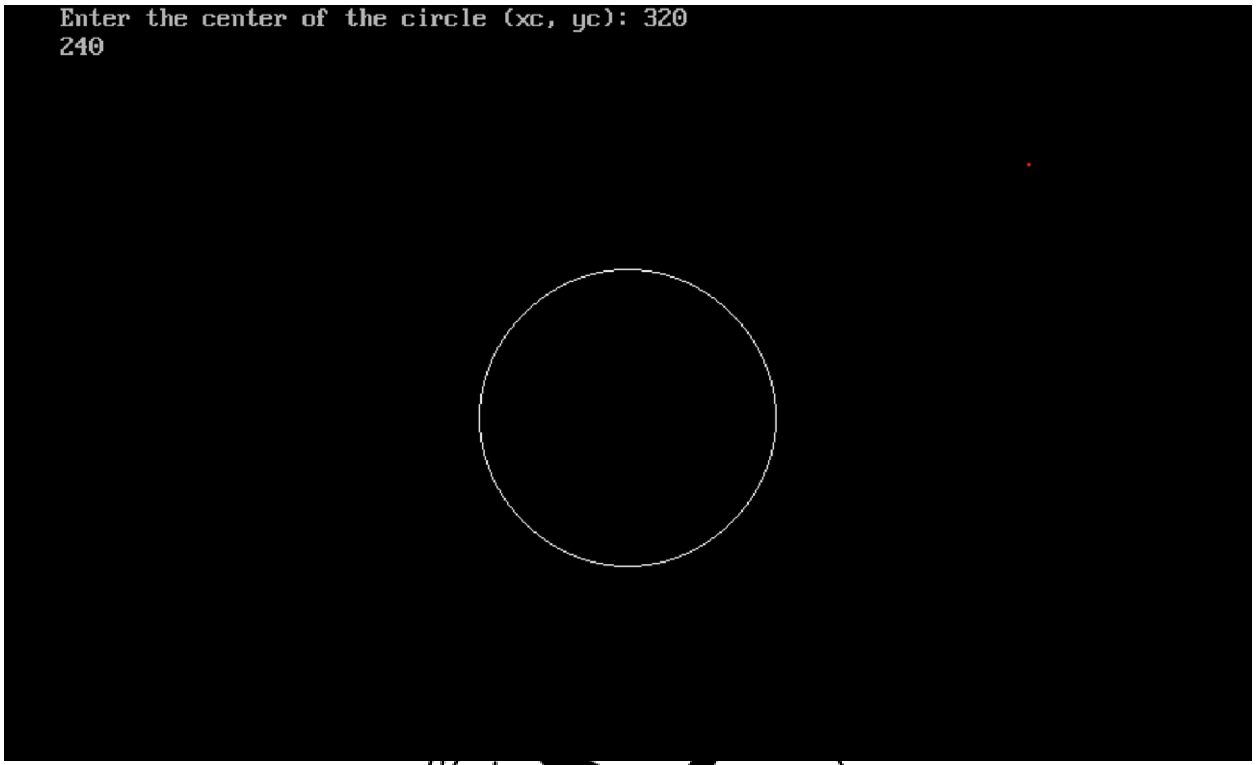
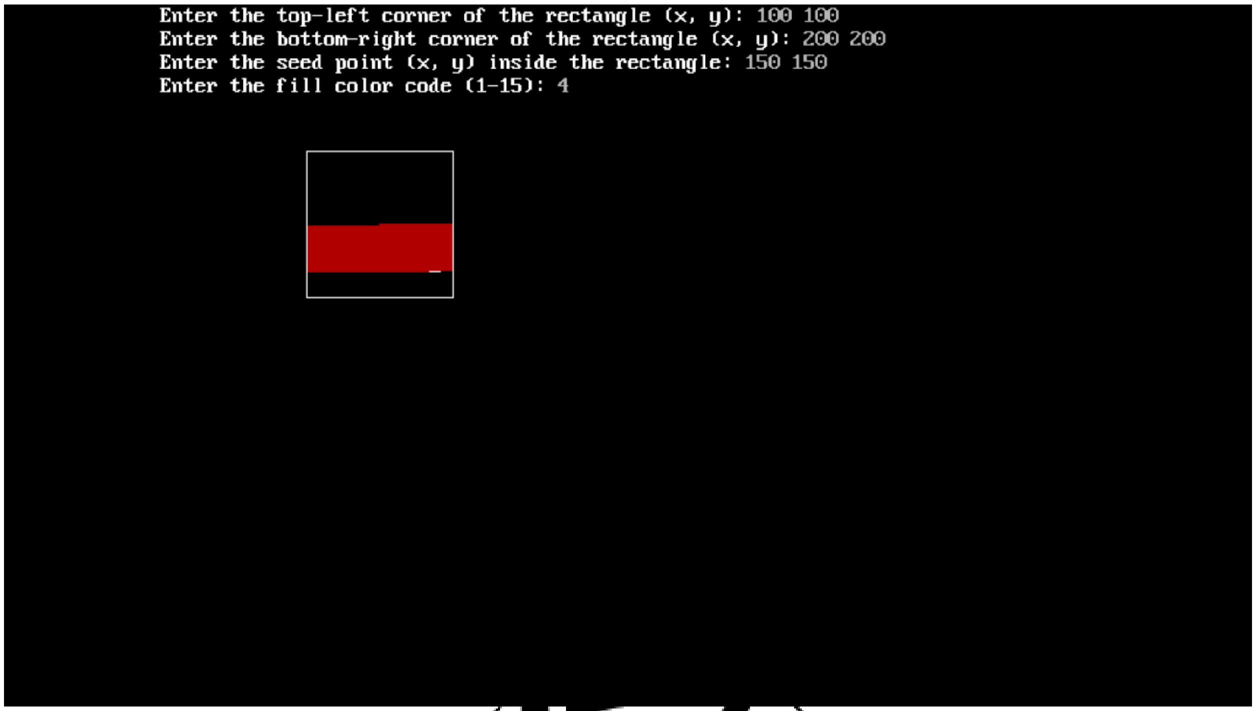
```
122 PRACTICLE 4
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
void bresenhamCircle(int xc, int yc, int r) {
    int x = 0, y = r;
    int d = 3 - 2 * r;
    while (x <= y) {
        putpixel(xc + x, yc + y, WHITE);
        putpixel(xc - x, yc + y, WHITE);
        putpixel(xc + x, yc - 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++;

        if (d > 0) {
            y--;
            d = d + 4 * (x - y) + 10;
        } else {
            d = d + 4 * x + 6;
        }
    }
}
int main() {
    int gd = DETECT, gm;
    int xc, yc, r;
    printf("Enter the center of the circle (xc, yc): ");
    scanf("%d %d", &xc, &yc);
    printf("Enter the radius of the circle: ");
    scanf("%d", &r);
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");
    bresenhamCircle(xc, yc, r);
    getch();
    closegraph();
    return 0;
}
```



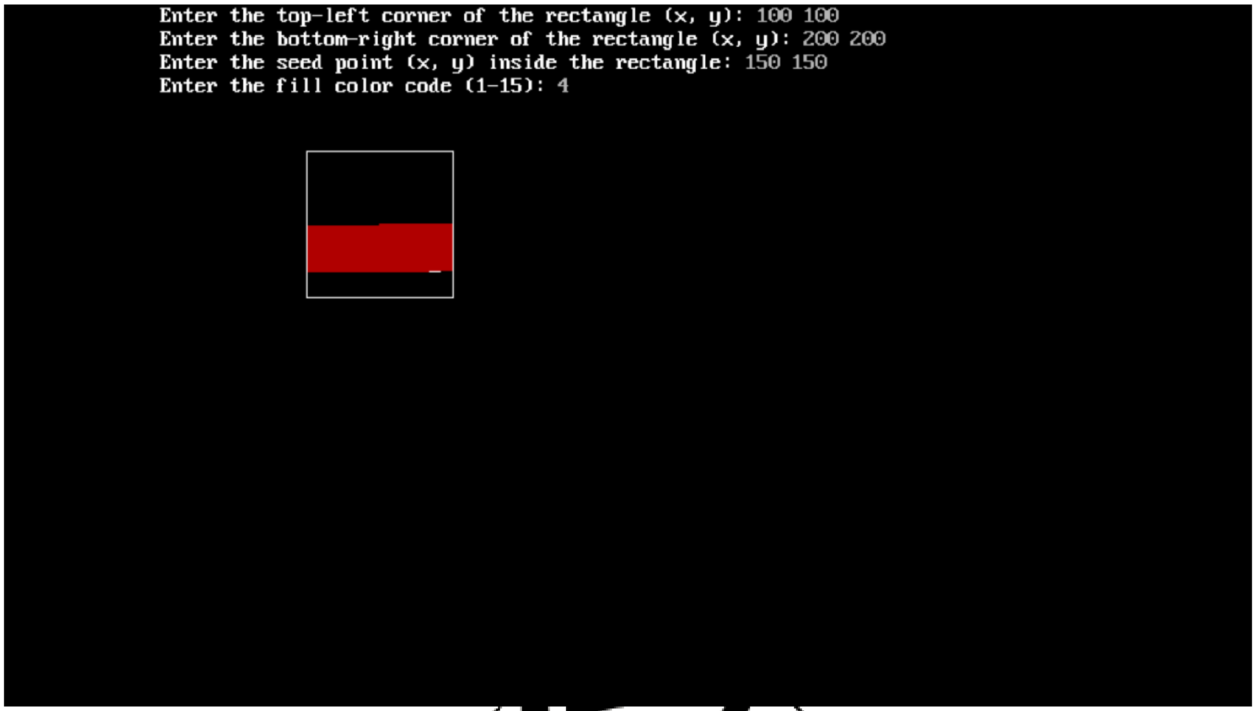
```
Enter the center of the circle (xc, yc): 320
240
```

```
122 PRACTICLE 5
#include <graphics.h>
#include <conio.h>
#include <dos.h>
void floodFill(int x, int y, int fillColor, int borderColor) {
    int currentColor = getpixel(x, y);
    if (currentColor != borderColor && currentColor != fillColor) {
        putpixel(x, y, fillColor);
        // Recursively fill neighboring pixels
        floodFill(x + 1, y, fillColor, borderColor);
        floodFill(x - 1, y, fillColor, borderColor);
        floodFill(x, y + 1, fillColor, borderColor);
        floodFill(x, y - 1, fillColor, borderColor);
    }
}
int main() {
    int gd = DETECT, gm;
    int borderColor = WHITE; // Define border color
    int left, top, right, bottom;
    int seedX, seedY;
    int fillColor;
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

    printf("Enter the top-left corner of the rectangle (x, y): ");
    scanf("%d %d", &left, &top);
    printf("Enter the bottom-right corner of the rectangle (x, y): ");
    scanf("%d %d", &right, &bottom);
    printf("Enter the seed point (x, y) inside the rectangle: ");
    scanf("%d %d", &seedX, &seedY);
    printf("Enter the fill color code (1-15): ");
    scanf("%d", &fillColor);

    setcolor(borderColor);
    rectangle(left, top, right, bottom);

    floodFill(seedX, seedY, fillColor, borderColor);
    getch();
    closegraph();
    return 0;
}
```



```
Enter the top-left corner of the rectangle (x, y): 100 100
Enter the bottom-right corner of the rectangle (x, y): 200 200
Enter the seed point (x, y) inside the rectangle: 150 150
Enter the fill color code (1-15): 4
```

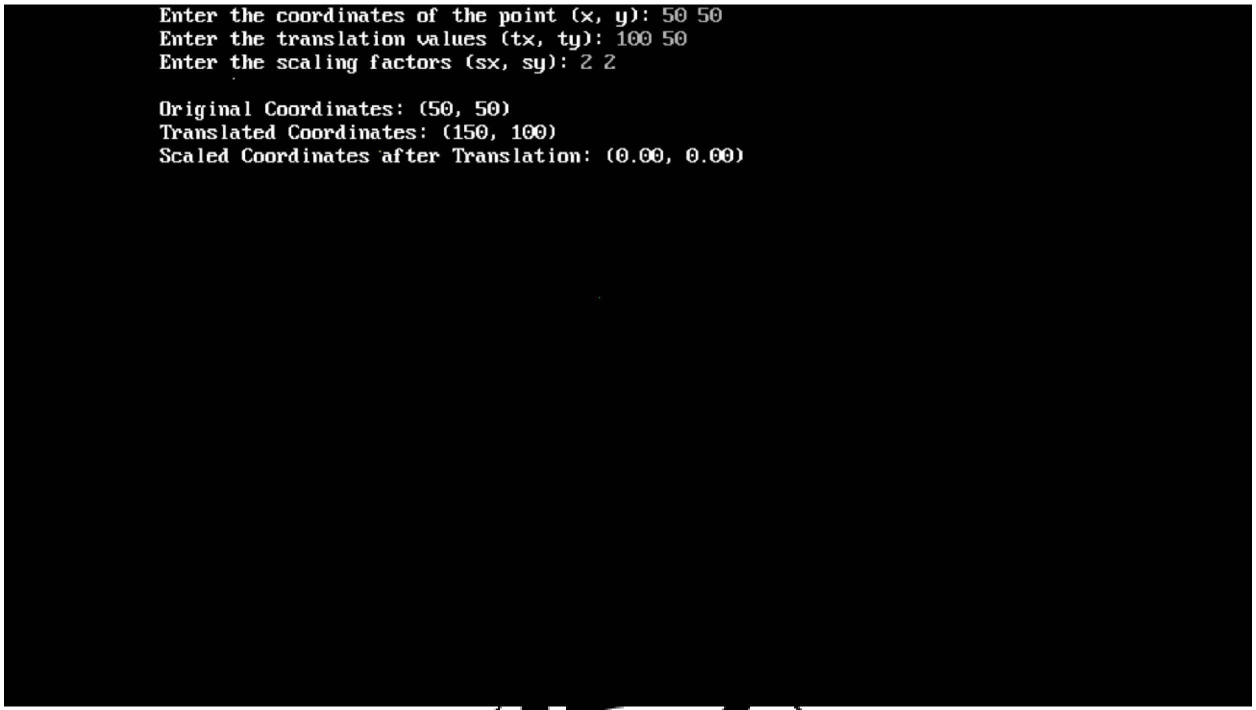
```
122 PRACTICLE 6
#include <graphics.h>
#include <conio.h>
#include <dos.h>
void boundaryFill(int x, int y, int fillColor, int boundaryColor) {
    int currentColor = getpixel(x, y);
    if (currentColor != boundaryColor && currentColor != fillColor) {
        putpixel(x, y, fillColor);

        // Recursively fill the neighboring pixels
        boundaryFill(x + 1, y, fillColor, boundaryColor);
        boundaryFill(x - 1, y, fillColor, boundaryColor);
        boundaryFill(x, y + 1, fillColor, boundaryColor);
        boundaryFill(x, y - 1, fillColor, boundaryColor);
    }
}
int main() {
    int gd = DETECT, gm;
    int boundaryColor = WHITE;
    int left, top, right, bottom;
    int seedX, seedY;
    int fillColor;
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");
    printf("Enter the top-left corner of the rectangle (x, y): ");
    scanf("%d %d", &left, &top);
    printf("Enter the bottom-right corner of the rectangle (x, y): ");
    scanf("%d %d", &right, &bottom);
    printf("Enter the seed point (x, y) inside the rectangle: ");
    scanf("%d %d", &seedX, &seedY);
    printf("Enter the fill color code (1-15): ");
    scanf("%d", &fillColor);
    setcolor(boundaryColor);
    rectangle(left, top, right, bottom);
    boundaryFill(seedX, seedY, fillColor, boundaryColor);
    getch();
    closegraph();
    return 0;
}
```



```
Enter the top-left corner of the rectangle (x, y): 100 100
Enter the bottom-right corner of the rectangle (x, y): 200 200
Enter the seed point (x, y) inside the rectangle: 150 150
Enter the fill color code (1-15): 4
```

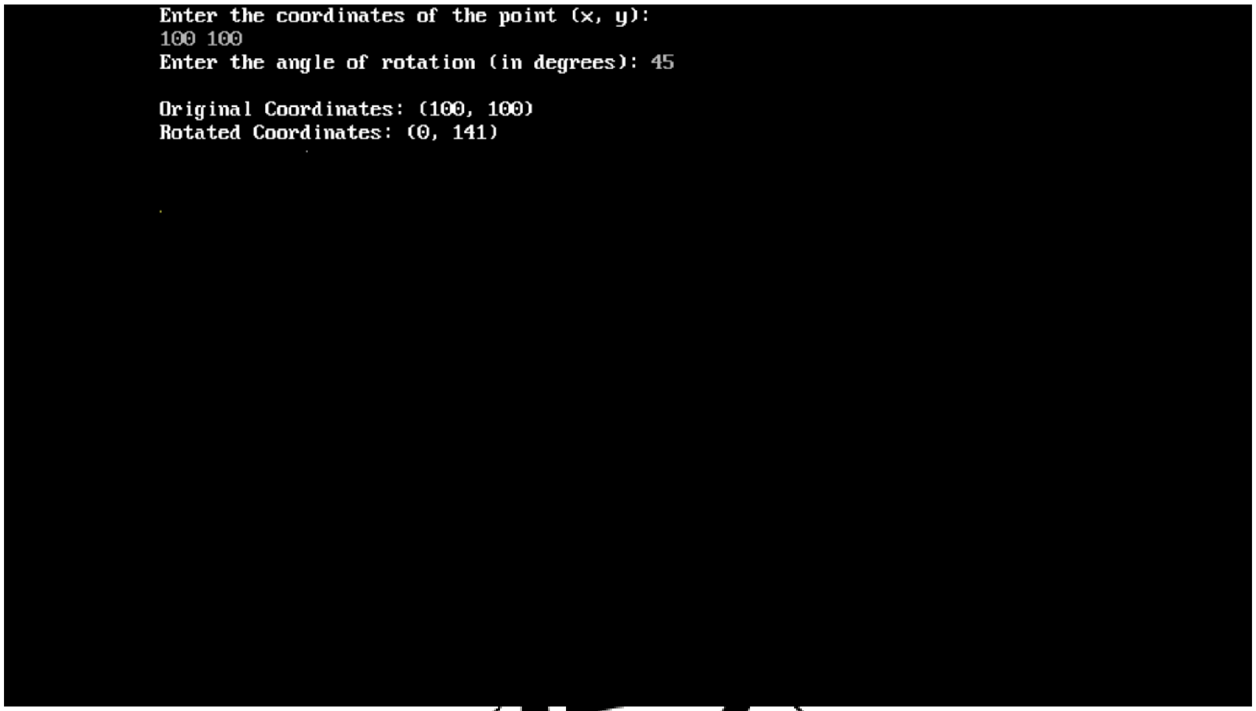
COMPUTER ENG.

```
122 PRACTICLE 7
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main() {
    int gd = DETECT, gm;
    int x, y;          // Original coordinates
    int tx, ty;        // Translation values
    float sx, sy;      // Scaling factors
    int translatedX, translatedY, scaledX, scaledY;
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");
    printf("Enter the coordinates of the point (x, y): ");
    scanf("%d %d", &x, &y);
    printf("Enter the translation values (tx, ty): ");
    scanf("%d %d", &tx, &ty);
    printf("Enter the scaling factors (sx, sy): ");
    scanf("%f %f", &sx, &sy);
    translatedX = x + tx;
    translatedY = y + ty;
    scaledX = translatedX * sx;
    scaledY = translatedY * sy;
    printf("\nOriginal Coordinates: (%d, %d)", x, y);
    printf("\nTranslated Coordinates: (%d, %d)", translatedX, translatedY);
    printf("\nScaled Coordinates after Translation: (%.2f, %.2f)", scaledX, scaledY);
    putpixel(x, y, WHITE);            // Original point in white
    putpixel(translatedX, translatedY, YELLOW);  // Translated point in yellow
    putpixel(scaledX, scaledY, GREEN);           // Scaled point in green
    getch();
    closegraph();
    return 0;
}
```



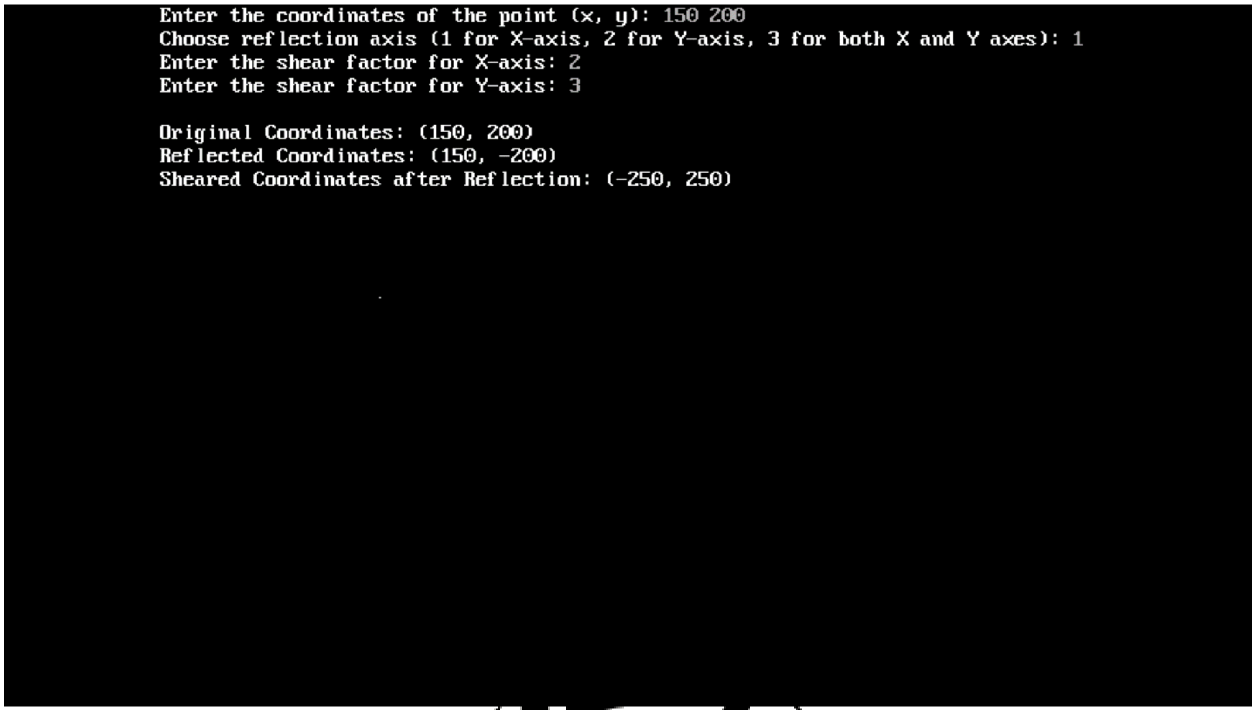
```
Enter the coordinates of the point (x, y): 50 50
Enter the translation values (tx, ty): 100 50
Enter the scaling factors (sx, sy): 2 2

Original Coordinates: (50, 50)
Translated Coordinates: (150, 100)
Scaled Coordinates after Translation: (0.00, 0.00)
```

```
122 PRACTICLE 8
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>

int main() {
    int gd = DETECT, gm;


    int x, y;           // Original coordinates
    float angle;        // Angle of rotation
    float radian;       // Angle in radians
    int rotatedX, rotatedY;
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

    printf("Enter the coordinates of the point (x, y): ");
    scanf("%d %d", &x, &y);

    printf("Enter the angle of rotation (in degrees): ");
    scanf("%f", &angle);

    radian = angle * (M_PI / 180);
    rotatedX = x * cos(radian) - y * sin(radian);
    rotatedY = x * sin(radian) + y * cos(radian);
    printf("\nOriginal Coordinates: (%d, %d)", x, y);
    printf("\nRotated Coordinates: (%d, %d)", rotatedX, rotatedY);
    putpixel(x, y, WHITE);             // Original point in white
    putpixel(rotatedX, rotatedY, YELLOW);  // Rotated point in yellow
    getch();
    closegraph();
    return 0;
}
```



```
Enter the coordinates of the point (x, y):
100 100
Enter the angle of rotation (in degrees): 45

Original Coordinates: (100, 100)
Rotated Coordinates: (0, 141)
```

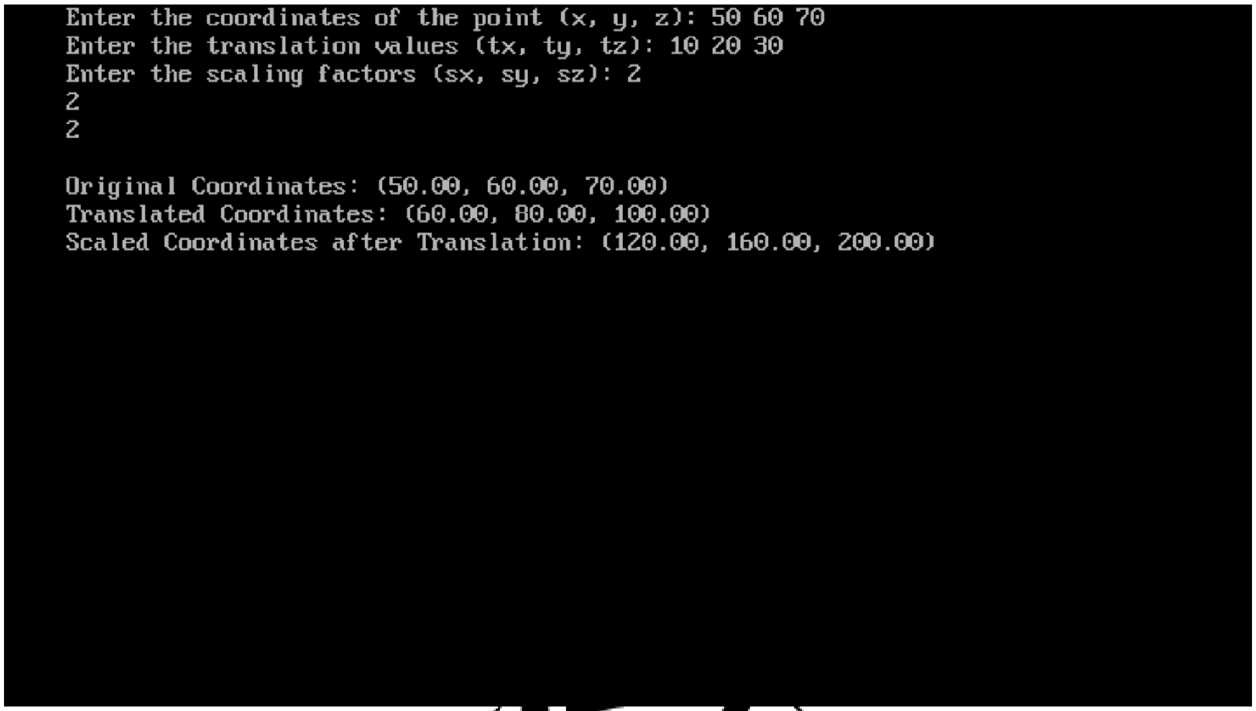
```
122 PRACTICLE 9
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
    int gd = DETECT, gm;
    int x, y;               // Original coordinates of the point
    int choice;             // Reflection axis choice
    float shearX, shearY;   // Shear factors for X and Y axes
    int reflectedX, reflectedY, shearedX, shearedY;
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");
    printf("Enter the coordinates of the point (x, y): ");
    scanf("%d %d", &x, &y);
    printf("Choose reflection axis (1 for X-axis, 2 for Y-axis, 3 for both X and Y axes): ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:  // Reflection across X-axis
            reflectedX = x;
            reflectedY = -y;
            break;
        case 2:  // Reflection across Y-axis
            reflectedX = -x;
            reflectedY = y;
            break;
        case 3:  // Reflection across both X and Y axes
            reflectedX = -x;
            reflectedY = -y;
            break;
        default:
            printf("Invalid choice! Using original coordinates.\n");
            reflectedX = x;
            reflectedY = y;
    }
    printf("Enter the shear factor for X-axis: ");
    scanf("%f", &shearX);
    printf("Enter the shear factor for Y-axis: ");
    scanf("%f", &shearY);
    shearedX = reflectedX + shearX * reflectedY;
    shearedY = reflectedY + shearY * reflectedX;
    printf("\nOriginal Coordinates: (%d, %d)", x, y);
    printf("\nReflected Coordinates: (%d, %d)", reflectedX, reflectedY);
    printf("\nSheared Coordinates after Reflection: (%d, %d)", shearedX, shearedY);
    putpixel(x, y, WHITE);                  // Original point in white
    putpixel(reflectedX, reflectedY, YELLOW);  // Reflected point in yellow
    putpixel(shearedX, shearedY, GREEN);       // Sheared point in green
    getch();
    closegraph();
    return 0;
}
```



```
Enter the coordinates of the point (x, y): 150 200
Choose reflection axis (1 for X-axis, 2 for Y-axis, 3 for both X and Y axes): 1
Enter the shear factor for X-axis: 2
Enter the shear factor for Y-axis: 3

Original Coordinates: (150, 200)
Reflected Coordinates: (150, -200)
Sheared Coordinates after Reflection: (-250, 250)
```

COMPUTER ENG.

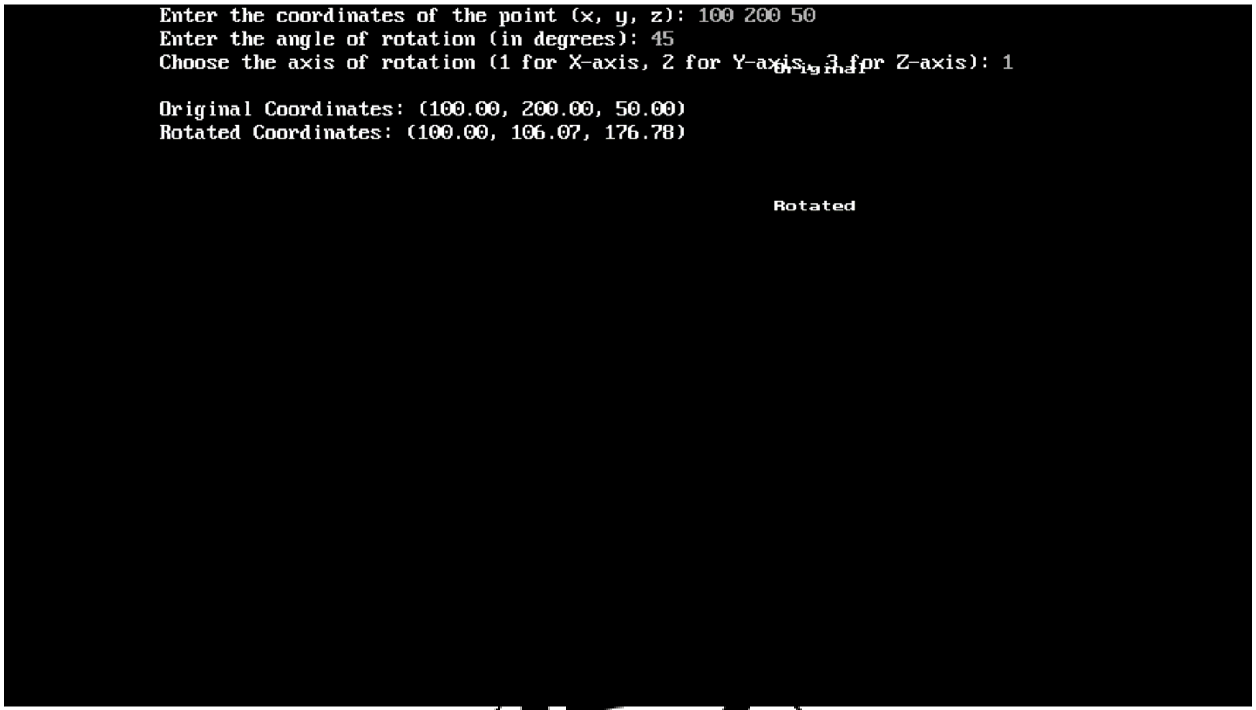
```
122 PRACTICLE 10
#include <stdio.h>

void main() {
    float x, y, z;          // Original coordinates of the point
    float tx, ty, tz;       // Translation values for x, y, and z
    float sx, sy, sz;       // Scaling factors for x, y, and z
    float translatedX, translatedY, translatedZ;  // Translated coordinates
    float scaledX, scaledY, scaledZ;
    clrscr();              // Scaled coordinates

    // Input for original coordinates of the point
    printf("Enter the coordinates of the point (x, y, z): ");
    scanf("%f %f %f", &x, &y, &z);

    // Input for translation values
    printf("Enter the translation values (tx, ty, tz): ");
    scanf("%f %f %f", &tx, &ty, &tz);

    // Perform translation
    translatedX = x + tx;
    translatedY = y + ty;
    translatedZ = z + tz;

    // Input for scaling factors
    printf("Enter the scaling factors (sx, sy, sz): ");
    scanf("%f %f %f", &sx, &sy, &sz);

    // Perform scaling on the translated coordinates
    scaledX = translatedX * sx;
    scaledY = translatedY * sy;
    scaledZ = translatedZ * sz;
    // Display results
    printf("\nOriginal Coordinates: (%.2f, %.2f, %.2f)", x, y, z);
    printf("\nTranslated Coordinates: (%.2f, %.2f, %.2f)", translatedX, translatedY, translatedZ);
    printf("\nScaled Coordinates after Translation: (%.2f, %.2f, %.2f)", scaledX, scaledY, scaledZ);

   getch();
}
```



```
Enter the coordinates of the point (x, y, z): 50 60 70
Enter the translation values (tx, ty, tz): 10 20 30
Enter the scaling factors (sx, sy, sz): 2
2
2

Original Coordinates: (50.00, 60.00, 70.00)
Translated Coordinates: (60.00, 80.00, 100.00)
Scaled Coordinates after Translation: (120.00, 160.00, 200.00)
```

COMPUTER ENG.

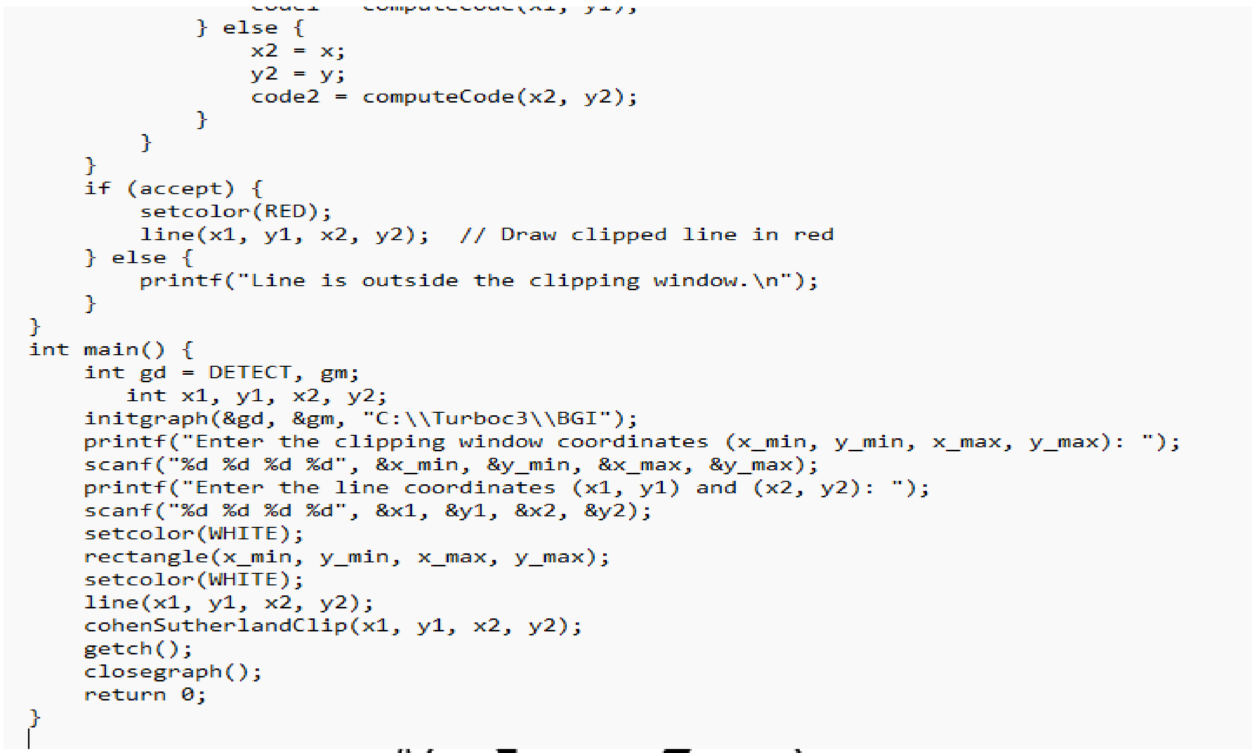
```
122 PRACTICLE 11
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>
#define PI 3.14159265
void plot_point(int x, int y, int color) {
    putpixel(320 + x, 240 - y, color);  // Offset to center the plot on screen
}
int main() {
    int gd = DETECT, gm;
    float x, y, z;              // Original coordinates of the point
    float angle;                // Rotation angle in degrees
    int choice;                 // Axis of rotation choice
    float radian;               // Angle in radians for rotation
    float rotatedX, rotatedY, rotatedZ;
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");
    printf("Enter the coordinates of the point (x, y, z): ");
    scanf("%f %f %f", &x, &y, &z);
    printf("Enter the angle of rotation (in degrees): ");
    scanf("%f", &angle);
    radian = angle * (PI / 180.0);
    printf("Choose the axis of rotation (1 for X-axis, 2 for Y-axis, 3 for Z-axis): ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:  // Rotation around X-axis
            rotatedX = x;
            rotatedY = y * cos(radian) - z * sin(radian);
            rotatedZ = y * sin(radian) + z * cos(radian);
            break;
        case 2:  // Rotation around Y-axis
            rotatedX = x * cos(radian) + z * sin(radian);
            rotatedY = y;
            rotatedZ = -x * sin(radian) + z * cos(radian);
            break;
        case 3:  // Rotation around Z-axis
            rotatedX = x * cos(radian) - y * sin(radian);
            rotatedY = x * sin(radian) + y * cos(radian);
            rotatedZ = z;
            break;
        default:
            printf("Invalid choice! No rotation applied.\n");
            rotatedX = x;
            rotatedY = y;
            rotatedZ = z;
    }
    plot_point((int)x, (int)y, WHITE);
    outtextxy(320 + (int)x, 240 - (int)y, "Original");
    plot_point((int)rotatedX, (int)rotatedY, YELLOW);
    outtextxy(320 + (int)rotatedX, 240 - (int)rotatedY, "Rotated");
    printf("\nOriginal Coordinates: (%.2f, %.2f, %.2f)", x, y, z);
    printf("\nRotated Coordinates: (%.2f, %.2f, %.2f)", rotatedX, rotatedY, rotatedZ);
    getch();
    closegraph();
    return 0;
}
```



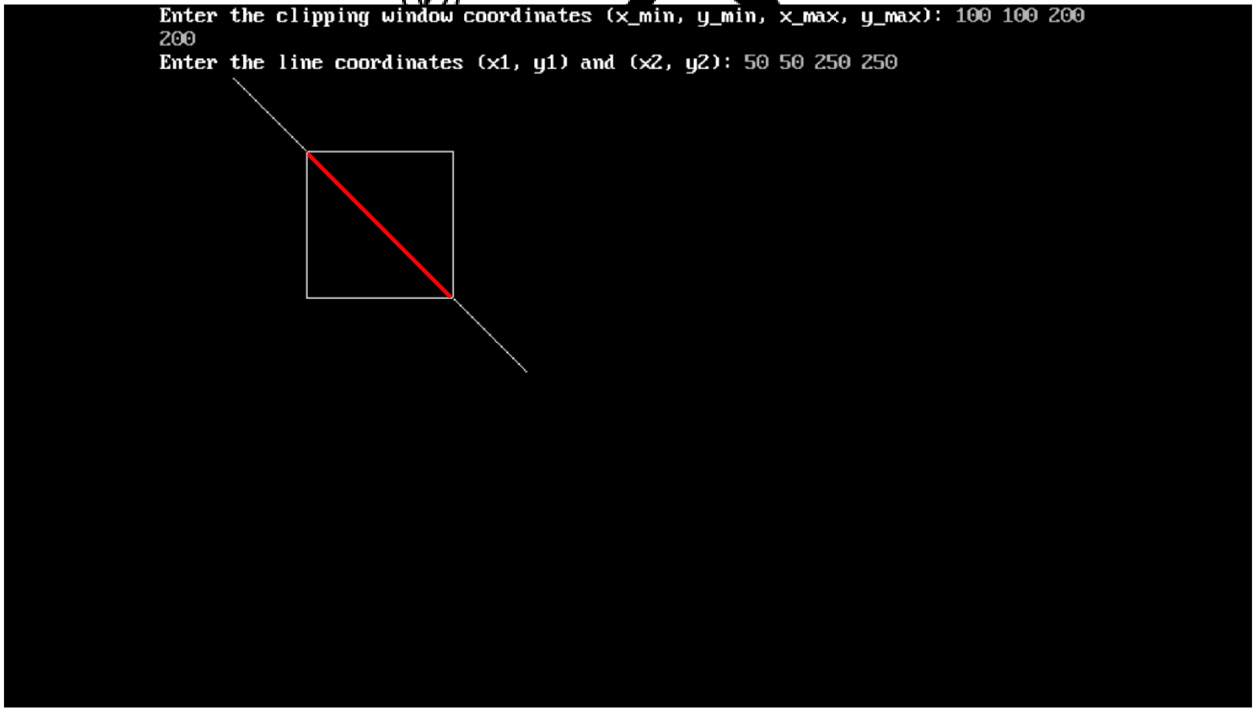
```
Enter the coordinates of the point (x, y, z): 100 200 50
Enter the angle of rotation (in degrees): 45
Choose the axis of rotation (1 for X-axis, 2 for Y-axis, 3 for Z-axis): 1

Original Coordinates: (100.00, 200.00, 50.00)
Rotated Coordinates: (100.00, 106.07, 176.78)
```

COMPUTER ENG.

```
122 PRACTICLE 12
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#define LEFT 1    // Binary codes for the 4 regions
#define RIGHT 2
#define BOTTOM 4
#define TOP 8
int x_min, y_min, x_max, y_max; // Clipping window coordinates
int computeCode(int x, int y) {
    int code = 0;
    if (x < x_min) code |= LEFT;
    if (x > x_max) code |= RIGHT;
    if (y < y_min) code |= BOTTOM;
    if (y > y_max) code |= TOP;
    return code;
}
void cohenSutherlandClip(int x1, int y1, int x2, int y2) {
    int code1 = computeCode(x1, y1);
    int code2 = computeCode(x2, y2);
    int accept = 0;
    while (1) {
        if ((code1 == 0) && (code2 == 0)) { // Both endpoints inside rectangle
            accept = 1;
            break;
        } else if (code1 & code2) {         // Both endpoints share an outside region
            break;
        } else {
            int code_out;
            int x, y;
            if (code1 != 0)
                code_out = code1;
            else
                code_out = code2;
            if (code_out & TOP) {           // Point above the clip rectangle
                x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1);
                y = y_max;
            } else if (code_out & BOTTOM) {  // Point below the clip rectangle
                x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1);
                y = y_min;
            } else if (code_out & RIGHT) {   // Point to the right of clip rectangle
                y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1);
                x = x_max;
            } else if (code_out & LEFT) {    // Point to the left of clip rectangle
                y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1);
                x = x_min;
            }
            if (code_out == code1) {
                x1 = x;
                y1 = y;
                code1 = computeCode(x1, y1);
            } else {
                x2 = x;
                y2 = y;
```

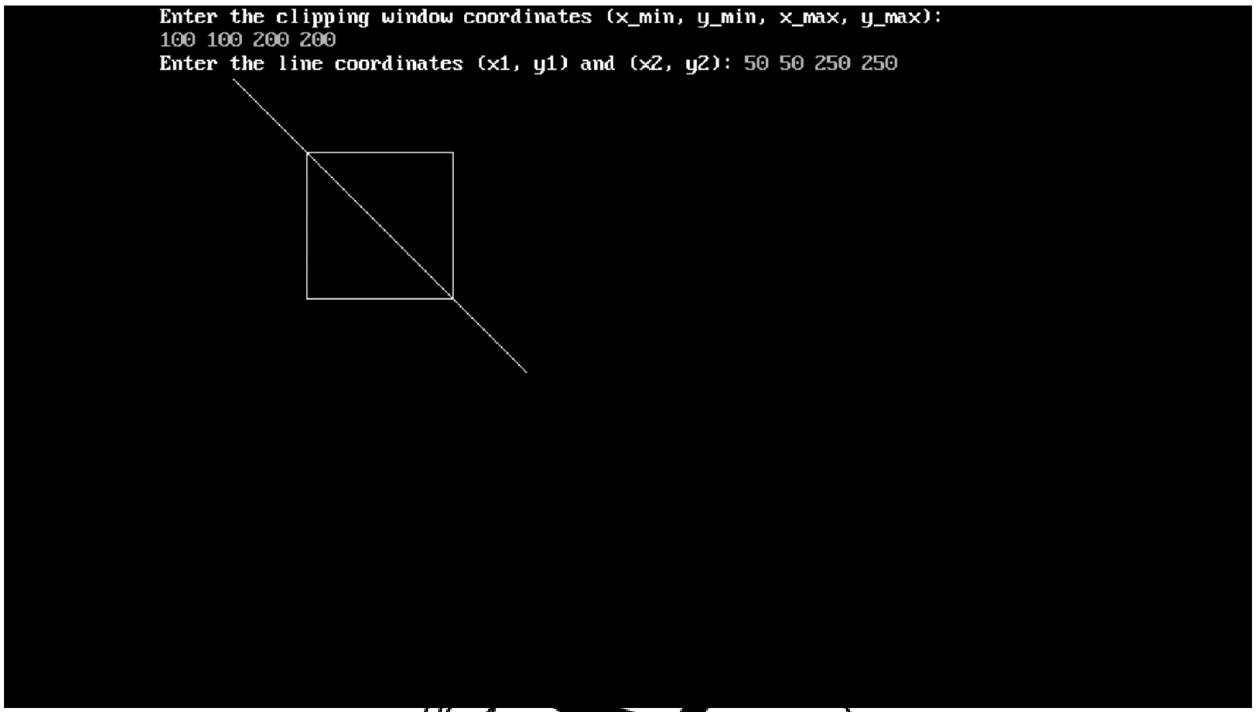


```
                code1 = computeCode(x1, y1);
            } else {
                x2 = x;
                y2 = y;
                code2 = computeCode(x2, y2);
            }
        }
    }
    if (accept) {
        setcolor(RED);
        line(x1, y1, x2, y2);  // Draw clipped line in red
    } else {
        printf("Line is outside the clipping window.\n");
    }
}
int main() {
    int gd = DETECT, gm;
       int x1, y1, x2, y2;
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");
    printf("Enter the clipping window coordinates (x_min, y_min, x_max, y_max): ");
    scanf("%d %d %d %d", &x_min, &y_min, &x_max, &y_max);
    printf("Enter the line coordinates (x1, y1) and (x2, y2): ");
    scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
    setcolor(WHITE);
    rectangle(x_min, y_min, x_max, y_max);
    setcolor(WHITE);
    line(x1, y1, x2, y2);
    cohenSutherlandClip(x1, y1, x2, y2);
    getch();
    closegraph();
    return 0;
}
```



```
Enter the clipping window coordinates (x_min, y_min, x_max, y_max): 100 100 200
200
Enter the line coordinates (x1, y1) and (x2, y2): 50 50 250 250
```

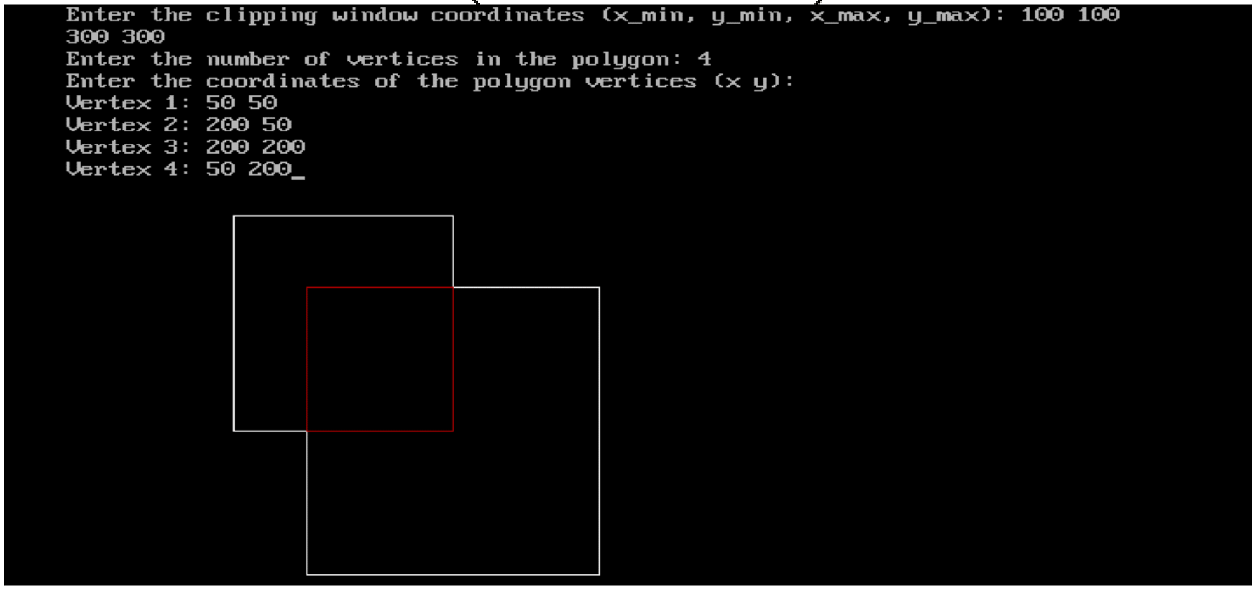
```
122 PRACTICLE 13
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int x_min, y_min, x_max, y_max; // Clipping window coordinates
int isInside(int x, int y) {
    return (x >= x_min && x <= x_max && y >= y_min && y <= y_max);
}
void midpointClip(int x1, int y1, int x2, int y2) {
    if (isInside(x1, y1) && isInside(x2, y2)) {
        setcolor(RED);
        line(x1, y1, x2, y2); // Draw the clipped line in red
    }
    else if ((x1 < x_min && x2 < x_min) || (x1 > x_max && x2 > x_max) ||
             (y1 < y_min && y2 < y_min) || (y1 > y_max && y2 > y_max)) {
        return; // The line is outside the window and completely invisible
    }
    else {
        int mid_x = (x1 + x2) / 2;
        int mid_y = (y1 + y2) / 2;
        midpointClip(x1, y1, mid_x, mid_y);
        midpointClip(mid_x, mid_y, x2, y2);
    }
}
int main() {
    int gd = DETECT, gm;
    int x1, y1, x2, y2;
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");
    // Input for clipping window coordinates
    printf("Enter the clipping window coordinates (x_min, y_min, x_max, y_max): ");
    scanf("%d %d %d %d", &x_min, &y_min, &x_max, &y_max);

    // Input for the line coordinates
    printf("Enter the line coordinates (x1, y1) and (x2, y2): ");
    scanf("%d %d %d %d", &x1, &y1, &x2, &y2);

    setcolor(WHITE);
    rectangle(x_min, y_min, x_max, y_max);

    setcolor(WHITE);
    line(x1, y1, x2, y2);

    midpointClip(x1, y1, x2, y2);

    getch();
    closegraph();
    return 0;
}
```



```
Enter the clipping window coordinates (x_min, y_min, x_max, y_max):
100 100 200 200
Enter the line coordinates (x1, y1) and (x2, y2): 50 50 250 250
```

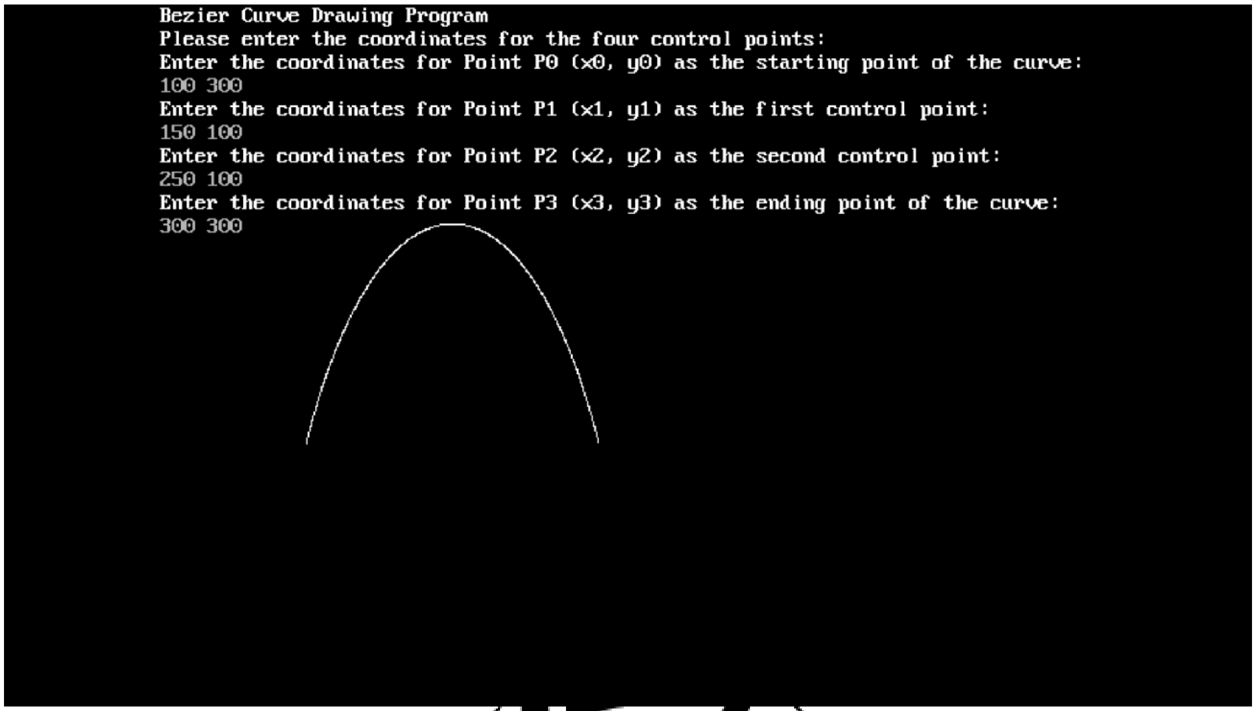
COMPUTER ENG.

```
122 PRACTICLE 14
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#define MAX_POINTS 20
int x_min, y_min, x_max, y_max;
typedef struct {
    int x, y;
} Point;
int isInside(Point p, int edge) {
    switch(edge) {
        case 1: return (p.x >= x_min); // Left
        case 2: return (p.x <= x_max); // Right
        case 3: return (p.y >= y_min); // Bottom
        case 4: return (p.y <= y_max); // Top
    }
    return 0;
}
Point intersect(Point p1, Point p2, int edge) {
    Point temp;
    float m;
    if (p1.x != p2.x) m = (float)(p2.y - p1.y) / (p2.x - p1.x);
    else m = 100000.0; // Large number to avoid division by zero
    switch(edge) {
        case 1: // Left
            temp.x = x_min;
            temp.y = p1.y + (x_min - p1.x) * m;
            break;
        case 2: // Right
            temp.x = x_max;
            temp.y = p1.y + (x_max - p1.x) * m;
            break;
        case 3: // Bottom
            temp.y = y_min;
            if (p1.x != p2.x) temp.x = p1.x + (y_min - p1.y) / m;
            else temp.x = p1.x;
            break;
        case 4: // Top
            temp.y = y_max;
            if (p1.x != p2.x) temp.x = p1.x + (y_max - p1.y) / m;
            else temp.x = p1.x;
            break;
    }
    return temp;
}
void clipEdge(Point *poly_points, int *poly_size, int edge) {
    Point new_points[MAX_POINTS];
    int new_size = 0;
    int i;

    Point v1 = poly_points[*poly_size - 1];
    for (i = 0; i < *poly_size; i++) {
        Point v2 = poly_points[i];
        if (isInside(v2, edge)) {
            if (isInside(v1, edge)) {
                new_points[new_size++] = v2;
            } else {
                new_points[new_size++] = intersect(v1, v2, edge);
                new_points[new_size++] = v2;
            }
        } else if (isInside(v1, edge)) {
            new_points[new_size++] = intersect(v1, v2, edge);
        }
        v1 = v2;
    }
    *poly_size = new_size;
    for (i = 0; i < new_size; i++) {
        poly_points[i] = new_points[i];
    }
}
void sutherlandHodgmanClip(Point *poly_points, int *poly_size) {
    clipEdge(poly_points, poly_size, 1); // Left edge
    clipEdge(poly_points, poly_size, 2); // Right edge
    clipEdge(poly_points, poly_size, 3); // Bottom edge
    clipEdge(poly_points, poly_size, 4); // Top edge
}
int main() {
    int gd = DETECT, gm;
    Point poly_points[MAX_POINTS];
    int poly_size, i;
    clrscr();
    printf("Enter the clipping window coordinates (x_min, y_min, x_max, y_max): ");
    scanf("%d %d %d %d", &x_min, &y_min, &x_max, &y_max);
    printf("Enter the number of vertices in the polygon: ");
    scanf("%d", &poly_size);
    printf("Enter the coordinates of the polygon vertices (x y):\n");
    for (i = 0; i < poly_size; i++) {
        printf("Vertex %d: ", i + 1);
        scanf("%d %d", &poly_points[i].x, &poly_points[i].y);
    }
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");
    setcolor(WHITE);
    rectangle(x_min, y_min, x_max, y_max);
    for (i = 0; i < poly_size - 1; i++) {
        line(poly_points[i].x, poly_points[i].y, poly_points[i + 1].x, poly_points[i + 1].y);
    }
    line(poly_points[poly_size - 1].x, poly_points[poly_size - 1].y, poly_points[0].x, poly_points[0].y);
    sutherlandHodgmanClip(poly_points, &poly_size);
    setcolor(RED);
    for (i = 0; i < poly_size - 1; i++) {
        line(poly_points[i].x, poly_points[i].y, poly_points[i + 1].x, poly_points[i + 1].y);
    }
    line(poly_points[poly_size - 1].x, poly_points[poly_size - 1].y, poly_points[0].x, poly_points[0].y);
    getch();
    closegraph();
    return 0;
}
```



```
Enter the clipping window coordinates (x_min, y_min, x_max, y_max): 100 100
300 300
Enter the number of vertices in the polygon: 4
Enter the coordinates of the polygon vertices (x y):
Vertex 1: 50 50
Vertex 2: 200 50
Vertex 3: 200 200
Vertex 4: 50 200_
```

```c
122 PRACTICLE 15
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>

void bezier(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3) {
    float t;
    int x, y, prevX, prevY;

    // Initialize the previous point with the starting point P0
    prevX = x0;
    prevY = y0;

    // Loop to calculate the Bezier curve and draw lines between the points
    for(t = 0; t <= 1; t += 0.001) {  // Decreased t increment to make the curve smoother
        // Calculate the x and y coordinates using the Bezier formula
        x = pow(1 - t, 3) * x0 + 3 * pow(1 - t, 2) * t * x1 + 3 * (1 - t) * t * t * x2 + pow(t, 3) * x3;
        y = pow(1 - t, 3) * y0 + 3 * pow(1 - t, 2) * t * y1 + 3 * (1 - t) * t * t * y2 + pow(t, 3) * y3;

        // Draw a line from the previous point to the current point
        line(prevX, prevY, x, y);

        // Update previous point to current point
        prevX = x;
        prevY = y;
    }
}

int main() {
    int gd = DETECT, gm;
    int x0, y0, x1, y1, x2, y2, x3, y3;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

    // Display clear prompts for the user to enter control points
    printf("Bezier Curve Drawing Program\n");
    printf("Please enter the coordinates for the four control points:\n");

    // User-friendly prompts for control points
    printf("Enter the coordinates for Point P0 (x0, y0) as the starting point of the curve: ");
    scanf("%d %d", &x0, &y0);
    printf("Enter the coordinates for Point P1 (x1, y1) as the first control point: ");
    scanf("%d %d", &x1, &y1);
    printf("Enter the coordinates for Point P2 (x2, y2) as the second control point: ");
    scanf("%d %d", &x2, &y2);
    printf("Enter the coordinates for Point P3 (x3, y3) as the ending point of the curve: ");
    scanf("%d %d", &x3, &y3);

    // Draw the Bezier curve
    bezier(x0, y0, x1, y1, x2, y2, x3, y3);

    getch();  // Wait for user input
    closegraph();  // Close graphics mode
    return 0;
}
```



```
Bezier Curve Drawing Program
Please enter the coordinates for the four control points:
Enter the coordinates for Point P0 (x0, y0) as the starting point of the curve:
100 300
Enter the coordinates for Point P1 (x1, y1) as the first control point:
150 100
Enter the coordinates for Point P2 (x2, y2) as the second control point:
250 100
Enter the coordinates for Point P3 (x3, y3) as the ending point of the curve:
300 300
```

COMPUTER ENG.