Lab 1: WAP to implement DDA line algorithm.

Source code:

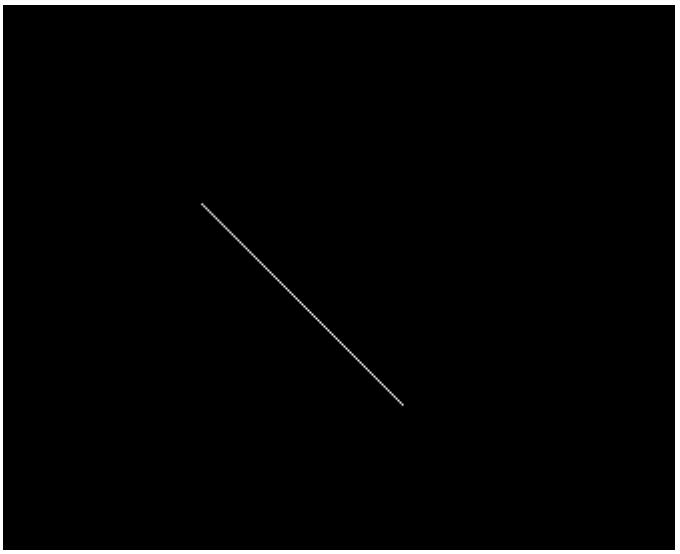
```
#include <graphics.h>
#include <math.h>
#include <stdio.h>

void lineDDA(int x1, int y1, int x2, int y2)
{
    int dx, dy, steps, k;
    float incrx, incry, x, y;

    dx = x2 - x1;
    dy = y2 - y1;

    if (abs(dx) > abs(dy))
    {
        steps = abs(dx);
    }
    else
    {
        steps = abs(dy);
    }

    incrx = (float)dx / steps;
    incry = (float)dy / steps;

    x = x1;
    y = y1;

    putpixel(round(x), round(y), WHITE);

    for (k = 1; k <= steps; k++)
    {
        x = x + incrx;
        y = y + incry;
        putpixel(round(x), round(y), WHITE);
    }
}
int main()
{
    int gd = DETECT, gm;
    int xco1, yco1, xco2, yco2;

    printf("Enter the coordinate x1: ");
    scanf("%d", &xco1);
    printf("Enter the coordinate y1: ");
    scanf("%d", &yco1);

    printf("Enter the coordinate x2: ");
    scanf("%d", &xco2);
    printf("Enter the coordinate y2: ");
    scanf("%d", &yco2);

    initgraph(&gd, &gm, "");

    lineDDA(xco1, yco1, xco2, yco2);
```

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

Input :

```
Enter the coordinate x1: 100
Enter the coordinate y1: 100
Enter the coordinate x2: 200
Enter the coordinate y2: 200
```

Output :



Lab 2: WAP to implement Bresenham's Line Algorithm.

Source code:

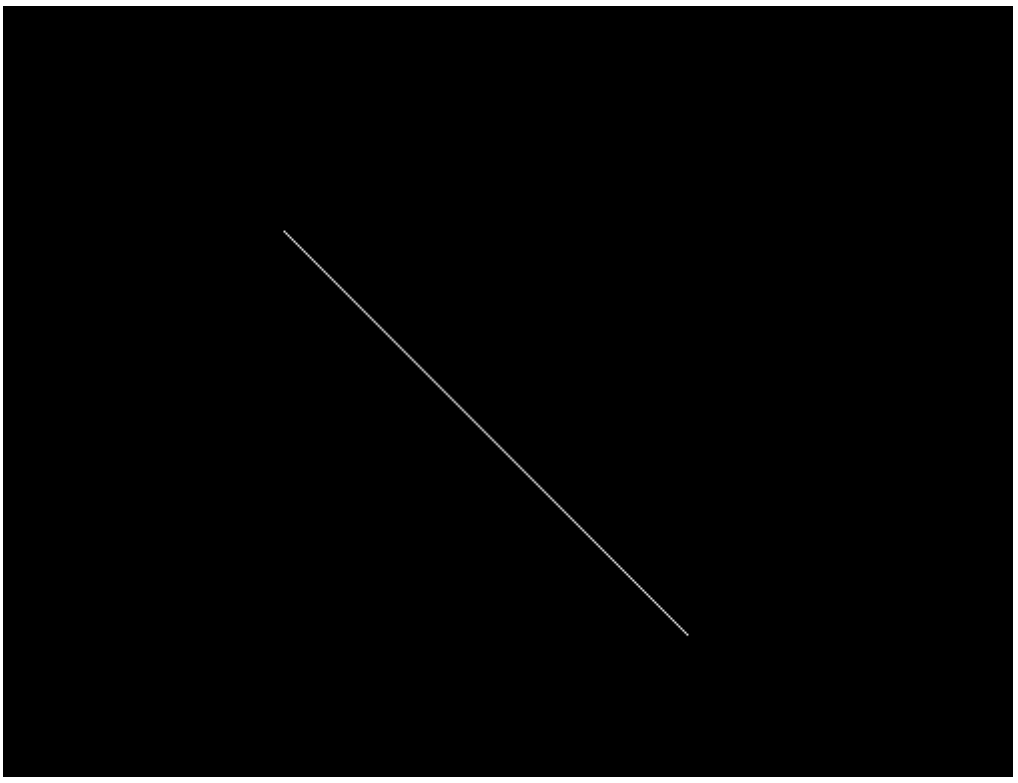
```
#include<stdio.h>
#include<graphics.h>
#include<math.h>

void lineBresenham(int x1, int y1, int x2, int y2){
    int x, y, dx, dy, pk, k, xend;

    dx=abs(x2-x1);
    dy=abs(y2-y1);

    if(x1>x2){
        x=x2;
        y=y2;

        xend=x1;
    }
    else{
        x=x1;
        y=y1;

        xend=x2;
    }
    putpixel(x, y, 15);
    pk=2*dy-dx;

    while (x<=xend)
    {
        if(pk==0){
            x=x+1;
            pk=pk+2*dy;
        }

        else{
            x=x+1;
            y=y+1;
            pk=pk+2*dy-2*dx;
        }

        putpixel(x, y, 15);
    }
}

int main(){
    int gd=DETECT, gm;
    int xco1, yco1, xco2, yco2;

    printf("Enter the coordinate x1: ");
    scanf("%d", &xco1);
    printf("Enter the coordinate y1: ");
    scanf("%d", &yco1);

    printf("Enter the coordinate x2: ");
```

```
        scanf("%d", &xco2);
        printf("Enter the coordinate y2: ");
        scanf("%d", &yco2);

        initgraph(&gd, &gm, "");

        lineBresenham(xco1, yco1, xco2, yco2);

        getch();

        return 0;
    }
```

Input :

```
Enter the coordinate x1: 150
Enter the coordinate y1: 150
Enter the coordinate x2: 350
Enter the coordinate y2: 350
```

Output :



Lab 3: WAP to implement midpoint circle algorithm.

Source code:

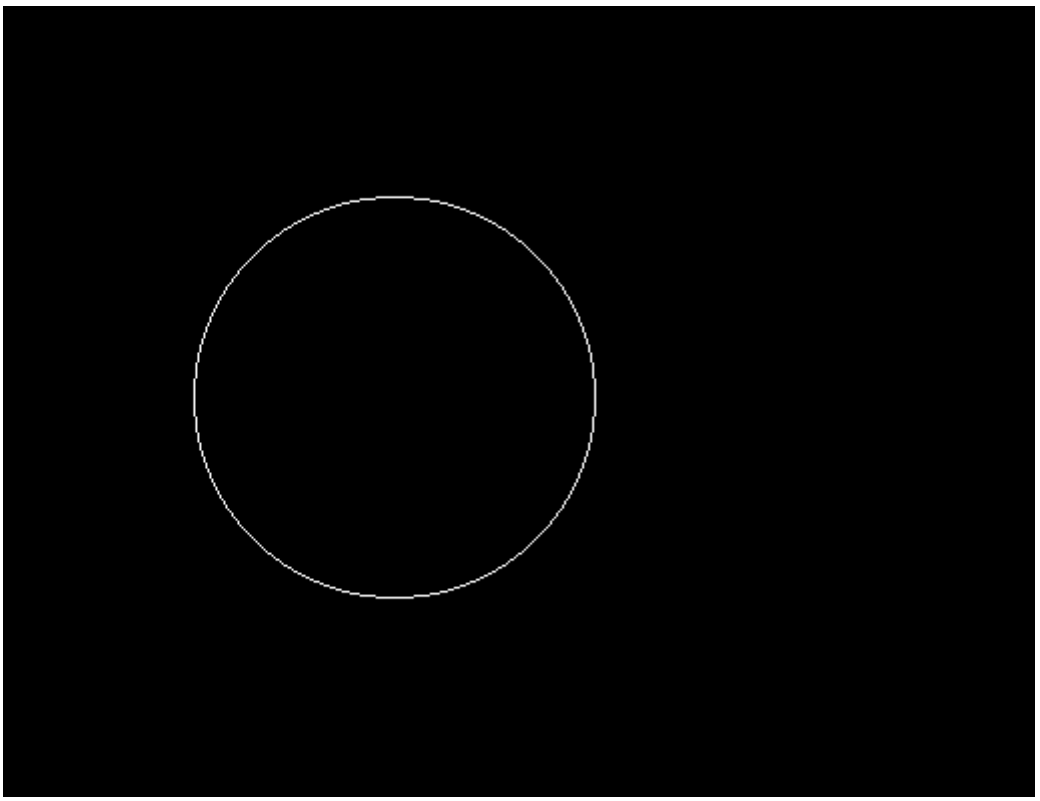
```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

// Function to plot all eight symmetric points of the circle
void drawpoints(int x, int y, int xc, int yc) {
    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);
}

// Function to implement the Midpoint Circle Drawing Algorithm
void drawcircle(int xc, int yc, int r) {
    int p, x, y;
    x = 0;
    y = r;
    drawpoints(x, y, xc, yc); // Draw initial points
    p = 1 - r; // Initial decision parameter

    // Loop until x < y
    while (x < y) {
        x = x + 1;
        if (p < 0) {
            // Midpoint is inside the circle
            p = p + 2 * x + 1;
        } else {
            // Midpoint is outside or on the circle
            y = y - 1;
            p = p + 2 * (x - y) + 1;
        }
        drawpoints(x, y, xc, yc); // Draw symmetric points
    }
}

int main(void) {
    int gdriver = DETECT, gmode, errorcode;
    int xc, yc, r;

    // Initialize graphics mode
    initgraph(&gdriver, &gmode, "");

    // Check for initialization error
    errorcode = graphresult();
    if (errorcode != grOk) {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);
```

```
        }

        // Input center and radius
        printf("Enter the center co-ordinates: ");
        scanf("%d%d", &xc, &yc);
        printf("Enter the radius: ");
        scanf("%d", &r);

        // Draw the circle
        drawcircle(xc, yc, r);

        getch(); // Wait for user input
        return 0;
    }
```

Input :

```
Enter the center co-ordinates: 200 200
Enter the radius: 100
```

Output :



Lab 4: WAP to implement midpoint ellipse algorithm.

Source code:

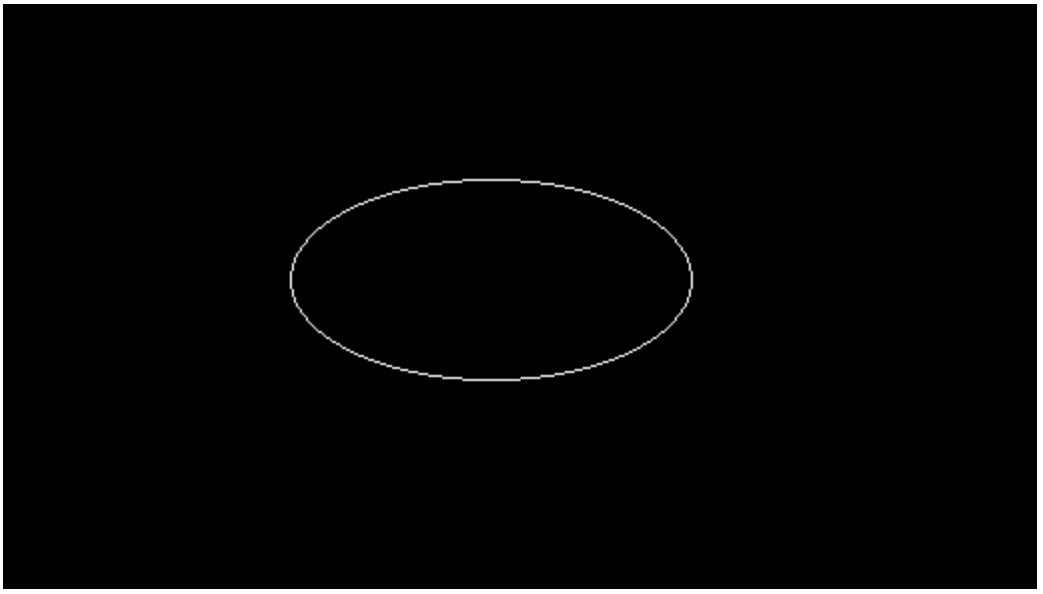
```
#include <stdio.h>
#include <graphics.h>
#include <math.h>

// Function to plot all four symmetric points
void plotEllipsePoints(int xc, int yc, int x, int 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);
}

int main() {
    int xc, yc, rx, ry;
    int gd = DETECT, gm;
    printf("Enter center of ellipse (xc, yc): ");
    scanf("%d %d", &xc, &yc);
    printf("Enter x-radius (rx): ");
    scanf("%d", &rx);
    printf("Enter y-radius (ry): ");
    scanf("%d", &ry);

    initgraph(&gd, &gm, "");

    int x = 0;
    int y = ry;

    // Initial decision parameter for region 1
    float rx2 = rx * rx;
    float ry2 = ry * ry;
    float two_rx2 = 2 * rx2;
    float two_ry2 = 2 * ry2;

    float p1 = ry2 - (rx2 * ry) + (0.25 * rx2);
    int dx = two_ry2 * x;
    int dy = two_rx2 * y;

    // Region 1
    while (dx < dy) {
        plotEllipsePoints(xc, yc, x, y);
        if (p1 < 0) {
            x++;
            dx += two_ry2;
            p1 += dx + ry2;
        } else {
            x++;
            y--;
            dx += two_ry2;
            dy -= two_rx2;
            p1 += dx - dy + ry2;
        }
    }

    // Initial decision parameter for region 2
```

```
    float p2 = (ry2) * (x + 0.5) * (x + 0.5) + (rx2) * (y - 1) * (y - 1)
- (rx2 * ry2);

    // Region 2
    while (y >= 0) {
        plotEllipsePoints(xc, yc, x, y);
        if (p2 > 0) {
            y--;
            dy -= two_rx2;
            p2 += rx2 - dy;
        } else {
            x++;
            y--;
            dx += two_ry2;
            dy -= two_rx2;
            p2 += dx - dy + rx2;
        }
    }

    getch();
    return 0;
}
```

Input :

```
Enter center of ellipse (xc, yc): 300 200
Enter x-radius (rx): 100
Enter y-radius (ry): 50
```

Output :



Lab 5: WAP in C to perform 2D translation.

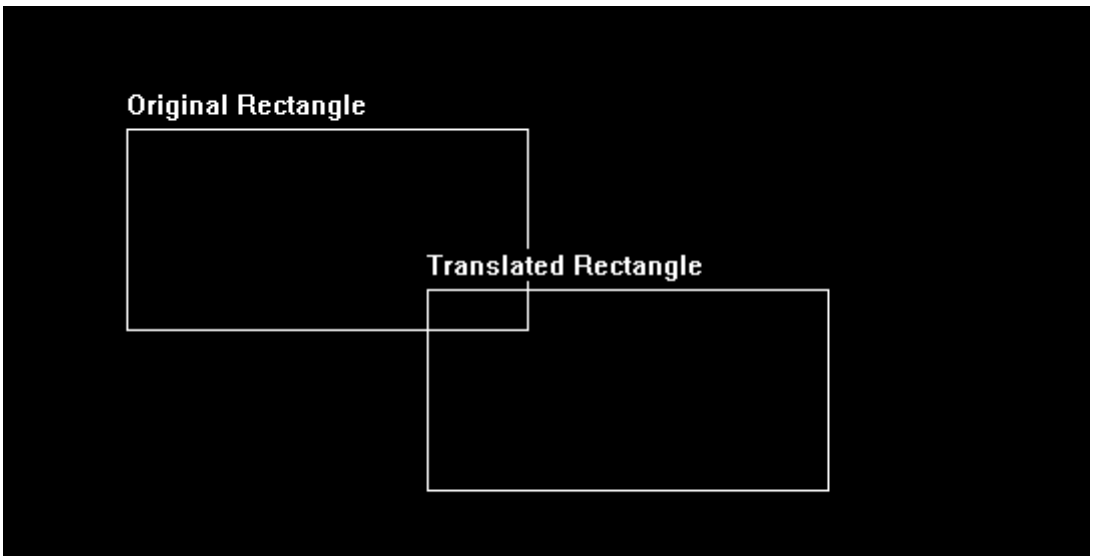
Source code:

```
#include <stdio.h>
#include <graphics.h>
#include <conio.h>

int main()
{
    int rect_x1, rect_y1, rect_x2, rect_y2;// Rectangle coordinates
    int trans_x, trans_y;// Translation values

    printf("Enter the coordinates of the first corner (x1 y1): ");
    scanf("%d %d", &rect_x1, &rect_y1);
    printf("Enter the coordinates of the opposite corner (x2 y2): ");
    scanf("%d %d", &rect_x2, &rect_y2);

    printf("Enter the translation values (tx ty): ");
    scanf("%d %d", &trans_x, &trans_y);

    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    rectangle(rect_x1, rect_y1, rect_x2, rect_y2);
    outtextxy(rect_x1, rect_y1 - 20, "Original Rectangle");

    rectangle(rect_x1 + trans_x, rect_y1 + trans_y, rect_x2 + trans_x,
rect_y2 + trans_y);
    outtextxy(rect_x1 + trans_x, rect_y1 + trans_y - 20, "Translated
Rectangle");

    getch();

    return 0;
}
```

Input :

```
Enter the coordinates of the first corner (x1 y1): 100 100
Enter the coordinates of the opposite corner (x2 y2): 300 200
Enter the translation values (tx ty): 150 80
```

Output :



Lab 6: WAP in C to perform 2D rotation.

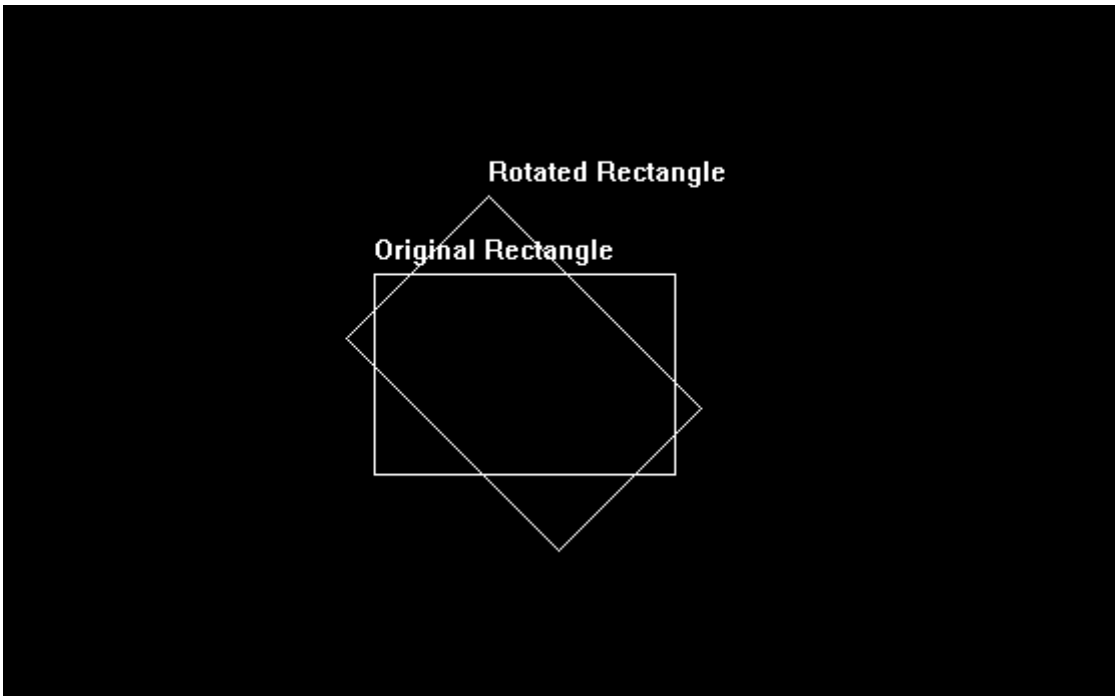
Source code:

```
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define M_PI 3.14159265358979323846

int main()
{
    int rect_x1, rect_y1, rect_x2, rect_y2; // Rectangle coordinates
    float angle_deg, angle_rad;

    printf("Enter the coordinates of the first corner (x1 y1): ");
    scanf("%d %d", &rect_x1, &rect_y1);
    printf("Enter the coordinates of the opposite corner (x2 y2): ");
    scanf("%d %d", &rect_x2, &rect_y2);

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

    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    // Draw original rectangle
    rectangle(rect_x1, rect_y1, rect_x2, rect_y2);
    outtextxy(rect_x1, rect_y1 - 20, "Original Rectangle");

    // Calculate center of rectangle
    float cx = (rect_x1 + rect_x2) / 2.0;
    float cy = (rect_y1 + rect_y2) / 2.0;

    // Convert angle to radians
    angle_rad = angle_deg * M_PI / 180.0;

    // Rectangle corners
    int x[4] = {rect_x1, rect_x2, rect_x2, rect_x1};
    int y[4] = {rect_y1, rect_y1, rect_y2, rect_y2};
    int xr[4], yr[4];

    // Rotate each corner around the center
    for (int i = 0; i < 4; i++) {
        float dx = x[i] - cx;
        float dy = y[i] - cy;
        xr[i] = (int)(cx + dx * cos(angle_rad) - dy * sin(angle_rad));
        yr[i] = (int)(cy + dx * sin(angle_rad) + dy * cos(angle_rad));
    }

    // Draw rotated rectangle
    for (int i = 0; i < 4; i++) {
        line(xr[i], yr[i], xr[(i+1)%4], yr[(i+1)%4]);
    }
    outtextxy(xr[0], yr[0] - 20, "Rotated Rectangle");

    getch();

    return 0;
}
```

Input :

```
Enter the coordinates of the first corner (x1 y1): 200 150
Enter the coordinates of the opposite corner (x2 y2): 350 250
Enter the rotation angle (degrees): 45
```

Output:



Lab 7: WAP in C to perform 2D Scaling.

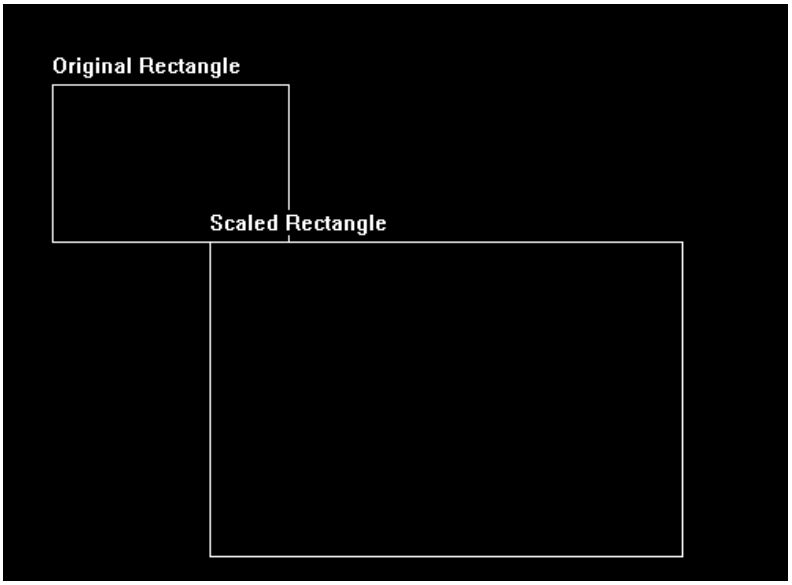
Source code:

```
#include <stdio.h>
#include <graphics.h>
#include <conio.h>

int main()
{
    int rect_x1, rect_y1, rect_x2, rect_y2; // Rectangle coordinates
    int scale_x, scale_y; // Scaling factors

    printf("Enter the coordinates of the first corner (x1 y1): ");
    scanf("%d %d", &rect_x1, &rect_y1);
    printf("Enter the coordinates of the opposite corner (x2 y2): ");
    scanf("%d %d", &rect_x2, &rect_y2);

    printf("Enter the scaling factors (sx sy): ");
    scanf("%d %d", &scale_x, &scale_y);

    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    rectangle(rect_x1, rect_y1, rect_x2, rect_y2);
    outtextxy(rect_x1, rect_y1 - 20, "Original Rectangle");

    rectangle(rect_x1 * scale_x, rect_y1 * scale_y, rect_x2 * scale_x,
rect_y2 * scale_y);
    outtextxy(rect_x1 * scale_x, rect_y1 * scale_y - 20, "Scaled
Rectangle");

    getch();

    return 0;
}
```

Input :

```
Enter the coordinates of the first corner (x1 y1): 100 100
Enter the coordinates of the opposite corner (x2 y2): 250 200
Enter the scaling factors (sx sy): 2 2
```

Output:



Lab 8: WAP in C to perform 2D reflection.

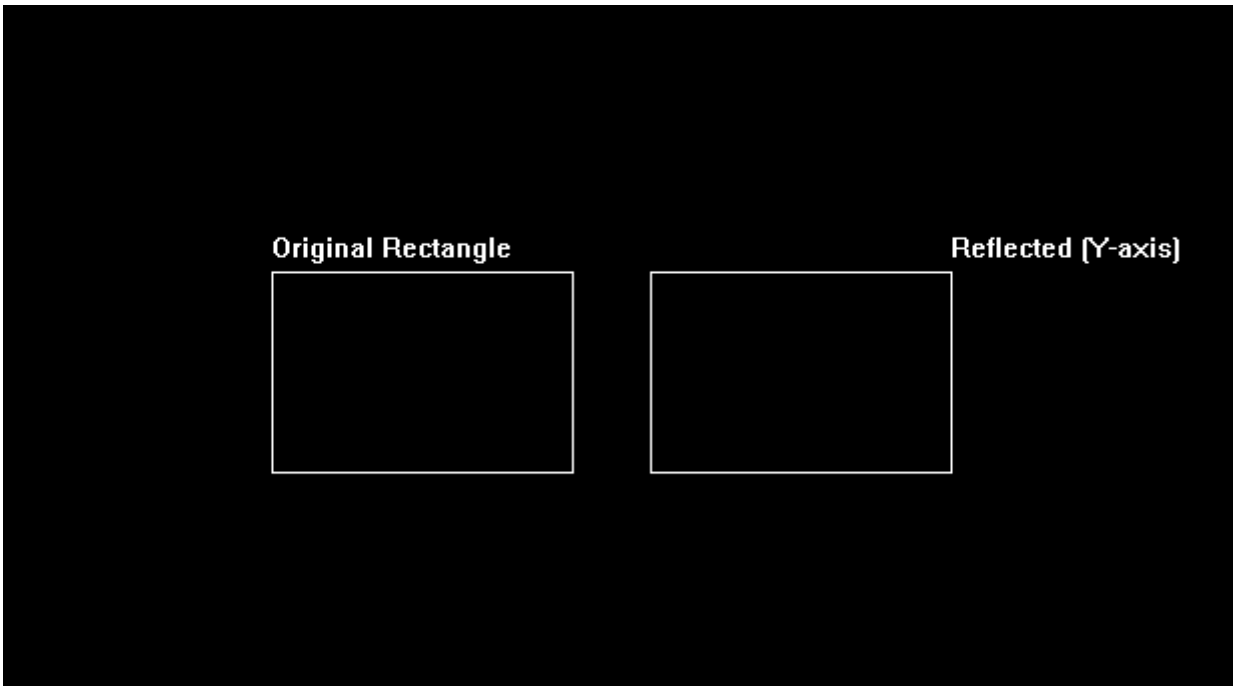
Source code:

```
#include <stdio.h>
#include <graphics.h>
#include <conio.h>

int main()
{
    int rect_x1, rect_y1, rect_x2, rect_y2; // Rectangle coordinates
    int choice;
    int gd = DETECT, gm;

    printf("Enter the coordinates of the first corner (x1 y1): ");
    scanf("%d %d", &rect_x1, &rect_y1);
    printf("Enter the coordinates of the opposite corner (x2 y2): ");
    scanf("%d %d", &rect_x2, &rect_y2);

    printf("Reflect about:\n1. X-axis\n2. Y-axis\nEnter your choice: ");
    scanf("%d", &choice);

    initgraph(&gd, &gm, "");

    rectangle(rect_x1, rect_y1, rect_x2, rect_y2);
    outtextxy(rect_x1, rect_y1 - 20, "Original Rectangle");

    int maxx = getmaxx();
    int maxy = getmaxy();

    if (choice == 1) {
        // Reflection about X-axis (invert y)
        rectangle(rect_x1, maxy - rect_y1, rect_x2, maxy - rect_y2);
        outtextxy(rect_x1, maxy - rect_y1 - 20, "Reflected (X-axis)");
    } else if (choice == 2) {
        // Reflection about Y-axis (invert x)
        rectangle(maxx - rect_x1, rect_y1, maxx - rect_x2, rect_y2);
        outtextxy(maxx - rect_x1, rect_y1 - 20, "Reflected (Y-axis)");
    } else {
        outtextxy(10, maxy - 30, "Invalid choice!");
    }

    getch();
    return 0;
}
```

Input :

```
Enter the coordinates of the first corner (x1 y1): 150 150
Enter the coordinates of the opposite corner (x2 y2): 300 250
Reflect about:
1. X-axis
2. Y-axis
Enter your choice: 2
```

Output:



Lab 9: WAP in C to perform 2D shearing.

Source code:

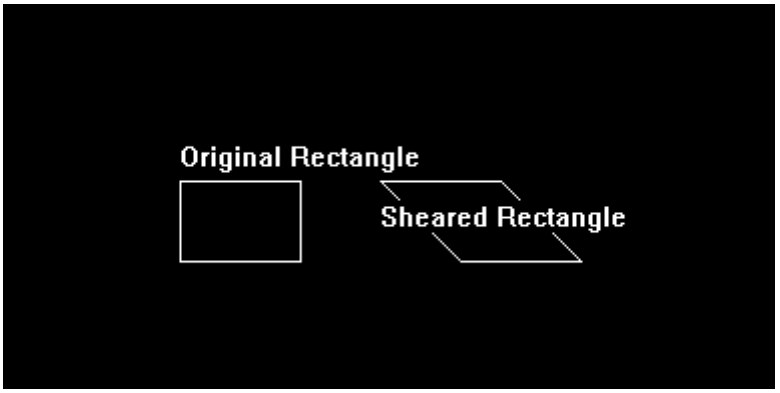
```
#include <stdio.h>
#include <graphics.h>
#include <dos.h> // For delay()

void draw_rect_with_label(int x[4], int y[4], const char *label, int
label_y_offset);

int main()
{
    int rect_x[4], rect_y[4]; // Rectangle corner coordinates
    int shear_factor;
    int choice;
    int sheared_x[4], sheared_y[4];
    int i;

    printf("Enter the coordinates of the rectangle corners (x y):\n");
    for (i = 0; i < 4; i++)
    {
        printf("Corner %d: ", i + 1);
        scanf("%d %d", &rect_x[i], &rect_y[i]);
    }

    printf("Shear along which axis?\n");
    printf("1 for X-axis\n");
    printf("2 for Y-axis\n");
    printf("3 for both X and Y axes\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    printf("Enter the shearing factor: ");
    scanf("%d", &shear_factor);

    // Copy original rectangle to sheared arrays
    for (i = 0; i < 4; i++) {
        sheared_x[i] = rect_x[i];
        sheared_y[i] = rect_y[i];
    }

    // Apply shearing
    if (choice == 1) {
        for (i = 0; i < 4; i++)
            sheared_x[i] = rect_x[i] + shear_factor * rect_y[i];
    } else if (choice == 2) {
        for (i = 0; i < 4; i++)
            sheared_y[i] = rect_y[i] + shear_factor * rect_x[i];
    } else if (choice == 3) {
        for (i = 0; i < 4; i++) {
            sheared_x[i] = rect_x[i] + shear_factor * rect_y[i];
            sheared_y[i] = rect_y[i] + shear_factor * rect_x[i];
        }
    }

    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    setbkcolor(BLACK);
```

```
    // Draw original rectangle
    draw_rect_with_label(rect_x, rect_y, "Original Rectangle", -20);

    // Draw sheared rectangle
    draw_rect_with_label(sheared_x, sheared_y, "Sheared Rectangle", 10);

    getch();

    return 0;
}

void draw_rect_with_label(int x[4], int y[4], const char *label, int
label_y_offset)
{
    line(x[0], y[0], x[1], y[1]);
    line(x[1], y[1], x[2], y[2]);
    line(x[2], y[2], x[3], y[3]);
    line(x[3], y[3], x[0], y[0]);
    // Place label near the first corner
    outtextxy(x[0], y[0] + label_y_offset, (char*)label);
}
```

Input :

```
Enter the coordinates of the rectangle corners (x y):
Corner 1: 100 100
Corner 2: 160 100
Corner 3: 160 140
Corner 4: 100 140
Shear along which axis?
1 for X-axis
2 for Y-axis
3 for both X and Y axes
Enter your choice: 1
Enter the shearing factor: 1
```

Output:



Lab 10: WAP in C to perform cohen sutherland line clipping algorithm.

Source code:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>

int main()
{
    int rcode_begin[4] = {0, 0, 0, 0}, rcode_end[4] = {0, 0, 0, 0},
region_code[4];
    int W_xmax, W_ymax, W_xmin, W_ymin, flag = 0;
    float slope;
    int x, y, x1, y1, i, xc, yc;
    int gr = DETECT, gm;
    initgraph(&gr, &gm, "");
    printf("\n****** Cohen Sutherland Line Clipping Algorithm
***********\n");
    printf("Enter XMin and YMin (space separated): ");
    scanf("%d %d", &W_xmin, &W_ymin);
    printf("Enter XMax and YMax (space separated): ");
    scanf("%d %d", &W_xmax, &W_ymax);
    printf("Enter initial point (x y): ");
    scanf("%d %d", &x, &y);
    printf("Enter final point (x1 y1): ");
    scanf("%d %d", &x1, &y1);
    cleardevice();
    rectangle(W_xmin, W_ymin, W_xmax, W_ymax);
    line(x, y, x1, y1);
    line(0, 0, 600, 0);
    line(0, 0, 0, 600);
    if (y > W_ymax)
    {
        rcode_begin[0] = 1; // Top
        flag = 1;
    }
    if (y < W_ymin)
    {
        rcode_begin[1] = 1; // Bottom
        flag = 1;
    }
    if (x > W_xmax)
    {
        rcode_begin[2] = 1; // Right
        flag = 1;
    }
    if (x < W_xmin)
    {
        rcode_begin[3] = 1; // Left
        flag = 1;
    }

    // end point of Line
    if (y1 > W_ymax)
    {
        rcode_end[0] = 1; // Top
```

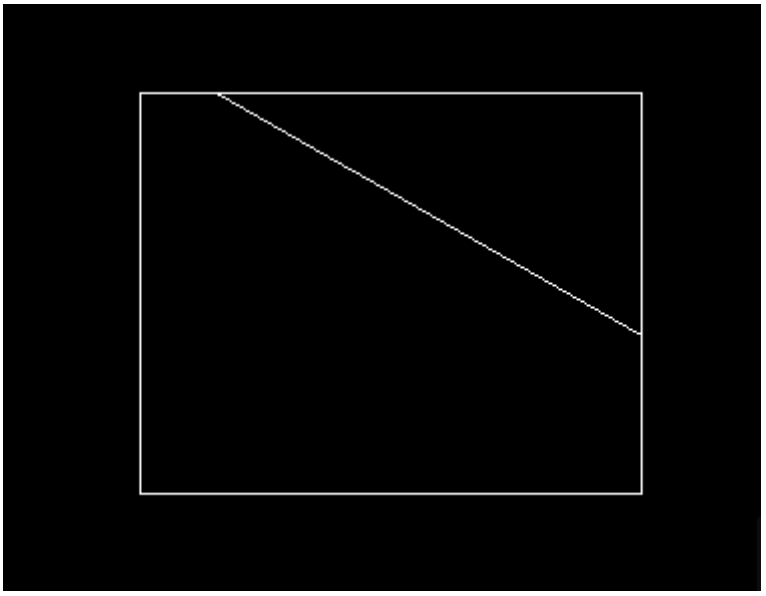
```
    flag = 1;
}
if (y1 < W_ymin)
{
    rcode_end[1] = 1; // Bottom
    flag = 1;
}
if (x1 > W_xmax)
{
    rcode_end[2] = 1; // Right
    flag = 1;
}
if (x1 < W_xmin)
{
    rcode_end[3] = 1; // Left
    flag = 1;
}
if (flag == 0)
{
    printf("No need of clipping as it is already in window");
}
flag = 1;
for (i = 0; i < 4; i++)
{
    region_code[i] = rcode_begin[i] && rcode_end[i];
    if (region_code[i] == 1)
        flag = 0;
}
if (flag == 0)
{
    printf("\n Line is completely outside the window");
}
else
{
    slope = (float)(y1 - y) / (x1 - x);
    if (rcode_begin[2] == 0 && rcode_begin[3] == 1) // left
    {
        y = y + (float)(W_xmin - x) * slope;
        x = W_xmin;
    }
    if (rcode_begin[2] == 1 && rcode_begin[3] == 0) // right
    {
        y = y + (float)(W_xmax - x) * slope;
        x = W_xmax;
    }
    if (rcode_begin[0] == 1 && rcode_begin[1] == 0) // top
    {
        x = x + (float)(W_ymax - y) / slope;
        y = W_ymax;
    }
    if (rcode_begin[0] == 0 && rcode_begin[1] == 1) // bottom
    {
        x = x + (float)(W_ymin - y) / slope;
        y = W_ymin;
    }
    // end points
    if (rcode_end[2] == 0 && rcode_end[3] == 1) // left
    {
        y1 = y1 + (float)(W_xmin - x1) * slope;
        x1 = W_xmin;
    }
```

```
            if (rcode_end[2] == 1 && rcode_end[3] == 0) // right
            {
                y1 = y1 + (float)(W_xmax - x1) * slope;
                x1 = W_xmax;
            }
            if (rcode_end[0] == 1 && rcode_end[1] == 0) // top
            {
                x1 = x1 + (float)(W_ymax - y1) / slope;
                y1 = W_ymax;
            }
            if (rcode_end[0] == 0 && rcode_end[1] == 1) // bottom
            {
                x1 = x1 + (float)(W_ymin - y1) / slope;
                y1 = W_ymin;
            }
        }
        clearviewport();
        rectangle(W_xmin, W_ymin, W_xmax, W_ymax);
        line(0, 0, 600, 0);
        line(0, 0, 0, 600);
        line(x, y, x1, y1);
        getch();
        return 0;
    }
```

Input :

```
****** Cohen Sutherland Line Clipping Algorithm ***********
Enter XMin and YMin (space separated): 150 150
Enter XMax and YMax (space separated): 400 350
Enter initial point (x y): 100 100
Enter final point (x1 y1): 450 300
```

Output:



Lab 11: WAP in C to perform bezier curve

Source code:

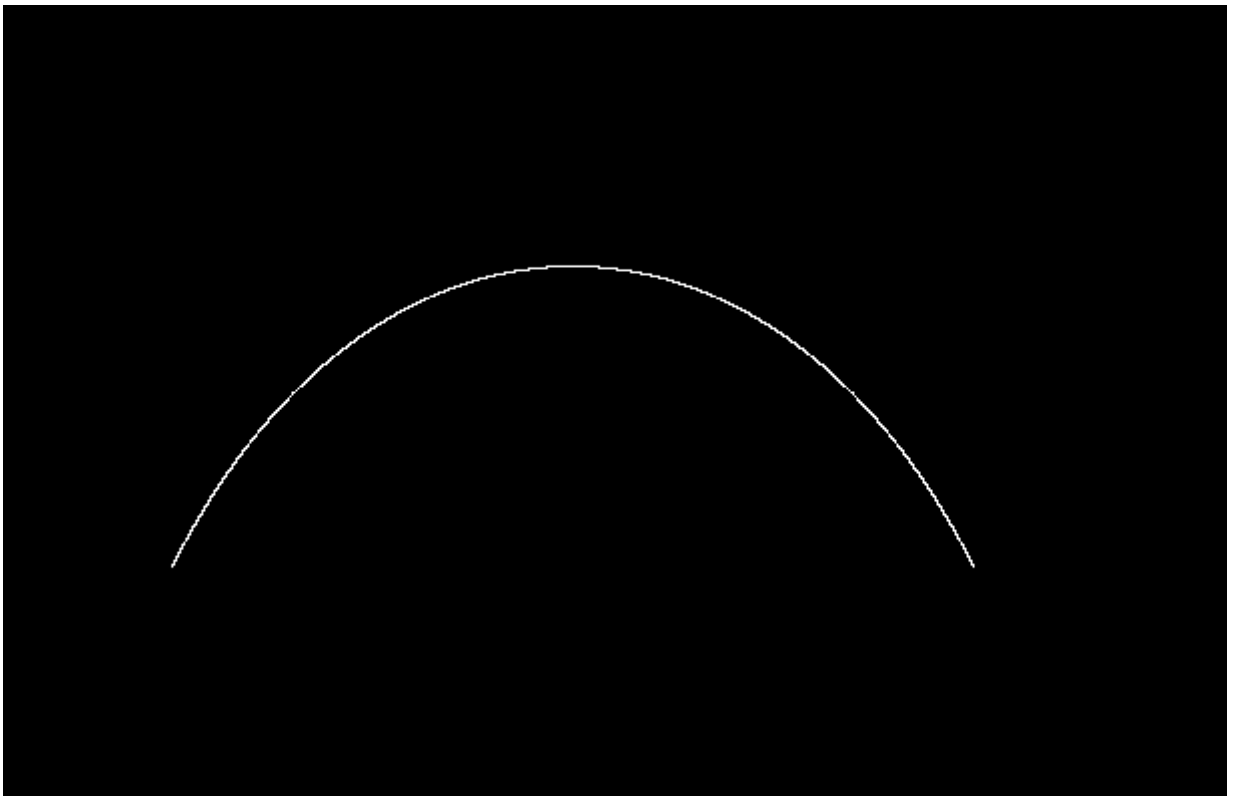
```
#include <stdio.h>
#include <math.h>
#include <graphics.h>

#define MAX_POINTS 20
#define MAX_ORDER  20

typedef struct Point
{
    int x;
    int y;
} Point;

// calculating bezier coefficients
void coefficients_calculate(int n, int *coefficients)
{
    int k, j;
    for (k = 0; k <= n; k++)
    {
        coefficients[k] = 1;
        for (j = n; j >= k + 1; j--)
        {
            coefficients[k] *= j;
        }
        for (j = n - k; j >= 2; j--)
        {
            coefficients[k] /= j;
        }
    }
}

// calculating bezier points
void bezier_points(float parameter, Point *curve, int
control_points_num, Point *control_points, int *coefficients)
{
    int k, n = control_points_num - 1;
    float blending;
    float x = 0.0, y = 0.0;

    for (k = 0; k < control_points_num; k++)
    {
        blending = coefficients[k] * pow(parameter, k) * pow(1 -
parameter, n - k);
        x += control_points[k].x * blending;
        y += control_points[k].y * blending;
    }
    putpixel((int)(x + 0.5), (int)(y + 0.5), WHITE);
}

// drawing bezier curve
void bezier_curve(int divisions, int control_points_num, Point
*control_points, int *coefficients)
{
    int i;
    coefficients_calculate(control_points_num - 1, coefficients);
    for (i = 0; i <= divisions; i++)
    {
```

```
            float param = i / (float)divisions;
            Point curve;
            bezier_points(param, &curve, control_points_num, control_points,
    coefficients);
        }
    }

    int main()
    {
        int gd = DETECT, gm;
        int control_points_num, order;
        int coefficients[MAX_ORDER];
        Point control_points[MAX_POINTS];

        printf("Enter number of control points (max %d): ", MAX_POINTS);
        scanf("%d", &control_points_num);

        if (control_points_num > MAX_POINTS || control_points_num < 2) {
            printf("Invalid number of control points.\n");
            return 1;
        }

        printf("Enter control points (x y):\n");
        for (int i = 0; i < control_points_num; i++)
        {
            scanf("%d %d", &control_points[i].x, &control_points[i].y);
        }

        printf("Enter the order of bezier curve (max %d): ", MAX_ORDER);
        scanf("%d", &order);

        if (order > MAX_ORDER || order < 1) {
            printf("Invalid order.\n");
            return 1;
        }

        initgraph(&gd, &gm, NULL);
        bezier_curve(10000, control_points_num, control_points,
    coefficients);
        getch();

        return 0;
    }
```

Input :

```
Enter number of control points (max 20): 4
Enter control points (x y):
100 300
200 100
400 100
500 300
Enter the order of bezier curve (max 20): 3
```

Output :



Lab 12: WAP in C to perform 3D transformations (translation, scaling, and rotation) on a cube using graphics.

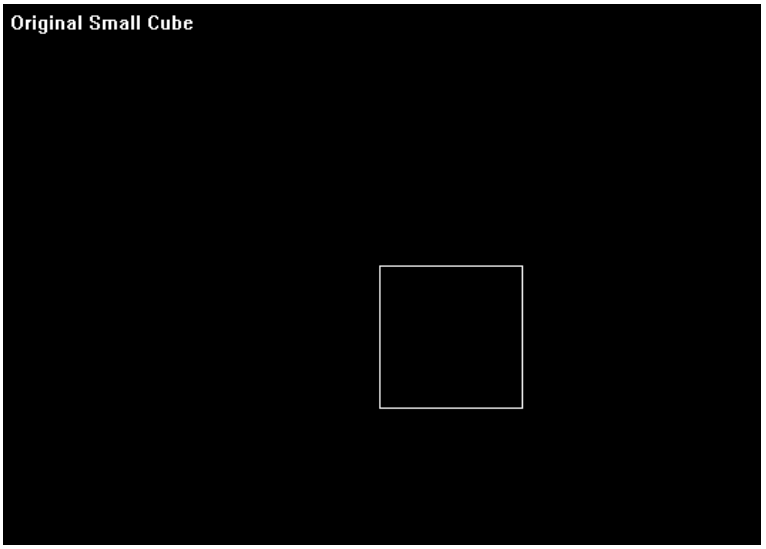
Source code:

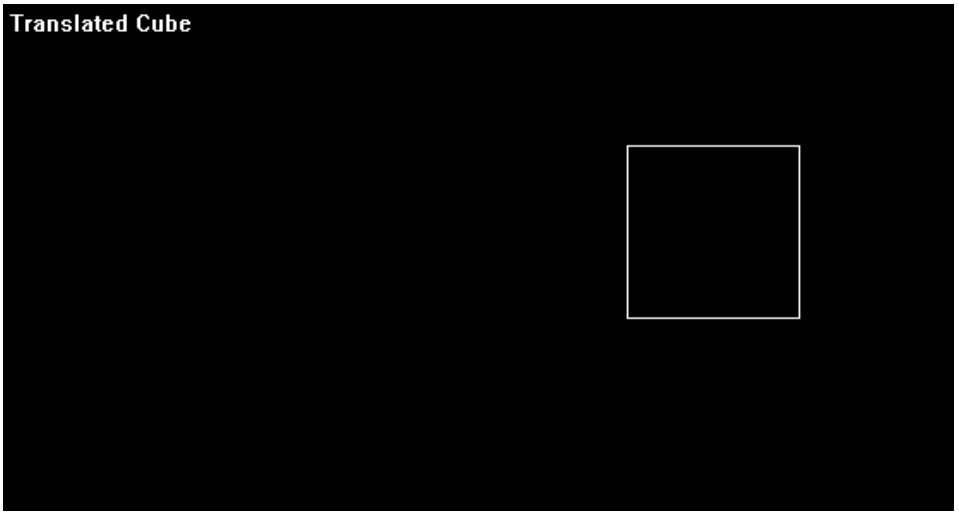
```
#include <graphics.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>

#define PI 3.14159

typedef struct {
    float x, y, z;
} Point3D;

// Original cube coordinates (from -1 to 1)
Point3D cube[8] = {
    {-1, -1, -1},
    { 1, -1, -1},
    { 1,  1, -1},
    {-1,  1, -1},
    {-1, -1,  1},
    { 1, -1,  1},
    { 1,  1,  1},
    {-1,  1,  1}
};

int edges[12][2] = {
    {0, 1}, {1, 2}, {2, 3}, {3, 0},
    {4, 5}, {5, 6}, {6, 7}, {7, 4},
    {0, 4}, {1, 5}, {2, 6}, {3, 7}
};

// Project 3D point to 2D (orthographic)
void project(Point3D p, int *x, int *y) {
    int scale = 100;
    *x = getmaxx()/2 + (int)(p.x * scale);
    *y = getmaxy()/2 - (int)(p.y * scale);
}

// Draw cube edges
void drawCube(Point3D cube[]) {
    cleardevice();
    int x1, y1, x2, y2;

    for (int i = 0; i < 12; i++) {
        project(cube[edges[i][0]], &x1, &y1);
        project(cube[edges[i][1]], &x2, &y2);
        line(x1, y1, x2, y2);
    }
}

// Apply translation
void translate(Point3D cube[], float tx, float ty, float tz) {
    for (int i = 0; i < 8; i++) {
        cube[i].x += tx;
```

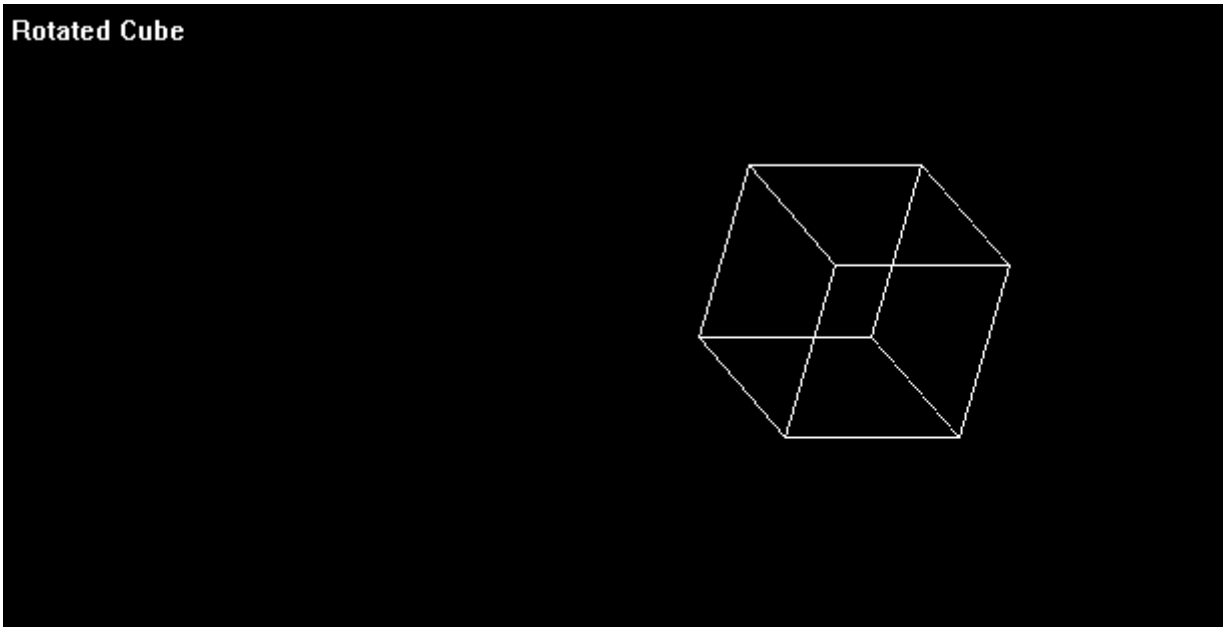
```
        cube[i].y += ty;
        cube[i].z += tz;
    }
}

// Apply scaling
void scale(Point3D cube[], float sx, float sy, float sz) {
    for (int i = 0; i < 8; i++) {
        cube[i].x *= sx;
        cube[i].y *= sy;
        cube[i].z *= sz;
    }
}

// Rotate around X-axis
void rotateX(Point3D cube[], float angle) {
    float rad = angle * PI / 180;
    for (int i = 0; i < 8; i++) {
        float y = cube[i].y;
        float z = cube[i].z;
        cube[i].y = y * cos(rad) - z * sin(rad);
        cube[i].z = y * sin(rad) + z * cos(rad);
    }
}

// Rotate around Y-axis
void rotateY(Point3D cube[], float angle) {
    float rad = angle * PI / 180;
    for (int i = 0; i < 8; i++) {
        float x = cube[i].x;
        float z = cube[i].z;
        cube[i].x = x * cos(rad) + z * sin(rad);
        cube[i].z = -x * sin(rad) + z * cos(rad);
    }
}

// Rotate around Z-axis
void rotateZ(Point3D cube[], float angle) {
    float rad = angle * PI / 180;
    for (int i = 0; i < 8; i++) {
        float x = cube[i].x;
        float y = cube[i].y;
        cube[i].x = x * cos(rad) - y * sin(rad);
        cube[i].y = x * sin(rad) + y * cos(rad);
    }
}

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    // Copy of original cube to transform
    Point3D transformedCube[8];
    for (int i = 0; i < 8; i++)
        transformedCube[i] = cube[i];

    // Step 1: Scale cube smaller (initially)
    scale(transformedCube, 0.5, 0.5, 0.5);
    drawCube(transformedCube);
    outtextxy(10, 10, "Original Small Cube");
    getch();
```

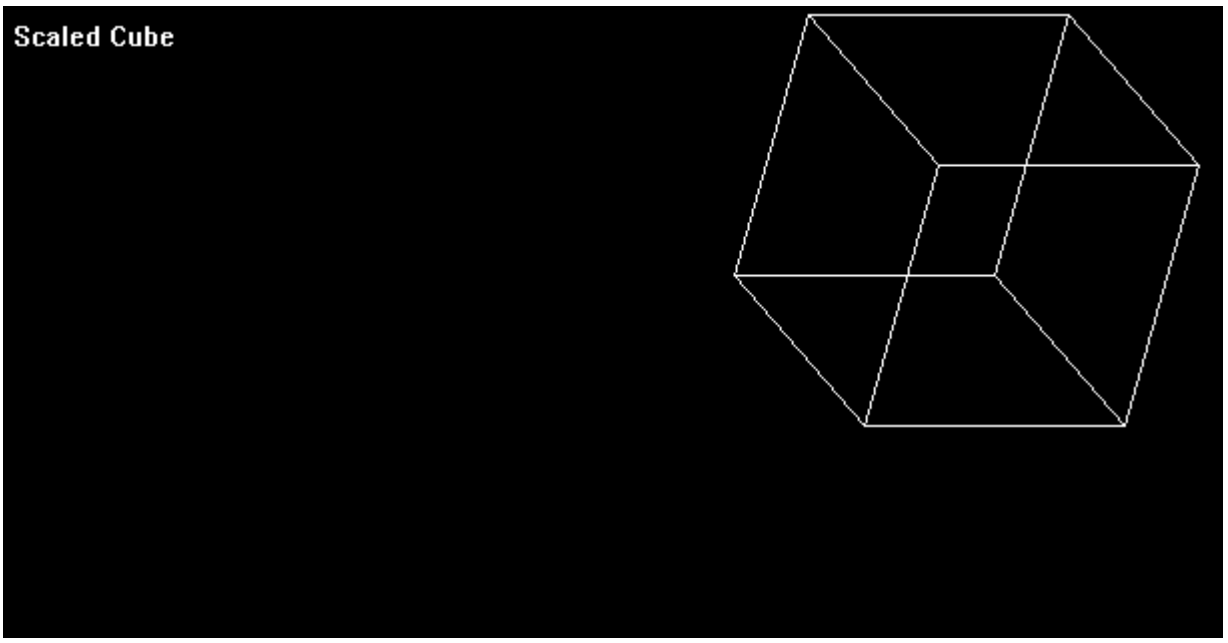
```
    // Step 2: Translate
    translate(transformedCube, 1.0, 1.0, 0.0);
    drawCube(transformedCube);
    outtextxy(10, 10, "Translated Cube");
    getch();

    // Step 3: Rotate around X and Y
    rotateX(transformedCube, 30);
    rotateY(transformedCube, 30);
    drawCube(transformedCube);
    outtextxy(10, 10, "Rotated Cube");
    getch();

    // Step 4: Scale again
    scale(transformedCube, 1.5, 1.5, 1.5);
    drawCube(transformedCube);
    outtextxy(10, 10, "Scaled Cube");
    getch();

    return 0;
}
```

Output :






Rotated Cube


Scaled Cube

Lab 13: WAP in C to implement polygon filling using the polygon table method.

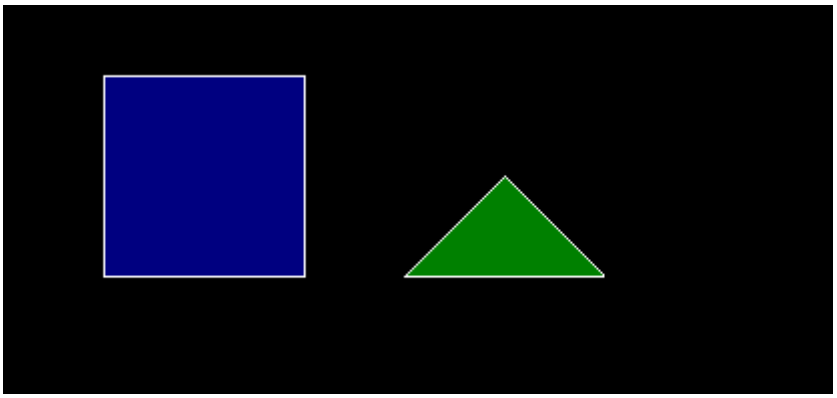
Source code:

```
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int main()
{
    int numVertices, numFaces, totalFaces;

    int gdriver = DETECT, gmode;
    initgraph(&gdriver, &gmode, "");

    printf("Enter the number of faces: ");
    scanf("%d", &numFaces);
    totalFaces = numFaces;

    for (int faceIndex = 0; numFaces != faceIndex; numFaces--)
    {
        printf("Enter the total number of vertices in face %d: ",
totalFaces - numFaces + 1);
        scanf("%d", &numVertices);
        int vertices[numVertices * 2 + 2], vertexIndex;

        printf("Enter vertices for face %d (x,y): \n", totalFaces -
numFaces + 1);
        for (vertexIndex = 0; vertexIndex < (numVertices * 2);
vertexIndex++)
        {
            printf("\tx%d , y%d : ", vertexIndex / 2, vertexIndex / 2);
            scanf("%d %d", &vertices[vertexIndex], &vertices[vertexIndex
+ 1]);
            vertexIndex++;
        }
        vertices[numVertices * 2] = vertices[0];
        vertices[(numVertices * 2) + 1] = vertices[1];

        setcolor(getmaxcolor());
        setfillstyle(SOLID_FILL, totalFaces - numFaces + 1);
        fillpoly(numVertices + 1, vertices);
    }

    getch();

    return 0;
}
```

Input :

```
Enter the number of faces: 2
Enter the total number of vertices in face 1: 4
Enter vertices for face 1 (x,y):
        x0 , y0 : 100 100
        x1 , y1 : 200 100
        x2 , y2 : 200 200
        x3 , y3 : 100 200
Enter the total number of vertices in face 2: 3
Enter vertices for face 2 (x,y):
        x0 , y0 : 300 150
        x1 , y1 : 350 200
        x2 , y2 : 250 200
```

Output:

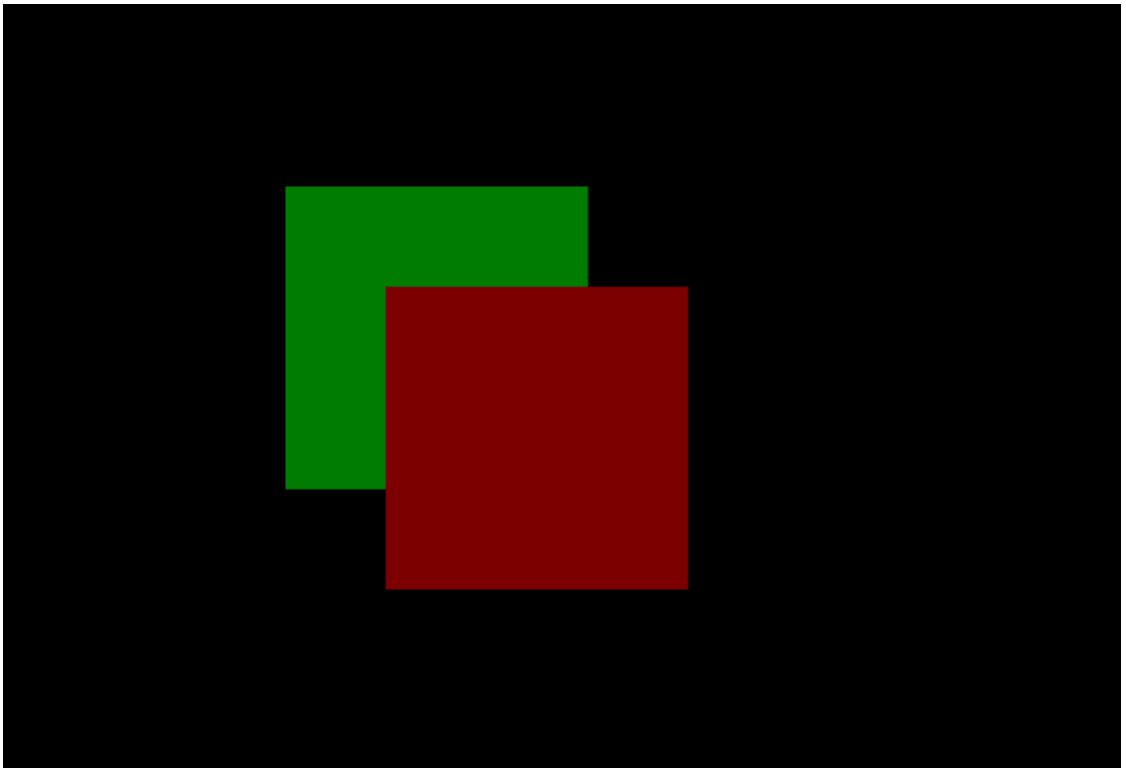


Lab 14: WAP in C to implement the Z-buffer algorithm for hidden surface removal.

Source code:

```
#include <graphics.h>
#include <stdio.h>
#include <conio.h>

#define WIDTH 640
#define HEIGHT 480

float zBuffer[WIDTH][HEIGHT];

// Function to initialize Z-buffer
void clearZBuffer() {
    for (int x = 0; x < WIDTH; x++) {
        for (int y = 0; y < HEIGHT; y++) {
            zBuffer[x][y] = 1e9; // Initialize with farthest depth
        }
    }
}

// Function to plot pixel with Z-buffer test
void putPixelZ(int x, int y, float z, int color) {
    if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT) {
        if (z < zBuffer[x][y]) {
            putpixel(x, y, color);
            zBuffer[x][y] = z;
        }
    }
}

// Draw a filled rectangle with depth (Z)
void drawRectangle3D(int x1, int y1, int x2, int y2, float z, int color)
{
    for (int x = x1; x <= x2; x++) {
        for (int y = y1; y <= y2; y++) {
            putPixelZ(x, y, z, color);
        }
    }
}

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    clearZBuffer();

    // Draw far rectangle first (green, z = 50)
    drawRectangle3D(150, 100, 300, 250, 50.0f, GREEN);

    // Draw near rectangle overlapping (red, z = 30)
    drawRectangle3D(200, 150, 350, 300, 30.0f, RED);

    getch();

    return 0;
}
```

Output:



Lab 15: Write an OpenGL program in C to demonstrate basic 2D and 3D drawing techniques.

Source code:

```
#include <GL/glut.h>

void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Draw a point
    glPointSize(5.0f);
    glBegin(GL_POINTS);
        glColor3f(1, 0, 0);
        glVertex3f(-0.8f, 0.8f, 0.0f);
    glEnd();

    // Draw a line
    glLineWidth(2.0f);
    glBegin(GL_LINES);
        glColor3f(0, 1, 0);
        glVertex3f(-0.8f, 0.6f, 0.0f);
        glVertex3f(-0.2f, 0.6f, 0.0f);
    glEnd();

    // Draw a triangle
    glBegin(GL_TRIANGLES);
        glColor3f(0, 0, 1);
        glVertex3f(-0.8f, 0.3f, 0.0f);
        glVertex3f(-0.6f, 0.5f, 0.0f);
        glVertex3f(-0.4f, 0.3f, 0.0f);
    glEnd();

    // Draw a quadrilateral
    glBegin(GL_QUADS);
        glColor3f(1, 1, 0);
        glVertex3f(0.2f, 0.8f, 0.0f);
        glVertex3f(0.6f, 0.8f, 0.0f);
        glVertex3f(0.6f, 0.6f, 0.0f);
        glVertex3f(0.2f, 0.6f, 0.0f);
    glEnd();

    // Draw a polygon (pentagon)
    glBegin(GL_POLYGON);
        glColor3f(0, 1, 1);
        glVertex3f(0.4f, 0.3f, 0.0f);
        glVertex3f(0.5f, 0.5f, 0.0f);
        glVertex3f(0.7f, 0.5f, 0.0f);
        glVertex3f(0.8f, 0.3f, 0.0f);
        glVertex3f(0.6f, 0.1f, 0.0f);
    glEnd();

    // Draw a 3D cube (filled)
    glPushMatrix();
    glTranslatef(0.0f, -0.5f, -2.0f);
    glRotatef(30, 1, 1, 0);
```

```
    // Filled cube
    glColor3f(1, 0, 1); // Magenta
    glutSolidCube(0.5);

    // Wireframe overlay
    glColor3f(1, 1, 1); // White
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glutWireCube(0.5);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL); // Reset to fill mode

    glPopMatrix();

    glutSwapBuffers();
}

void init() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background to black
    glEnable(GL_DEPTH_TEST);
}

void reshape(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (double)w/h, 1.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0,0,2, 0,0,0, 0,1,0);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Basic Drawing Techniques in OpenGL");
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
    return 0;
}
```

Output: