190001016
Garvit Galgat
Assignment 2

Q1.)

Screenshot

Code

```
// 190001016
// Garvit Galgat

#include <GL/glut.h>
#include <iostream>
#include <math.h>
#include <vector>
#include <utility>

/**
 * initialPoint.first  = x coordinate
 * initialPoint.second = y coordinate
 */
std::pair<int, int> initialPoint;

/**
 * finalPoint.first  = x coordinate
 * finalPoint.second = y coordinate
 */
std::pair<int, int> finalPoint;

/**
 * @brief drawing lines using polynomial method
 *
 * @return std::vector<std::pair<int, int>> vector containing all points in order
 */
std::vector<std::pair<int, int>> drawWithPoly() {
    int delta_x = finalPoint.first - initialPoint.first;
    int delta_y = finalPoint.second - initialPoint.second;

    std::vector<std::pair<int, int>> poly_points;
    if (delta_x == 0) {
        for (int y = initialPoint.second; y <= finalPoint.second; y++) {
            poly_points.push_back({initialPoint.first, y});
        }
        return poly_points;
    }

    double slope = delta_y * 1.0 / delta_x;
    double intercept = 1.0 * (finalPoint.first * initialPoint.second - finalPoint.second *
initialPoint.first) / (finalPoint.first - initialPoint.first);
```

```
    // if abs of slope is less than 1, we increment x by 1 and calculate y
    if (abs(slope) <= 1) {
        int x = initialPoint.first;
        double y = initialPoint.second;
        poly_points.push_back({x, (int)y});
        while (x + 1 <= finalPoint.first) {
            y = slope * (x + 1) + intercept;
            poly_points.push_back({x + 1, round(y)});
            x += 1;
        }
    } else {
        // otherwise we increment y by 1 (if slope is positive)
        // else we decrement y be 1 and calculate x
        double x = initialPoint.first;
        int y = initialPoint.second;
        poly_points.push_back({(int)x, y});
        if (slope > 0) {
            while (y + 1 <= finalPoint.second) {
                x = ((y + 1) - intercept) / slope;
                poly_points.push_back({round(x), y + 1});
                y += 1;
            }
        } else {
            while (y - 1 >= finalPoint.second) {
                x = ((y - 1) - intercept) / slope;
                poly_points.push_back({round(x), y - 1});
                y -= 1;
            }
        }
    }
    return poly_points;
}

/**
 * @brief drawing lines using DDA method
 *
 * @return std::vector<std::pair<int, int>> vector containing all points in order
 */
std::vector<std::pair<int, int>> drawWithDDA() {
    int delta_x = finalPoint.first - initialPoint.first;
    int delta_y = finalPoint.second - initialPoint.second;

    std::vector<std::pair<int, int>> dda_points;
    if (delta_x == 0) {
```

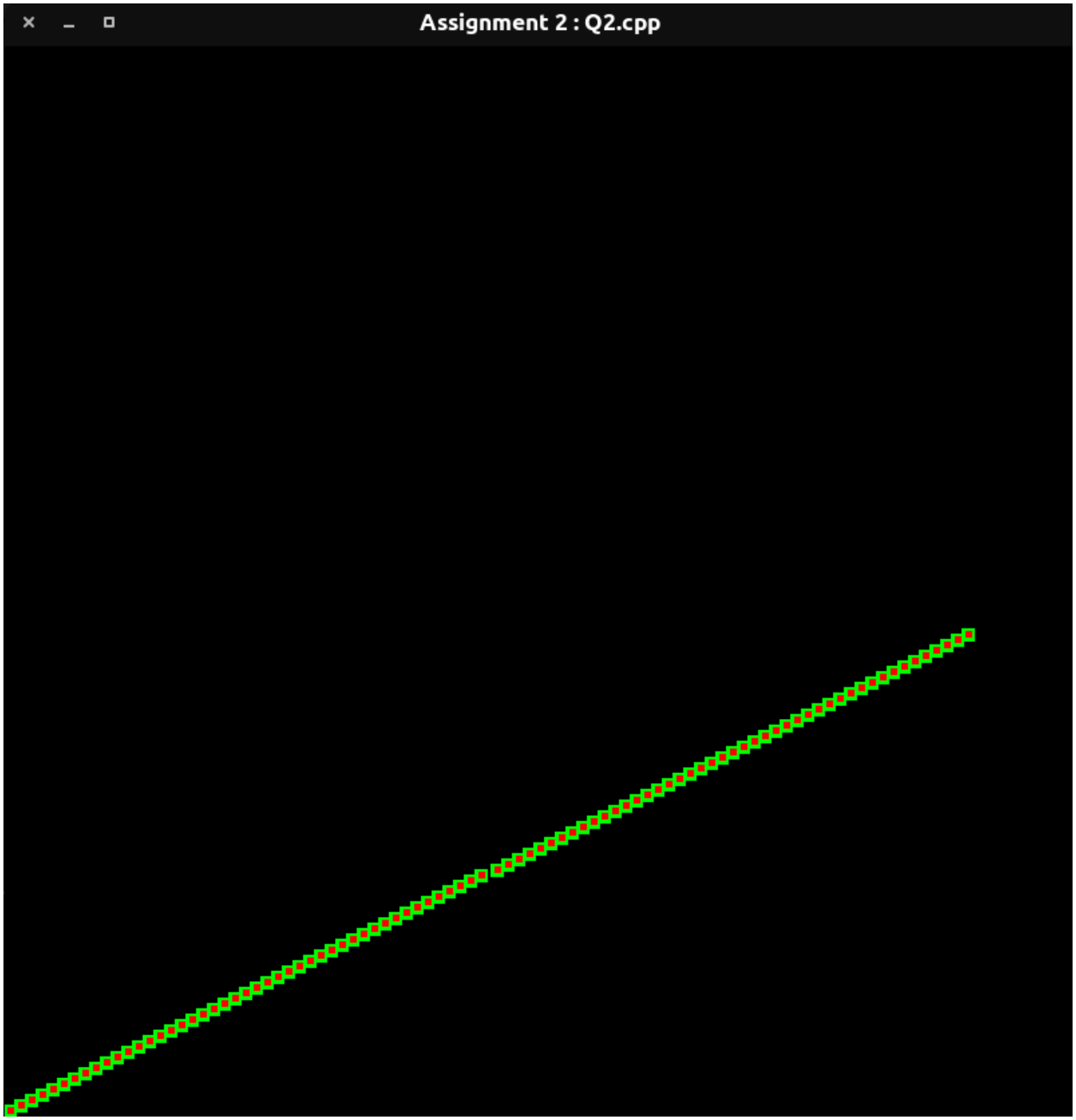
```
        for (int y = initialPoint.second; y <= finalPoint.second; y++) {
            dda_points.push_back({initialPoint.first, y});
        }
        return dda_points;
    }

    double slope = delta_y * 1.0 / delta_x;
    double intercept = 1.0 * (finalPoint.first * initialPoint.second - finalPoint.second *
initialPoint.first) / (finalPoint.first - initialPoint.first);
    // if abs of slope is less than 1, we increment x by 1, and
    // increment y by slope
    if (abs(slope) < 1) {
        int x = initialPoint.first;
        double y = initialPoint.second;
        dda_points.push_back({x, (int)y});
        while (x + 1 <= finalPoint.first) {
            dda_points.push_back({x + 1, round(y + slope)});
            x += 1;
            y += slope;
        }
    } else {
        // otherwise we calculate the inverse of slope
        // and increment x by inverse of slope and increment y
        // by 1 is slope is positive else we decrement the slope by 1
        double x = initialPoint.first;
        int y = initialPoint.second;
        double inverseSlope = abs(1.0 / slope);
        dda_points.push_back({(int)x, y});
        if (slope > 0) {
            while (y + 1 <= finalPoint.second) {
                dda_points.push_back({round(x + inverseSlope), y + 1});
                x += inverseSlope;
                y += 1;
            }
        } else {
            while (y - 1 >= finalPoint.second) {
                dda_points.push_back({round(x + inverseSlope), y - 1});
                x += inverseSlope;
                y -= 1;
            }
        }
    }
    return dda_points;
}
```

```
void draw() {
    auto poly_points = drawWithPoly();
    auto dda_points = drawWithDDA();

    glClear(GL_COLOR_BUFFER_BIT);

    glPointSize(10);
    glBegin(GL_POINTS);
    glColor3f(0, 1, 0);
    for (auto i : poly_points) {
        glVertex2f(i.first, i.second);
    }
    glEnd();

    glPointSize(5);
    glBegin(GL_POINTS);
    for (auto i : dda_points) {
        glColor3f(1, 0, 0);
        glVertex2f(i.first, i.second);
    }
    glEnd();

    glFlush();
}

int main(int argc, char* argv[]) {
    int x1, x2, y1, y2;
    std::cin >> x1 >> y1 >> x2 >> y2;

    initialPoint = {x1, y1};
    finalPoint = {x2, y2};

    // using initial point as the one on the left side
    if (initialPoint.first > finalPoint.first) swap(initialPoint, finalPoint);

    glutInit(&argc, argv);
    glutInitWindowPosition(300, 300);
    glutInitWindowSize(800, 800);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutCreateWindow("Assignment 2 : Q1.cpp");
    glutDisplayFunc(draw);
    gluOrtho2D(0, 200, 0, 200);
    glutMainLoop();
```

```
}
```

Q2.)

Screenshot



Code

```
// 190001016
// Garvit Galgat

#include <GL/glut.h>
#include <iostream>
#include <math.h>
#include <vector>
#include <utility>

/**
 * initialPoint.first  = x coordinate
 * initialPoint.second = y coordinate
 */
std::pair<int, int> initialPoint;

/**
 * finalPoint.first  = x coordinate
 * finalPoint.second = y coordinate
 */
std::pair<int, int> finalPoint;

/**
 * @brief drawing lines using polynomial method
 *
 * @return std::vector<std::pair<int, int>> vector containing all points in order
 */
std::vector<std::pair<int, int>> drawWithPoly() {
    int delta_x = finalPoint.first - initialPoint.first;
    int delta_y = finalPoint.second - initialPoint.second;

    std::vector<std::pair<int, int>> poly_points;
    if (delta_x == 0) {
        for (int y = initialPoint.second; y <= finalPoint.second; y++) {
            poly_points.push_back({initialPoint.first, y});
        }
        return poly_points;
    }

    double slope = delta_y * 1.0 / delta_x;
    double intercept = 1.0 * (finalPoint.first * initialPoint.second - finalPoint.second *
initialPoint.first) / (finalPoint.first - initialPoint.first);

    // inverse of Q1
```

```
    if (abs(slope) <= 1) {
        double x = initialPoint.first;
        int y = initialPoint.second;
        poly_points.push_back({(int)x, y});
        if (slope > 0) {
            while (y + 1 <= finalPoint.second) {
                x = ((y + 1) - intercept) / slope;
                poly_points.push_back({round(x), y + 1});
                y += 1;
            }
        } else {
            while (y - 1 >= finalPoint.second) {
                x = ((y - 1) - intercept) / slope;
                poly_points.push_back({round(x), y - 1});
                y -= 1;
            }
        }
    } else {
        int x = initialPoint.first;
        double y = initialPoint.second;
        poly_points.push_back({x, (int)y});
        while (x + 1 <= finalPoint.first) {
            y = slope * (x + 1) + intercept;
            poly_points.push_back({x + 1, round(y)});
            x += 1;
        }
    }
    return poly_points;
}

/**
 * @brief drawing lines using DDA method
 *
 * @return std::vector<std::pair<int, int>> vector containing all points in order
 */
std::vector<std::pair<int, int>> drawWithDDA() {
    int delta_x = finalPoint.first - initialPoint.first;
    int delta_y = finalPoint.second - initialPoint.second;

    std::vector<std::pair<int, int>> dda_points;
    if (delta_x == 0) {
        for (int y = initialPoint.second; y <= finalPoint.second; y++) {
            dda_points.push_back({initialPoint.first, y});
        }
```

```
        return dda_points;
    }

    double slope = delta_y * 1.0 / delta_x;
    double intercept = 1.0 * (finalPoint.first * initialPoint.second - finalPoint.second *
initialPoint.first) / (finalPoint.first - initialPoint.first);

    // inverse of Q1
    if (abs(slope) < 1) {
        double x = initialPoint.first;
        int y = initialPoint.second;
        double inverseSlope = abs(1.0 / slope);
        dda_points.push_back({(int)x, y});
        if (slope > 0) {
            while (y + 1 <= finalPoint.second) {
                dda_points.push_back({round(x + inverseSlope), y + 1});
                x += inverseSlope;
                y += 1;
            }
        } else {
            while (y - 1 >= finalPoint.second) {
                dda_points.push_back({round(x + inverseSlope), y - 1});
                x += inverseSlope;
                y -= 1;
            }
        }
    } else {
        int x = initialPoint.first;
        double y = initialPoint.second;
        dda_points.push_back({x, (int)y});
        while (x + 1 <= finalPoint.first) {
            dda_points.push_back({x + 1, round(y + slope)});
            x += 1;
            y += slope;
        }
    }
    return dda_points;
}

void draw() {
    auto poly_points = drawWithPoly();
    auto dda_points = drawWithDDA();

    glClear(GL_COLOR_BUFFER_BIT);
```

```
    glPointSize(10);
    glBegin(GL_POINTS);
    glColor3f(0, 1, 0);
    for (auto i : poly_points) {
        glVertex2f(i.first, i.second);
    }
    glEnd();

    glPointSize(5);
    glBegin(GL_POINTS);
    glColor3f(1, 0, 0);
    for (auto i : dda_points) {
        glVertex2f(i.first, i.second);
    }
    glEnd();
    glFlush();
}

int main(int argc, char* argv[]) {
    int x1, x2, y1, y2;
    std::cin >> x1 >> y1 >> x2 >> y2;

    initialPoint = {x1, y1};
    finalPoint = {x2, y2};

    // using initial point as the one on the left side
    if (initialPoint.first > finalPoint.first) swap(initialPoint, finalPoint);

    glutInit(&argc, argv);
    glutInitWindowPosition(300, 300);
    glutInitWindowSize(800, 800);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutCreateWindow("Assignment 2 : Q2.cpp");
    glutDisplayFunc(draw);
    gluOrtho2D(0, 200, 0, 200);
    glutMainLoop();
}
```