Ram Lal Anand College

**(University of Delhi)**

**Department of Computer Science**

**Practical File**

**Session- Jan-May 2023**

Name of Program/Course: - B.Sc. (H) Computer Science Semester: - VI

Name of the Paper :- Computer Graphics Paper Code: - 32341602

Name of the Student: HARSH JAISWAL Examination Roll No: - 20058570011

Submitted To – Ms. Sakshi Taaresh Khanna

**1. Write a program to implement DDA and Bresenham’s line drawing algorithm.**

**DDA.cpp**

#include <graphics.h>

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <iostream>

using namespace std;

int main()

{

    int gd = DETECT, gm;

    int x1, y1, x2, y2, dx, dy, steps, k;

    float x, y, xinc, yinc;

    initgraph(&gd, &gm, "C:\\MinGW\\lib\\libbgi.a");

    cout << "Enter the value of x1 and y1: ";

    cin >> x1 >> y1;

    cout << "Enter the value of x2 and y2: ";

    cin >> x2 >> y2;

    dx = x2 - x1;

    dy = y2 - y1;

    if (abs(dx) > abs(dy))

    {

        steps = abs(dx);

    }

    else

    {

        steps = abs(dy);

    }

    xinc = dx / (float)steps;

    yinc = dy / (float)steps;

    x = x1;

    y = y1;

    putpixel(x, y, 15);

    for (k = 0; k < steps; k++)

    {

        x += xinc;

        y += yinc;

        putpixel(x, y, RED);

        delay(100);

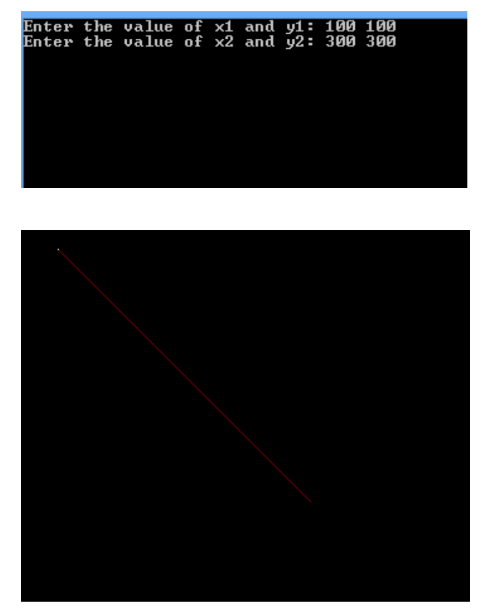
    }

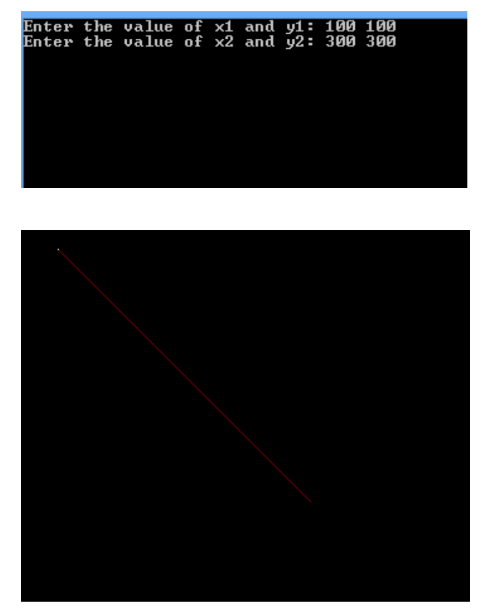
    getch();

    closegraph();

    return 0;

}





**Breshnham.cpp:-**

#include <graphics.h>

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <iostream>

using namespace std;

int main()

{

int gd = DETECT, gm;

int x1, y1, x2, y2, dx, dy, p, x, y;

initgraph(&gd, &gm, "C:\\MinGW\\lib\\libbgi.a");

cout << "Enter the value of x1 and y1: ";

cin >> x1 >> y1;

cout << "Enter the value of x2 and y2: ";

cin >> x2 >> y2;

dx = x2 - x1;

dy = y2 - y1;

p = 2 \* dy - dx;

x = x1;

y = y1;

putpixel(x, y, YELLOW);

while (x < x2)

{

if (p < 0)

{

p = p + 2 \* dy;

}

else

{

p = p + 2 \* (dy - dx);

y++;

}

x++;

putpixel(x, y, YELLOW);

delay(100);

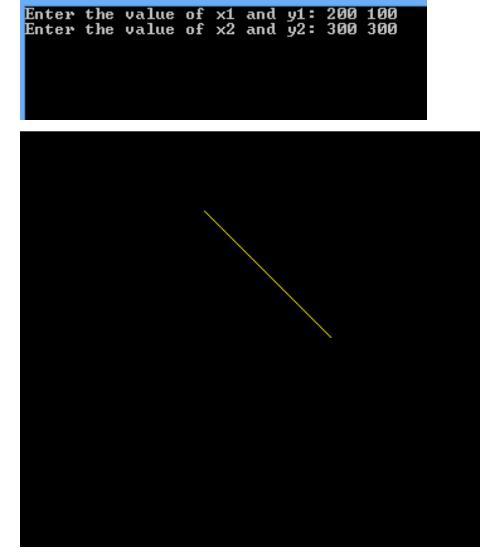
}

getch();

closegraph();

return 0;

}

****

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

#include <graphics.h>

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <iostream>

using namespace std;

void Bcircle(int r)

{

    int x, y, xc = getmaxx() / 2, yc = getmaxy() / 2, p;

    x = 0;

    y = r;

    p = 3 - (2 \* r);

    for (x = 0; x <= y; x++)

    {

        putpixel(xc + x, yc - y, YELLOW);

        if (p < 0)

        {

            y = y;

            p = (p + (4 \* x) + 1);

        }

        else

        {

            y = y - 1;

            p = (p + (4 \* (x - y) + 1));

        }

        putpixel(xc + x, yc - y, YELLOW);

        putpixel(xc - x, yc - y, YELLOW);

        putpixel(xc + x, yc + y, YELLOW);

        putpixel(xc - x, yc + y, YELLOW);

        putpixel(xc + y, yc - x, YELLOW);

        putpixel(xc - y, yc - x, YELLOW);

        putpixel(xc + y, yc + x, YELLOW);

        putpixel(xc - y, yc + x, YELLOW);

    }

}

int main()

{

    int gd = DETECT, gm;

    int r;

    initgraph(&gd, &gm, "C:\\MinGW\\lib\\libbgi.a");

    cout << "Enter the radius of circle: ";

    cin >> r;

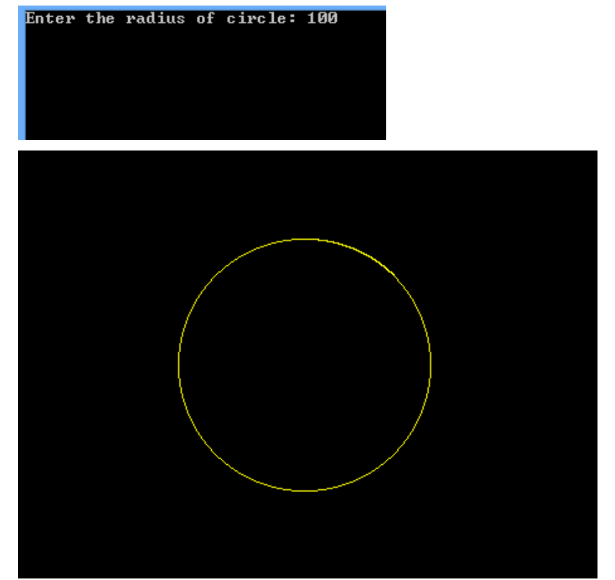
    Bcircle(r);

    getch();

    closegraph();

    return 0;

}

****

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

#include <iostream>

#include <graphics.h>

#include <math.h>

using namespace std;

void Window(int x1, int y1, int x2, int y2)

{

    rectangle(x1, y1, x2, y2);

}

void Code(int x, int y, int x1, int y1, int x2, int y2, int \*c)

{

    if (y > y2)

        c[0] = 1;

    else

        c[0] = 0;

    if (y < y1)

        c[1] = 1;

    else

        c[1] = 0;

    if (x > x2)

        c[2] = 1;

    else

        c[2] = 0;

    if (x < x1)

        c[3] = 1;

    else

        c[3] = 0;

}

void Cohen(int x1, int y1, int x2, int y2, int xwmin, int ywmin, int xwmax, int ywmax)

{

    int c1[4], c2[4], i;

    int accept = 0, done = 0;

    Code(x1, y1, xwmin, ywmin, xwmax, ywmax, c1);

    Code(x2, y2, xwmin, ywmin, xwmax, ywmax, c2);

    do

    {

        if ((c1[0] == 0 && c1[1] == 0 && c1[2] == 0 && c1[3] == 0) &&

            (c2[0] == 0 && c2[1] == 0 && c2[2] == 0 && c2[3] == 0))

        {

            accept = 1;

            done = 1;

        }

        else if ((c1[0] & c2[0]) || (c1[1] & c2[1]) || (c1[2] & c2[2]) ||

                 (c1[3] & c2[3]))

        {

            done = 1;

        }

        else

        {

            float m = (float)(y2 - y1) / (x2 - x1);

            float xi, yi;

            for (i = 0; i < 4; i++)

            {

                if (c1[i])

                {

                    switch (i)

                    {

                    case 0:

                        xi = x1 + (x2 - x1) \* (ywmax - y1) / (y2 - y1);

                        yi = ywmax;

                        break;

                    case 1:

                        xi = x1 + (x2 - x1) \* (ywmin - y1) / (y2 - y1);

                        yi = ywmin;

                        break;

                    case 2:

                        yi = y1 + (y2 - y1) \* (xwmax - x1) / (x2 - x1);

                        xi = xwmax;

                        break;

                    case 3:

                        yi = y1 + (y2 - y1) \* (xwmin - x1) / (x2 - x1);

                        xi = xwmin;

                        break;

                    }

                    x1 = xi;

                    y1 = yi;

                    Code(x1, y1, xwmin, ywmin, xwmax, ywmax, c1);

                }

                else if (c2[i])

                {

                    switch (i)

                    {

                    case 0:

                        xi = x1 + (x2 - x1) \* (ywmax - y1) / (y2 - y1);

                        yi = ywmax;

                        break;

                    case 1:

                        xi = x1 + (x2 - x1) \* (ywmin - y1) / (y2 - y1);

                        yi = ywmin;

                        break;

                    case 2:

                        yi = y1 + (y2 - y1) \* (xwmax - x1) / (x2 - x1);

                        xi = xwmax;

                        break;

                    case 3:

                        yi = y1 + (y2 - y1) \* (xwmin - x1) / (x2 - x1);

                        xi = xwmin;

                        break;

                    }

                    x2 = xi;

                    y2 = yi;

                    Code(x2, y2, xwmin, ywmin, xwmax, ywmax, c2);

                }

            }

        }

    } while (done == 0);

    if (accept)

    {

        setcolor(RED);

        line(x1, y1, x2, y2);

    }

}

int main()

{

    int gd = DETECT, gm;

    initgraph(&gd, &gm, "C:\\MinGW\\lib\\libbgi.a");

    int x1, y1, x2, y2, xwmin, ywmin, xwmax, ywmax;

    cout << "Enter the coordinates of the line: ";

    cin >> x1 >> y1 >> x2 >> y2;

    cout << "Enter the coordinates of the window: ";

    cin >> xwmin >> ywmin >> xwmax >> ywmax;

    setcolor(GREEN);

    cout << "Before clipping: ";

    Window(xwmin, ywmin, xwmax, ywmax);

    line(x1, y1, x2, y2);

    delay(8000);

    cleardevice();

    cout << "After clipping: ";

    Window(xwmin, ywmin, xwmax, ywmax);

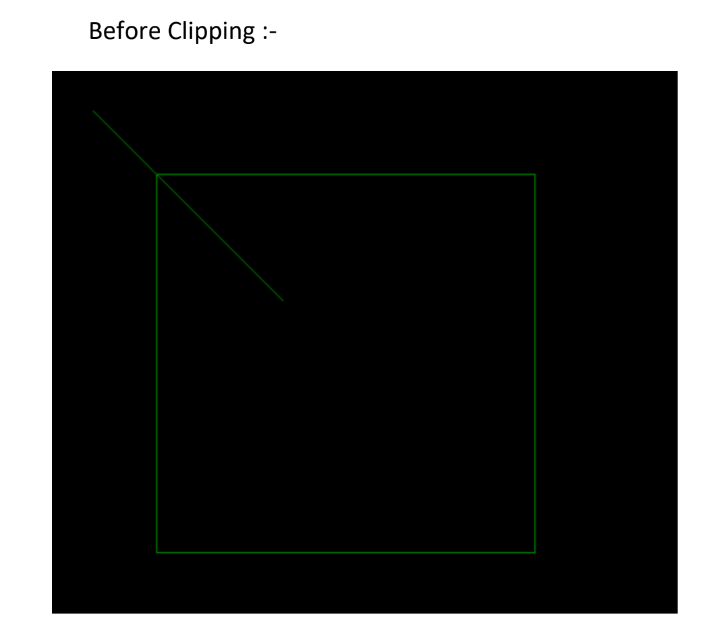
    Cohen(x1, y1, x2, y2, xwmin, ywmin, xwmax, ywmax);

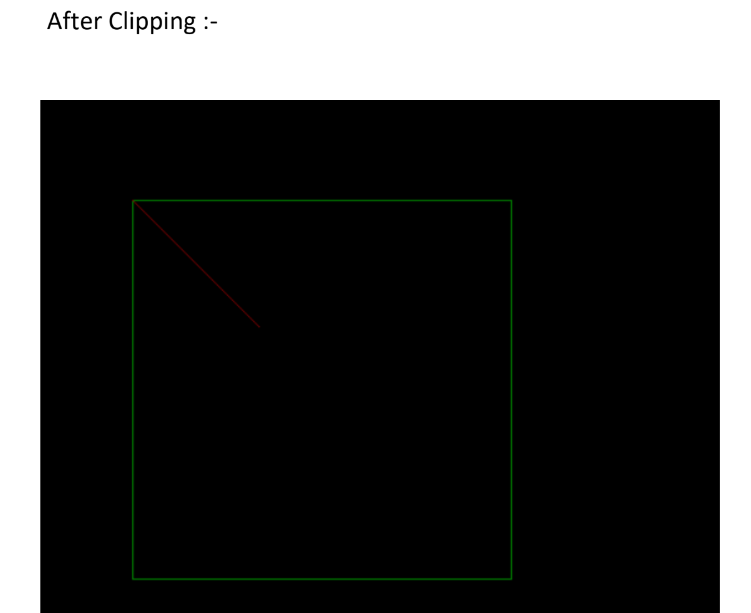
    getch();

    closegraph();

    return 0;

}





**4. Write a program to clip a polygon using Sutherland Hodgeman algorithm.**

#include <iostream>

#include <graphics.h>

using namespace std;

int xmin, xmax, ymin, ymax;

class Coordinate

{

public:

    int X;

    int Y;

    Coordinate() {}

    Coordinate(float x, float y)

    {

        X = x;

        Y = y;

    }

};

Coordinate pointOfIntersection(Coordinate s1, Coordinate e1, Coordinate s2, Coordinate e2)

{

    float a1 = e1.Y - s1.Y;

    float b1 = s1.X - e1.X;

    float c1 = a1 \* s1.X + b1 \* s1.Y;

    float a2 = e2.Y - s2.Y;

    float b2 = s2.X - e2.X;

    float c2 = a2 \* s2.X + b2 \* s2.Y;

    float delta = a1 \* b2 - a2 \* b1;

    return Coordinate((b2 \* c1 - b1 \* c2) / delta, (a1 \* c2 - a2 \* c1) / delta);

}

int main()

{

    int gd = DETECT, gm;

    initgraph(&gd, &gm, "C:\\MinGW\\lib\\libbgi.a");

    int i, j, choice = 1;

    cout << "Enter co-ordinates for window ABCD:\n\n";

    cout << "xmin, xmax, ymin, ymax: ";

    cin >> xmin >> xmax >> ymin >> ymax;

    rectangle(xmin, ymin, xmax, ymax);

    do

    {

        int n;

        cout << "\n\nEnter number of vertices (max 10): ";

        cin >> n;

        Coordinate V[n];

        cout << "Enter co-ordinates of vertices\n";

        int drawPolygon[n \* 2], newPolygon[100];

        for (i = 0, j = 0; i < n; i++)

        {

            cout << "Enter x" << i + 1 << " and y" << i + 1 << ": ";

            cin >> V[i].X >> V[i].Y;

            drawPolygon[j++] = V[i].X;

            drawPolygon[j++] = V[i].Y;

        }

        drawPolygon[j++] = V[0].X;

        drawPolygon[j] = V[0].Y;

        setcolor(MAGENTA);

        drawpoly(n + 1, drawPolygon);

        Coordinate top\_left(xmin, ymin), top\_right(xmax, ymin),

            bot\_left(xmin, ymax), bot\_right(xmax, ymax);

        int newN;

        cout << endl;

        for (i = 0, j = 0, newN = 0; i < n; i++)

        {

            Coordinate temp;

            if (V[i].X < xmin)

            {

                if (V[(i + 1) % n].X < xmin)

                    cout << "\nOut->Out";

                else

                {

                    temp = pointOfIntersection(V[i], V[(i + 1) % n], top\_left, bot\_left);

                    newN++;

                    newPolygon[j++] = temp.X;

                    newPolygon[j++] = temp.Y;

                    newN++;

                    newPolygon[j++] = V[(i + 1) % n].X;

                    newPolygon[j++] = V[(i + 1) % n].Y;

                    cout << "\nOut->In";

                }

            }

            else

            {

                if (V[(i + 1) % n].X < xmin)

                {

                    temp = pointOfIntersection(V[i], V[(i + 1) % n], top\_left,bot\_left);

                    newN++;

                    newPolygon[j++] = temp.X;

                    newPolygon[j++] = temp.Y;

                    cout << "\nIn->Out";

                }

                else

                {

                    newN++;

                    newPolygon[j++] = V[(i + 1) % n].X;

                    newPolygon[j++] = V[(i + 1) % n].Y;

                    cout << "\nIn->In";

                }

            }

        }

        newPolygon[j++] = newPolygon[0];

        newPolygon[j] = newPolygon[1];

        setcolor(RED);

        cout << "\nRed coordinates: ";

        for (i = 0; i <= j; i++)

            cout << newPolygon[i] << " ";

        drawpoly(newN + 1, newPolygon);

        for (i = 0, j = 0; i < newN; i++)

        {

            V[i].X = newPolygon[j++];

            V[i].Y = newPolygon[j++];

        }

        n = newN;

        cout << endl;

        for (i = 0, j = 0, newN = 0; i < n; i++)

        {

            Coordinate temp;

            if (V[i].X > xmax)

            {

                if (V[(i + 1) % n].X > xmax)

                    cout << "\nOut->Out";

                else

                {

                    temp = pointOfIntersection(V[i], V[(i + 1) % n], top\_right,bot\_right);

                    newN++;

                    newPolygon[j++] = temp.X;

                    newPolygon[j++] = temp.Y;

                    newN++;

                    newPolygon[j++] = V[(i + 1) % n].X;

                    newPolygon[j++] = V[(i + 1) % n].Y;

                    cout << "\nOut->In";

                }

            }

            else

            {

                if (V[(i + 1) % n].X > xmax)

                {

                    temp = pointOfIntersection(V[i], V[(i + 1) % n], top\_right,bot\_right);

                    newN++;

                    newPolygon[j++] = temp.X;

                    newPolygon[j++] = temp.Y;

                    cout << "\nIn->Out";

                }

                else

                {

                    newN++;

                    newPolygon[j++] = V[(i + 1) % n].X;

                    newPolygon[j++] = V[(i + 1) % n].Y;

                    cout << "\nIn->In";

                }

            }

        }

        newPolygon[j++] = newPolygon[0];

        newPolygon[j] = newPolygon[1];

        setcolor(GREEN);

        cout << "\nGreen coordinates: ";

        for (i = 0; i <= j; i++)

            cout << newPolygon[i] << " ";

        drawpoly(newN + 1, newPolygon);

        for (i = 0, j = 0; i < newN; i++)

        {

            V[i].X = newPolygon[j++];

            V[i].Y = newPolygon[j++];

        }

        n = newN;

        cout << endl;

        for (i = 0, j = 0, newN = 0; i < n; i++)

        {

            Coordinate temp;

            if (V[i].Y > ymax)

            {

                if (V[(i + 1) % n].Y > ymax)

                    cout << "\nOut->Out";

                else

                {

                    temp = pointOfIntersection(V[i], V[(i + 1) % n], bot\_left,bot\_right);

                    newN++;

                    newPolygon[j++] = temp.X;

                    newPolygon[j++] = temp.Y;

                    newN++;

                    newPolygon[j++] = V[(i + 1) % n].X;

                    newPolygon[j++] = V[(i + 1) % n].Y;

                    cout << "\nOut->In";

                }

            }

            else

            {

                if (V[(i + 1) % n].Y > ymax)

                {

                    temp = pointOfIntersection(V[i], V[(i + 1) % n], bot\_left, bot\_right);

                    newN++;

                    newPolygon[j++] = temp.X;

                    newPolygon[j++] = temp.Y;

                    cout << "\nIn->Out";

                }

                else

                {

                    newN++;

                    newPolygon[j++] = V[(i + 1) % n].X;

                    newPolygon[j++] = V[(i + 1) % n].Y;

                    cout << "\nIn->In";

                }

            }

        }

        newPolygon[j++] = newPolygon[0];

        newPolygon[j] = newPolygon[1];

        setcolor(BLUE);

        cout << "\nBlue coordinates: ";

        for (i = 0; i <= j; i++)

            cout << newPolygon[i] << " ";

        drawpoly(newN + 1, newPolygon);

        for (i = 0, j = 0; i < newN; i++)

        {

            V[i].X = newPolygon[j++];

            V[i].Y = newPolygon[j++];

        }

        n = newN;

        cout << endl;

        for (i = 0, j = 0, newN = 0; i < n; i++)

        {

            Coordinate temp;

            if (V[i].Y < ymin)

            {

                if (V[(i + 1) % n].Y < ymin)

                    cout << "\nOut->Out";

                else

                {

                    temp = pointOfIntersection(V[i], V[(i + 1) % n], top\_left,

                                               top\_right);

                    newN++;

                    newPolygon[j++] = temp.X;

                    newPolygon[j++] = temp.Y;

                    newN++;

                    newPolygon[j++] = V[(i + 1) % n].X;

                    newPolygon[j++] = V[(i + 1) % n].Y;

                    cout << "\nOut->In";

                }

            }

            else

            {

                if (V[(i + 1) % n].Y < ymin)

                {

                    temp = pointOfIntersection(V[i], V[(i + 1) % n], top\_left,

                                               top\_right);

                    newN++;

                    newPolygon[j++] = temp.X;

                    newPolygon[j++] = temp.Y;

                    cout << "\nIn->Out";

                }

                else

                {

                    newN++;

                    newPolygon[j++] = V[(i + 1) % n].X;

                    newPolygon[j++] = V[(i + 1) % n].Y;

                    cout << "\nIn->In";

                }

            }

        }

        newPolygon[j++] = newPolygon[0];

        newPolygon[j] = newPolygon[1];

        for (i = 0, j = 0; i < newN; i++)

        {

            V[i].X = newPolygon[j++];

            V[i].Y = newPolygon[j++];

        }

        n = newN;

        setcolor(YELLOW);

        drawpoly(newN + 1, newPolygon);

        cout << "\n\nContinue?\t1.Yes\t0.No:\t";

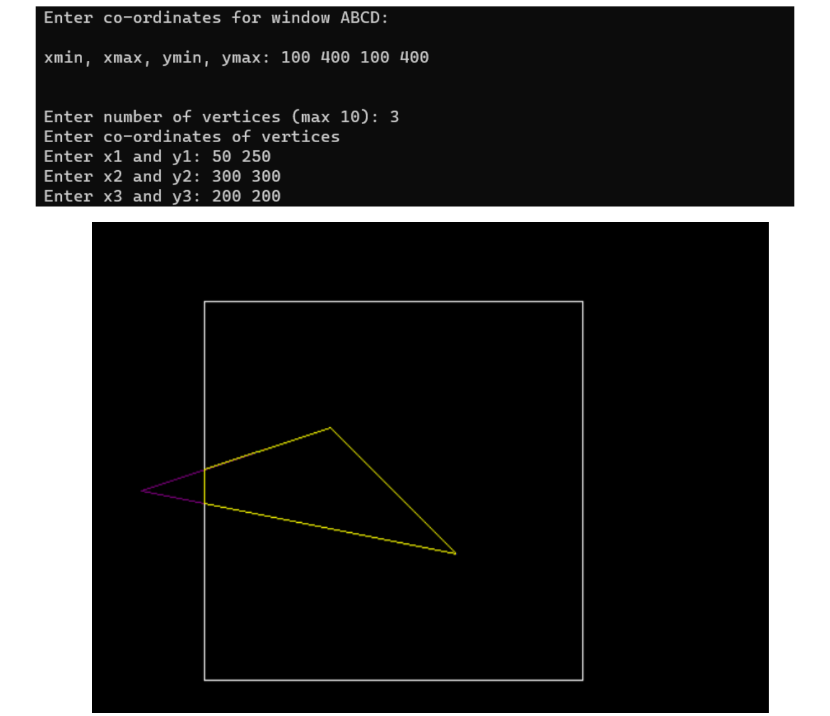
        cin >> choice;

    } while (choice != 0);

    getch();

    closegraph();

}



**5. Write a program to fill a polygon using Scan line fill algorithm.**

#include <graphics.h>

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <math.h>

using namespace std;

void polygonScanline(int n, int x[], int y[])

{

    int i, j, k, ymax = 0, ymin = 480, x1, y1, x2, y2, temp, flag = 0;

    float m[20], xi[20];

    for (i = 0; i < n; i++)

    {

        if (y[i] > ymax)

            ymax = y[i];

        if (y[i] < ymin)

            ymin = y[i];

    }

    for (i = 0; i < n; i++)

    {

        j = (i + 1) % n;

        x1 = x[i];

        y1 = y[i];

        x2 = x[j];

        y2 = y[j];

        if (y2 != y1)

            m[i] = (float)(x2 - x1) / (y2 - y1);

        else

            m[i] = 0;

    }

    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)))

            {

                xi[k] = x[j] + m[j] \* (i - y[j]);

                k++;

            }

        }

        for (j = 0; j < k - 1; j++)

        {

            for (int l = 0; l < k - j - 1; l++)

            {

                if (xi[l] > xi[l + 1])

                {

                    temp = xi[l];

                    xi[l] = xi[l + 1];

                    xi[l + 1] = temp;

                }

            }

        }

        setcolor(RED);

        for (j = 0; j < k; j += 2)

        {

            line(xi[j], i, xi[j + 1] + 1, i);

        }

    }

}

int main()

{

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    initgraph(&gd, &gm, data);

    int n, x[20], y[20];

    cout << "---Polygon Scanline---" << endl;

    cout << "Enter number of vertices: ";

    cin >> n;

    for (int i = 0; i < n; i++)

    {

        cout << "Enter Point " << i + 1 << " (x, y): ";

        cin >> x[i] >> y[i];

    }

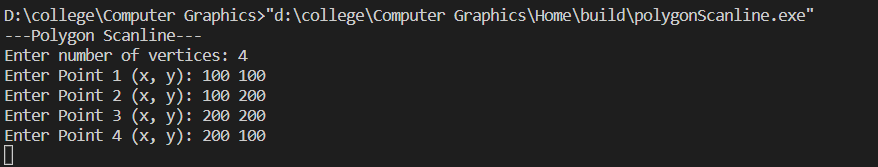
    polygonScanline(n, x, y);

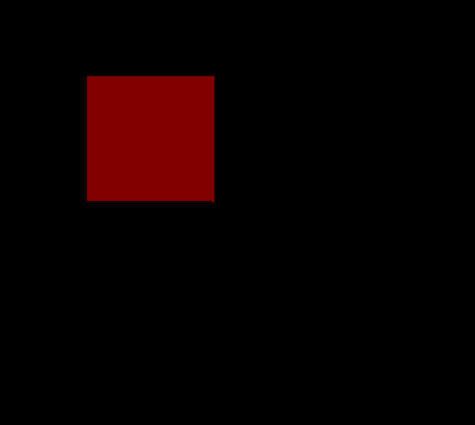
    getch();

    closegraph();

    return 0;

}





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

#include <graphics.h>

#include <iostream>

#include <stdlib.h>

#include <conio.h>

#include <math.h>

using namespace std;

void draw\_Polygon(int n, int \*drawPolygon)

{

    setcolor(MAGENTA);

    drawpoly(n + 1, drawPolygon);

}

void drawLine(int x1, int y1, int x2, int y2)

{

    setcolor(GREEN);

    line(x1, y1, x2, y2);

}

void translate(int n, int \*drawPolygon, int tx, int ty)

{

    int i, j;

    int newPolygon[100];

    int T[3][3] = {{1, 0, tx}, {0, 1, ty}, {0, 0, 1}};

    for (i = 0, j = 0; i < n; i++)

    {

        int v[3] = {drawPolygon[i \* 2], drawPolygon[i \* 2 + 1], 1};

        int v\_out[3] = {0, 0, 0};

        for (int k = 0; k < 3; k++)

        {

            for (int l = 0; l < 3; l++)

            {

                v\_out[k] += T[k][l] \* v[l];

            }

        }

        newPolygon[j++] = v\_out[0];

        newPolygon[j++] = v\_out[1];

    }

    newPolygon[j++] = newPolygon[0];

    newPolygon[j] = newPolygon[1];

    setcolor(RED);

    drawpoly(n + 1, newPolygon);

}

void scale(int n, int \*drawPolygon, int sx, int sy)

{

    int i, j;

    int newPolygon[100];

    for (i = 0, j = 0; i < n; i++)

    {

        newPolygon[j++] = drawPolygon[i \* 2] \* sx;

        newPolygon[j++] = drawPolygon[i \* 2 + 1] \* sy;

    }

    newPolygon[j++] = newPolygon[0];

    newPolygon[j] = newPolygon[1];

    int tfx = (-sx + 1) \* drawPolygon[0], tfy = (-sy + 1) \* drawPolygon[1];

    translate(n, newPolygon, tfx, tfy);

}

void rotate(int n, int \*drawPolygon, int angle)

{

    int i, j;

    int newPolygon[100];

    for (i = 0, j = 0; i < n; i++)

    {

        newPolygon[j++] = drawPolygon[i \* 2] \* cos(angle \* 3.14 / 180) - drawPolygon[i \* 2 + 1] \* sin(angle \* 3.14 / 180);

        newPolygon[j++] = drawPolygon[i \* 2] \* sin(angle \* 3.14 / 180) + drawPolygon[i \* 2 + 1] \* cos(angle \* 3.14 / 180);

    }

    newPolygon[j++] = newPolygon[0];

    newPolygon[j] = newPolygon[1];

    int tfx = drawPolygon[0] - newPolygon[0];

    int tfy = drawPolygon[1] - newPolygon[1];

    translate(n, newPolygon, tfx, tfy);

}

void shear(int n, int \*drawPolygon, int shx, int shy)

{

    int i, j;

    int newPolygon[100];

    for (i = 0, j = 0; i < n; i++)

    {

        newPolygon[j++] = drawPolygon[i \* 2] + shx \* drawPolygon[i \* 2 + 1];

        newPolygon[j++] = drawPolygon[i \* 2 + 1] + shy \* drawPolygon[i \* 2];

    }

    newPolygon[j++] = newPolygon[0];

    newPolygon[j] = newPolygon[1];

    setcolor(RED);

    drawpoly(n + 1, newPolygon);

}

void reflect(int n, int \*drawPolygon)

{

    int i, j;

    int newPolygon[2 \* n + 2];

    int x = getmaxx();

    for (i = 0; i < 2 \* n + 2; i++)

    {

        if (i % 2 == 0)

        {

            newPolygon[i] = x - drawPolygon[i];

        }

        else

        {

            newPolygon[i] = drawPolygon[i];

        }

    }

    setcolor(RED);

    drawpoly(n + 1, newPolygon);

}

int main()

{

    int gd = DETECT, gm;

    initgraph(&gd, &gm, "C:\\MinGW\\lib\\libbgi.a");

    int n, i, j, k, tx, ty, sx, sy, angle, shx, shy, x1, y1, x2, y2;

    int drawPolygon[100];

    cout << "Enter the number of vertices: ";

    cin >> n;

    cout << "Enter the vertices: ";

    for (i = 0, j = 0; i < n; i++)

    {

        cin >> drawPolygon[j++] >> drawPolygon[j++];

    }

    drawPolygon[j++] = drawPolygon[0];

    drawPolygon[j] = drawPolygon[1];

    draw\_Polygon(n, drawPolygon);

    do

    {

        cout << "1. Translation \t 2. Scaling \t 3. Rotation \t 4. Shearing \t 5. Reflection \t 6. Exit" << endl;

        cout << "Enter your choice: ";

        cin >> k;

        cleardevice();

        draw\_Polygon(n, drawPolygon);

        switch (k)

        {

        case 1:

            cout << "Enter the translation factor: ";

            cin >> tx >> ty;

            translate(n, drawPolygon, tx, ty);

            break;

        case 2:

            cout << "Enter the scaling factor: ";

            cin >> sx >> sy;

            scale(n, drawPolygon, sx, sy);

            break;

        case 3:

            cout << "Enter the angle of rotation: ";

            cin >> angle;

            rotate(n, drawPolygon, angle);

            break;

        case 4:

            cout << "Enter the shearing factor: ";

            cin >> shx >> shy;

            shear(n, drawPolygon, shx, shy);

            break;

        case 5:

            x1 = getmaxx() / 2;

            y1 = 0, y2 = getmaxy();

            drawLine(x1, y1, x1, y2);

            reflect(n, drawPolygon);

            break;

        case 6:

            break;

        default:

            cout << "Invalid choice" << endl;

        }

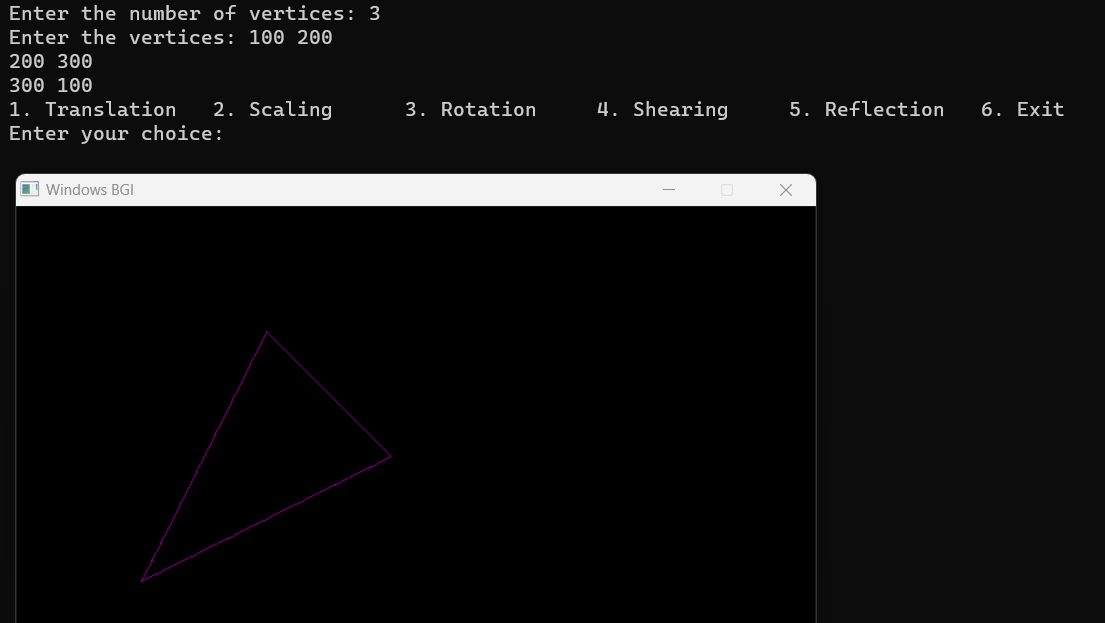
    } while (k != 6);

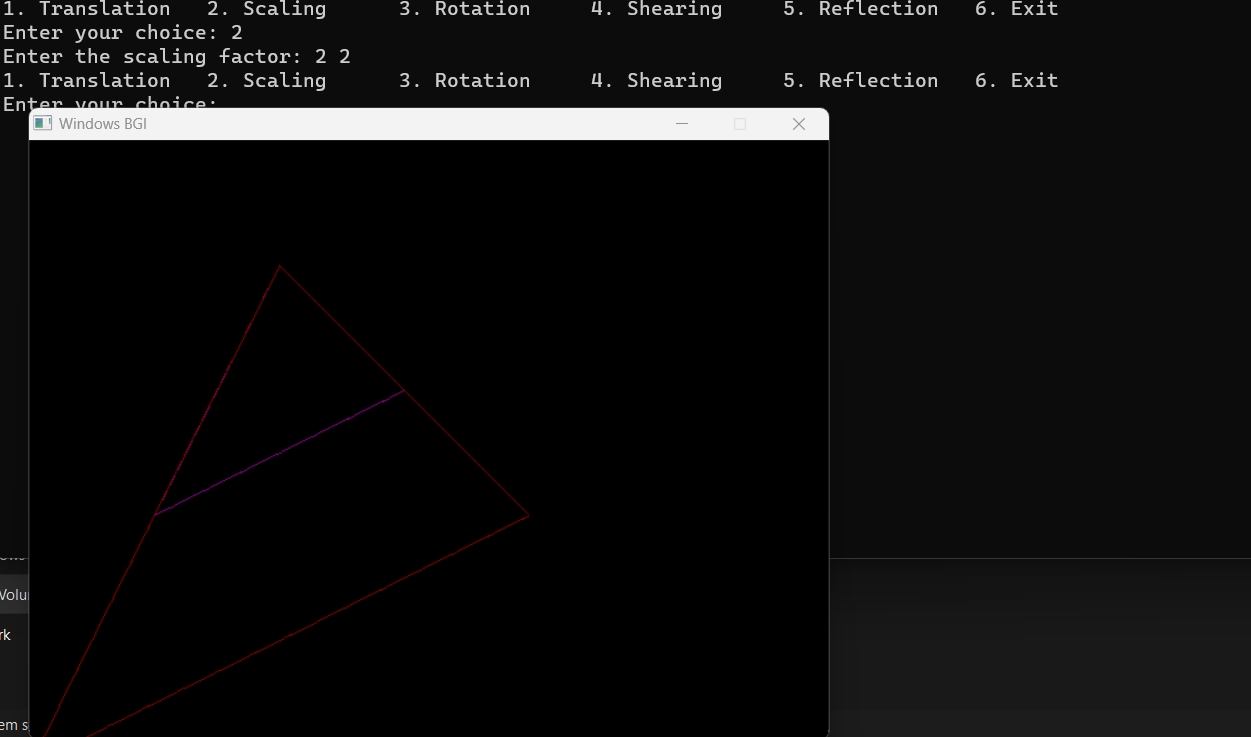
    getch();

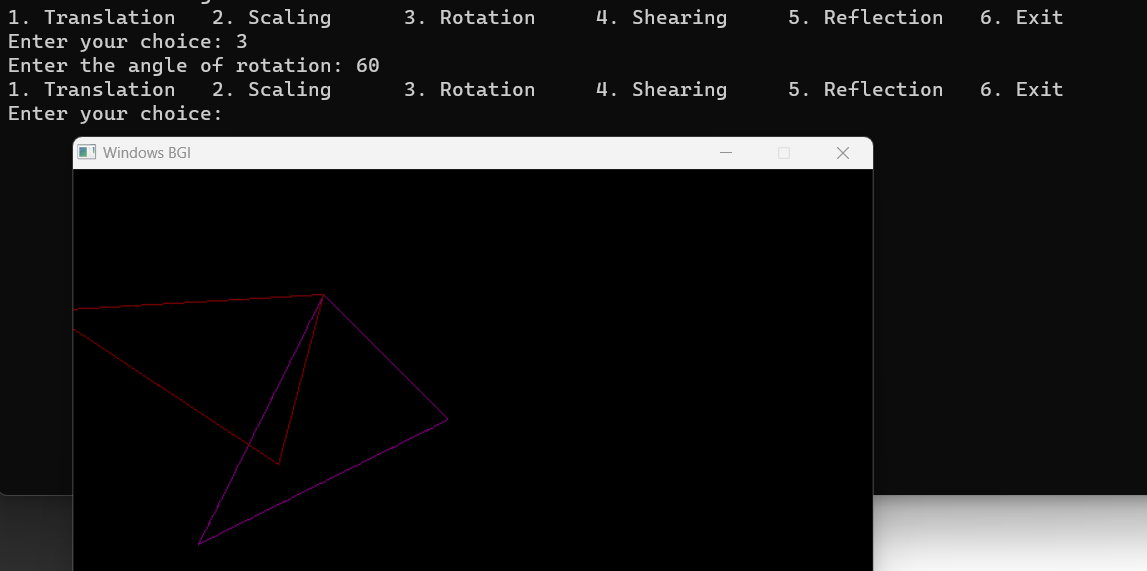
    closegraph();

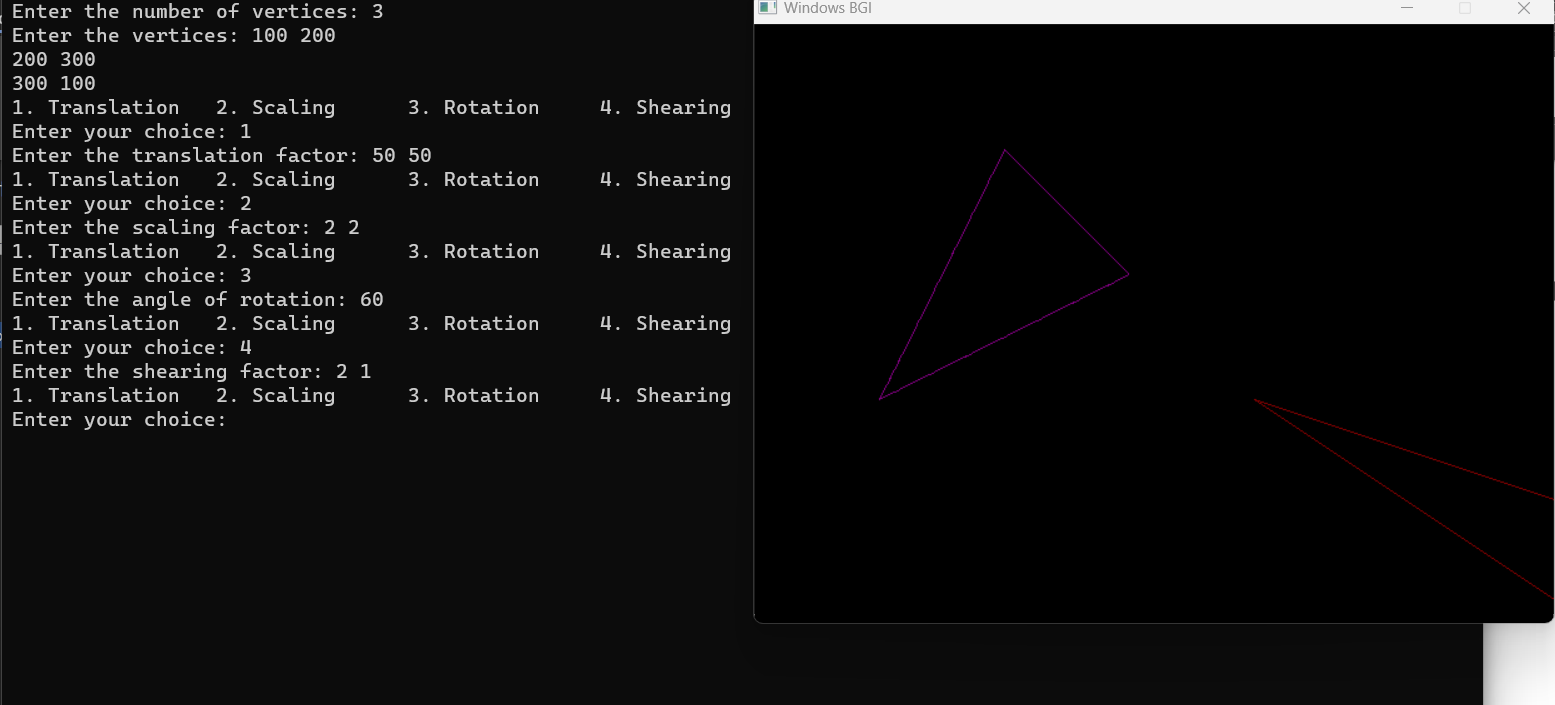
    return 0;

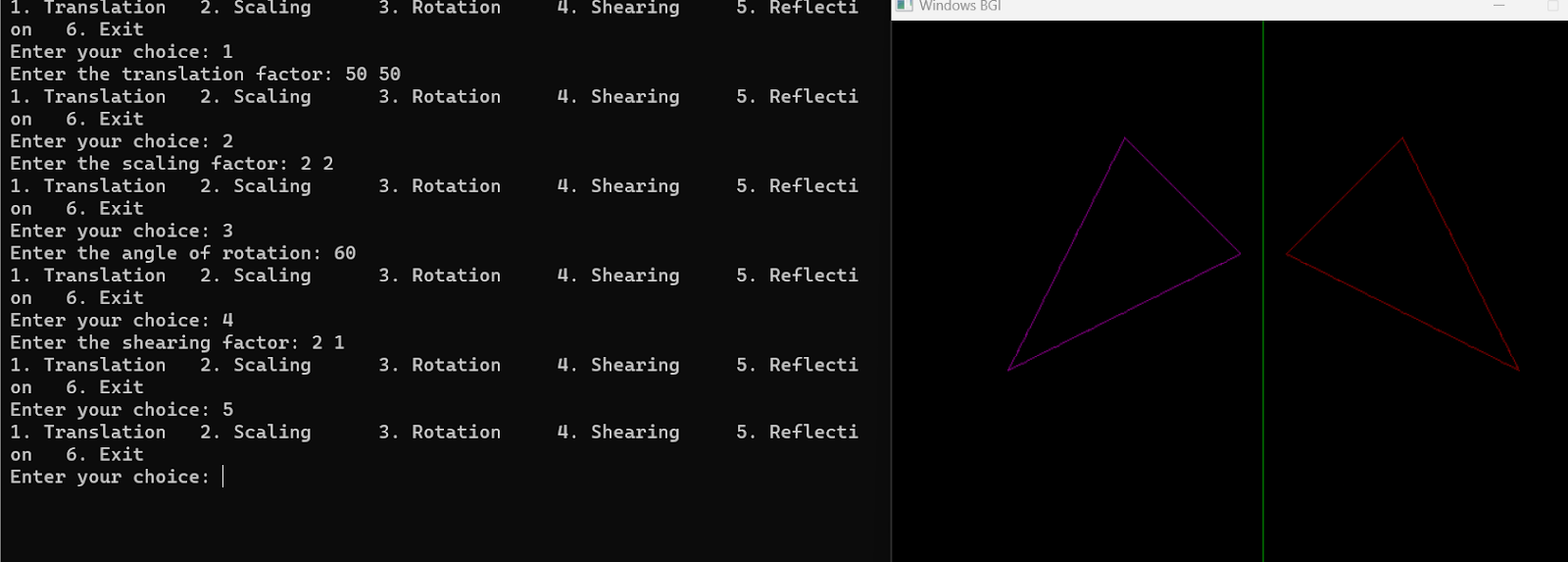
}











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

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <math.h>

#include <iostream>

#include <graphics.h>

using namespace std;

int s=1;

double x1, x2, y2;

void draw\_cube(double edge[20][4])

{

  double y1;

  int i;

  clearviewport();

  for (i = 0; i < 19; i++)

  {

    x1 = edge[i][0] + edge[i][2] \* (cos(2.3562));

    y1 = edge[i][1] - edge[i][2] \* (sin(2.3562));

    x2 = edge[i + 1][0] + edge[i + 1][2] \* (cos(2.3562));

    y2 = edge[i + 1][1] - edge[i + 1][2] \* (sin(2.3562));

    line(x1 + 320, 240 - y1, x2 + 320, 240 - y2);

  }

  line(320, 240, 320, 25);

  line(320, 240, 550, 240);

  line(320, 240, 150, 410);

}

void scale(double edge[20][4])

{

  double a, b, c;

  int i;

  cout << "Enter The Scaling Factors: ";

  cin >> a >> b >> c;

  clearviewport();

  double scaleMat[4][4] = {

    {a, 0, 0, 0},

    {0, b, 0, 0},

    {0, 0, c, 0},

    {0, 0, 0, 1}

  };

  for (i = 0; i < 20; i++)

  {

    double tempVec[4] = {edge[i][0], edge[i][1], edge[i][2], edge[i][3]};

    for(int j=0; j<4; j++){

      edge[i][j] = 0;

      for(int k=0; k<4; k++){

        edge[i][j] += tempVec[k]\*scaleMat[j][k];

      }

    }

  }

  draw\_cube(edge);

}

void translate(double edge[20][4])

{

  double Tx, Ty, Tz;

  int i;

  cout << "Enter The Translation Factors: ";

  cin >> Tx >> Ty >> Tz;

  clearviewport();

  double transMat[4][4] = {

    {1, 0, 0, Tx},

    {0, 1, 0, Ty},

    {0, 0, 1, Tz},

    {0, 0, 0, 1}

  };

  for (i = 0; i < 20; i++)

  {

    double tempVec[4] = {edge[i][0], edge[i][1], edge[i][2], 1};

    for(int j=0; j<4; j++){

      edge[i][j] = 0;

      for(int k=0; k<4; k++){

        edge[i][j] += tempVec[k]\*transMat[j][k];

      }

    }

  }

  draw\_cube(edge);

}

void matrixMult(double edge[20][4], double mat[4][4])

{

  double res[20][4];

  int i, j, k;

  for (i = 0; i < 20; i++)

  {

    for (j = 0; j < 4; j++)

    {

      res[i][j] = 0;

      for (k = 0; k < 4; k++)

      {

        res[i][j] += edge[i][k] \* mat[k][j];

      }

    }

  }

  for (i = 0; i < 20; i++)

  {

    for (j = 0; j < 4; j++)

    {

      edge[i][j] = res[i][j];

    }

  }

}

void rotateX(double edge[20][4], double angle)

{

  double c = cos(angle);

  double s = sin(angle);

  double rot\_mat[4][4] = {{1, 0, 0, 0},

                          {0, c, -s, 0},

                          {0, s, c, 0},

                          {0, 0, 0, 1}};

  matrixMult(edge, rot\_mat);

  draw\_cube(edge);

}

void rotateY(double edge[20][4], double angle)

{

  double c = cos(angle);

  double s = sin(angle);

  double rot\_mat[4][4] = {{c, 0, s, 0},

                          {0, 1, 0, 0},

                          {-s, 0, c, 0},

                          {0, 0, 0, 1}};

  matrixMult(edge, rot\_mat);

  draw\_cube(edge);

}

void rotateZ(double edge[20][4], double angle)

{

  double c = cos(angle);

  double s = sin(angle);

  double rot\_mat[4][4] = {{c, -s, 0, 0},

                          {s, c, 0, 0},

                          {0, 0, 1, 0},

                          {0, 0, 0, 1}};

  matrixMult(edge, rot\_mat);

  draw\_cube(edge);

}

void rotate(double edge[20][4])

{

  int ch;

  double angle;

  cout << "-=[ Rotation About ]=-" << endl;

  cout << "1:==> X-Axis " << endl;

  cout << "2:==> Y-Axis" << endl;

  cout << "3:==> Z-Axis " << endl;

  cout << "Enter Your Choice: ";

  cin >> ch;

  switch (ch)

  {

  case 1:

    cout << "Enter The Angle (in degrees): ";

    cin >> angle;

    angle = (angle \* M\_PI) / 180; // Convert angle to radians

    rotateX(edge, angle);

    break;

  case 2:

    cout << "Enter The Angle (in degrees): ";

    cin >> angle;

    angle = (angle \* M\_PI) / 180; // Convert angle to radians

    rotateY(edge, angle);

    break;

  case 3:

    cout << "Enter The Angle (in degrees): ";

    cin >> angle;

    angle = (angle \* M\_PI) / 180; // Convert angle to radians

    rotateZ(edge, angle);

    break;

  default:

    cout << "Invalid Choice!" << endl;

    break;

  }

}

void reflect(double edge[20][4]) {

    int ch;

    cout << "-=[ Reflection About ]=-" << endl;

    cout << "1:==> X-Axis" << endl;

    cout << "2:==> Y-Axis " << endl;

    cout << "3:==> Z-Axis " << endl;

    cout << "Enter Your Choice: ";

    cin >> ch;

    double reflect\_mat[4][4] = {{1, 0, 0, 0},

                                {0, 1, 0, 0},

                                {0, 0, 1, 0},

                                {0, 0, 0, 1}};

    switch (ch) {

        case 1:

            reflect\_mat[1][1] = -1;

            reflect\_mat[2][2] = -1;

            break;

        case 2:

            reflect\_mat[0][0] = -1;

            reflect\_mat[2][2] = -1;

            break;

        case 3:

            reflect\_mat[0][0] = -1;

            reflect\_mat[1][1] = -1;

            break;

        default:

            cout << "Invalid Choice. Reflection aborted." << endl;

            return;

    }

    matrixMult(edge, reflect\_mat);

    draw\_cube(edge);

}

void perspect(double edge[20][4]) {

  int ch;

  double p, q, r;

  cout << "-=[ Perspective Projection About ]=-" << endl;

  cout << "1:==> X-Axis " << endl;

  cout << "2:==> Y-Axis " << endl;

  cout << "3:==> Z-Axis" << endl;

  cout << "Enter Your Choice := ";

  cin >> ch;

  switch (ch) {

    case 1:

      cout << "Enter P := ";

      cin >> p;

      for (int i = 0; i < 20; i++) {

        double w = p \* edge[i][0] + 1;

        double x = edge[i][0];

        double y = edge[i][1];

        double z = edge[i][2];

        edge[i][0] = x / w;

        edge[i][1] = y / w;

        edge[i][2] = z / w;

        edge[i][3] = 1.0 / w;

      }

      break;

    case 2:

      cout << "Enter Q := ";

      cin >> q;

      for (int i = 0; i < 20; i++) {

        double w = q \* edge[i][1] + 1;

        double x = edge[i][0];

        double y = edge[i][1];

        double z = edge[i][2];

        edge[i][0] = x / w;

        edge[i][1] = y / w;

        edge[i][2] = z / w;

        edge[i][3] = 1.0 / w;

      }

      break;

    case 3:

      cout << "Enter R := ";

      cin >> r;

      for (int i = 0; i < 20; i++) {

        double w = r \* edge[i][2] + 1;

        double x = edge[i][0];

        double y = edge[i][1];

        double z = edge[i][2];

        edge[i][0] = x / w;

        edge[i][1] = y / w;

        edge[i][2] = z / w;

        edge[i][3] = 1.0 / w;

      }

      break;

  }

  draw\_cube(edge);

}

int main()

{

   int gm,gd = DETECT;

char data[] = "C:\\MinGW\\lib\\libbgi.a";

initgraph(&gd, &gm, data);

    int choice;

  double edge[20][4] = {

      100, 0, 0,1,

      100, 100, 0,1,

      0, 100, 0,1,

      0, 100, 100,1,

      0, 0, 100,1,

      0, 0, 0,1,

      100, 0, 0,1,

      100, 0, 100,1,

      100, 75, 100,1,

      75, 100, 100,1,

      100, 100, 75,1,

      100, 100, 0,1,

      100, 100, 75,1,

      100, 75, 100,1,

      75, 100, 100,1,

      0, 100, 100,1,

      0, 100, 0,1,

      0, 0, 0,1,

      0, 0, 100,1,

      100, 0, 100,1};

    while (s==1)

  {

    cout << "1:==> Draw Cube " << endl;

    cout << "2:==> Scaling " << endl;

    cout << "3:==> Rotation " << endl;

    cout << "4:==> Reflection " << endl;

    cout << "5:==> Translation " << endl;

    cout << "6:==> Perspective Projection " << endl;

    cout << "7:==> Exit " << endl;

    cout << "Enter Your Choice := ";

    cin >> choice;

    switch (choice)

    {

    case 1:

      draw\_cube(edge);

      break;

    case 2:

      scale(edge);

      break;

    case 3:

      rotate(edge);

      break;

    case 4:

      reflect(edge);

      break;

    case 5:

      translate(edge);

      break;

    case 6:

      perspect(edge);

      break;

    case 7:

       exit(0);

      break;

    default:

      cout << "\nPress A Valid Key...!!! ";

      break;

    }

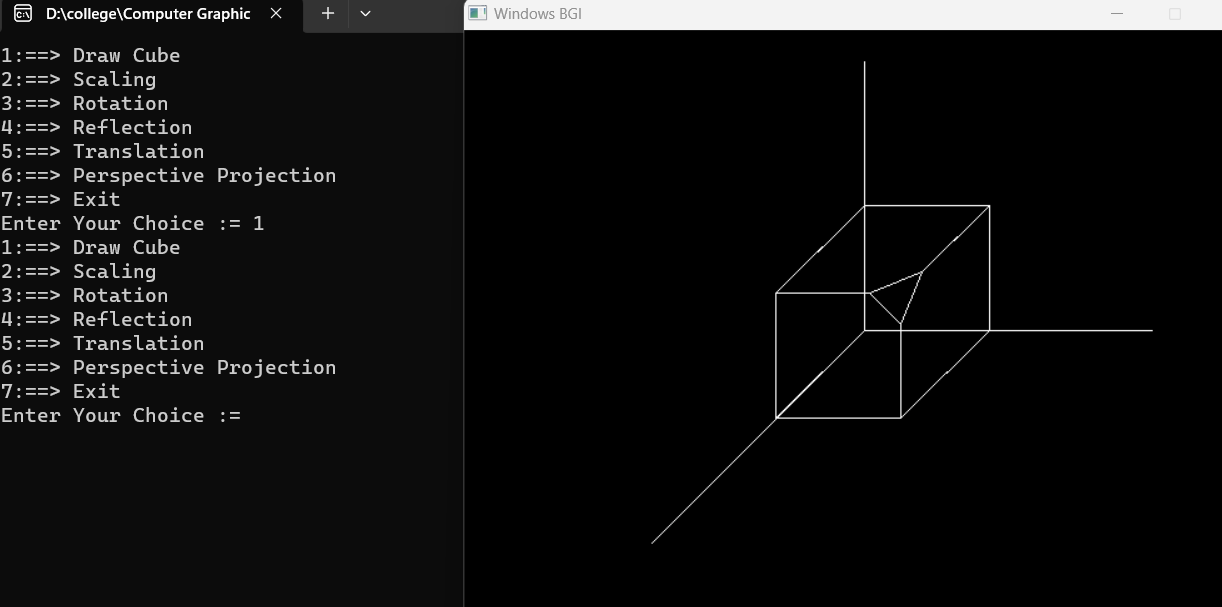
  }

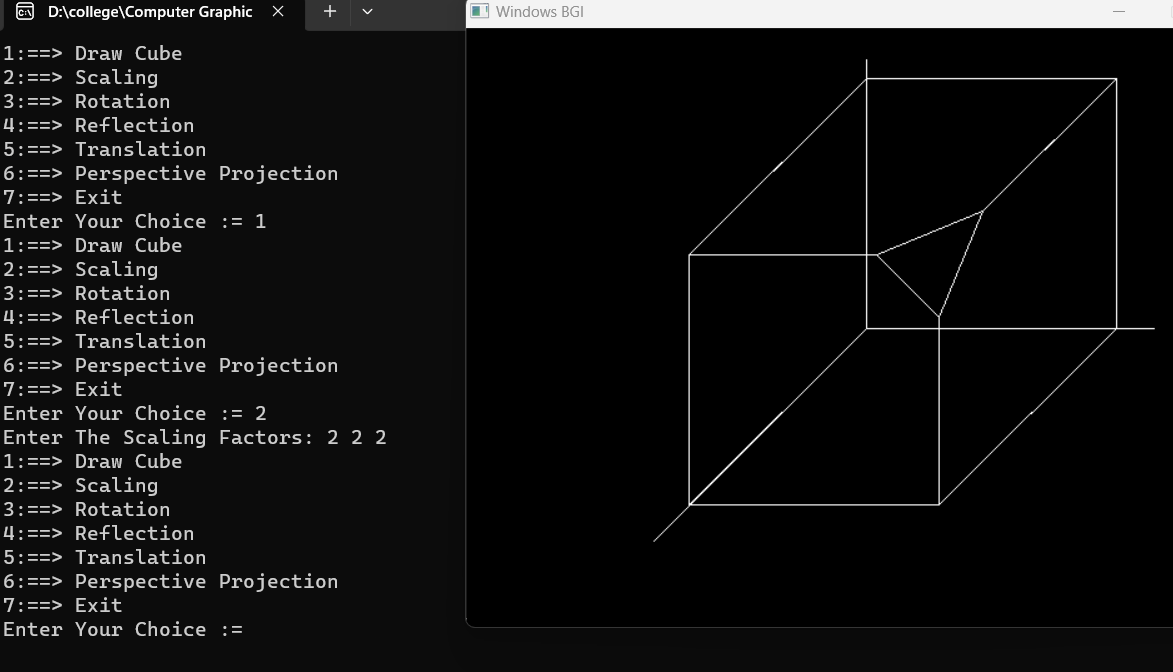
  getch();

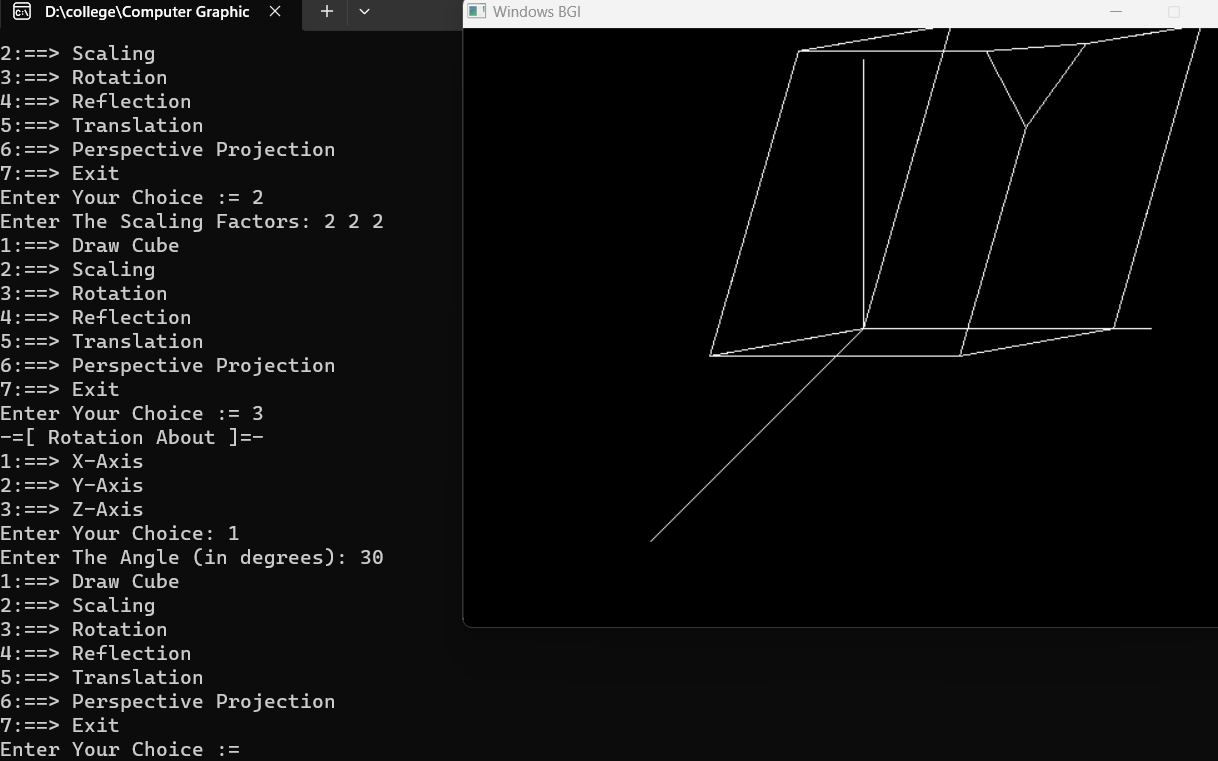
  closegraph();

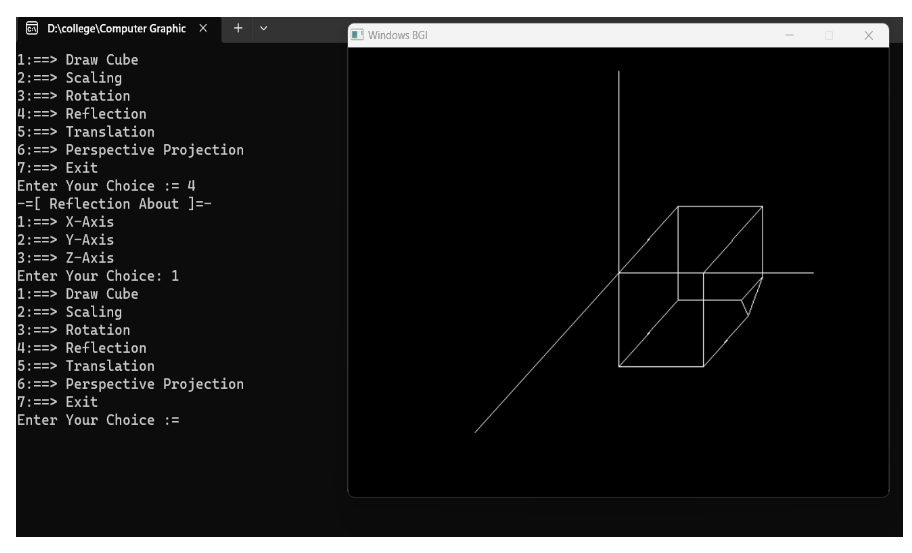
  return 0;

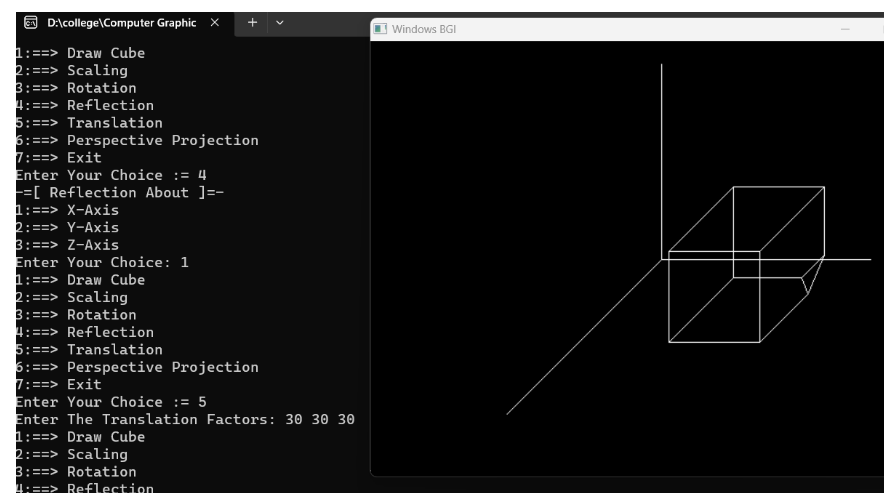
}

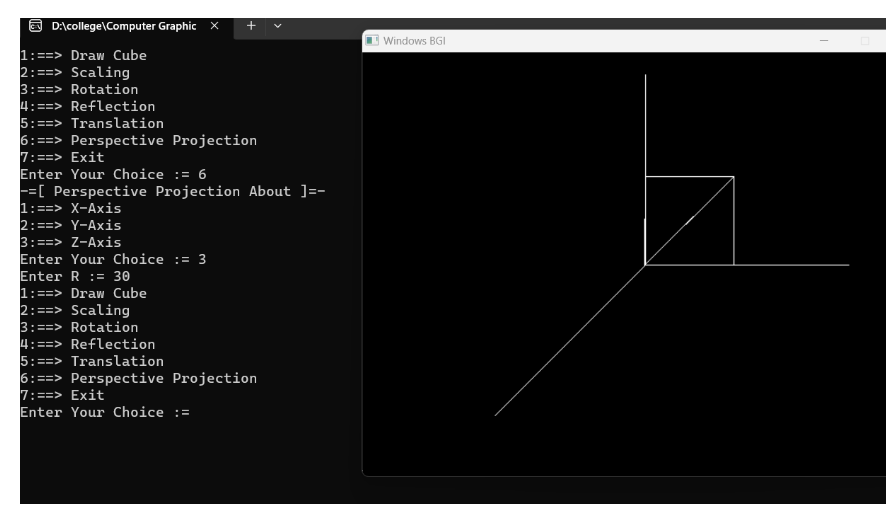












**8. Write a program to draw Hermite /Bezier curve**

**Hermite.cpp**

#include <graphics.h>

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <math.h>

using namespace std;

struct point

{

  int x, y;

};

void hermite(point p1, point p4, double r1, double r4)

{

  float x, y, t;

  for (t = 0.0; t <= 1.0; t += 0.00005)

  {

    x = (2 \* pow(t, 3) - 3 \* pow(t, 2) + 1) \* p1.x +

        (-2 \* pow(t, 3) + 3 \* pow(t, 2)) \* p4.x +

        (pow(t, 3) - 2 \* pow(t, 2) + t) \* r1 +

        (pow(t, 3) - pow(t, 2)) \* r4;

    y = (2 \* pow(t, 3) - 3 \* pow(t, 2) + 1) \* p1.y +

        (-2 \* pow(t, 3) + 3 \* pow(t, 2)) \* p4.y +

        (pow(t, 3) - 2 \* pow(t, 2) + 1) \* r1 +

        (pow(t, 3) - pow(t, 2)) \* r4;

    putpixel(x, y, WHITE);

  }

  circle(p1.x, p1.y, 3);

  circle(p4.x, p4.y, 3);

}

int main()

{

  point p1, p4;

  double r1, r4;

  int gd = DETECT, gm;

  char data[] = "C:\\MinGW\\lib\\libbgi.a";

  initgraph(&gd, &gm, data);

    cout << "---Hermite Curve---" << endl;

  cout << "Enter Point 1 (x, y): ";

  cin >> p1.x >> p1.y;

  cout << "Enter Point 2 (x, y): ";

  cin >> p4.x >> p4.y;

  cout << "Enter Tangent at Point 1: ";

  cin >> r1;

  cout << "Enter Tangent at Point 4: ";

  cin >> r4;

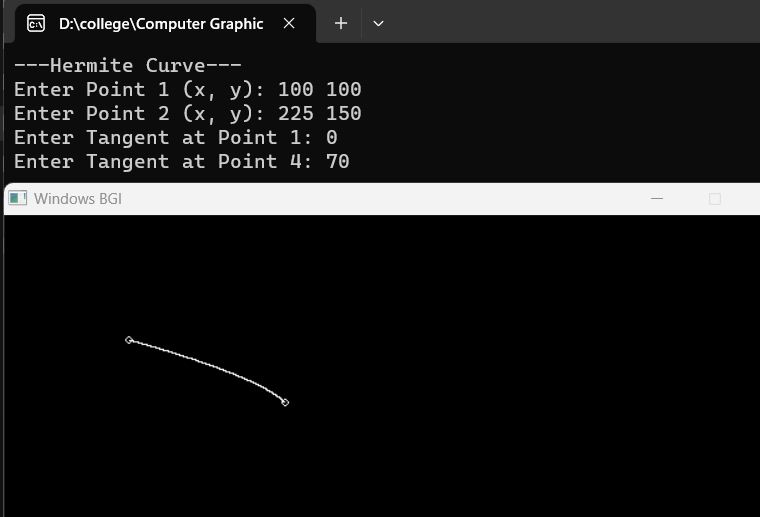
  hermite(p1, p4, r1, r4);

  getch();

  closegraph();

  return 0;

}



**Bezier.cpp**

#include <graphics.h>

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <math.h>

using namespace std;

void bezier(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)

{

    float x, y, t;

    for (t = 0.0; t <= 1.0; t += 0.00005)

    {

        x = pow(1 - t, 3) \* x1 + 3 \* t \* pow(1 - t, 2) \* x2 + 3 \* pow(t, 2) \* (1 - t) \* x3 + pow(t, 3) \* x4;

        y = pow(1 - t, 3) \* y1 + 3 \* t \* pow(1 - t, 2) \* y2 + 3 \* pow(t, 2) \* (1 - t) \* y3 + pow(t, 3) \* y4;

        putpixel(x, y, WHITE);

    }

    circle(x1, y1, 3);

    circle(x2, y2, 3);

    circle(x3, y3, 3);

    circle(x4, y4, 3);

}

int main()

{

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    initgraph(&gd, &gm, data);

     int x1, y1, x2, y2, x3, y3, x4, y4;

     cout << "---Bezier Curve---" << endl;

    cout << "Enter Point 1 (x, y): ";

    cin >> x1 >> y1;

    cout << "Enter Point 2 (x, y): ";

    cin >> x2 >> y2;

    cout << "Enter Point 3 (x, y): ";

    cin >> x3 >> y3;

    cout << "Enter Point 4 (x, y): ";

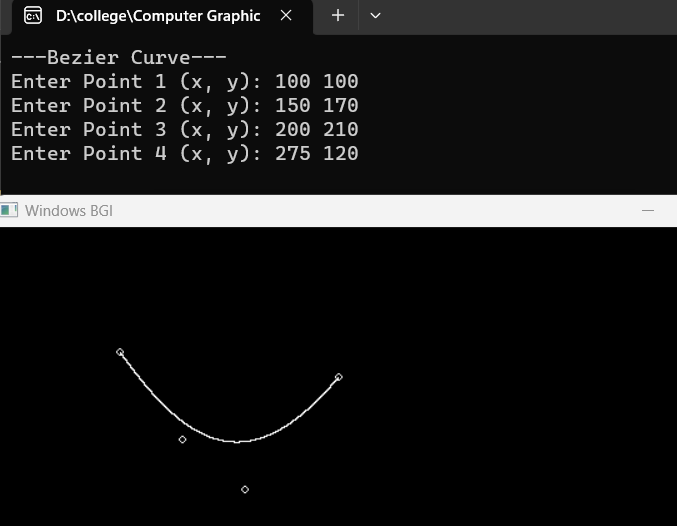
    cin >> x4 >> y4;

    bezier(x1, y1, x2, y2, x3, y3, x4, y4);

    getch();

    closegraph();

    return 0;

}