**DAY-6**

**CODE:**

#include "mainwindow.h"

#include <QPainter>

#include <QStack>

#include <QPaintEvent>

#include <QDebug>

#include <algorithm>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent), canvas(400, 400, QImage::Format\_RGB32)

{

setFixedSize(400, 400);

initCanvas();

drawShapes();

}

MainWindow::~MainWindow(){}

void MainWindow::initCanvas()

{

canvas.fill(Qt::white);

}

void MainWindow::drawShapes()

{

QPainter painter(&canvas);

painter.setPen(Qt::black);

// --- Draw a rectangle outline ---

QRect rect(50, 50, 100, 80);

painter.drawRect(rect);

// --- Draw a circle (ellipse) outline ---

QRect ellipseRect(200, 50, 80, 80);

painter.drawEllipse(ellipseRect);

// --- Flood Fill: Fill the circle with red ---

QPoint circleSeed(ellipseRect.center());

floodFill(circleSeed.x(), circleSeed.y(), Qt::white, Qt::red);

// --- Boundary Fill: Fill the rectangle with green ---

QPoint rectSeed(rect.center());

boundaryFill(rectSeed.x(), rectSeed.y(), Qt::green, Qt::black);

QPolygon polygon;

polygon << QPoint(50, 200) << QPoint(150, 200) << QPoint(130, 300) << QPoint(70, 300);

painter.drawPolygon(polygon);

scanLineFill(polygon, Qt::blue);

}

// Flood Fill Algorithm (iterative using a stack)

void MainWindow::floodFill(int x, int y, const QColor &targetColor, const QColor &replacementColor)

{

if (targetColor == replacementColor)

return;

QStack<QPoint> stack;

stack.push(QPoint(x, y));

while (!stack.isEmpty()) {

QPoint p = stack.pop();

int px = p.x(), py = p.y();

if (px < 0 || px >= canvas.width() || py < 0 || py >= canvas.height())

continue;

if (QColor(canvas.pixel(px, py)) != targetColor)

continue;

canvas.setPixel(px, py, replacementColor.rgb());

stack.push(QPoint(px + 1, py));

stack.push(QPoint(px - 1, py));

stack.push(QPoint(px, py + 1));

stack.push(QPoint(px, py - 1));

}

}

// Boundary Fill Algorithm (recursive)

void MainWindow::boundaryFill(int x, int y, const QColor &fillColor, const QColor &boundaryColor)

{

if (x < 0 || x >= canvas.width() || y < 0 || y >= canvas.height())

return;

QColor currentColor = QColor(canvas.pixel(x, y));

if (currentColor == boundaryColor || currentColor == fillColor)

return;

canvas.setPixel(x, y, fillColor.rgb());

boundaryFill(x + 1, y, fillColor, boundaryColor);

boundaryFill(x - 1, y, fillColor, boundaryColor);

boundaryFill(x, y + 1, fillColor, boundaryColor);

boundaryFill(x, y - 1, fillColor, boundaryColor);

}

// Scan-Line Polygon Fill Algorithm

void MainWindow::scanLineFill(const QPolygon &polygon, const QColor &fillColor)

{

int yMin = canvas.height(), yMax = 0;

for (int i = 0; i < polygon.size(); i++) {

int y = polygon[i].y();

yMin = std::min(yMin, y);

yMax = std::max(yMax, y);

}

QPainter painter(&canvas);

painter.setPen(fillColor);

for (int y = yMin; y <= yMax; y++) {

QList<int> intersections;

for (int i = 0; i < polygon.size(); i++) {

QPoint p1 = polygon[i];

QPoint p2 = polygon[(i + 1) % polygon.size()];

if ((p1.y() <= y && p2.y() > y) || (p2.y() <= y && p1.y() > y)) {

int x = p1.x() + (y - p1.y()) \* (p2.x() - p1.x()) / (p2.y() - p1.y());

intersections.append(x);

}

}

std::sort(intersections.begin(), intersections.end());

for (int i = 0; i < intersections.size(); i += 2) {

if (i + 1 < intersections.size()) {

int xStart = intersections[i];

int xEnd = intersections[i + 1];

painter.drawLine(xStart, y, xEnd, y);

}

}

}

}

void MainWindow::paintEvent(QPaintEvent \*event)

{

floodFill(240, 90, Qt::white, Qt::red);

boundaryFill(100, 90, Qt::green, Qt::black);

scanLineFill(QPolygon({{50, 200}, {150, 200}, {130, 300}, {70, 300}}), Qt::blue);

QPainter painter(this);

painter.drawImage(0, 0, canvas);

}

**OUTPUT:**

