EXPERIMENT 3

AIM : Write a program to implement the Mid-Point Circle Drawing Algorithm

THEORY :

The midpoint circle drawing algorithm is an algorithm used to determine the points needed for rasterizing a circle. We use the midpoint 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. This will work because a circle is symmetric about its center.

ALGORITHM :

We assign the starting point coordinates (X_0, Y_0)

as $X_0 = 0$

$Y_0 = R$

We calculate decision parameter P_0

$P_0 = 1 – R$

We calculate P_k, (X_k, Y_k), and (X_{k+1},

Y_{k+1}) If $P_k < 0$

$P_{k+1} = P_k + 2X_{k+1} +$

1 $X_{k+1} = X_k + 1$

Yk+1 = Yk

Else

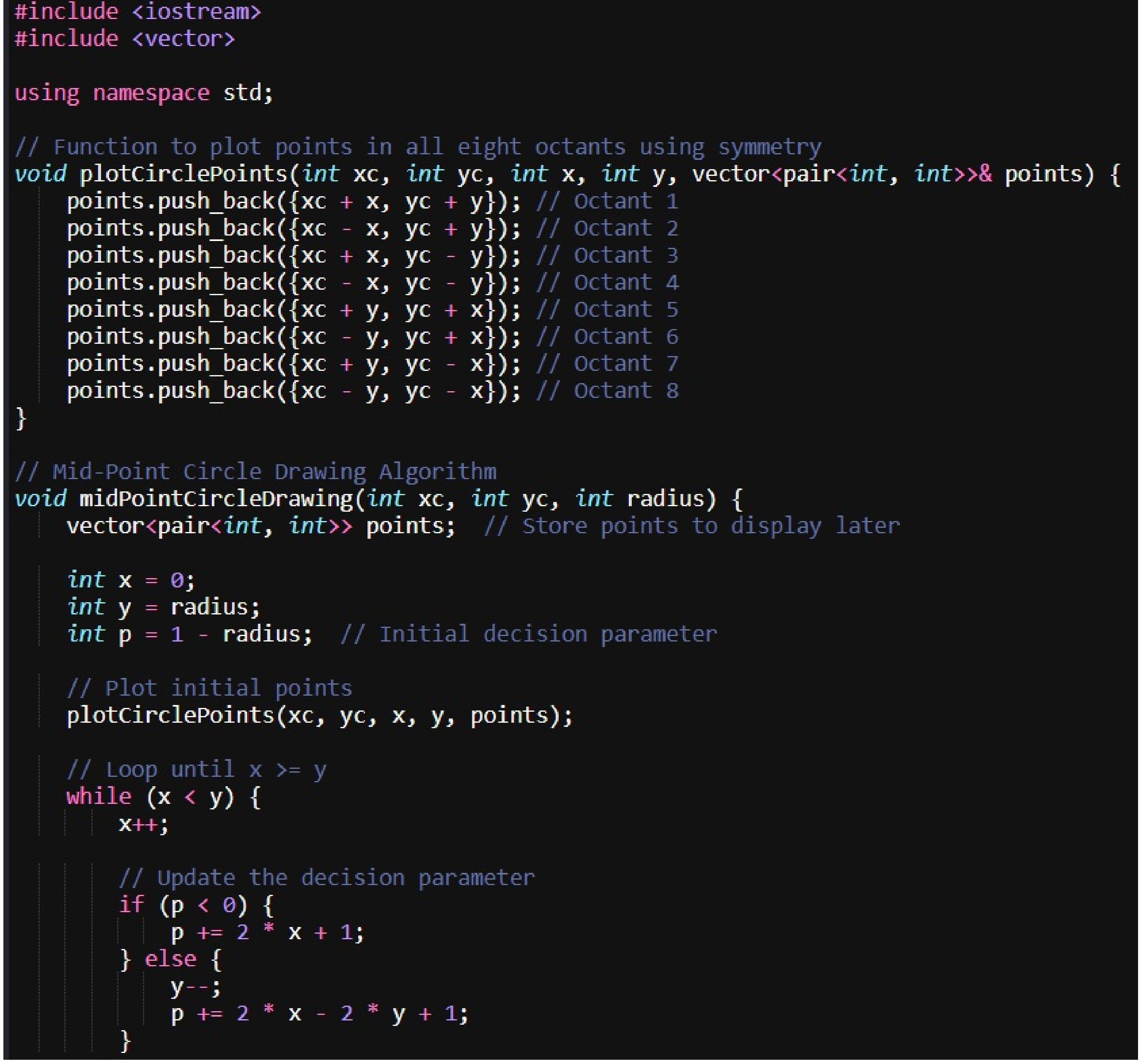
$P_{k+1} = P_k + 2X_{k+1} + 2Y_{k+1} +$

1 $X_{k+1} = X_k + 1$

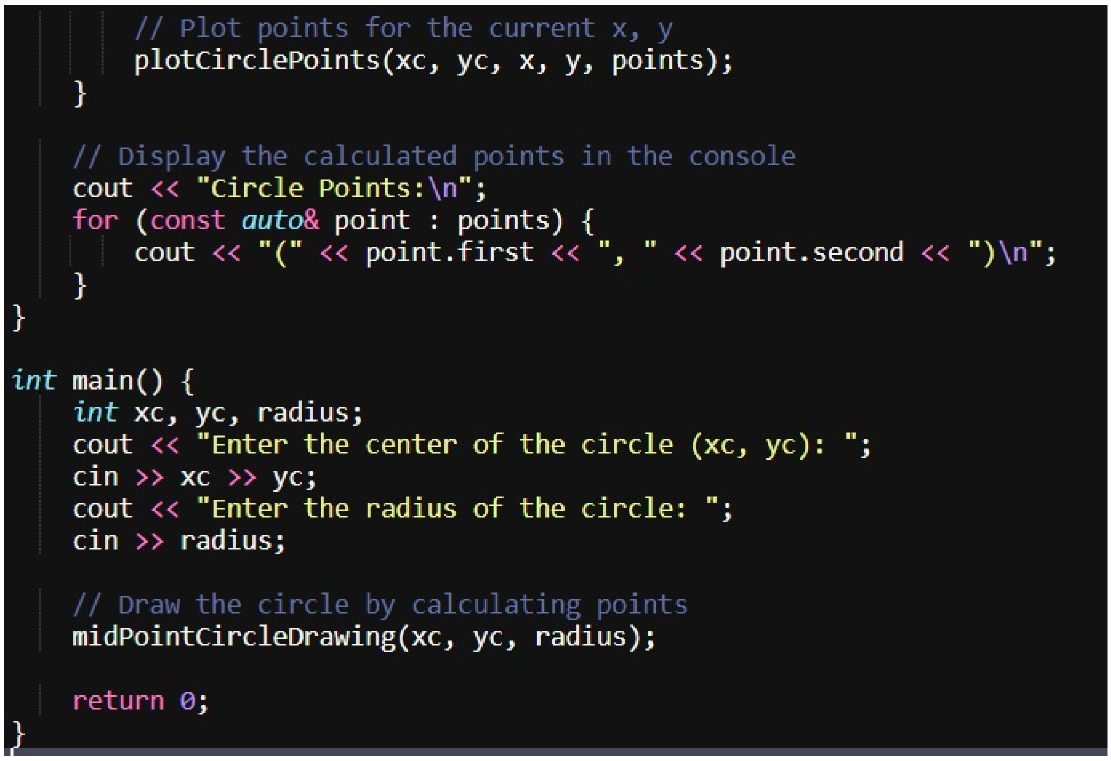
$Y_{k+1} = Y_k - 1$

This will give points in the first quadrant. We will then change signs and get the points in other quadrants.

CODE :

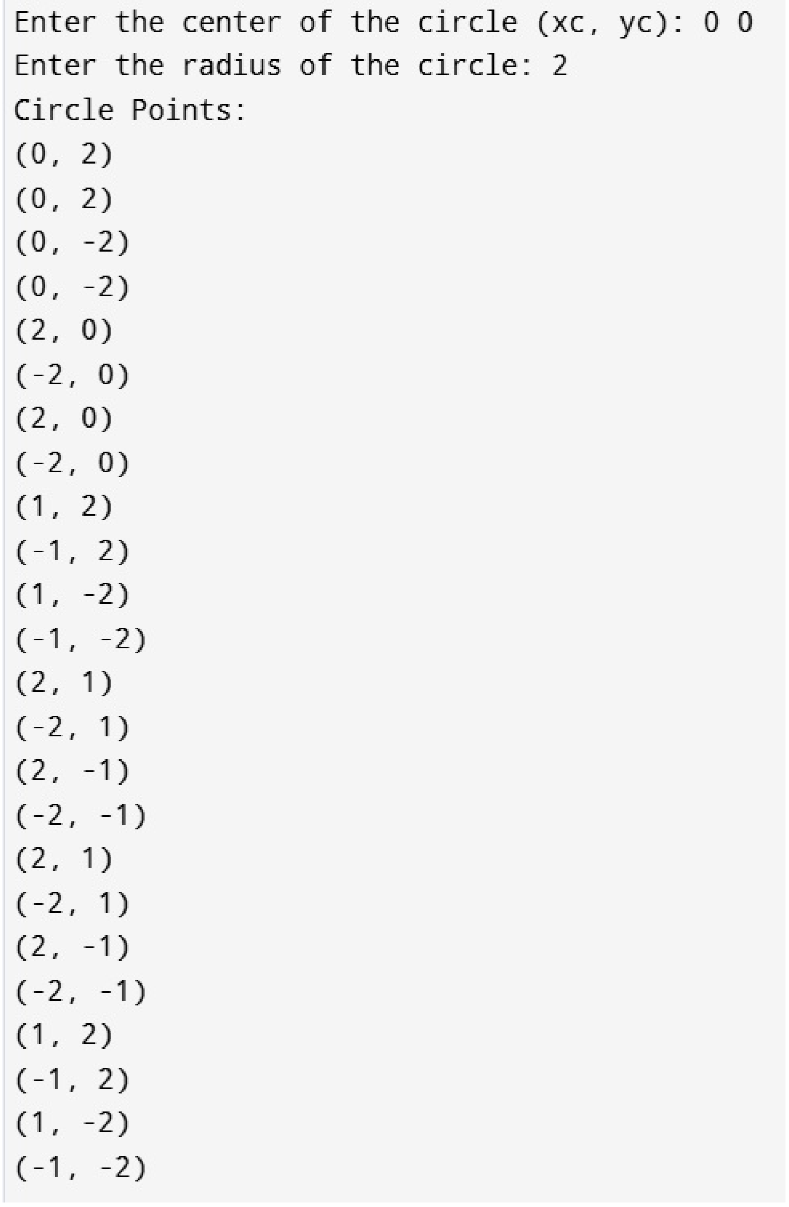


```cpp
#include <iostream>
#include <vector>

using namespace std;

// Function to plot points in all eight octants using symmetry
void plotCirclePoints(int xc, int yc, int x, int y, vector<pair<int, int>>& points) {
    points.push_back({xc + x, yc + y}); // Octant 1
    points.push_back({xc - x, yc + y}); // Octant 2
    points.push_back({xc + x, yc - y}); // Octant 3
    points.push_back({xc - x, yc - y}); // Octant 4
    points.push_back({xc + y, yc + x}); // Octant 5
    points.push_back({xc - y, yc + x}); // Octant 6
    points.push_back({xc + y, yc - x}); // Octant 7
    points.push_back({xc - y, yc - x}); // Octant 8
}

// Mid-Point Circle Drawing Algorithm
void midPointCircleDrawing(int xc, int yc, int radius) {
    vector<pair<int, int>> points;  // Store points to display later

    int x = 0;
    int y = radius;
    int p = 1 - radius;  // Initial decision parameter

    // Plot initial points
    plotCirclePoints(xc, yc, x, y, points);

    // Loop until x >= y
    while (x < y) {
        x++;

        // Update the decision parameter
        if (p < 0) {
            p += 2 * x + 1;
        } else {
            y--;
            p += 2 * x - 2 * y + 1;
        }
```



```cpp
        // Plot points for the current x, y
        plotCirclePoints(xc, yc, x, y, points);
    }

    // Display the calculated points in the console
    cout << "Circle Points:\n";
    for (const auto& point : points) {
        cout << "(" << point.first << ", " << point.second << ")\n";
    }
}

int main() {
    int xc, yc, radius;
    cout << "Enter the center of the circle (xc, yc): ";
    cin >> xc >> yc;
    cout << "Enter the radius of the circle: ";
    cin >> radius;

    // Draw the circle by calculating points
    midPointCircleDrawing(xc, yc, radius);

    return 0;
}
```

OUTPUT :



```
Enter the center of the circle (xc, yc): 0 0
Enter the radius of the circle: 2
Circle Points:
(0, 2)
(0, 2)
(0, -2)
(0, -2)
(2, 0)
(-2, 0)
(2, 0)
(-2, 0)
(1, 2)
(-1, 2)
(1, -2)
(-1, -2)
(2, 1)
(-2, 1)
(2, -1)
(-2, -1)
(2, 1)
(-2, 1)
(2, -1)
(-2, -1)
(1, 2)
(-1, 2)
(1, -2)
(-1, -2)
```

EXPERIMENT 4

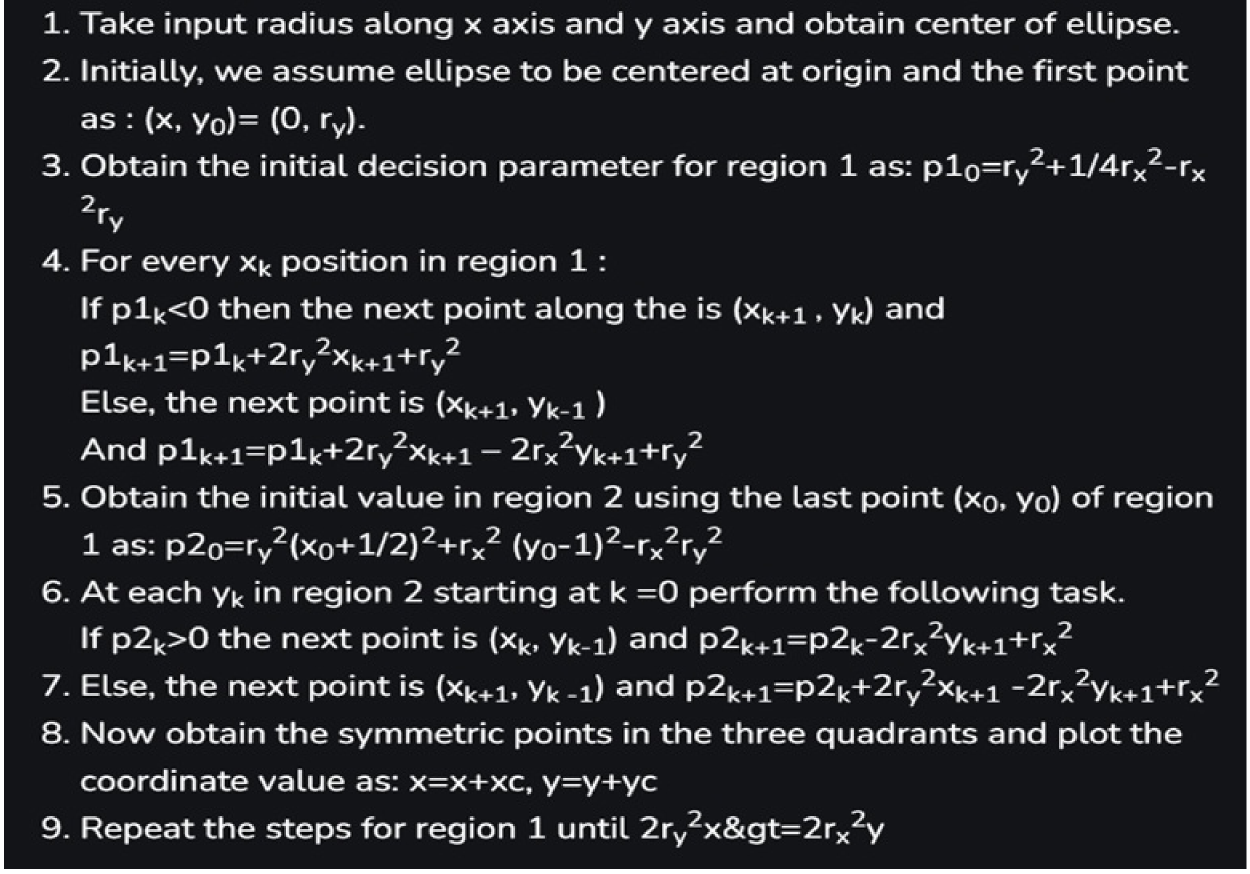
AIM : Write a program to implement MidPoint Ellipse Algorithm

THEORY:

Midpoint ellipse algorithm plots(finds) points of an ellipse on the first quadrant by dividing the quadrant into two regions. Each point(x, y) is then projected into other three quadrants

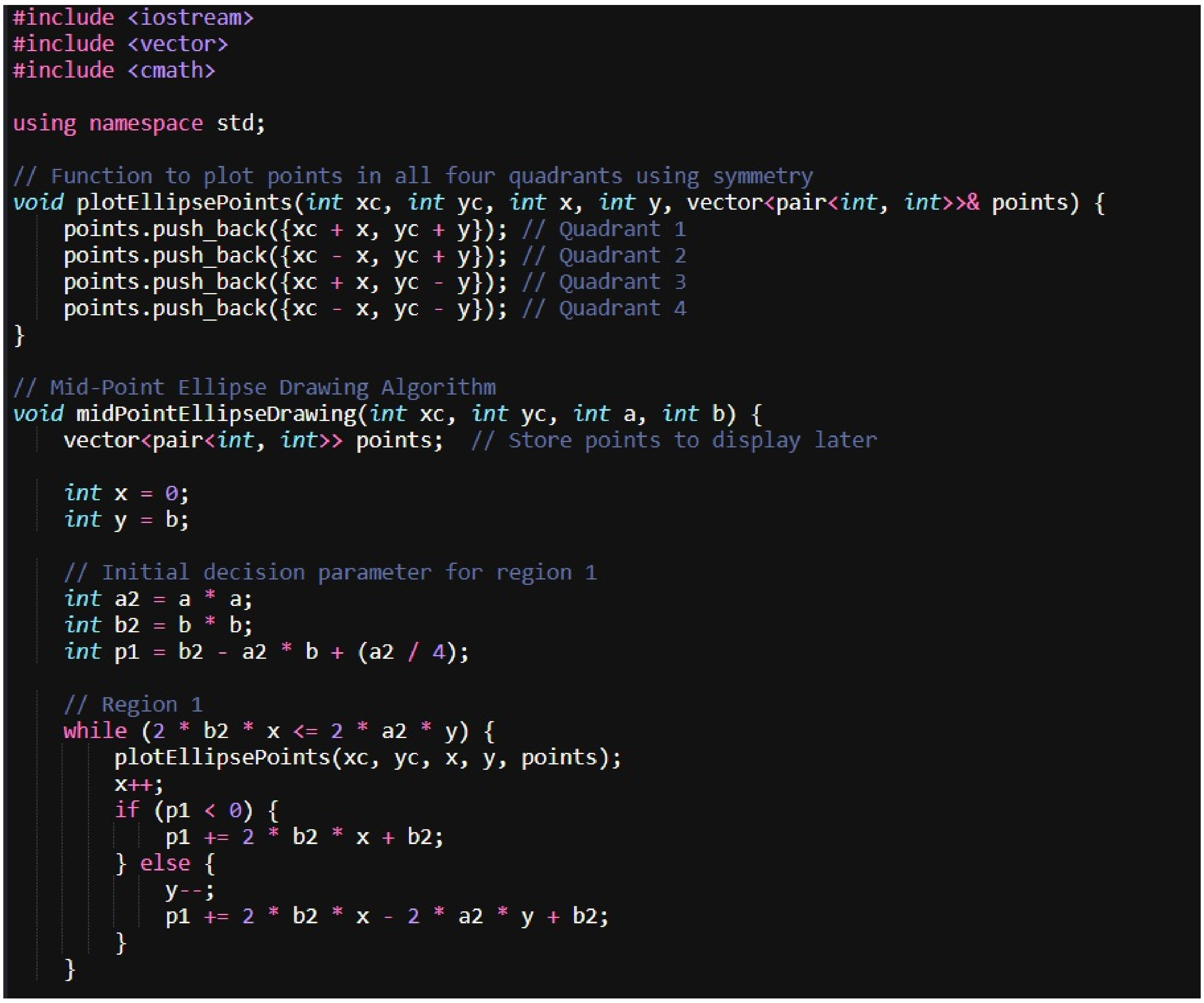
(-x, y), (x, -y), (-x, -y).

ALGORITHM :

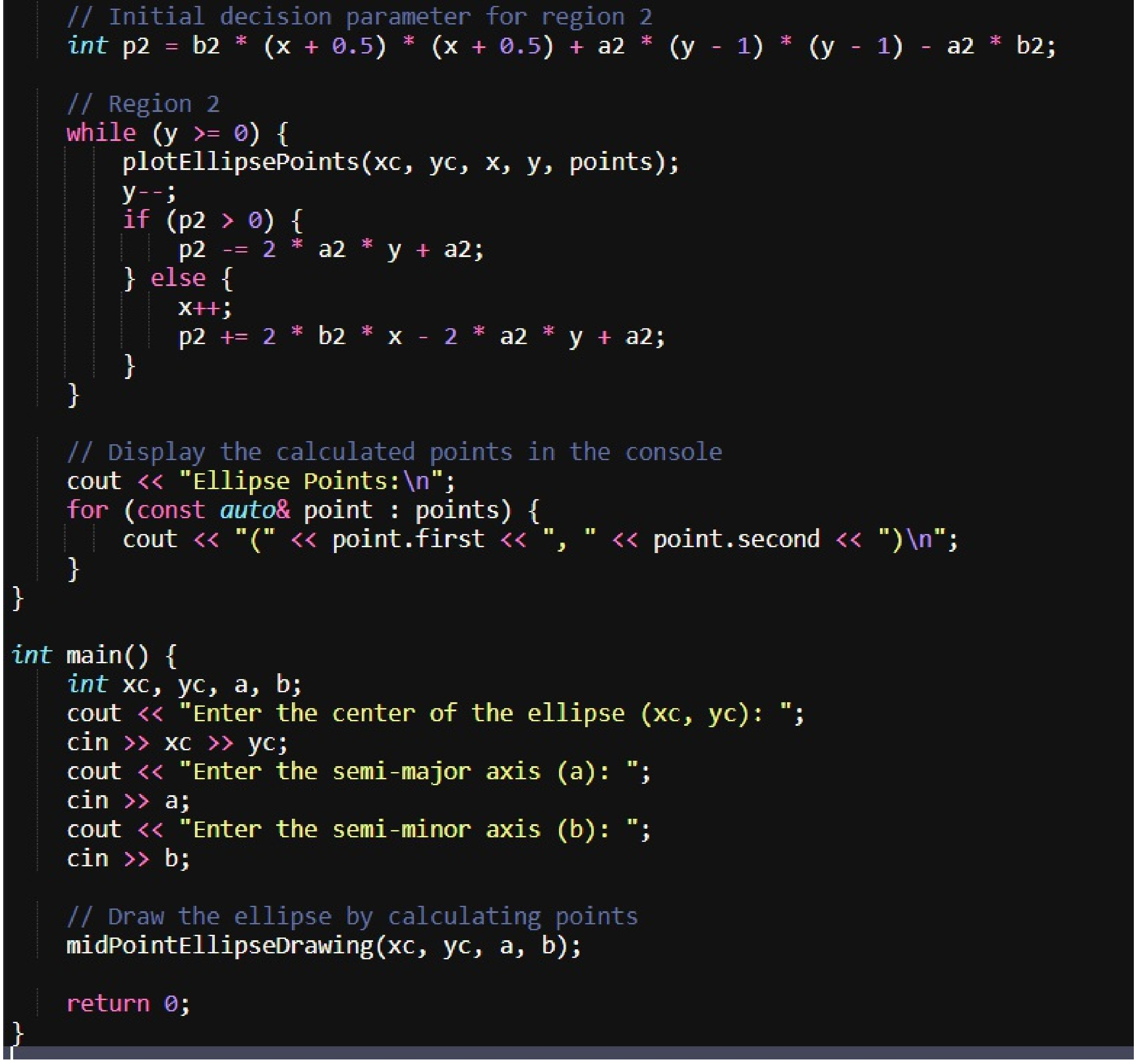


1. Take input radius along x axis and y axis and obtain center of ellipse.
2. Initially, we assume ellipse to be centered at origin and the first point as : $(x, y_0)= (0, r_y)$.
3. Obtain the initial decision parameter for region 1 as: $p1_0=r_y^2+1/4r_x^2-r_x^2r_y$
4. For every x_k position in region 1 :
   If $p1_k<0$ then the next point along the is (x_{k+1}, y_k) and $p1_{k+1}=p1_k+2r_y^2x_{k+1}+r_y^2$
   Else, the next point is (x_{k+1}, y_{k-1})
   And $p1_{k+1}=p1_k+2r_y^2x_{k+1} - 2r_x^2y_{k+1}+r_y^2$
5. Obtain the initial value in region 2 using the last point (x_0, y_0) of region 1 as: $p2_0=r_y^2(x_0+1/2)^2+r_x^2 (y_0-1)^2-r_x^2r_y^2$
6. At each y_k in region 2 starting at k =0 perform the following task.
   If $p2_k>0$ the next point is (x_k, y_{k-1}) and $p2_{k+1}=p2_k-2r_x^2y_{k+1}+r_x^2$
7. Else, the next point is (x_{k+1}, y_{k-1}) and $p2_{k+1}=p2_k+2r_y^2x_{k+1} -2r_x^2y_{k+1}+r_x^2$
8. Now obtain the symmetric points in the three quadrants and plot the coordinate value as: x=x+xc, y=y+yc
9. Repeat the steps for region 1 until $2r_y^2x$&gt=$2r_x^2y$

CODE :



```cpp
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

// Function to plot points in all four quadrants using symmetry
void plotEllipsePoints(int xc, int yc, int x, int y, vector<pair<int, int>>& points) {
    points.push_back({xc + x, yc + y}); // Quadrant 1
    points.push_back({xc - x, yc + y}); // Quadrant 2
    points.push_back({xc + x, yc - y}); // Quadrant 3
    points.push_back({xc - x, yc - y}); // Quadrant 4
}

// Mid-Point Ellipse Drawing Algorithm
void midPointEllipseDrawing(int xc, int yc, int a, int b) {
    vector<pair<int, int>> points;  // Store points to display later

    int x = 0;
    int y = b;

    // Initial decision parameter for region 1
    int a2 = a * a;
    int b2 = b * b;
    int p1 = b2 - a2 * b + (a2 / 4);

    // Region 1
    while (2 * b2 * x <= 2 * a2 * y) {
        plotEllipsePoints(xc, yc, x, y, points);
        x++;
        if (p1 < 0) {
            p1 += 2 * b2 * x + b2;
        } else {
            y--;
            p1 += 2 * b2 * x - 2 * a2 * y + b2;
        }
    }
```



```cpp
    // Initial decision parameter for region 2
    int p2 = b2 * (x + 0.5) * (x + 0.5) + a2 * (y - 1) * (y - 1) - a2 * b2;

    // Region 2
    while (y >= 0) {
        plotEllipsePoints(xc, yc, x, y, points);
        y--;
        if (p2 > 0) {
            p2 -= 2 * a2 * y + a2;
        } else {
            x++;
            p2 += 2 * b2 * x - 2 * a2 * y + a2;
        }
    }

    // Display the calculated points in the console
    cout << "Ellipse Points:\n";
    for (const auto& point : points) {
        cout << "(" << point.first << ", " << point.second << ")\n";
    }
}

int main() {
    int xc, yc, a, b;
    cout << "Enter the center of the ellipse (xc, yc): ";
    cin >> xc >> yc;
    cout << "Enter the semi-major axis (a): ";
    cin >> a;
    cout << "Enter the semi-minor axis (b): ";
    cin >> b;

    // Draw the ellipse by calculating points
    midPointEllipseDrawing(xc, yc, a, b);

    return 0;
}
```

OUTPUT :

```
Enter the center of the ellipse (xc, yc): 0 0
Enter the semi-major axis (a): 2
Enter the semi-minor axis (b): 3
Ellipse Points:
(0, 3)
(0, 3)
(0, -3)
(0, -3)
(1, 3)
(-1, 3)
(1, -3)
(-1, -3)
(2, 2)
(-2, 2)
(2, -2)
(-2, -2)
(2, 1)
(-2, 1)
(2, -1)
(-2, -1)
(2, 0)
(-2, 0)
(2, 0)
(-2, 0)
```

```
Enter the center of the ellipse (xc, yc): 0 0
Enter the semi-major axis (a): 2
Enter the semi-minor axis (b): 3
Ellipse Points:
(0, 3)
(0, 3)
(0, -3)
(0, -3)
(1, 3)
(-1, 3)
(1, -3)
(-1, -3)
(2, 2)
(-2, 2)
(2, -2)
(-2, -2)
(2, 1)
(-2, 1)
(2, -1)
(-2, -1)
(2, 0)
(-2, 0)
(2, 0)
(-2, 0)
```