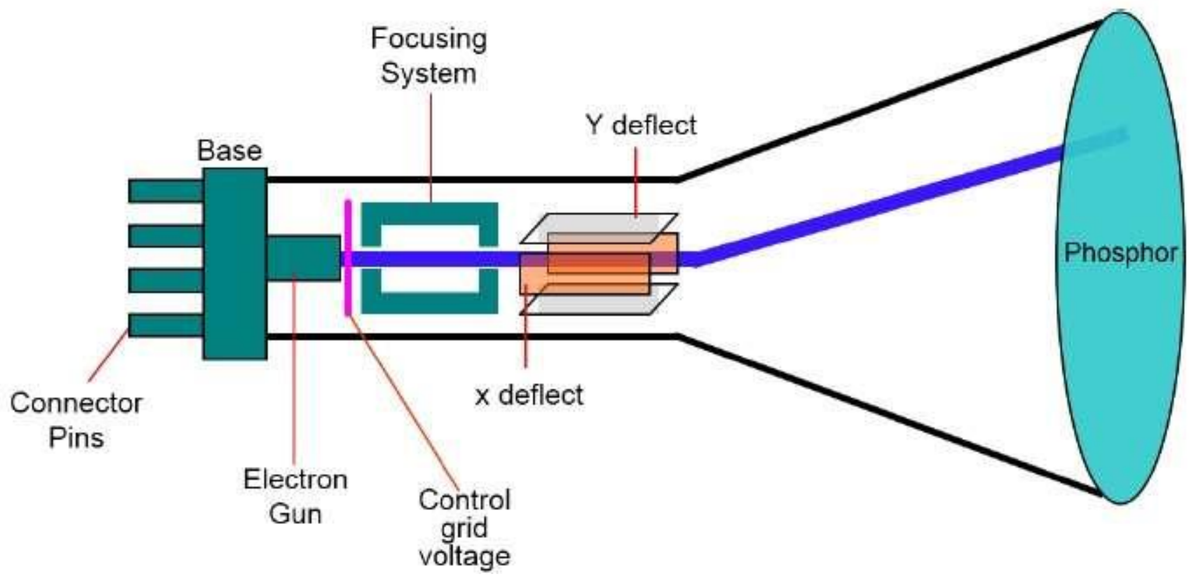
Atma Ram Sanatan Dharma College
University of Delhi

Computer Graphics Practical File



Neeraj

Roll No.: 18088

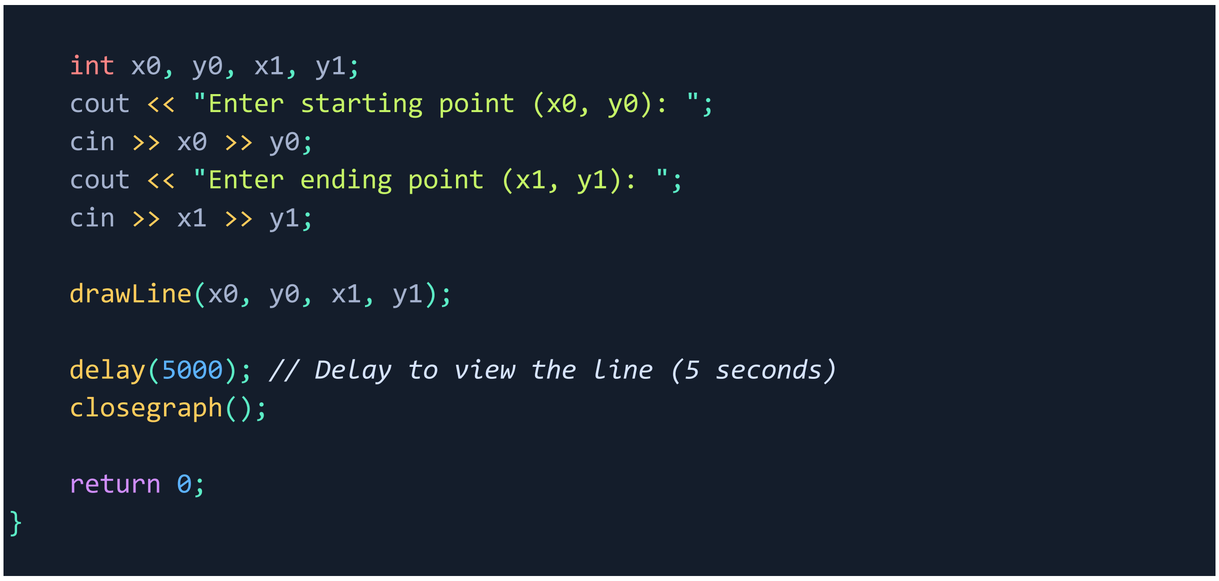
B.Sc.(H) computer science

Submitted to: Ms Manisha Bagri

Q1. Write a program to implement Bresenham’s line drawing algorithm.

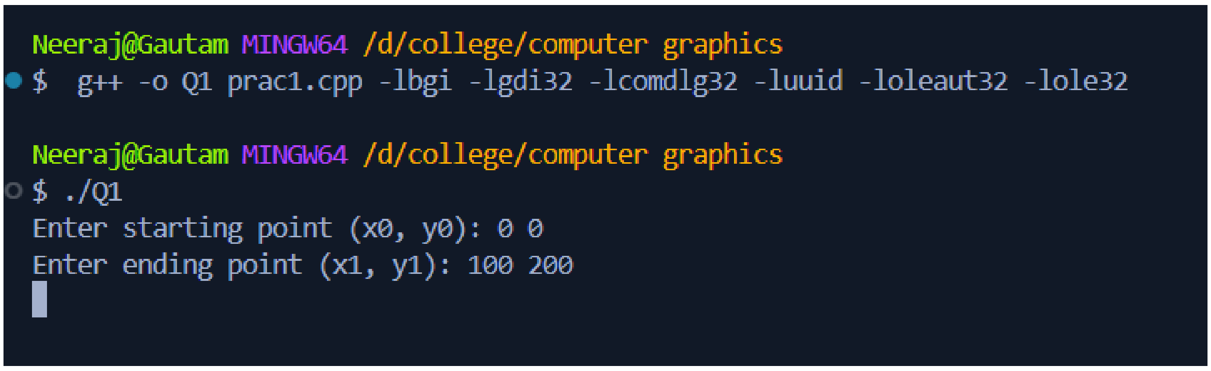
Code

```
#include <iostream>

#include <cmath>
#include <graphics.h>

using namespace std;

void drawLine(int x0, int y0, int x1, int y1)
{
    int dx = abs(x1 - x0);
    int dy = abs(y1 - y0);
    int sx = (x0 < x1) ? 1 : -1;
    int sy = (y0 < y1) ? 1 : -1;
    int err = dx - dy;
    int x = x0;
    int y = y0;

    while (true)
    {
        // Plot pixel
        putpixel(x, y, WHITE);

        if (x == x1 && y == y1)
            break;

        int e2 = 2 * err;
        if (e2 > -dy)
        {
            err -= dy;
            x += sx;
        }
        if (e2 < dx)
        {
            err += dx;
            y += sy;
        }
    }
}

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

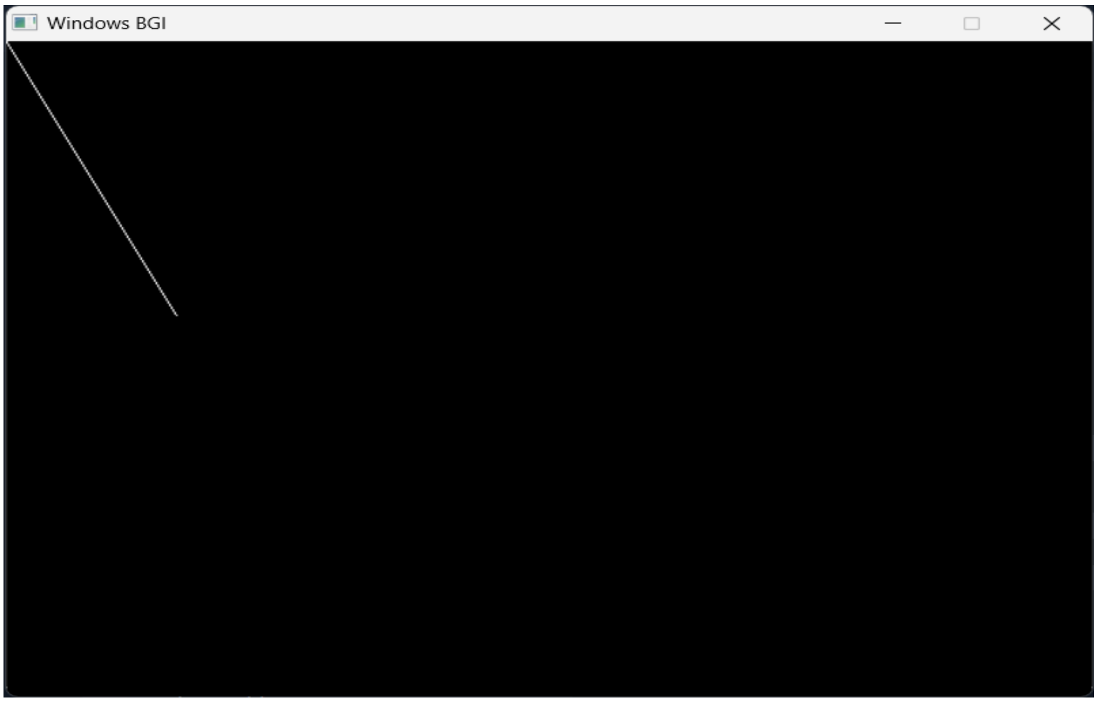


```cpp
    int x0, y0, x1, y1;
    cout << "Enter starting point (x0, y0): ";
    cin >> x0 >> y0;
    cout << "Enter ending point (x1, y1): ";
    cin >> x1 >> y1;

    drawLine(x0, y0, x1, y1);

    delay(5000); // Delay to view the line (5 seconds)
    closegraph();

    return 0;
}
```

Output:-



```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$  g++ -o Q1 prac1.cpp -lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32

Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q1
Enter starting point (x0, y0): 0 0
Enter ending point (x1, y1): 100 200
```



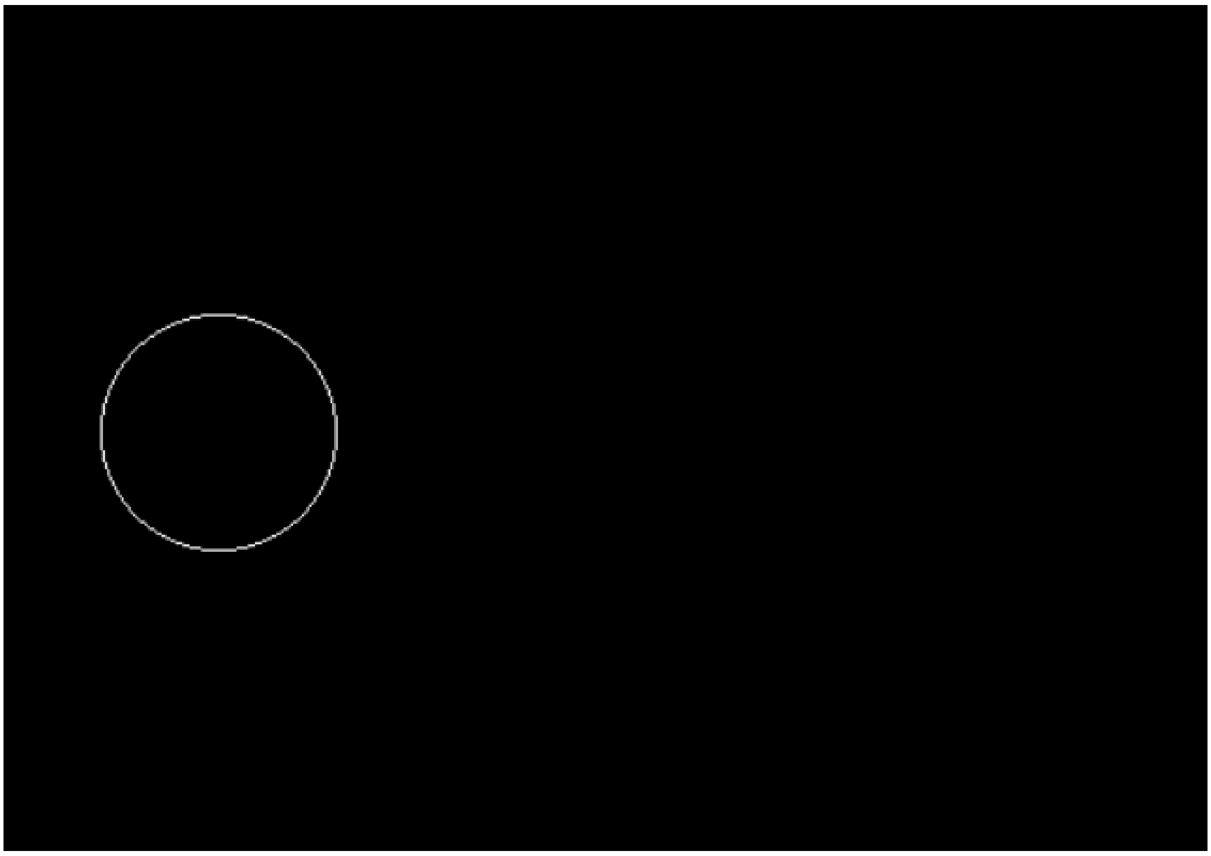
Q2. Write a program to implement mid-point circle drawing algorithm.

Code

```cpp
#include <iostream>
#include <graphics.h>

using namespace std;

void drawCircle(int xc, int yc, int r) {
    int x = 0, y = r;
    int p = 1 - r; // Initial decision parameter

    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);

        if (p <= 0) {
            x++;
            p += 2 * x + 1;
        } else {
            y--;
            x++;
            p += 2 * (x - y) + 1;
        }
    }
}

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

    int xc, yc, r;
    cout << "Enter center coordinates (xc, yc): ";
    cin >> xc >> yc;
    cout << "Enter radius (r): ";
    cin >> r;

    drawCircle(xc, yc, r);
    delay(5000); // Delay to view the circle (5 seconds)
    closegraph();

    return 0;
}
```

Output:-

```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$  g++ -o Q2 prac2.cpp -lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32

Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q2
Enter center coordinates (xc, yc): 100 200
Enter radius (r): 50
```



Q3. Write a program to clip a line using Cohen and Sutherland line clipping algorithm.

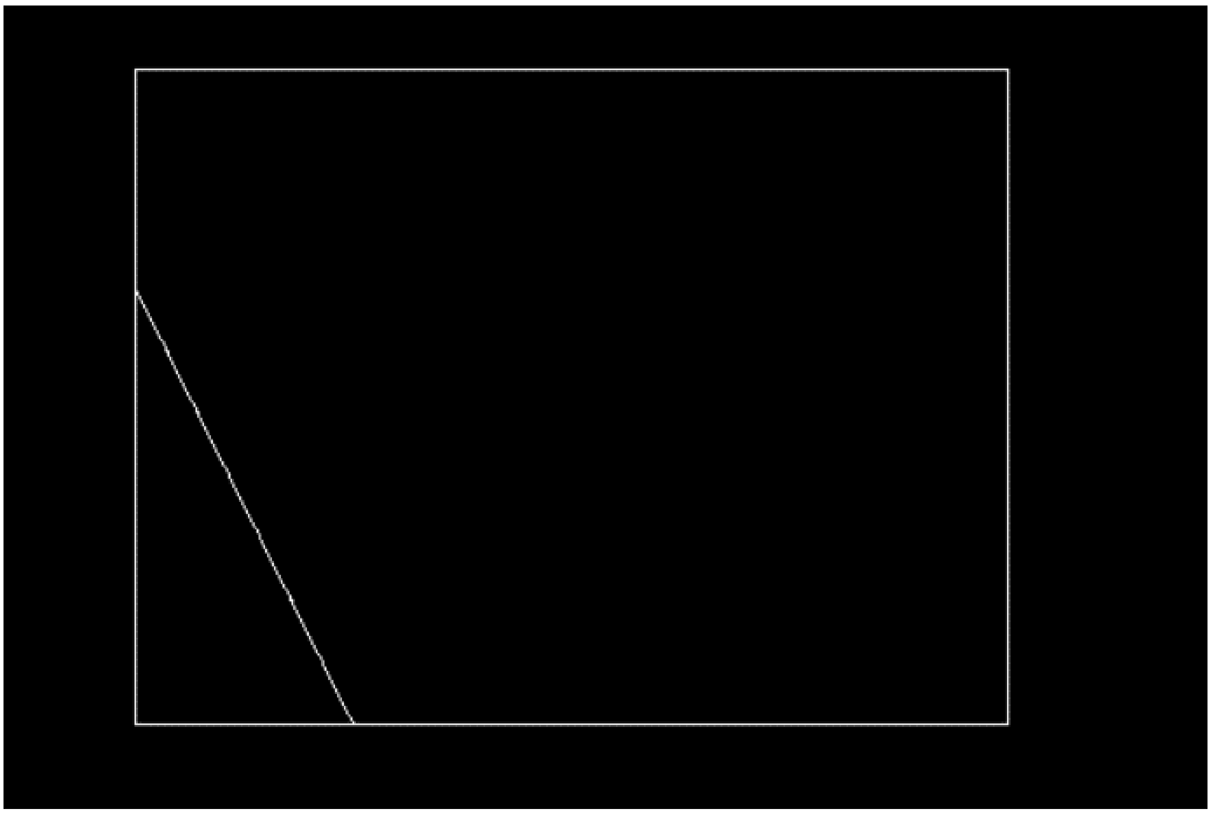
Code

```
#include <iostream>
#include <graphics.h>

using namespace std;

// Define region codes for Cohen-Sutherland algorithm
const int INSIDE = 0; // 0000
const int LEFT = 1;   // 0001
const int RIGHT = 2;  // 0010
const int BOTTOM = 4; // 0100
const int TOP = 8;    // 1000

// Define window boundaries
const int X_MIN = 100;
const int X_MAX = 500;
const int Y_MIN = 100;
const int Y_MAX = 400;

// Calculate region code for a point
int calculateCode(int x, int y) {
    int code = INSIDE; // Initialize as inside

    if (x < X_MIN)      // to the left of window
        code |= LEFT;
    else if (x > X_MAX) // to the right of window
        code |= RIGHT;

    if (y < Y_MIN)       // below window
        code |= BOTTOM;
    else if (y > Y_MAX)  // above window
        code |= TOP;

    return code;
}

// Clip the line using Cohen-Sutherland algorithm
void cohenSutherland(int x0, int y0, int x1, int y1) {
    int code0 = calculateCode(x0, y0);
    int code1 = calculateCode(x1, y1);
    bool accept = false;

    while (true) {
        if (!(code0 | code1)) { // Both endpoints are inside
            accept = true;
            break;
```

```cpp
        } else if (code0 & code1) { // Both endpoints are outside, trivially reject
            break;
        } else {
            int codeOut = code0 ? code0 : code1; // Pick the outside point
            int x, y;

            if (codeOut & TOP) {           // Point is above the clip window
                x = x0 + (x1 - x0) * (Y_MAX - y0) / (y1 - y0);
                y = Y_MAX;
            } else if (codeOut & BOTTOM) { // Point is below the clip window
                x = x0 + (x1 - x0) * (Y_MIN - y0) / (y1 - y0);
                y = Y_MIN;
            } else if (codeOut & RIGHT) {  // Point is to the right of the clip window
                y = y0 + (y1 - y0) * (X_MAX - x0) / (x1 - x0);
                x = X_MAX;
            } else if (codeOut & LEFT) {   // Point is to the left of the clip window
                y = y0 + (y1 - y0) * (X_MIN - x0) / (x1 - x0);
                x = X_MIN;
            }

            if (codeOut == code0) {
                x0 = x;
                y0 = y;
                code0 = calculateCode(x0, y0);
            } else {
                x1 = x;
                y1 = y;
                code1 = calculateCode(x1, y1);
            }
        }
    }

    if (accept) {
        line(x0, y0, x1, y1);
    }
}

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

    int x0, y0, x1, y1;
    cout << "Enter line coordinates (x0, y0) and (x1, y1): ";
    cin >> x0 >> y0 >> x1 >> y1;
    rectangle(X_MIN, Y_MIN, X_MAX, Y_MAX); // Draw the clipping window
    cohenSutherland(x0, y0, x1, y1);

    delay(5000); // Delay to view the clipped line (5 seconds)
    closegraph();

    return 0;
}
```

Output:-

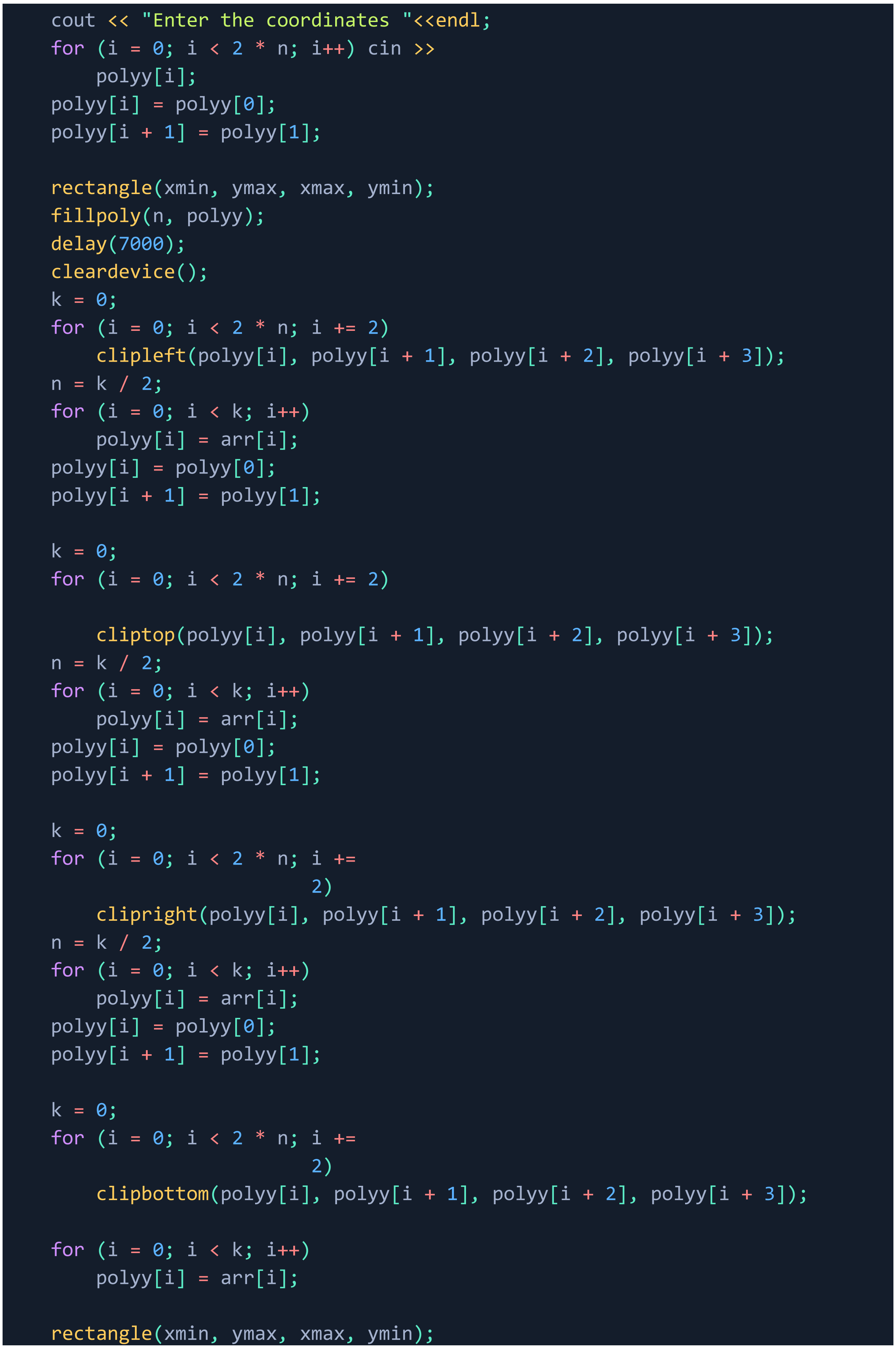
```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$  g++ -o Q3 prac3.cpp -lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32

Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q3
Enter line coordinates (x0, y0) and (x1, y1): 100 200 200 400
```



Q4. Write a program to clip a polygon using Sutherland Hodgeman algorithm.

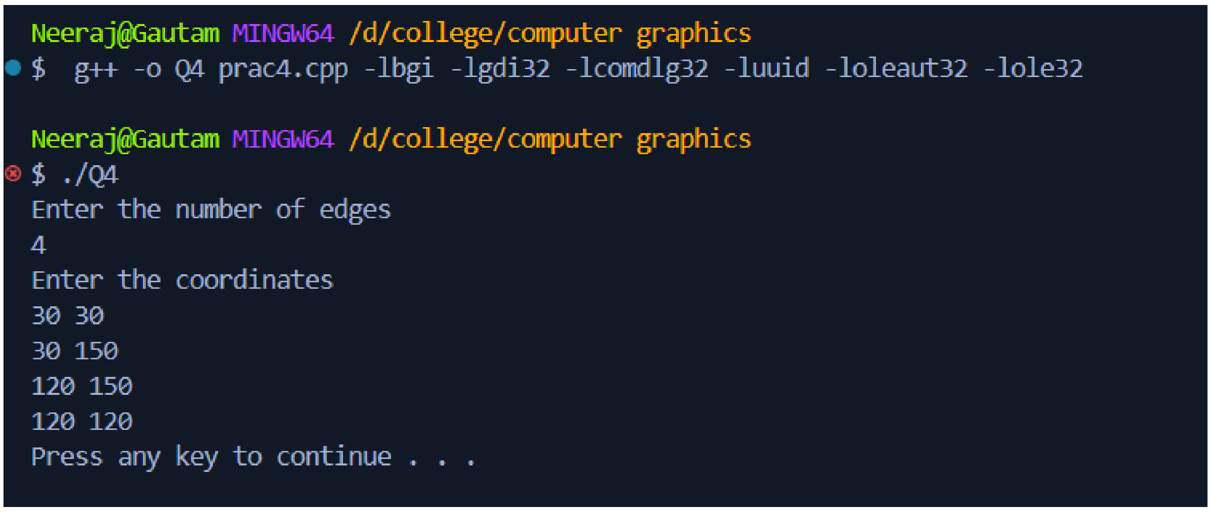
Code

```cpp
#include <iostream>
#include <graphics.h>
using namespace std;
#define round(a) ((int)(a + 0.5))
int
    xmin = 100,
    xmax = 500, ymin = 100, ymax = 500, arr[20],
    m;
int k;
void clipleft(int x1, int y1, int x2, int y2)
{
    if (x2 - x1)
        m = (y2 - y1) / (x2-x1);
    else
        m = 10000;

    if (x1 >= xmin && x2 >= xmin)
    {
        arr[k] = x2;
        arr[k + 1] = y2;
        k += 2;
    }
    if (x1 < xmin && x2 >= xmin)
    {
        arr[k] = xmin;

        arr[k + 1] = y1 + m * (xmin - x1);
        arr[k + 2] = x2;
        arr[k + 3] = y2;
        k += 4;
    }
    if (x1 >= xmin && x2 < xmin)
    {
        arr[k] = xmin;
        arr[k + 1] = y1 + m * (xmin - x1);
        k += 2;
    }
}

void cliptop(int x1, int y1, int x2, int y2)
{
    if (y2 - y1)
        m = (x2 - x1) / (y2 - y1);
    else
        m = 10000;
    if (y1 <= ymax && y2 <= ymax)
    {

        arr[k] = x2;
        arr[k + 1] = y2;
        k += 2;
    }
    if (y1 > ymax && y2 <= ymax)
    {
        arr[k] = x1 + m * (ymax - y1);
        arr[k + 1] = ymax;
        arr[k + 2] = x2;
        arr[k + 3] = y2;
        k += 4;
    }
    if (y1 <= ymax && y2 > ymax)
    {
        arr[k] = x1 + m * (ymax - y1);
        arr[k + 1] = ymax;
        k += 2;
    }
}

void clipright(int x1, int y1, int x2, int y2)
{
    if (x2 - x1)
        m = (y2 -
             y1) /
            (x2 - x1);
    else

        m = 10000;

    if (x1 <= xmax && x2 <= xmax)
    {
        arr[k] = x2;
        arr[k + 1] = y2;
        k += 2;
    }
    if (x1 > xmax && x2 <= xmax)
    {
        arr[k] = xmax;

        arr[k + 1] = y1 + m * (xmax - x1);

        arr[k + 2] = x2;

        arr[k + 3] = y2;
        k += 4;
    }
    if (x1 <= xmax && x2 > xmax)
    {
        arr[k] = xmax;
        arr[k + 1] = y1 + m * (xmax - x1);
        k += 2;
    }
}

void clipbottom(int x1, int y1, int x2, int y2)
{
    if (y2 - y1)
        m = (x2 - x1) / (y2 - y1);
    else
        m = 10000;

    if (y1 >= ymin && y2 >= ymin)
    {
        arr[k] = x2;

        arr[k + 1] = y2;
        k += 2;
    }
    if (y1 < ymin && y2 >= ymin)
    {
        arr[k] = x1 + m * (ymin -
                           y1);
        arr[k + 1] = ymin;
        arr[k + 2] = x2;
        arr[k + 3] = y2;
        k += 4;
    }
    if (y1 >= ymin && y2 < ymin)
    {
        arr[k] = x1 + m * (ymin - y1);
        arr[k + 1] = ymin;
        k += 2;
    }
}

int main()
{
    int polyy[20];
    int window1 = initwindow(800, 800);
    int n, i;
    cout << "Enter the number of edges "<<endl;
    cin >> n;
```

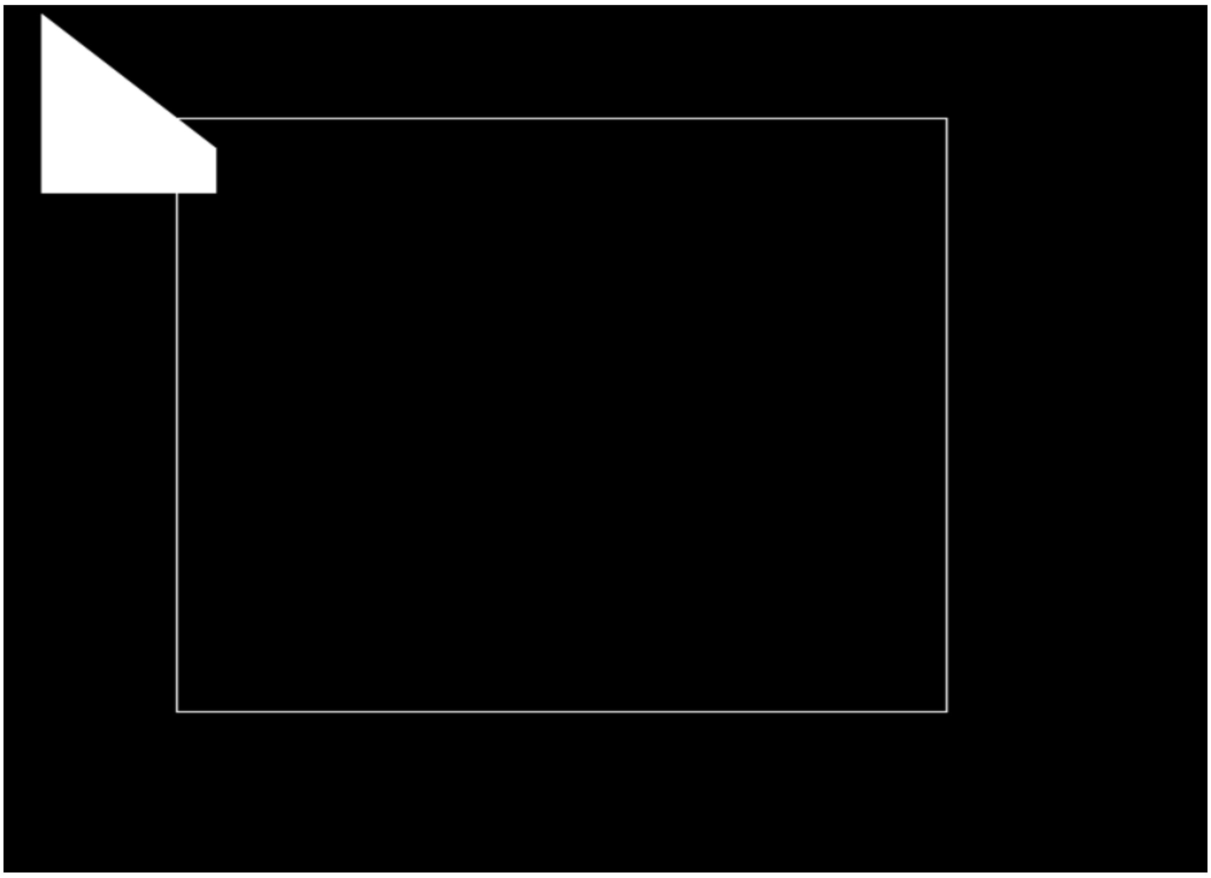


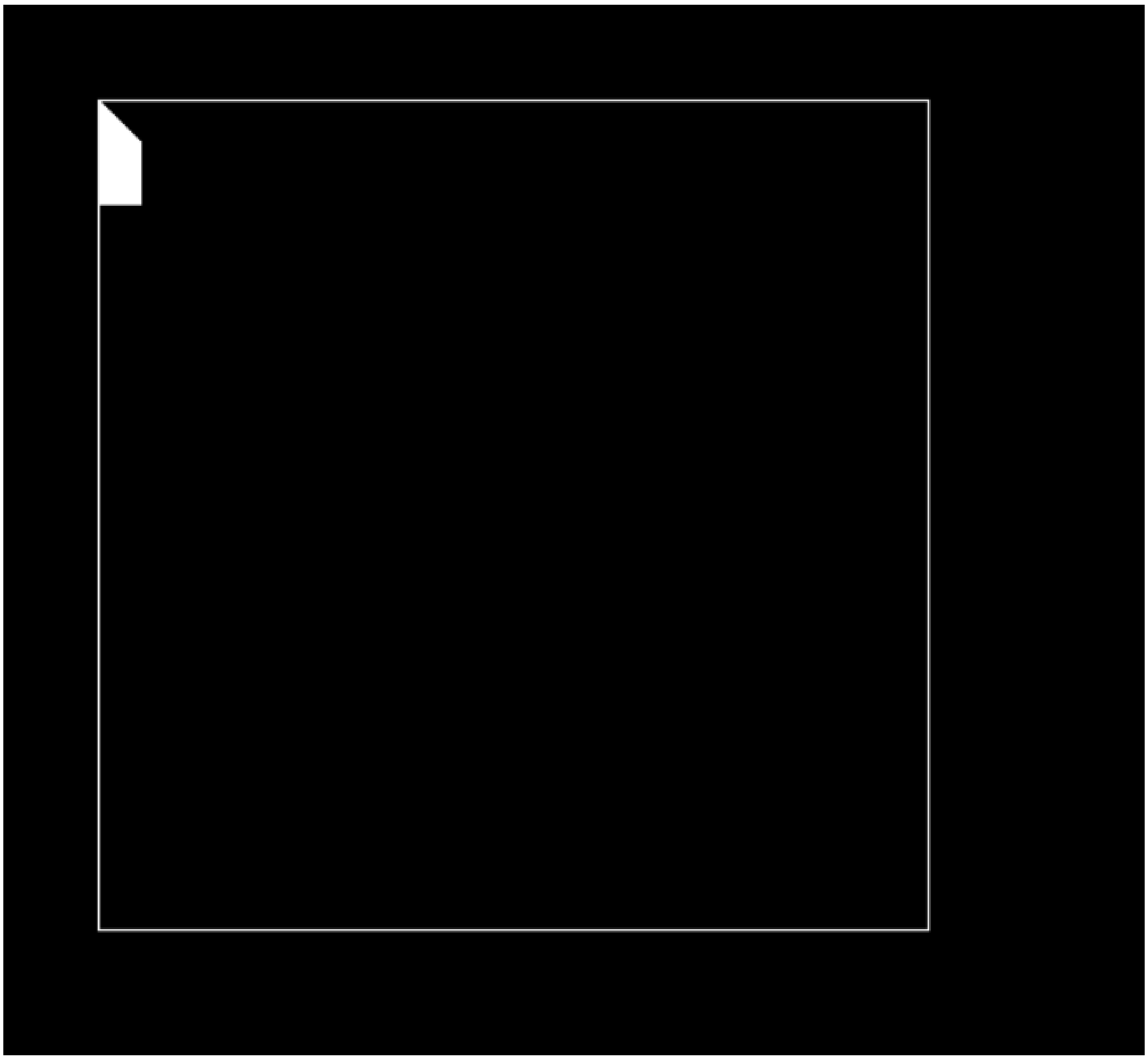
```
    cout << "Enter the coordinates "<<endl;
    for (i = 0; i < 2 * n; i++) cin >>
        polyy[i];
    polyy[i] = polyy[0];
    polyy[i + 1] = polyy[1];

    rectangle(xmin, ymax, xmax, ymin);
    fillpoly(n, polyy);
    delay(7000);
    cleardevice();
    k = 0;
    for (i = 0; i < 2 * n; i += 2)
        clipleft(polyy[i], polyy[i + 1], polyy[i + 2], polyy[i + 3]);
    n = k / 2;
    for (i = 0; i < k; i++)
        polyy[i] = arr[i];
    polyy[i] = polyy[0];
    polyy[i + 1] = polyy[1];

    k = 0;
    for (i = 0; i < 2 * n; i += 2)

        cliptop(polyy[i], polyy[i + 1], polyy[i + 2], polyy[i + 3]);
    n = k / 2;
    for (i = 0; i < k; i++)
        polyy[i] = arr[i];
    polyy[i] = polyy[0];
    polyy[i + 1] = polyy[1];

    k = 0;
    for (i = 0; i < 2 * n; i +=
                           2)
        clipright(polyy[i], polyy[i + 1], polyy[i + 2], polyy[i + 3]);
    n = k / 2;
    for (i = 0; i < k; i++)
        polyy[i] = arr[i];
    polyy[i] = polyy[0];
    polyy[i + 1] = polyy[1];

    k = 0;
    for (i = 0; i < 2 * n; i +=
                           2)
        clipbottom(polyy[i], polyy[i + 1], polyy[i + 2], polyy[i + 3]);

    for (i = 0; i < k; i++)
        polyy[i] = arr[i];

    rectangle(xmin, ymax, xmax, ymin);
    if (k)
        fillpoly(k / 2, polyy);

    system("pause");
    return 1;
}
```

Output:-



```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$  g++ -o Q4 prac4.cpp -lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32

Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q4
Enter the number of edges
4
Enter the coordinates
30 30
30 150
120 150
120 120
Press any key to continue . . .
```

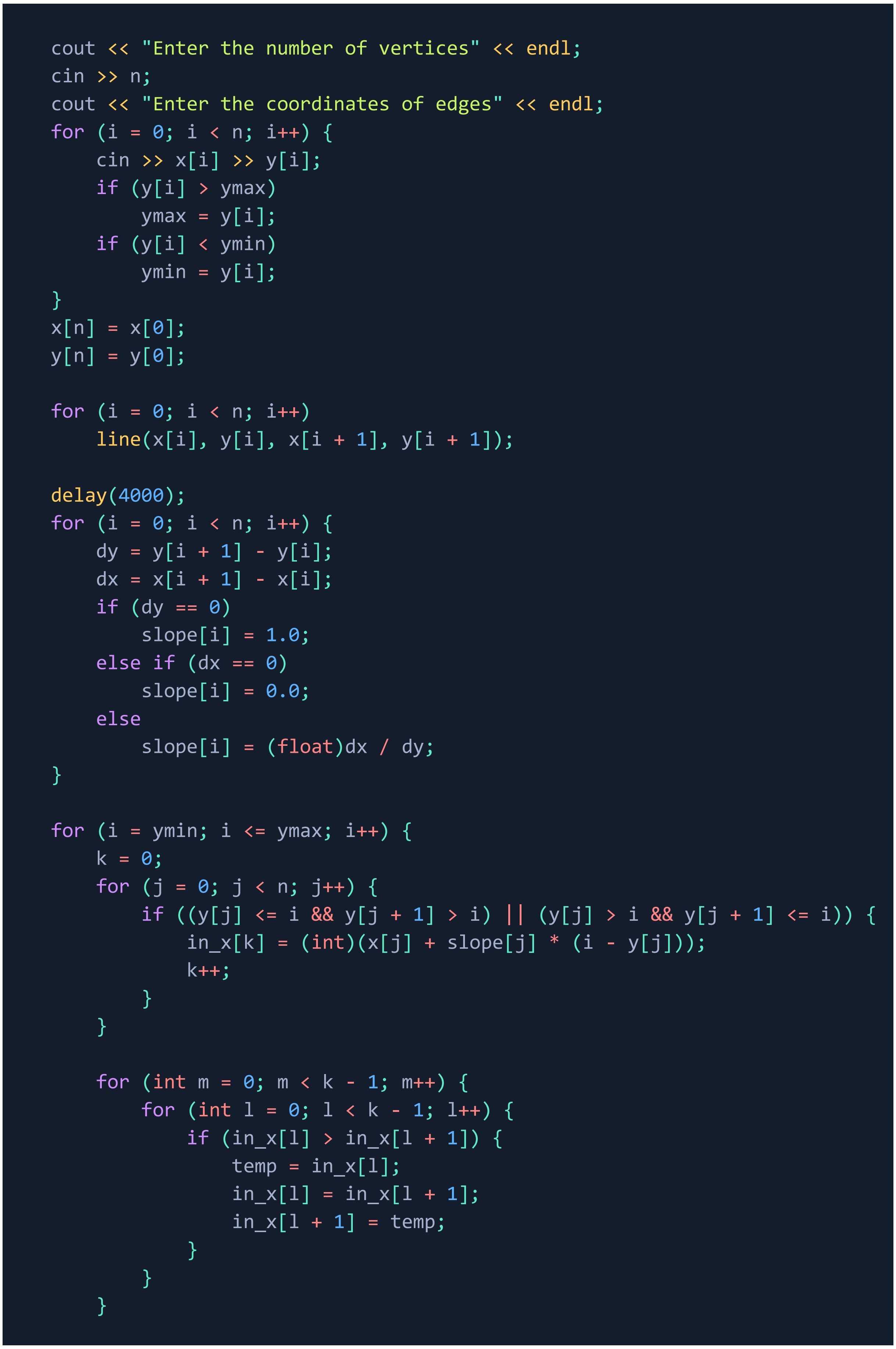




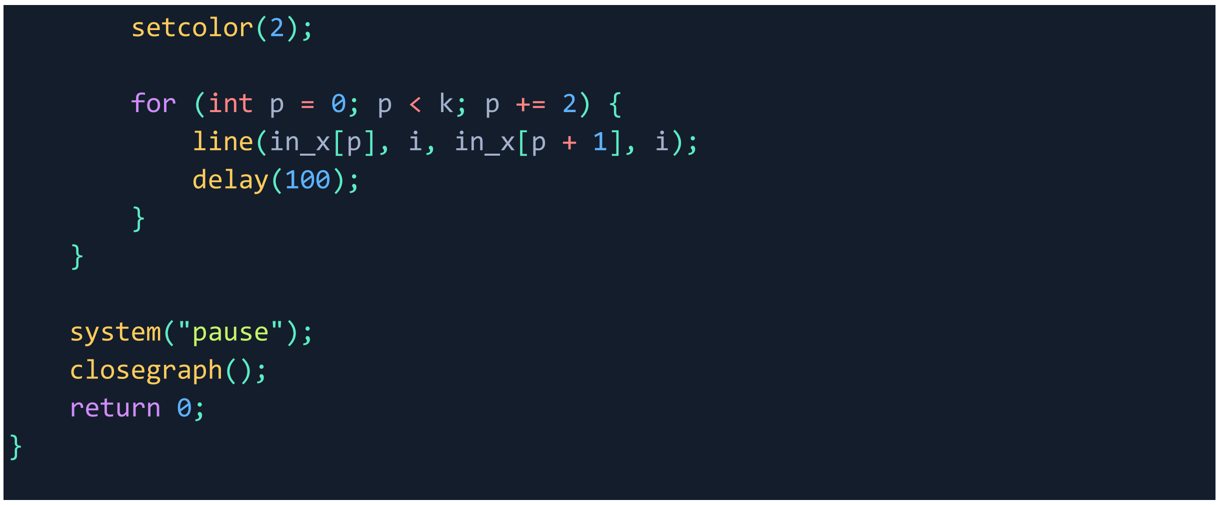
Q5. Write a program to fill a polygon using Scan line fill algorithm.

Code

```
#include <iostream>
#include <graphics.h>

using namespace std;

int main() {
    int i, j, n, k, x[20], y[20], ymin = 10000, ymax = 0, dy, dx, in_x[100],
temp;
    float slope[100];
    int window1 = initwindow(800, 800);
```

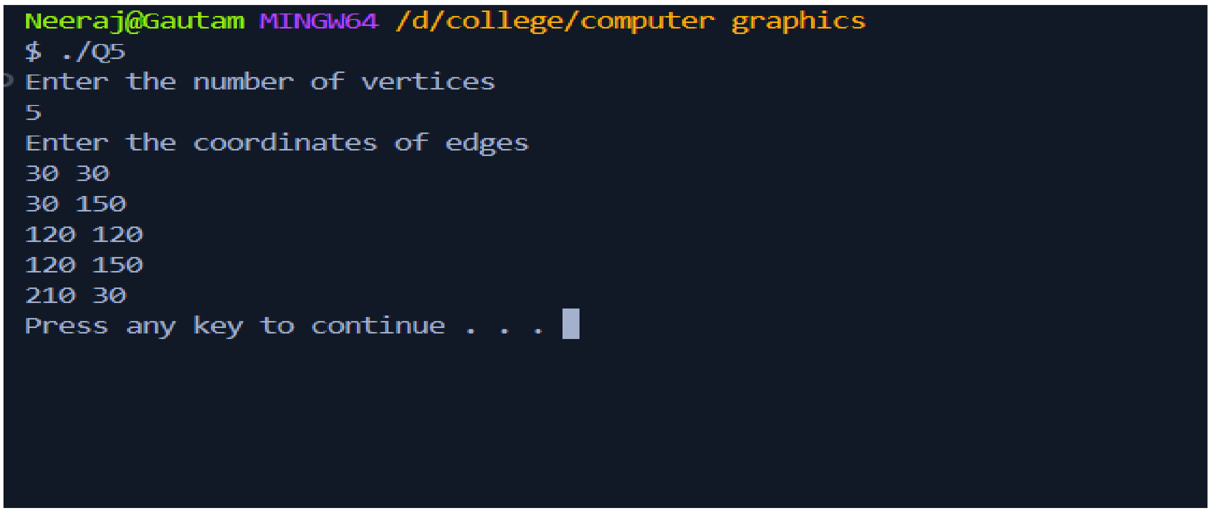


```cpp
cout << "Enter the number of vertices" << endl;
cin >> n;
cout << "Enter the coordinates of edges" << endl;
for (i = 0; i < n; i++) {
    cin >> x[i] >> y[i];
    if (y[i] > ymax)
        ymax = y[i];
    if (y[i] < ymin)
        ymin = y[i];
}
x[n] = x[0];
y[n] = y[0];

for (i = 0; i < n; i++)
    line(x[i], y[i], x[i + 1], y[i + 1]);

delay(4000);
for (i = 0; i < n; i++) {
    dy = y[i + 1] - y[i];
    dx = x[i + 1] - x[i];
    if (dy == 0)
        slope[i] = 1.0;
    else if (dx == 0)
        slope[i] = 0.0;
    else
        slope[i] = (float)dx / dy;
}

for (i = ymin; i <= ymax; i++) {
    k = 0;
    for (j = 0; j < n; j++) {
        if ((y[j] <= i && y[j + 1] > i) || (y[j] > i && y[j + 1] <= i)) {
            in_x[k] = (int)(x[j] + slope[j] * (i - y[j]));
            k++;
        }
    }

    for (int m = 0; m < k - 1; m++) {
        for (int l = 0; l < k - 1; l++) {
            if (in_x[l] > in_x[l + 1]) {
                temp = in_x[l];
                in_x[l] = in_x[l + 1];
                in_x[l + 1] = temp;
            }
        }
    }
```

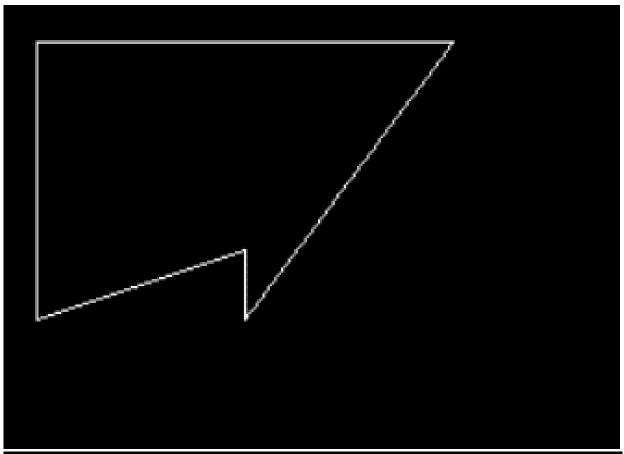
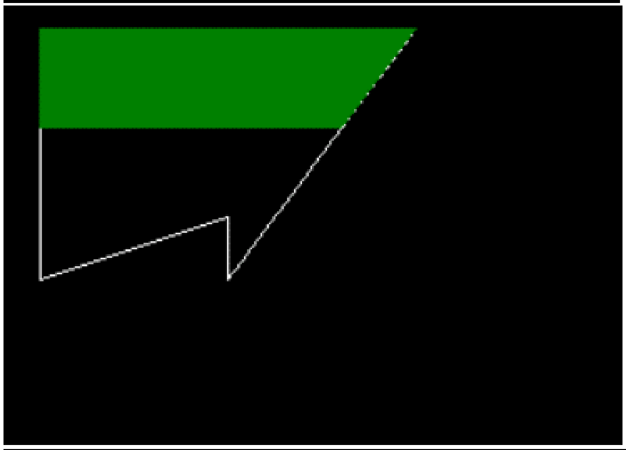
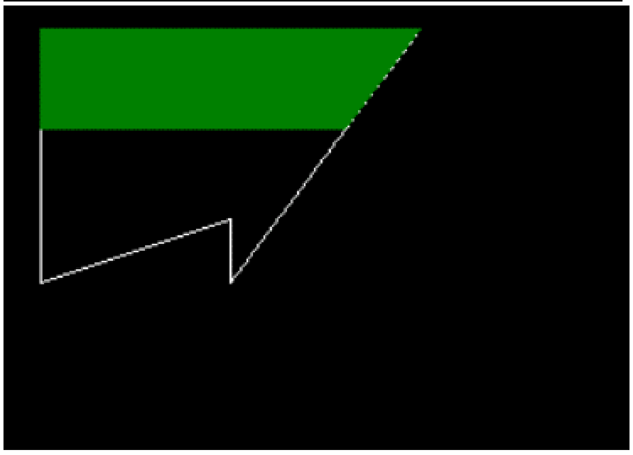
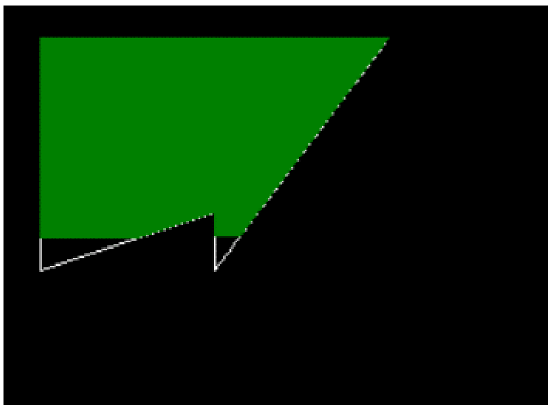


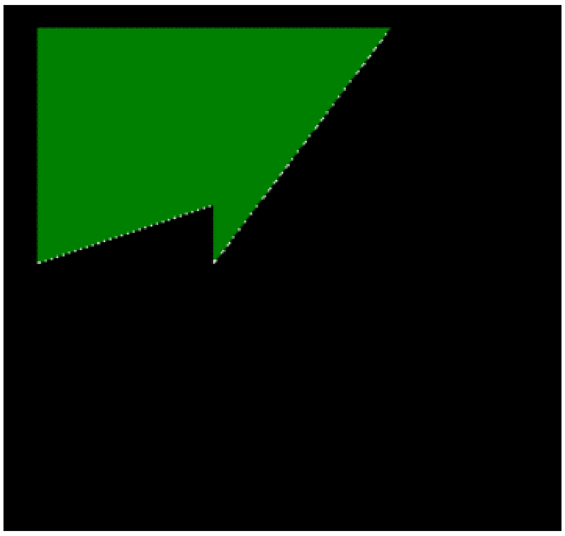
```c
        setcolor(2);

        for (int p = 0; p < k; p += 2) {
            line(in_x[p], i, in_x[p + 1], i);
            delay(100);
        }
    }

    system("pause");
    closegraph();
    return 0;
}
```

Output:-



```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q5
Enter the number of vertices
5
Enter the coordinates of edges
30 30
30 150
120 120
120 150
210 30
Press any key to continue . . .
```

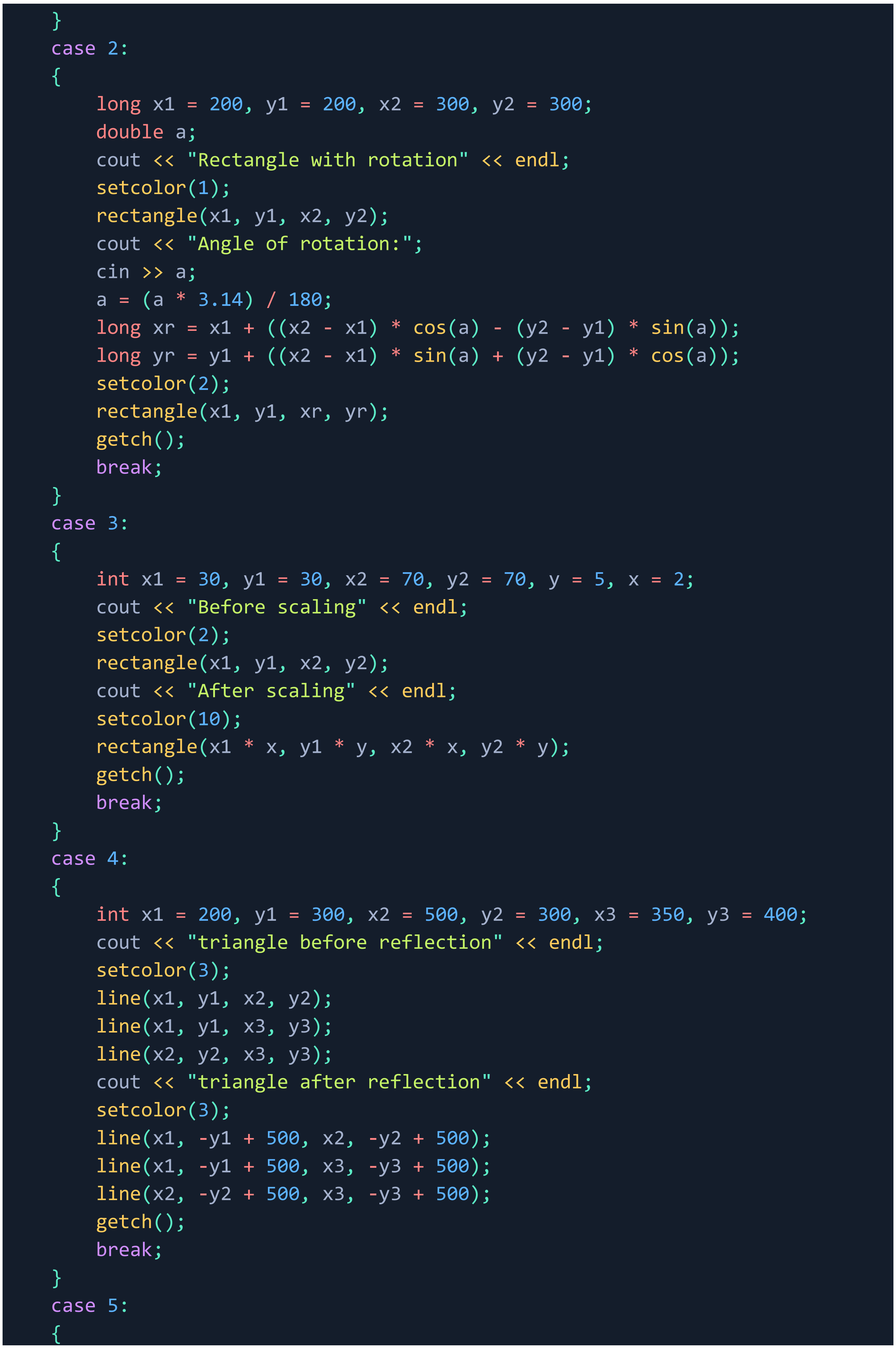







Q6. Write a program to apply various 2D transformations on a 2D object (use homogenous Coordinates).

Code

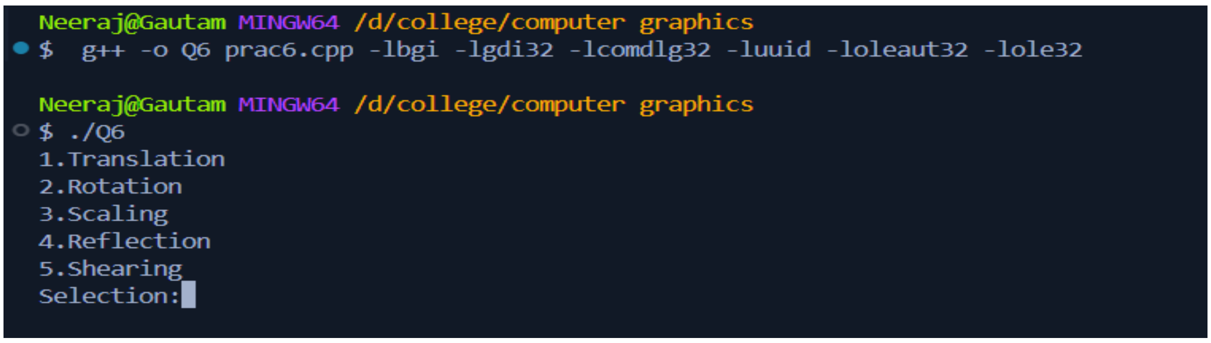
```
#include <iostream>
#include <graphics.h>
#include <math.h>
using namespace std;
int main()
{
    int gd = DETECT, gm, s;
    initgraph(&gd, &gm, (char *)"");
    cout <<
"1.Translation\n2.Rotation\n3.Scaling\n4.Reflection\n5.Shearing   " << endl;
    cout << "Selection:";
    cin >> s;
    switch (s)
    {
    case 1:
    {
        int x1 = 200, y1 = 150, x2 = 300, y2 = 250;
        int tx = 200, ty = 100;
        cout << "Rectangle before translation" << endl;
        setcolor(3);
        rectangle(x1, y1, x2, y2);
        setcolor(10);
        cout << "Rectangle after translation" << endl;
        rectangle(x1 + tx, y1 + ty, x2 + tx, y2 + ty);
        getch();
        break;
```



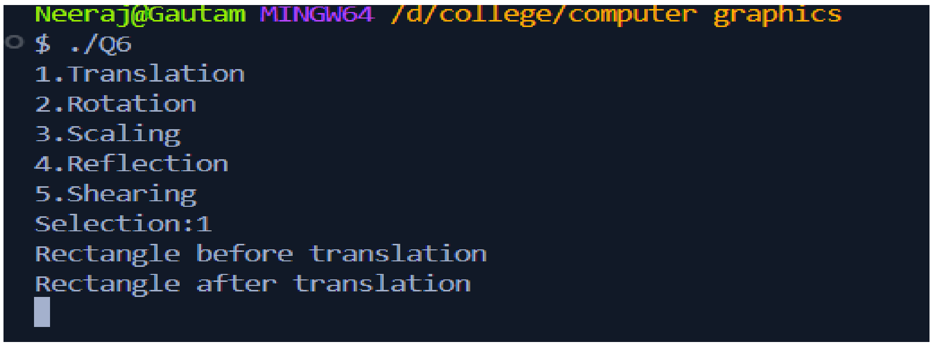
```
    }
    case 2:
    {
        long x1 = 200, y1 = 200, x2 = 300, y2 = 300;
        double a;
        cout << "Rectangle with rotation" << endl;
        setcolor(1);
        rectangle(x1, y1, x2, y2);
        cout << "Angle of rotation:";
        cin >> a;
        a = (a * 3.14) / 180;
        long xr = x1 + ((x2 - x1) * cos(a) - (y2 - y1) * sin(a));
        long yr = y1 + ((x2 - x1) * sin(a) + (y2 - y1) * cos(a));
        setcolor(2);
        rectangle(x1, y1, xr, yr);
        getch();
        break;
    }
    case 3:
    {
        int x1 = 30, y1 = 30, x2 = 70, y2 = 70, y = 5, x = 2;
        cout << "Before scaling" << endl;
        setcolor(2);
        rectangle(x1, y1, x2, y2);
        cout << "After scaling" << endl;
        setcolor(10);
        rectangle(x1 * x, y1 * y, x2 * x, y2 * y);
        getch();
        break;
    }
    case 4:
    {
        int x1 = 200, y1 = 300, x2 = 500, y2 = 300, x3 = 350, y3 = 400;
        cout << "triangle before reflection" << endl;
        setcolor(3);
        line(x1, y1, x2, y2);
        line(x1, y1, x3, y3);
        line(x2, y2, x3, y3);
        cout << "triangle after reflection" << endl;
        setcolor(3);
        line(x1, -y1 + 500, x2, -y2 + 500);
        line(x1, -y1 + 500, x3, -y3 + 500);
        line(x2, -y2 + 500, x3, -y3 + 500);
        getch();
        break;
    }
    case 5:
    {
```

```cpp
        int x1 = 400, y1 = 100, x2 = 600, y2 = 100, x3 = 400, y3 = 200, x4 =
600, y4 = 200, shx = -1;
        cout << "Before shearing of rectangle" << endl;
        setcolor(3);
        line(x1, y1, x2, y2);
        line(x1, y1, x3, y3);
        line(x3, y3, x4, y4);
        line(x2, y2, x4, y4);
        cout << "After shearing of rectangle" << endl;
        x1 = x1 + shx * y1;
        x2 = x2 + shx * y2;
        x3 = x3 + shx * y3;
        x4 = x4 + shx * y4;
        setcolor(10);
        line(x1, y1, x2, y2);
        line(x1, y1, x3, y3);
        line(x3, y3, x4, y4);
        line(x2, y2, x4, y4);
        getch();
    }
    default:
    {
        cout << "Invalid Selection" << endl;
        break;
    }
    }
    closegraph();
    return 0;
}
```

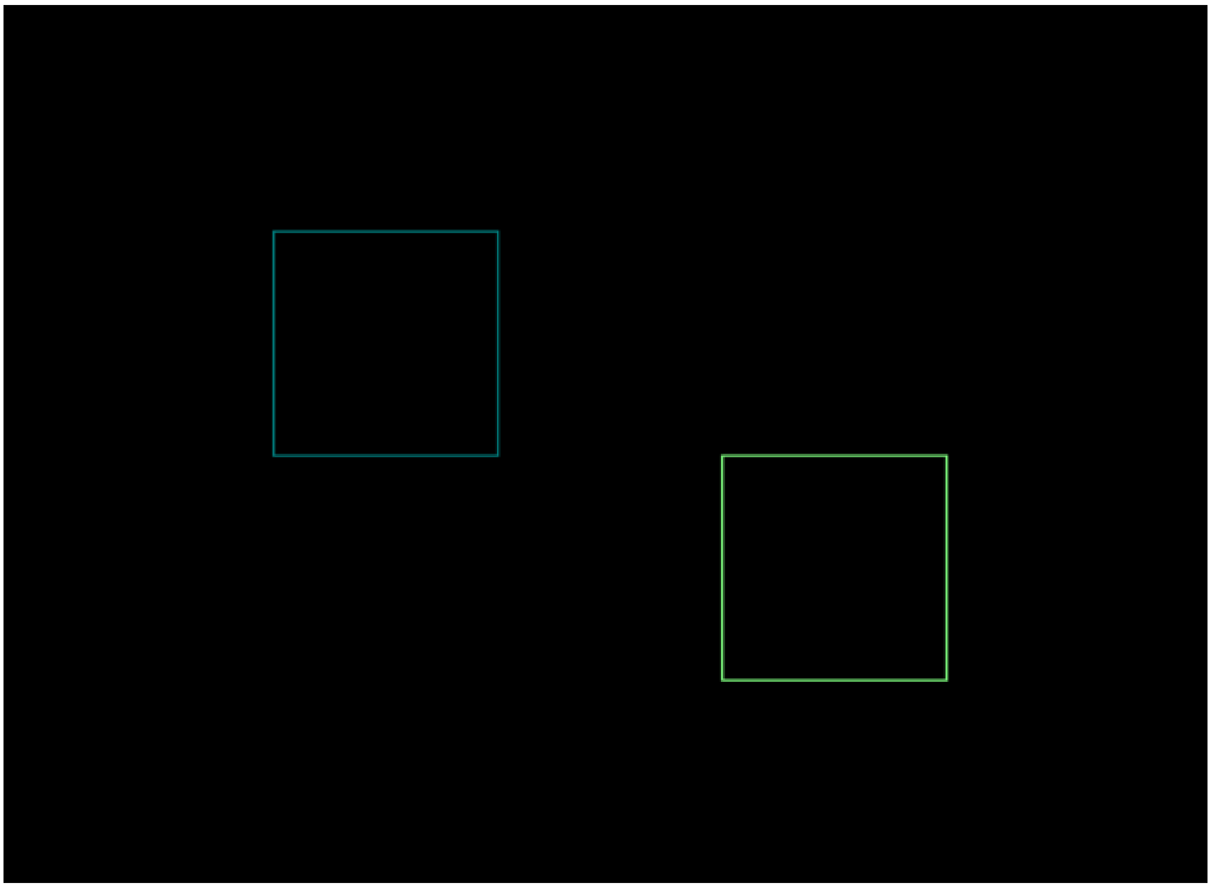
Output:-

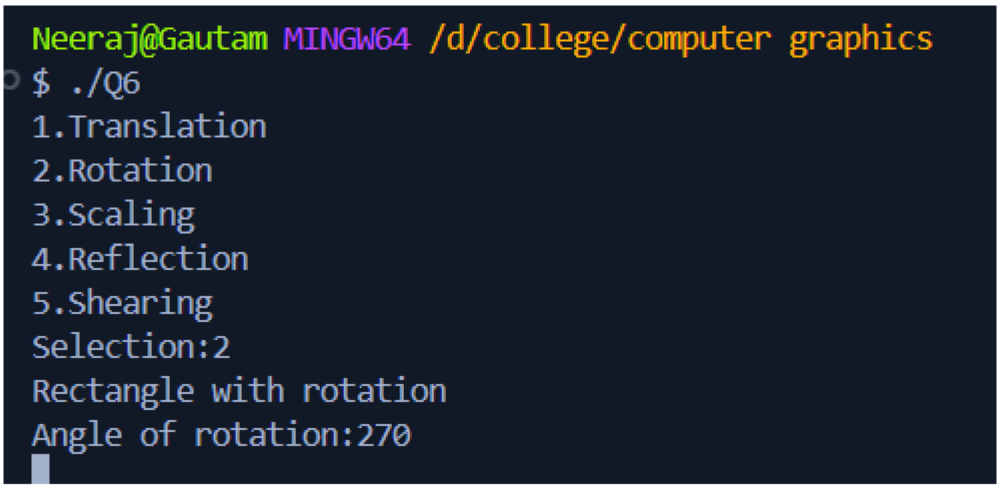


```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$  g++ -o Q6 prac6.cpp -lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32

Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q6
1.Translation
2.Rotation
3.Scaling
4.Reflection
5.Shearing
Selection:
```

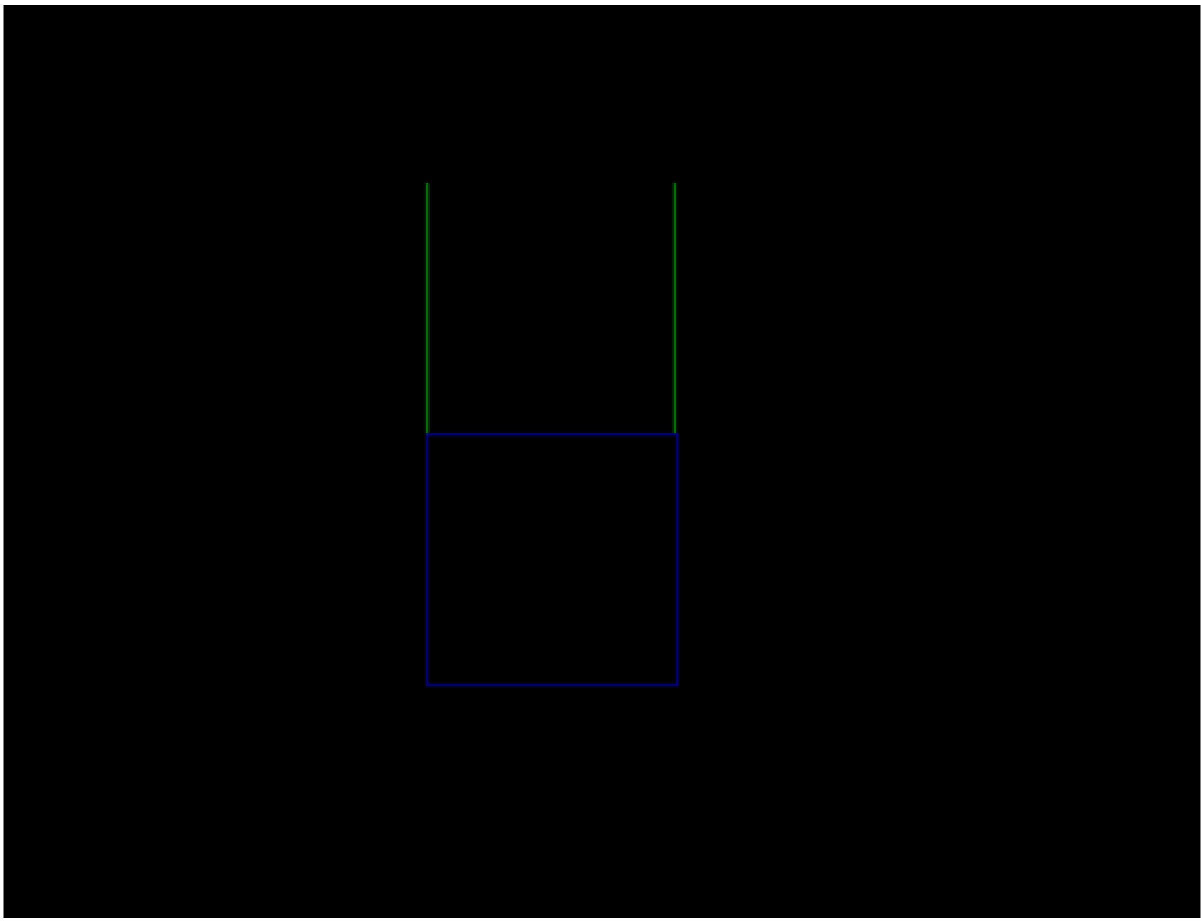


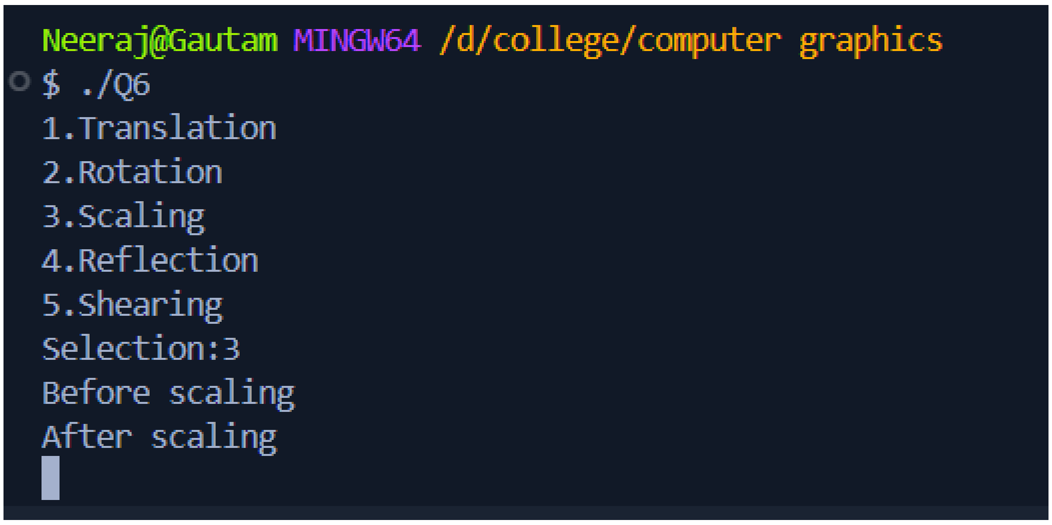
```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q6
1.Translation
2.Rotation
3.Scaling
4.Reflection
5.Shearing
Selection:1
Rectangle before translation
Rectangle after translation
```



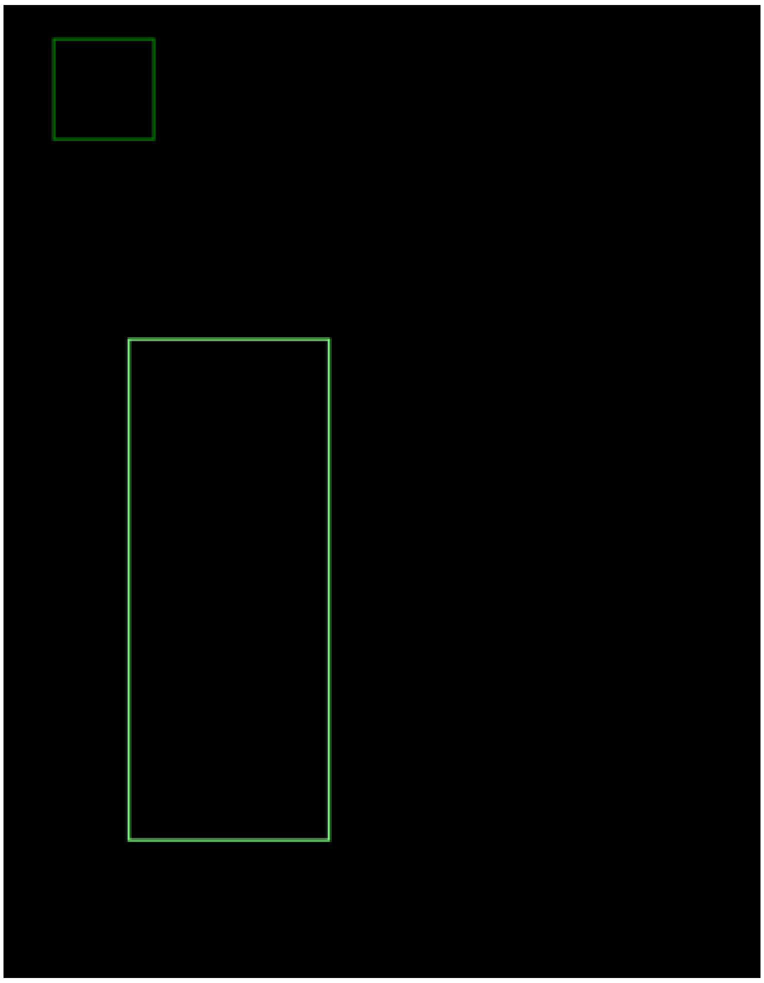


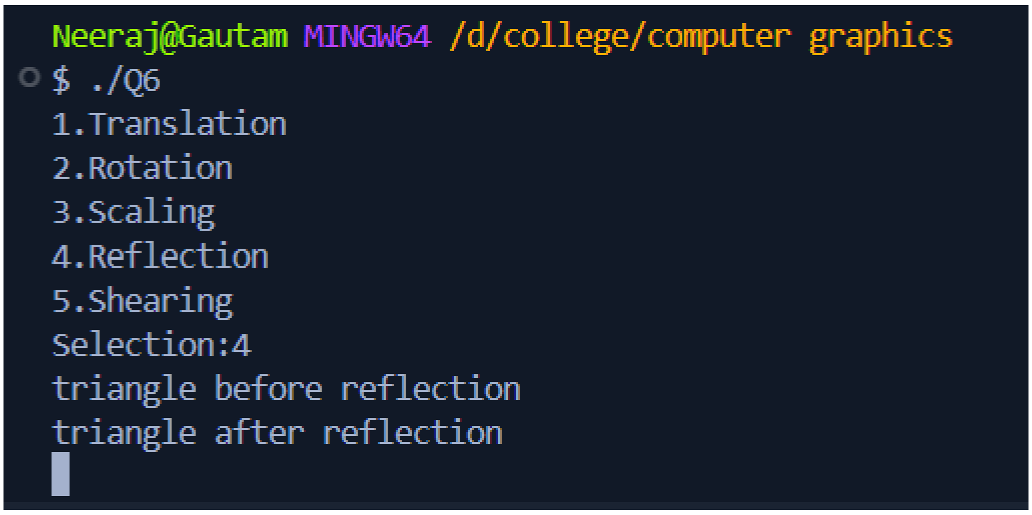
```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q6
1.Translation
2.Rotation
3.Scaling
4.Reflection
5.Shearing
Selection:2
Rectangle with rotation
Angle of rotation:270
```



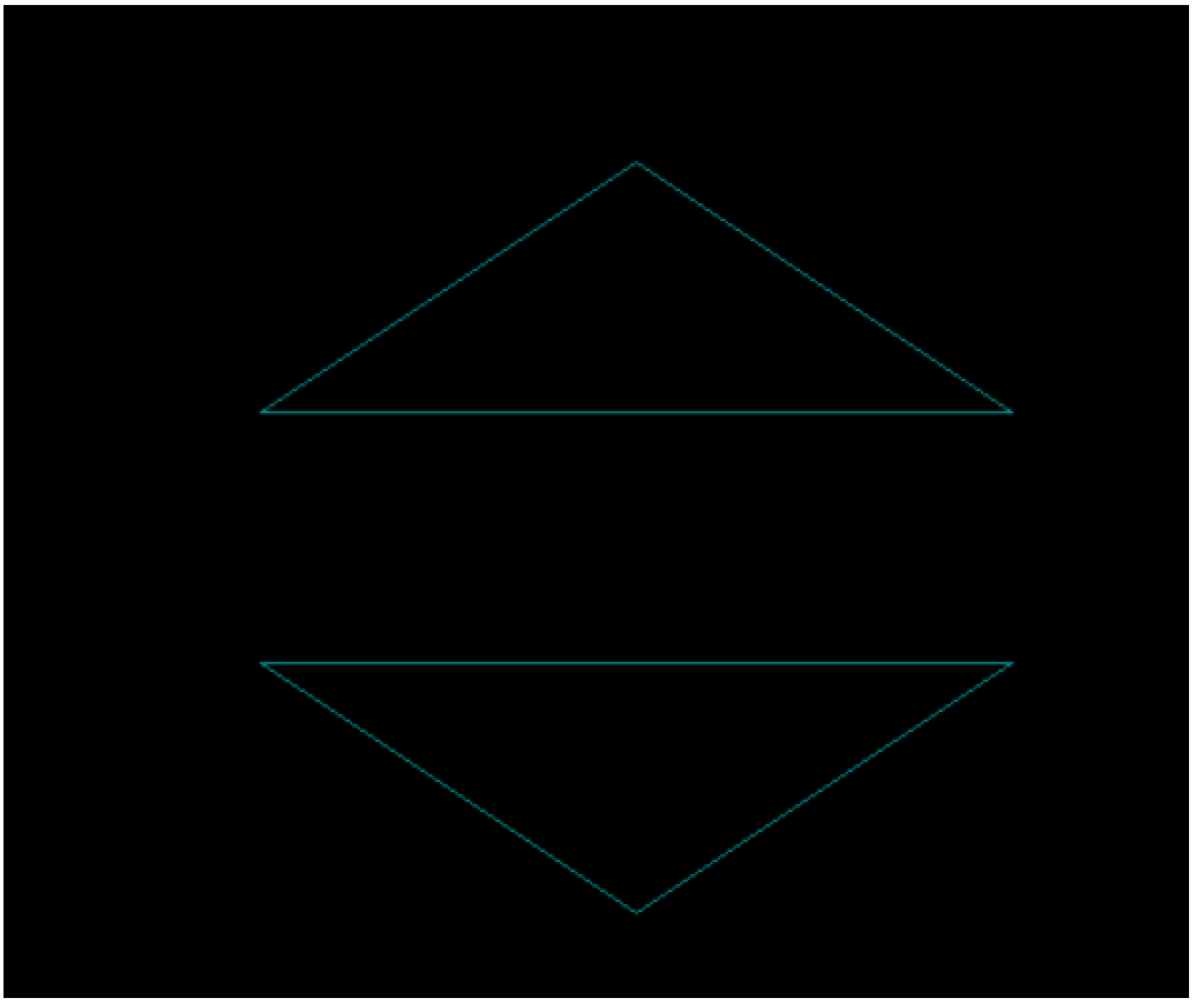


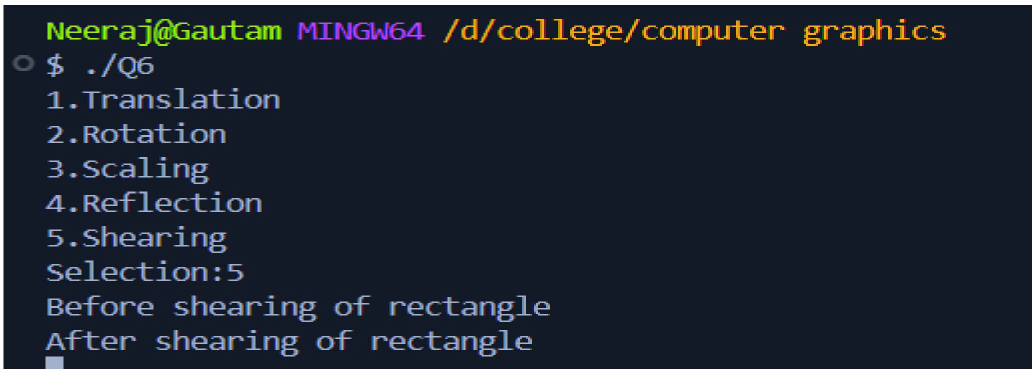
```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q6
1.Translation
2.Rotation
3.Scaling
4.Reflection
5.Shearing
Selection:3
Before scaling
After scaling
```



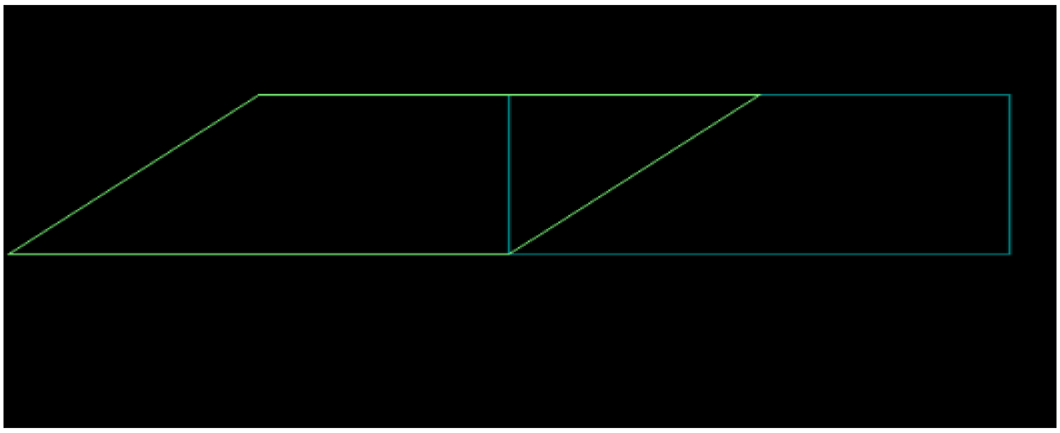


```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q6
1.Translation
2.Rotation
3.Scaling
4.Reflection
5.Shearing
Selection:4
triangle before reflection
triangle after reflection
```





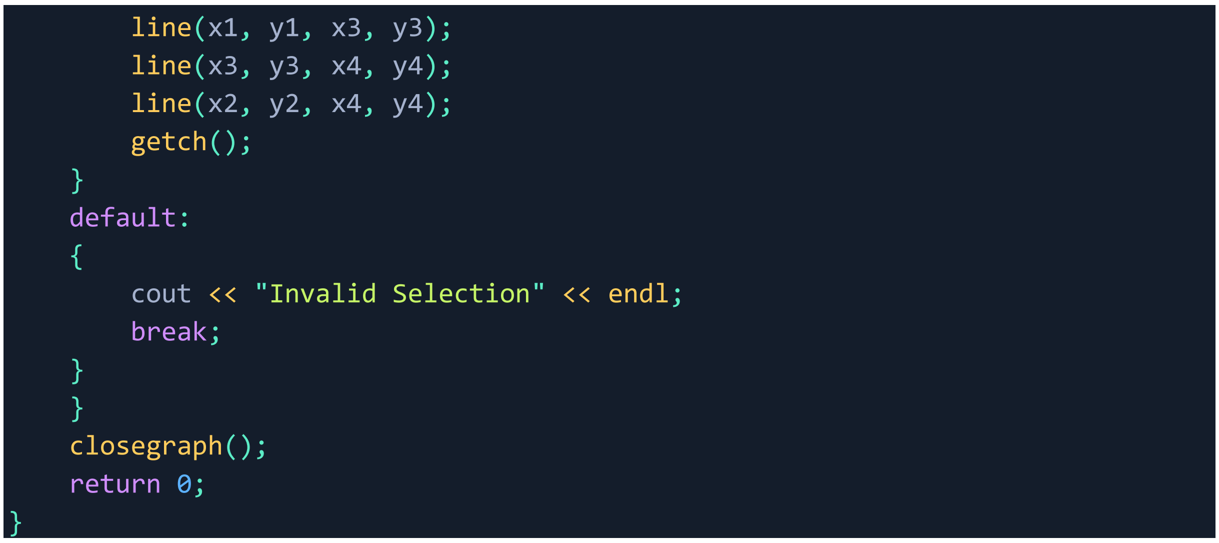
```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q6
1.Translation
2.Rotation
3.Scaling
4.Reflection
5.Shearing
Selection:5
Before shearing of rectangle
After shearing of rectangle
```



Q7. Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.

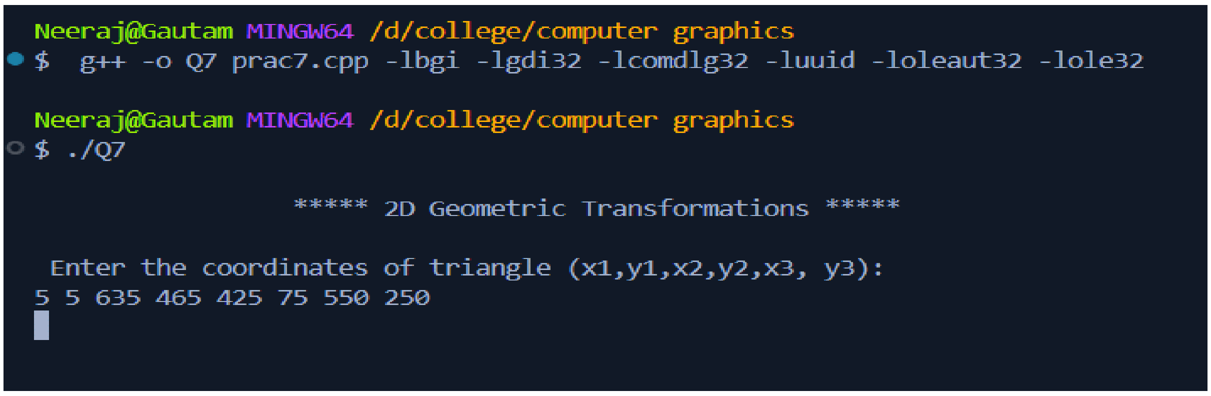
Code

```
#include <iostream>
#include <graphics.h>
#include <math.h>
using namespace std;
int main()
{
    int gd = DETECT, gm, s;
    initgraph(&gd, &gm, (char *)"");
    cout <<
"1.Translation\n2.Rotation\n3.Scaling\n4.Reflection\n5.Shearing   " << endl;
    cout << "Selection:";
    cin >> s;
    switch (s)
    {
    case 1:
    {
        int x1 = 200, y1 = 150, x2 = 300, y2 = 250;
        int tx = 200, ty = 100;
        cout << "Rectangle before translation" << endl;
        setcolor(3);
        rectangle(x1, y1, x2, y2);
        setcolor(10);
        cout << "Rectangle after translation" << endl;
        rectangle(x1 + tx, y1 + ty, x2 + tx, y2 + ty);
        getch();
        break;
    }
    case 2:
    {
        long x1 = 200, y1 = 200, x2 = 300, y2 = 300;
        double a;
        cout << "Rectangle with rotation" << endl;
        setcolor(1);
        rectangle(x1, y1, x2, y2);
        cout << "Angle of rotation:";
        cin >> a;
        a = (a * 3.14) / 180;
        long xr = x1 + ((x2 - x1) * cos(a) - (y2 - y1) * sin(a));
        long yr = y1 + ((x2 - x1) * sin(a) + (y2 - y1) * cos(a));
        setcolor(2);
        rectangle(x1, y1, xr, yr);
        getch();
        break;
    }
    case 3:
    {
        int x1 = 30, y1 = 30, x2 = 70, y2 = 70, y = 5, x = 2;
        cout << "Before scaling" << endl;
        setcolor(2);
        rectangle(x1, y1, x2, y2);
        cout << "After scaling" << endl;
        setcolor(10);
        rectangle(x1 * x, y1 * y, x2 * x, y2 * y);
        getch();
        break;
    }
    case 4:
    {
        int x1 = 200, y1 = 300, x2 = 500, y2 = 300, x3 = 350, y3 = 400;
        cout << "triangle before reflection" << endl;
        setcolor(3);
        line(x1, y1, x2, y2);
        line(x1, y1, x3, y3);
        line(x2, y2, x3, y3);
        cout << "triangle after reflection" << endl;
        setcolor(3);
        line(x1, -y1 + 500, x2, -y2 + 500);
        line(x1, -y1 + 500, x3, -y3 + 500);
        line(x2, -y2 + 500, x3, -y3 + 500);
        getch();
        break;
    }
    case 5:
    {
        int x1 = 400, y1 = 100, x2 = 600, y2 = 100, x3 = 400, y3 = 200, x4 =
600, y4 = 200, shx = -1;
        cout << "Before shearing of rectangle" << endl;
        setcolor(3);
        line(x1, y1, x2, y2);
        line(x1, y1, x3, y3);
        line(x3, y3, x4, y4);
        line(x2, y2, x4, y4);
        cout << "After shearing of rectangle" << endl;
        x1 = x1 + shx * y1;
        x2 = x2 + shx * y2;
        x3 = x3 + shx * y3;
        x4 = x4 + shx * y4;
        setcolor(10);
        line(x1, y1, x2, y2);
```

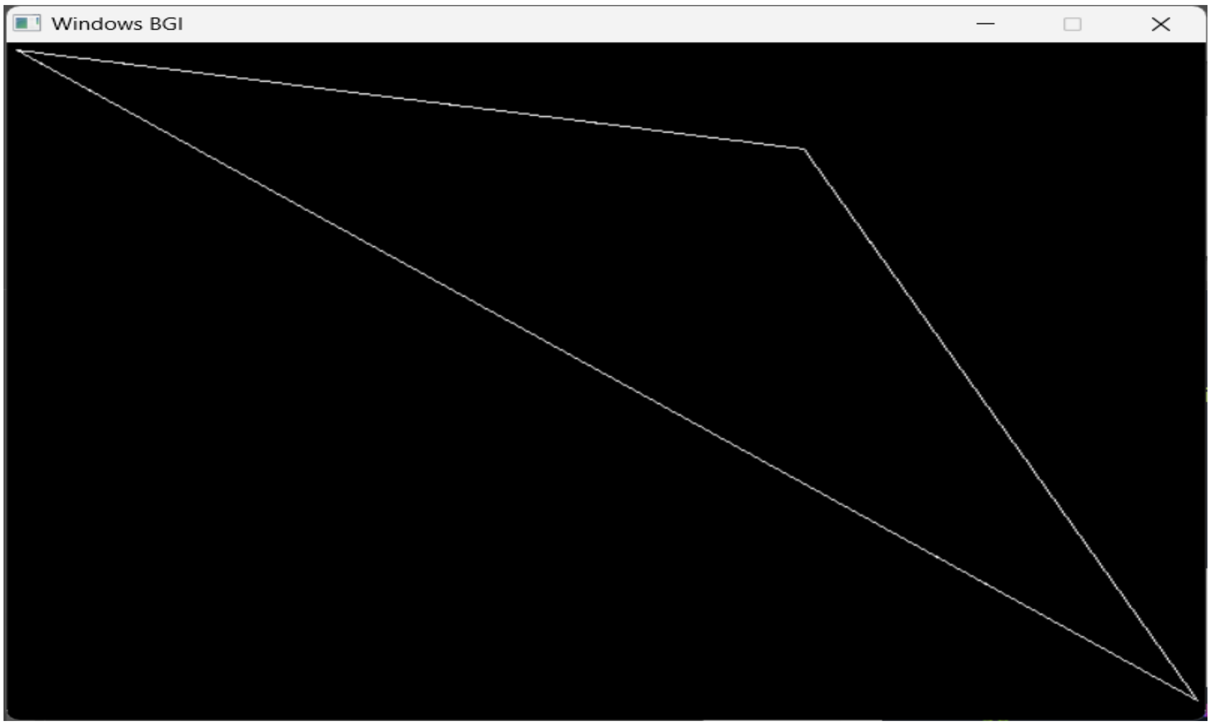


```
        line(x1, y1, x3, y3);
        line(x3, y3, x4, y4);
        line(x2, y2, x4, y4);
        getch();
    }
    default:
    {
        cout << "Invalid Selection" << endl;
        break;
    }
    }
    closegraph();
    return 0;
}
```

Output:-



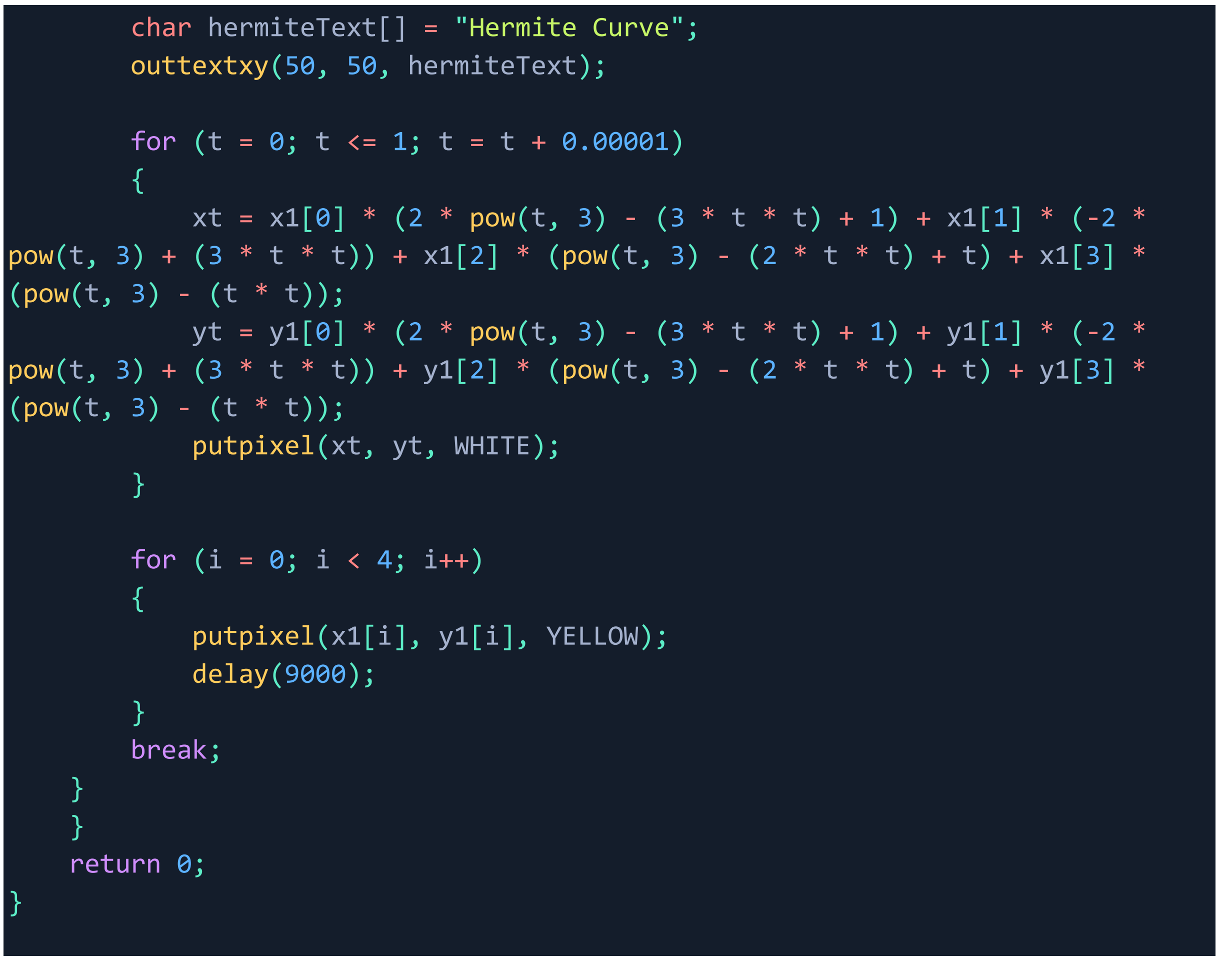
```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$  g++ -o Q7 prac7.cpp -lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32

Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q7

                  ***** 2D Geometric Transformations *****

 Enter the coordinates of triangle (x1,y1,x2,y2,x3, y3):
5 5 635 465 425 75 550 250
```



Q8. Write a program to draw Hermite /Bezier curve.

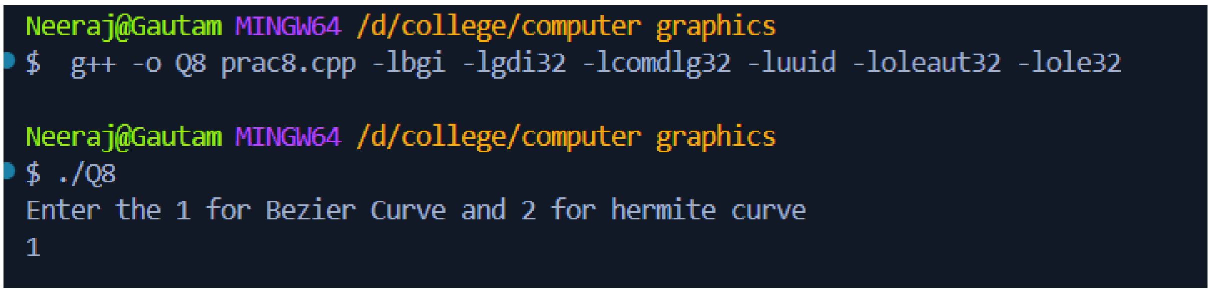
Code

```
#include <iostream>
#include <graphics.h>
#include <cmath>

using namespace std;

int main()
{
    int i;
    double t, xt, yt;
    int window1 = initwindow(800, 800);

    int ch;
    cout << "Enter the 1 for Bezier Curve and 2 for hermite curve" << endl;
    cin >> ch;
    switch (ch)
    {
    case 1:
    {
        int x[4] = {400, 300, 400, 450};
        int y[4] = {400, 350, 275, 300};
        char bezierText[] = "Bezier Curve";
        outtextxy(50, 50, bezierText);

        for (t = 0; t <= 1; t = t + 0.0005)
        {
            xt = pow(1 - t, 3) * x[0] + 3 * t * pow(1 - t, 2) * x[1] + 3 *
pow(t, 2) * (1 - t) * x[2] + pow(t, 3) * x[3];
            yt = pow(1 - t, 3) * y[0] + 3 * t * pow(1 - t, 2) * y[1] + 3 *
pow(t, 2) * (1 - t) * y[2] + pow(t, 3) * y[3];
            putpixel(xt, yt, WHITE);
        }

        for (i = 0; i < 4; i++)
        {
            putpixel(x[i], y[i], YELLOW);
            delay(4000);
        }
        break;
    }
    case 2:
    {
        int x1[4] = {200, 100, 200, 250};
        int y1[4] = {200, 150, 75, 100};
```

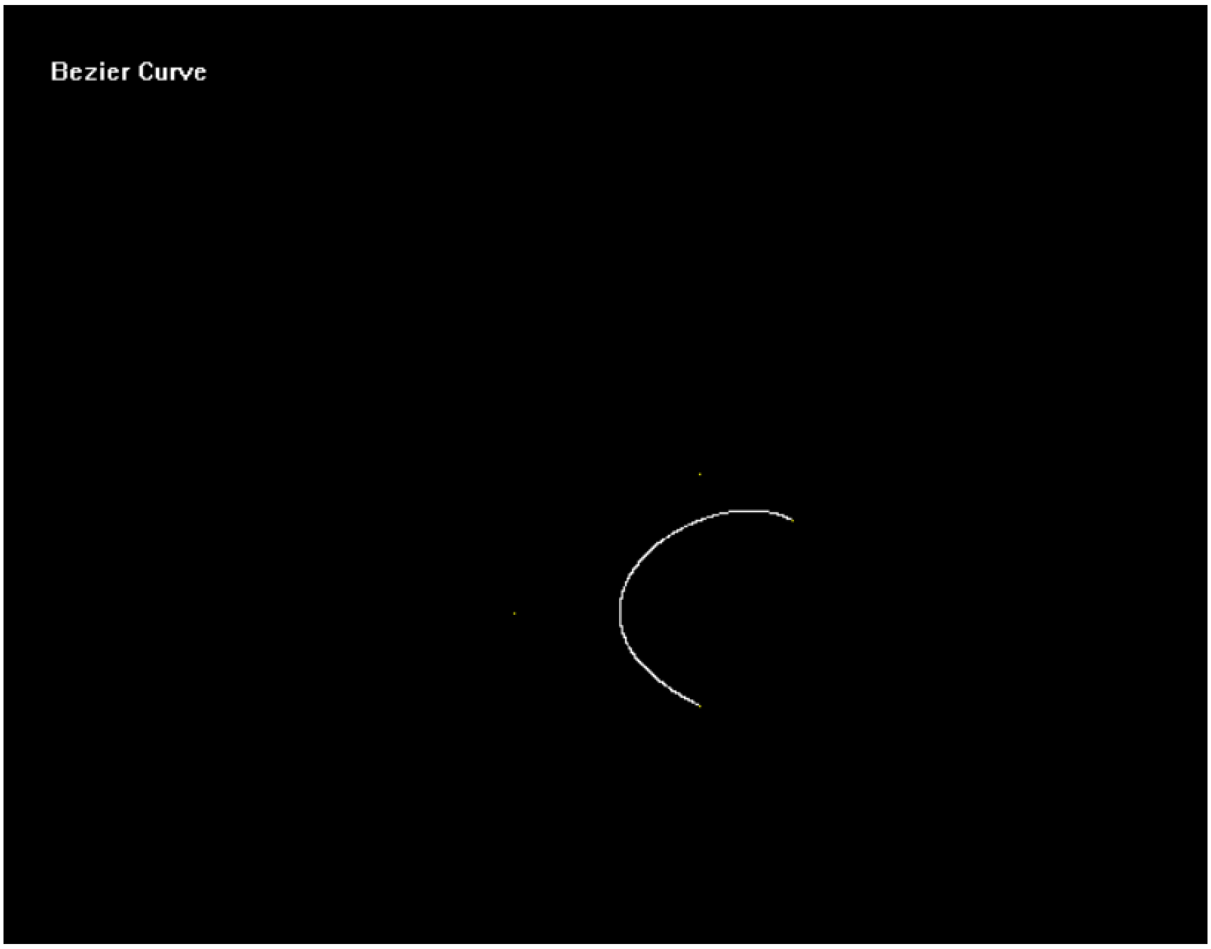


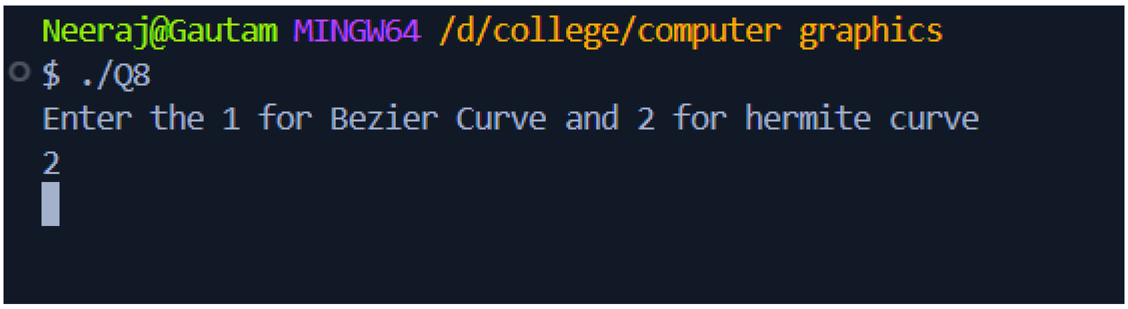
```
        char hermiteText[] = "Hermite Curve";
        outtextxy(50, 50, hermiteText);

        for (t = 0; t <= 1; t = t + 0.00001)
        {
            xt = x1[0] * (2 * pow(t, 3) - (3 * t * t) + 1) + x1[1] * (-2 *
pow(t, 3) + (3 * t * t)) + x1[2] * (pow(t, 3) - (2 * t * t) + t) + x1[3] *
(pow(t, 3) - (t * t));
            yt = y1[0] * (2 * pow(t, 3) - (3 * t * t) + 1) + y1[1] * (-2 *
pow(t, 3) + (3 * t * t)) + y1[2] * (pow(t, 3) - (2 * t * t) + t) + y1[3] *
(pow(t, 3) - (t * t));
            putpixel(xt, yt, WHITE);
        }

        for (i = 0; i < 4; i++)
        {
            putpixel(x1[i], y1[i], YELLOW);
            delay(9000);
        }
        break;
    }
    }
    return 0;
}
```

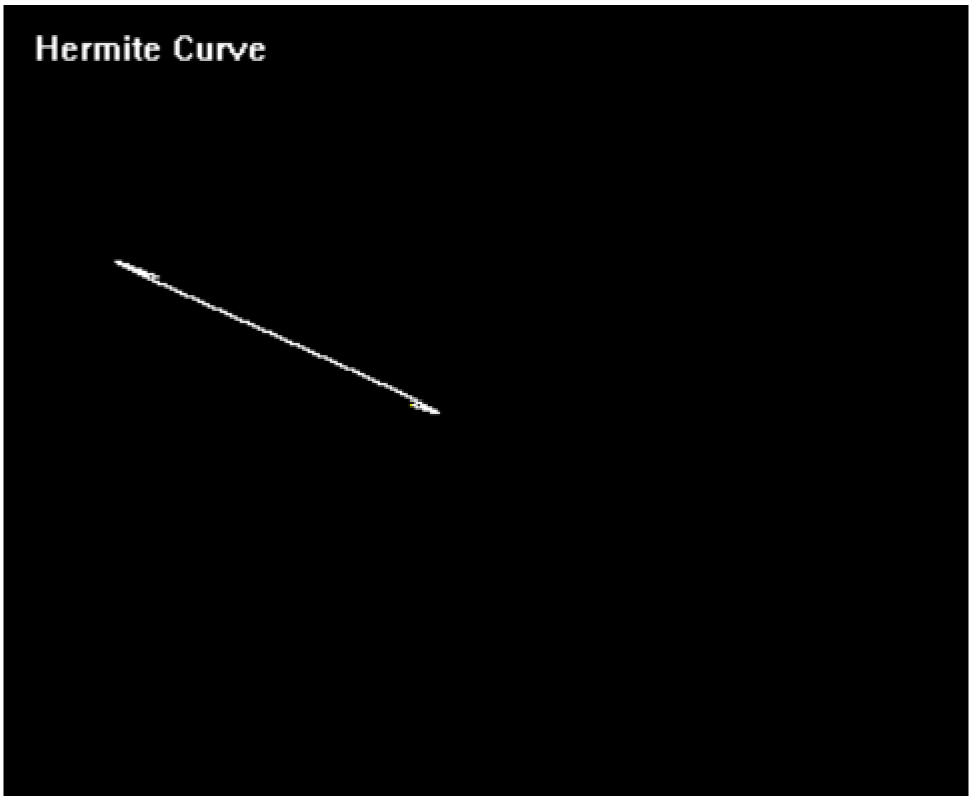
Output:-



```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$  g++ -o Q8 prac8.cpp -lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32

Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q8
Enter the 1 for Bezier Curve and 2 for hermite curve
1
```




```
Neeraj@Gautam MINGW64 /d/college/computer graphics
$ ./Q8
Enter the 1 for Bezier Curve and 2 for hermite curve
2
```


Hermite Curve