```
//  Q.10 Write C++ program to draw the following pattern. Use DDA line and
// Bresenham's circle drawing algorithm. Apply the concept of encapsulation.

#include <graphics.h>
#include <math.h>
#include <iostream>
using namespace std;

class DDA {
private:
   int x1, y1, x2, y2;

public:
   // Constructor to initialize coordinates
   DDA(int x1, int y1, int x2, int y2) {
      this->x1 = x1;
      this->y1 = y1;
      this->x2 = x2;
      this->y2 = y2;
   }

   // Function to draw line using DDA algorithm
   void drawLine() {
      float dx = x2 - x1;
      float dy = y2 - y1;
      float steps;
      float xIncrement, yIncrement;

      // Determine the number of steps required
      if (abs(dx) > abs(dy))
         steps = abs(dx);
      else
         steps = abs(dy);

      // Calculate the increment in x and y
      xIncrement = dx / steps;
      yIncrement = dy / steps;

      // Draw the points of the line
      float x = x1, y = y1;
      for (int i = 0; i <= steps; i++) {
         putpixel(round(x), round(y), WHITE);
         x += xIncrement;
         y += yIncrement;
      }
   }
};
```

```
class BresenhamCircle {
private:
    int xc, yc, radius;

public:
    // Constructor to initialize center and radius
    BresenhamCircle(int xc, int yc, int radius) {
        this->xc = xc;
        this->yc = yc;
        this->radius = radius;
    }

    // Function to draw the circle using Bresenham's Circle algorithm
    void drawCircle() {
        int x = 0, y = radius;
        int p = 3 - 2 * radius;

        // Draw the 8 symmetric points of the circle
        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);

            x++;

            // Mid-point decision to decide next point
            if (p <= 0)
                p = p + 4 * x + 6;
            else {
                y--;
                p = p + 4 * (x - y) + 10;
            }
        }
    }
};

void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3) {
    DDA line1(x1, y1, x2, y2);
    DDA line2(x2, y2, x3, y3);
    DDA line3(x3, y3, x1, y1);

    line1.drawLine();
    line2.drawLine();
    line3.drawLine();
}

int main() {
    // Initialize graphics mode
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    // 1. Draw Outer Circle (Radius = 100, Center = (300, 300))
    BresenhamCircle outerCircle(300, 300, 100);
    outerCircle.drawCircle();

    // 2. Draw Triangle inside the outer circle (Equilateral triangle)
    int radius = 100;
    // Calculate the three vertices of the equilateral triangle
    int x1 = 300, y1 = 300 - radius;
    int x2 = 300 + radius * sin(M_PI / 3), y2 = 300 + radius * cos(M_PI / 3);
    int x3 = 300 - radius * sin(M_PI / 3), y3 = 300 + radius * cos(M_PI / 3);

    drawTriangle(x1, y1, x2, y2, x3, y3);

    // 3. Calculate the incenter (centroid of the incircle) and inradius
    // Lengths of the sides of the triangle
    float a = sqrt(pow(x2 - x3, 2) + pow(y2 - y3, 2));  // side BC
    float b = sqrt(pow(x1 - x3, 2) + pow(y1 - y3, 2));  // side AC
    float c = sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));  // side AB

    // Semi-perimeter of the triangle
    float s = (a + b + c) / 2;

    // Area of the triangle using Heron's formula
    float area = sqrt(s * (s - a) * (s - b) * (s - c));

    // Inradius (radius of the incircle)
    float r = area / s;

    // Incenter (centroid of the incircle)
    int Ix = (a * x1 + b * x2 + c * x3) / (a + b + c);
    int Iy = (a * y1 + b * y2 + c * y3) / (a + b + c);

    // Draw the inner circle that touches the sides of the triangle
    BresenhamCircle innerCircle(Ix, Iy, r);
    innerCircle.drawCircle();

    // Wait for user input to close the window
    getch();
    closegraph();
    return 0;
}
```