial decision parameter
    while (x <= y) {
        putpixel(xc + x, yc + y, WHITE);
        putpixel(xc - x, yc + y, 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 + y, yc - x, WHITE);
        putpixel(xc - y, yc - x, WHITE);
        if (p < 0)
            p += 2 * x + 3;
        else {
            p += 2 * (x - y) + 5;
            y--; }
        x++;}}
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");
    int xc = 250, yc = 250, radius = 100;
    drawCircle(xc, yc, radius);
    delay(5000); // Delay to display the circle
    closegraph();
    return 0;}
```

**Output:**



## DISCUSSION:

In this lab, we implemented the Midpoint Circle Algorithm in C to draw a circle on a graphics window. The algorithm efficiently determines the points along the circle's perimeter by exploiting symmetry and avoiding costly trigonometric calculations. While other algorithms may offer better efficiency in certain cases, the Midpoint Circle Algorithm remains a fundamental and effective method for circle drawing in computer graphics.

## CONCLUSION:

The above experiment in C programming, we have understood the concept of Mid-Point Circle Algorithm and how it work.

# EXPERIMENT NO :4

## OBJECTIVE: TO IMPLEMENTING THE MIDPOINT ELLIPSE ALGORITHM

## THEORY:

The Midpoint Ellipse Algorithm is a method for drawing ellipses on a raster grid display device such as a computer screen. The algorithm utilizes the midpoint formula to determine the next pixel to be plotted as it traverses along the ellipse's perimeter.

The general equation of an ellipse is:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

Where (a,b) are the radii along the x-axis and y-axis respectively. The algorithm essentially divides the ellipse into eight regions and calculates the decision parameter to determine which pixel to plot in each region.

## ALGORITHM:

*Midpoint Ellipse Algorithm:*

*1. Initialize the ellipse parameters: a,b, center Coordinate(xc,yc).*

*2. Calculate the initial decision parameter :*$P_0 = b^2 - a^2 b + \frac{a^2}{4}$

*3. At each step, move along the major axis(x) and choose either the pixel at(x,y) or (x,y-1) depending on the decision parameter.*

*4. Update the decision parameter at each step based on the region and current pixel position.*

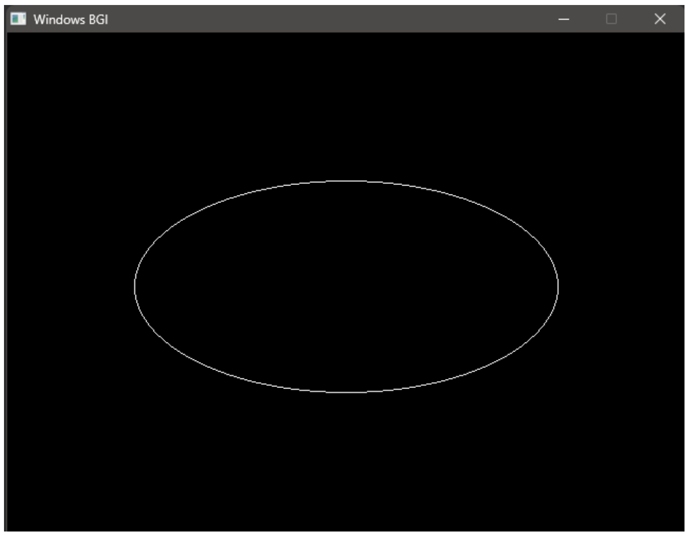
## Implementation in C:

//Program

```
#include <stdio.h>
#include <graphics.h>
void drawEllipse(int xc, int yc, int a, int b) {
  int gm;
  detectgraph(&gm);
  initgraph(&gm, &gm, "");
  int x = 0, y = b;
  float p = b * b - a * a * b + 0.25 * a * a;
  while (2 * b * b * x <= 2 * a * a * y) {
    putpixel(xc + x, yc + y, WHITE);
    putpixel(xc - x, yc + y, WHITE);
    putpixel(xc + x, yc - y, WHITE);
    putpixel(xc - x, yc - y, WHITE);
    if (p < 0) {
      x++;
      p += 2 * b * b * x + b * b;
```

```
        } else {
            x++;
            y--;
            p += 2 * b * b * x - 2 * a * a * y + b * b; }}
    p = b * b * (x + 0.5) * (x + 0.5) + a * a * (y - 1) * (y - 1) - a * a * b * b;
    while (y >= 0) {
        putpixel(xc + x, yc + y, WHITE);
        putpixel(xc - x, yc + y, WHITE);
        putpixel(xc + x, yc - y, WHITE);
        putpixel(xc - x, yc - y, WHITE);
        if (p > 0) {
            y--;
            p -= 2 * a * a * y + a * a;
        } else {
            y--;
            x++;
            p += 2 * b * b * x - 2 * a * a * y + a * a; }    }}
int main() {
    int xc = 320, yc = 240; // Center coordinates
    int a = 150, b = 100;    // Semi-major and semi-minor axes lengths
    drawEllipse(xc, yc, a, b);
    getch();
    closegraph();
    return 0;
}
```

## Output:



## DISCUSSION:

In this lab, we implemented Midpoint Ellipse Algorithm successfully draws an ellipse on the screen using the provided parameters. The algorithm efficiently plots points along the ellipse's perimeter, resulting in a smooth and accurate ellipse shape. By dividing the ellipse into regions and calculating the decision parameter, the algorithm minimizes computational complexity.

## CONCLUSION:

The above experiment in C programming, we have understood the concept of Mid-Point Ellipse  Algorithm and how it work.