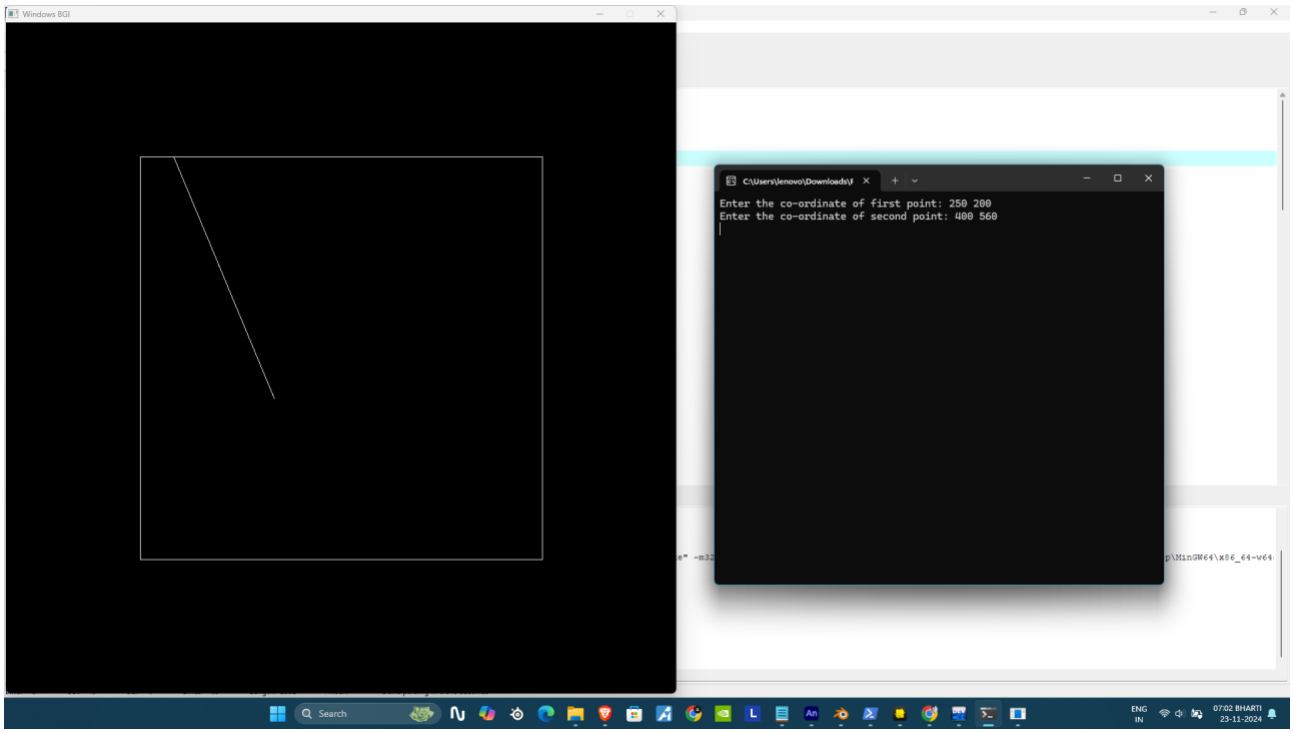
Q1. Write a program to implement Bresenham's Line Drawing Algorithm.

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

using namespace std;

// Function to draw a line using Bresenham's algorithm
void bresenhamLine(int x0, int y0, int x1, int y1, int val) {
    // Check if the endpoints are the same
    if (x0 == x1 && y0 == y1) {
        putpixel(x1, y1, val);
    } else {
        int dx = x1 - x0; // Change in x
        int dy = y1 - y0; // Change in y

        float m = float(dy) / float(dx); // Calculate slope

        // Check if the slope is valid
        if (m >= 1 || m <= 0) {
            cout << "ERROR: Slope must be between 0 and 1." << endl;
            exit(1);
        }

        // Bresenham's algorithm initialization
        int d = 2 * dy - dx;
        int del_E = 2 * dy;
        int del_NE = 2 * (dy - dx);

        int x = x0;
        int y = y0;
        putpixel(x, y, val); // Plot the initial point

        // Loop through each pixel along the line
        while (x < x1) {
            if (d <= 0) {
                d += del_E; // Move east
            } else {
                d += del_NE; // Move northeast
                y += 1;     // Increment y
            }
            x += 1; // Increment x
            putpixel(x, y, val); // Plot the current point
        }
    }
}

int main(void) {
    int x0, y0, x1, y1;
```

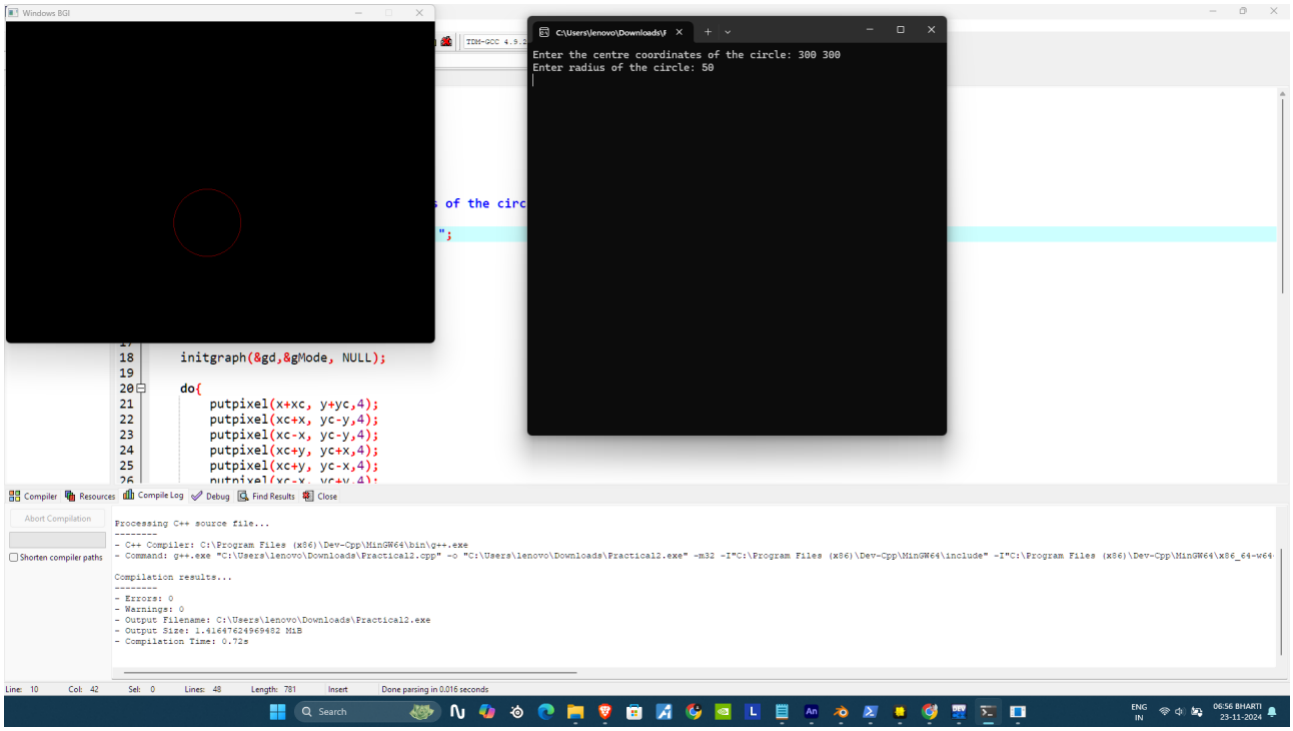
```
    // Input endpoints from the user
    cout << "Enter Left Endpoint (x0 y0): ";
    cin >> x0 >> y0;
    cout << "Enter Right Endpoint (x1 y1): ";
    cin >> x1 >> y1;

    cout << "Drawing Line..." << endl;

    int gd = DETECT, gm;
    initgraph(&gd, &gm, NULL); // Initialize graphics mode

    bresenhamLine(x0, y0, x1, y1, WHITE); // Draw the line

    delay(5000); // Wait for 5 seconds before closing
    closegraph(); // Close graphics mode

    cout << "Finished..." << endl;

    return 0; // End of program
}
```



Q2. Write a program to implement Midpoint Circle Drawing Theorem.

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

using namespace std;

int main() {
    int c, r, xc, yc;

    // Input center coordinates and radius of the circle
    cout << "Enter the center coordinates of the circle: ";
    cin >> xc >> yc;
    cout << "Enter the radius of the circle: ";
    cin >> r;

    // Initial values for Bresenham's circle algorithm
    int x = 0;
    int y = r;
    int p = 1 - r; // Decision parameter

    int gd = DETECT, gMode;
    initgraph(&gd, &gMode, NULL); // Initialize graphics mode

    // Draw the circle using Bresenham's algorithm
    do {
        // Plotting the points in all octants
        putpixel(x + xc, y + yc, 2);  // Octant 1
        putpixel(xc + x, yc - y, 2);  // Octant 2
        putpixel(xc - x, yc - y, 2);  // Octant 3
        putpixel(xc + y, yc + x, 2);  // Octant 4
        putpixel(xc + y, yc - x, 2);  // Octant 5
        putpixel(xc - x, yc + y, 2);  // Octant 6
        putpixel(xc - y, yc + x, 2);  // Octant 7
        putpixel(xc - y, yc - x, 2);  // Octant 8

        // Update decision parameter and coordinates
        if (p < 0) {
            x = x + 1;              // Move to next point in x direction
            p = p + 2 * x + 1;      // Update decision parameter
        } else {
            x = x + 1;              // Move to next point in x direction
            y = y - 1;              // Move down in y direction
            p = p + 2 * x - 2 * y + 1; // Update decision parameter
        }
    } while (x <= y); // Continue until we reach the midpoint

    delay(10000); // Wait for a while before closing
    closegraph(); // Close graphics mode

    return 0; // End of program
```

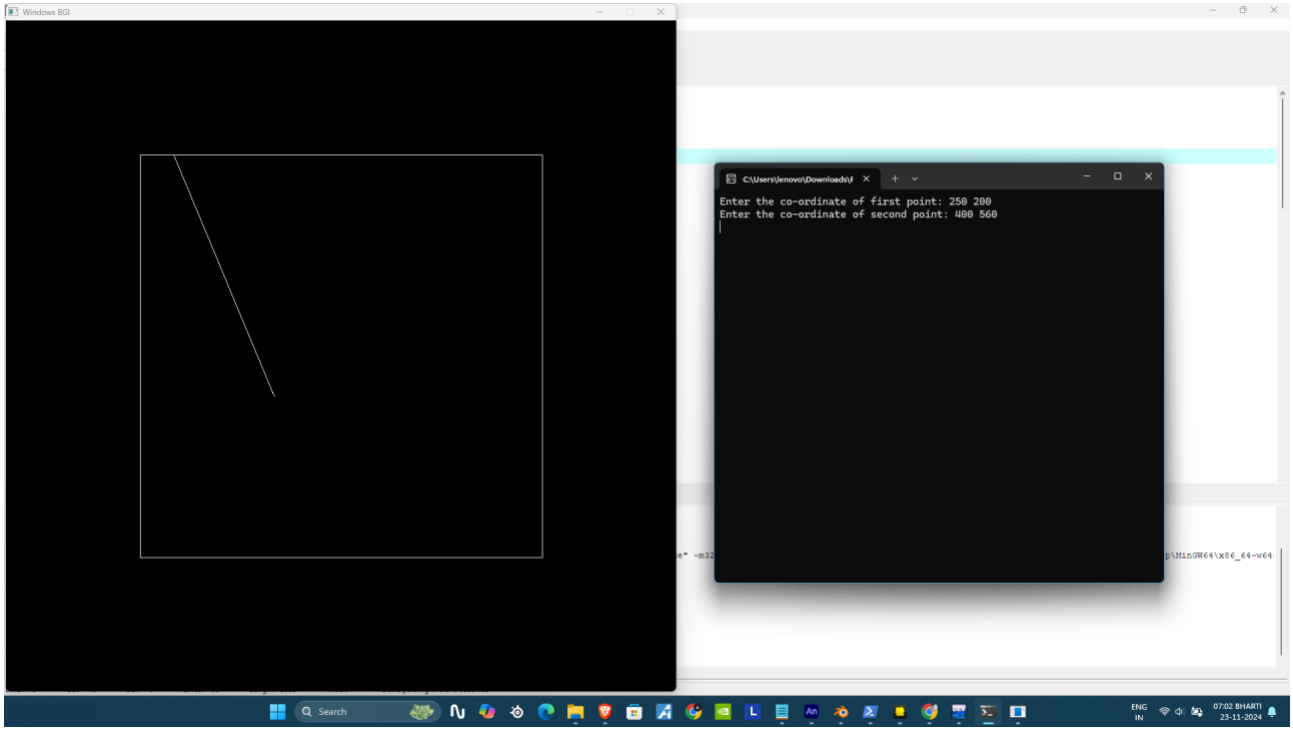
```
}
```


Windows BGI
C:\Users\lenovo\Downloads\
Enter the centre coordinates of the circle: 300 300
Enter radius of the circle: 50
initgraph(&gd,&gMode, NULL);
do{
putpixel(x+xc, y+yc,4);
putpixel(xc+x, yc-y,4);
putpixel(xc-x, yc-y,4);
putpixel(xc+y, yc+x,4);
putpixel(xc+y, yc-x,4);
Compiler
Resources
Compile Log
Debug
Find Results
Close
Abort Compilation
Shorten compiler paths
Processing C++ source file...
- C++ Compiler: C:\Program Files (x86)\Dev-Cpp\MinGW64\bin\g++.exe
- Output Filename: C:\Users\lenovo\Downloads\Practical2.exe
- Output Size: 1.41647624969482 MiB
- Compilation Time: 0.72s
Compilation results...
- Errors: 0
- Warnings: 0
Line: 10
Col: 42
Sel: 0
Lines: 48
Length: 781
Insert
Done parsing in 0.016 seconds
Search
ENG
IN
06:56 BHARTI
23-11-2024

Q3. Write a program to implement Cohen and Sutherland Hodgemann Algorithm.

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

using namespace std;

int xmin = 100, ymin = 300, xmax = 500, ymax = 500;

const int Left = 1;
const int Right = 2;
const int Top = 8;
const int Bottom = 4;

int computecode(int x, int y) {
   int code = 0;
   if (x < xmin) code |= Left;
   else if (y < ymin) code |= Bottom;
   if (x > xmax) code |= Right;
   else if (y > ymax) code |= Top;
   return code;
}

void clip(int x0, int x1, int y0, int y1) {
   int code1, code2;
   int accept, flag = 0;

   code1 = computecode(x0, y0);
   code2 = computecode(x1, y1);

   double m = (y1 - y0) / (x1 - x0);

   if ((code1 & code2) != 0) {
      accept = false;
   } else {
      do {
         if (code1 == 0 && code2 == 0) {
            accept = true;
            flag = 1;
         } else {
            int x, y, temp;

            if (code1 == 0) temp = code2;
            else temp = code1;

            if (temp & Top) {
               x = x0 + (1 / m) * (ymax - y0);
               y = ymax;
            } else if (temp & Bottom) {
               x = x0 + (1 / m) * (ymin - y0);
               y = ymin;
            } else if (temp & Left) {
               y = y0 + m * (xmin - x0);
               x = xmin;
            } else if (temp & Right) {
               y = y0 + m * (xmax - x0);
               x = xmax;
            }
```

```
            if (temp == code1) {
               x0 = x;
               y0 = y;
               code1 = computecode(x0, y0);
            } else {
               x1 = x;
               y1 = y;
               code2 = computecode(x1, y1);
            }
         }
      } while (!flag);
   }

   if (accept) {
      cleardevice();
      line(x0, y0, x1, y1);
      rectangle(xmin, ymin, xmax, ymax);
   }
}

int main() {
   int window1 = initwindow(800, 800);
   int x0, x1, y0, y1;

   cout << "Enter the co-ordinate of first point: ";
   cin >> x0 >> y0;
   cout << "Enter the co-ordinate of second point: ";
   cin >> x1 >> y1;

   line(x0, y0, x1, y1);
   rectangle(xmin, ymin, xmax, ymax);

   delay(7000);
   clip(x0, x1, y0, y1);

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



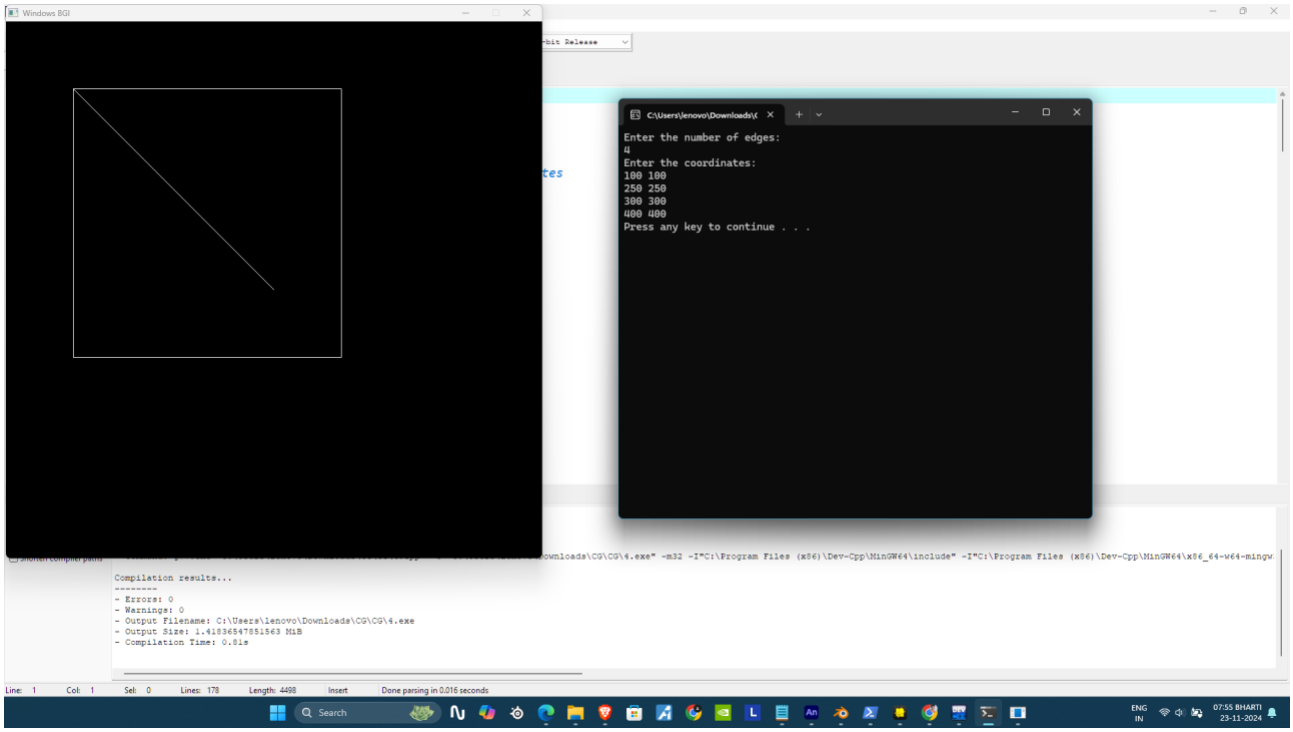
Q4. Write a program to clip a polygon using Sutherland and Hodgemann algorithm.

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

using namespace std;

// Global variables for clipping window and coordinates
int xmin = 100, xmax = 500, ymin = 100, ymax = 500;
int arr[20], m, k;

// Function to clip a line segment from the left
void clipLeft(int x1, int y1, int x2, int y2) {
    if (x2 - x1) {
        m = (y2 - y1) / (x2 - x1);
    } else {
        m = 10000; // Infinite slope
    }

    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;
    }
}

// Function to clip a line segment from the top
void clipTop(int x1, int y1, int x2, int y2) {
    if (y2 - y1) {
        m = (x2 - x1) / (y2 - y1);
    } else {
        m = 10000; // Infinite slope
    }

    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;
    }
}

// Function to clip a line segment from the right
void clipRight(int x1, int y1, int x2, int y2) {
    if (x2 - x1) {
        m = (y2 - y1) / (x2 - x1);
    } else {
        m = 10000; // Infinite slope
    }

    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;
    }
}

// Function to clip a line segment from the bottom
void clipBottom(int x1, int y1, int x2, int y2) {
    if (y2 - y1) {
        m = (x2 - x1) / (y2 - y1);
    } else {
        m = 10000; // Infinite slope
    }

    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 poly[20];
    int window1 = initwindow(800, 800); // Initialize graphics window
    int n, i;

    cout << "Enter the number of edges: " << endl; // User input for number of edges
    cin >> n;

    cout << "Enter the coordinates: " << endl; // User input for polygon coordinates
    for (i = 0; i < 2 * n; i++)
      cin >> poly[i];
```

```
  poly[i]    = poly[0]; // Closing the polygon
  poly[i + 1] = poly[1];

  rectangle(xmin, ymax, xmax, ymin); // Draw clipping rectangle
  fillpoly(n, poly);              // Fill the polygon
  delay(1000);                    // Delay for visibility
  cleardevice();                  // Clear the device

  k = 0; // Reset index for clipped coordinates

  // Clipping process
  for(i = 0; i < 2 * n; i += 2)
    clipLeft(poly[i], poly[i+1], poly[i+2], poly[i+3]);

  n = k / 2; // Update number of vertices after clipping
  for(i = 0; i < k; i++)
    poly[i] = arr[i];

  poly[i]    = poly[0]; // Closing the polygon again
  poly[i + 1] = poly[1];

  k = 0; // Reset index again for next clipping

 for(int i=0; i<2*n; i+=2)
    clipRight(poly[i], poly[i+1], poly[i+2], poly[i+3]);

 n=k/2;
 for(int i=0;i<k;i++)
    poly[i]=arr[i];

 poly[i]=poly[0];
 poly[i+1]=poly[1];

 k=0;

 for(int i=0;i<2*n;i+=2)
    clipBottom(poly[i],poly[i+1],poly[i+2],poly[i+3]);

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

 rectangle(xmin,ymax,xmax,ymin); // Draw clipping rectangle again

 if(k)
    fillpoly(k/2,poly); // Fill the clipped polygon

 system("pause"); // Wait for user input before closing
 return 0;      // End of program
}
```

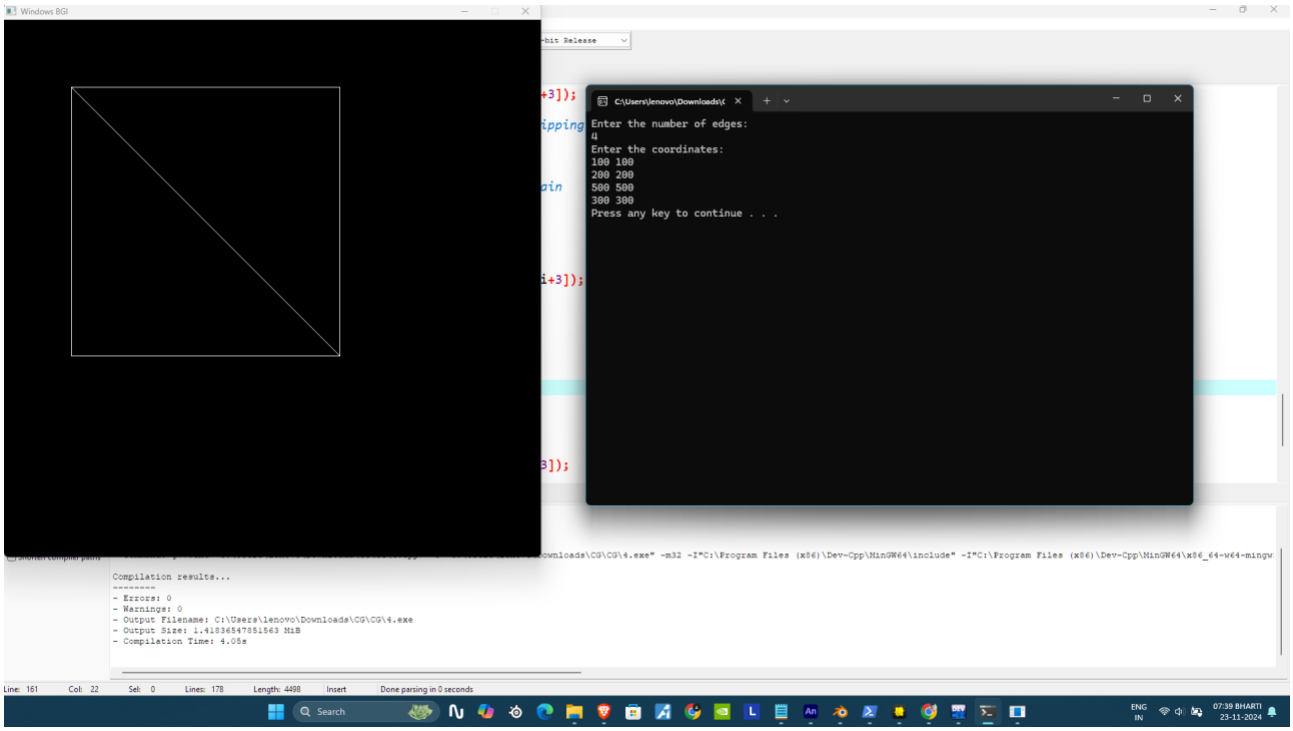

Windows BGI
C:\Users\lenovo\Downloads\C
Enter the number of edges:
4
Enter the coordinates:
100 100
250 250
300 300
400 400
Press any key to continue . . .
Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\lenovo\Downloads\CG\CG\4.exe
- Output Size: 1.41836547851563 MiB
- Compilation Time: 0.81s
Line: 1 Col: 1 Sel: 0 Lines: 178 Length: 4498 Insert Done parsing in 0.016 seconds
Search
ENG
IN
07:55 BHARTI
23-11-2024

Q5. Write a program to fill a polygon using the Scan Line fill algorithm.

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

using namespace std;

int main() {
    int n, i, j, k, gd, gm, dy, dx;
    int x, y, temp;
    int a[20][2], xi[20];
    float slope[20];
    int temp1 = 0;

    cout << "\nEnter the number of edges: ";
    cin >> n;

    // Input coordinates for the polygon
    for (i = 0; i < n; i++) {
        cout << "Enter the coordinate x" << i + 1 << ": ";
        cin >> a[i][0];
        cout << "Enter the coordinate y" << i + 1 << ": ";
        cin >> a[i][1];
    }

    // Closing the polygon by repeating the first vertex
    a[n][0] = a[0][0];
    a[n][1] = a[0][1];

    // Initialize graphics
    initgraph(&gd, &gm, NULL);
    setcolor(YELLOW);

    // Draw the polygon
    for (i = 0; i < n; i++) {
        line(a[i][0], a[i][1], a[i + 1][0], a[i + 1][1]);
    }

    // Calculate slopes for each edge
    for (i = 0; i < n; i++) {
        dy = a[i + 1][1] - a[i][1];
        dx = a[i + 1][0] - a[i][0];

        if (dy == 0) {
            slope[i] = 1.0; // Horizontal line
        } else if (dx == 0) {
            slope[i] = 0.0; // Vertical line
        } else {
            slope[i] = (float)dx / dy; // Calculate slope
        }
    }
```

```
    // Scanline algorithm to fill the polygon
    for (y = 0; y < 400; y++) {
        k = 0;

        // Find intersection points with the scanline
        for (i = 0; i < n; i++) {
            if (((a[i][1] <= y) && (a[i + 1][1] > y)) ||
                ((a[i][1] > y) && (a[i + 1][1] <= y))) ) {
                xi[k] = (int)(a[i][0] + slope[i] * (y - a[i][1]));
                k++;
            }
        }

        // Sort intersection points
        for (j = 0; j < k; j++) {
            for (i = 0; i < k - 1; i++) { // Fixed loop condition to avoid out-of-bounds access
                if (xi[i] > xi[i + 1]) {
                    temp = xi[i];
                    xi[i] = xi[i + 1];
                    xi[i + 1] = temp;
                }
            }
        }

        // Draw horizontal lines between pairs of intersection points
        setcolor(YELLOW);
        for (i = 0; i < k; i += 2) {
            line(xi[i], y, xi[i + 1] + 1, y); // Fill between intersections
            temp1 = i;
        }
    }

    delay(7000); // Wait before closing the graphics window
    return 0;
}
```



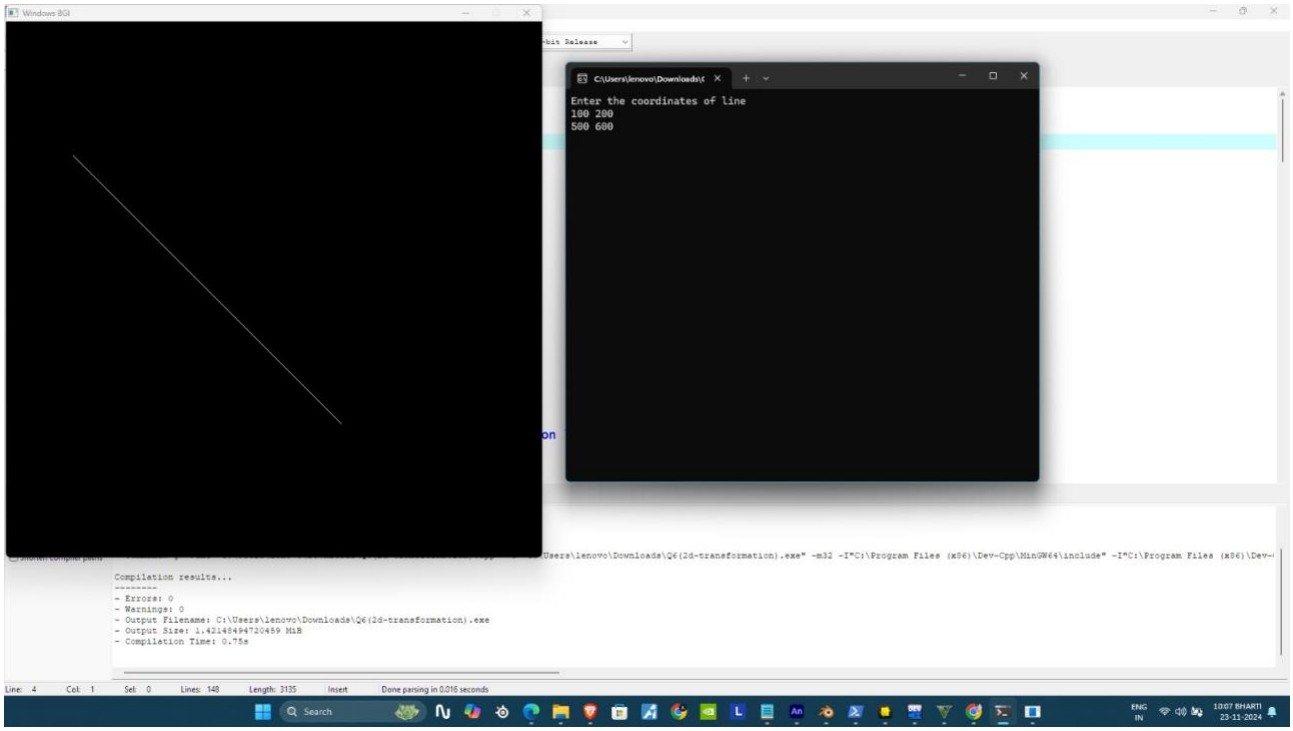
Q6. Write a program to apply various 2D Transformations on a 2D object.

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

using namespace std;

int main() {
    int tx = 2, ty = 5; // Translation factors
    int window1 = initwindow(800, 800);
    int i, j, k;
    float P[2][3]; // Array to hold the coordinates of the line in homogeneous form

    cout << "Enter the coordinates of the line (x1 y1 x2 y2): " << endl;
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++) {
            cin >> P[i][j];
            P[i][2] = 1; // Homogeneous coordinate
        }
    }

    // Draw the original line
    line(P[0][0], P[0][1], P[1][0], P[1][1]);
    delay(7000);

    float pp[2][3] = {0}; // Array for transformed coordinates
```

```
int ch;

do {
    cout << "Enter the 2D transformation:" << endl;
    cout << "1. Translation\n2. Shearing\n3. Reflection\n4. Rotation\n5. Scaling\n6. Exit" << endl;
    cin >> ch;

    switch (ch) {
        case 1: { // Translation
            cout << "Enter the translating factors (tx ty): ";
            cin >> tx >> ty;
            int T[3][3] = {{1, 0, 0},
                           {0, 1, 0},
                           {tx, ty, 1}};

            // Apply transformation
            for (i = 0; i < 2; i++) {
                for (j = 0; j < 3; j++) {
                    pp[i][j] = 0; // Initialize transformed coordinates
                    for (k = 0; k < 3; k++) {
                        pp[i][j] += P[i][k] * T[k][j];
                    }
                }
            }
            line(pp[0][0], pp[0][1], pp[1][0], pp[1][1]);
            system("pause");
            break;
        }
        case 2: { // Shearing
            int sh;
            char ax;
            cout << "Enter the shearing axis (x/y): ";
            cin >> ax;
            cout << "Enter the shearing factor: ";
            cin >> sh;

            int T[3][3];
            if (ax == 'x') {
                T[3][3] = {{1, sh, 0},
                           {0, 1, 0},
                           {0, 0, 1}};
            } else if (ax == 'y') {
                T[3][3] = {{1, 0, 0},
                           {sh, 1, 0},
                           {0, 0, 1}};
            }

            // Apply transformation
            for (i = 0; i < 2; i++) {
                for (j = 0; j < 3; j++) {
                    pp[i][j] = 0; // Initialize transformed coordinates
```

```
                for (k = 0; k < 3; k++) {
                    pp[i][j] += P[i][k] * T[k][j];
                }
            }
        }
        line(pp[0][0], pp[0][1], pp[1][0], pp[1][1]);
        system("pause");
        break;
    }
    case 3: { // Reflection
        int midx = getmaxx() / 2;
        int midy = getmaxy() / 2;
        char ax;

        cout << "Enter the axis for reflection (x/y): ";
        cin >> ax;

        if (ax == 'y') {
            pp[0][0] = (midx - P[0][0]) + midx;
            pp[0][1] = P[0][1];
            pp[1][0] = (midx - P[1][0]) + midx;
            pp[1][1] = P[1][1];
        } else if (ax == 'x') {
            pp[0][0] = P[0][0];
            pp[0][1] = (midy - P[0][1]) + midy;
            pp[1][0] = P[1][0];
            pp[1][1] = (midy - P[1][1]) + midy;
        }

        line(pp[0][0], pp[0][1], pp[1][0], pp[1][1]);
        system("pause");
        break;
    }
    case 4: { // Rotation
        float theta;
        cout << "Enter the angle of rotation in degrees: ";
        cin >> theta;

        float rx = (theta * M_PI) / 180.0; // Convert degrees to radians
        float T[3][3] = {{cos(rx), sin(rx), 0},
                         {-sin(rx), cos(rx), 0},
                         {0,     0,      1}};

        // Apply transformation
        for (i = 0; i < 2; i++) {
            for (j = 0; j < 3; j++) {
                pp[i][j] = 0; // Initialize transformed coordinates
                for (k = 0; k < 3; k++) {
                    pp[i][j] += P[i][k] * T[k][j];
                }
            }
```

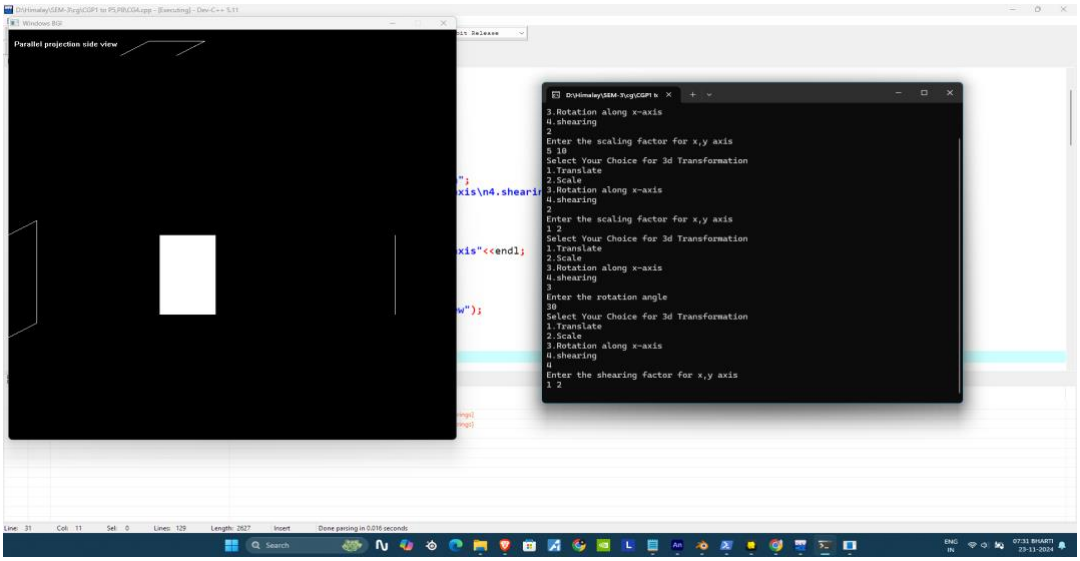
```
            }

            line(pp[0][0], pp[0][1], pp[1][0], pp[1][1]);
            system("pause");
            break;
        }
        case 5: { // Scaling
            int Sx, Sy;

            cout << "Enter the scaling factor for x-axis: ";
            cin >> Sx;

            cout << "Enter the scaling factor for y-axis: ";
            cin >> Sy;

            int T[3][3] = {{Sx,   0,   0},
                           {0,   Sy,   0},
                           {0,   0,   1}};

            // Apply transformation
            for (i = 0; i < 2; i++) {
                for (j = 0; j < 3; j++) {
                    pp[i][j] = 0; // Initialize transformed coordinates
                    for (k = 0; k < 3; k++) {
                        pp[i][j] += P[i][k] * T[k][j];
                    }
                }
            }

            line(pp[0][0], pp[0][1], pp[1][0], pp[1][1]);
            system("pause");
            break;
        }
        case 6:
            return 0; // Exit program
        default:
            cout << "Invalid choice!" << endl;
      }
   } while(ch !=6);

   return 0;
}
```

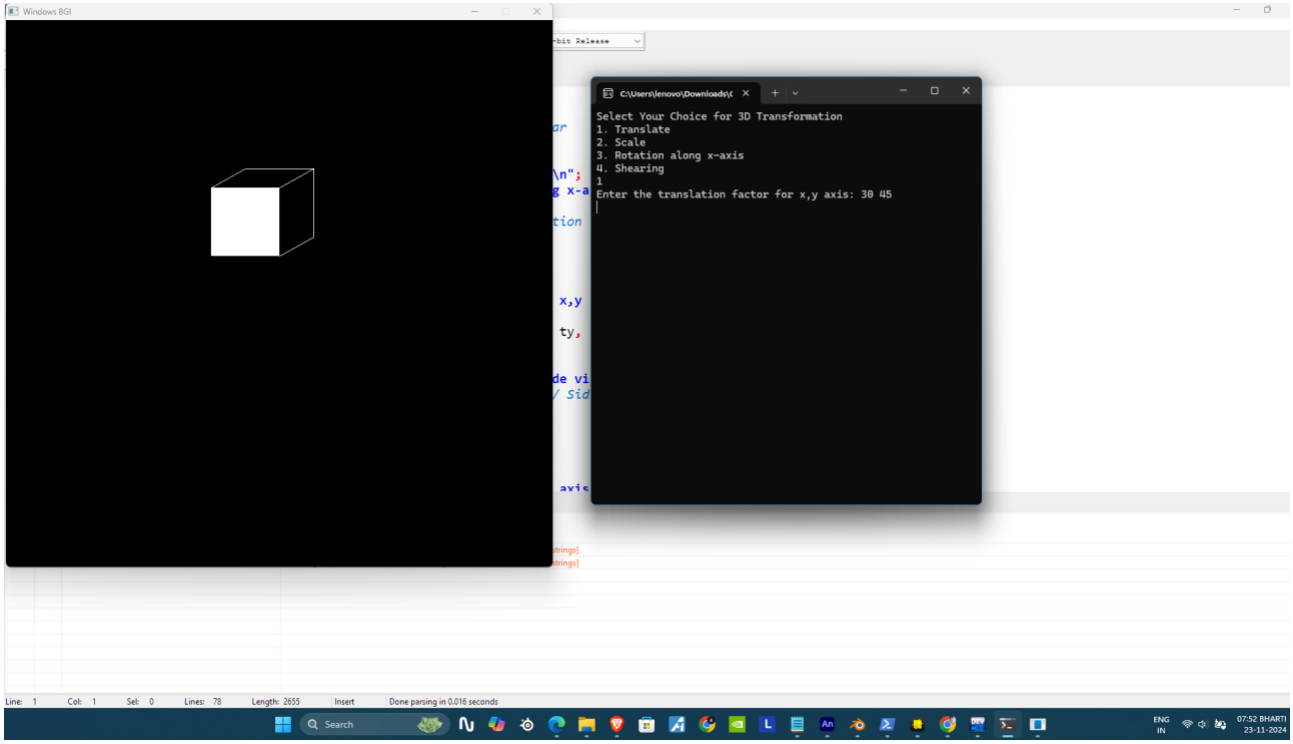

Windows BGI
C:\Users\lenovo\Downloads\(
Enter the coordinates of line
100 200
500 600
Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\lenovo\Downloads\Q6(2d-transformation).exe
- Output Size: 1.4214849472045898 MiB
- Compilation Time: 0.75s
Line: 4 Col: 1 Sel: 0 Lines: 148 Length: 3135 Insert Done parsing in 0.016 seconds
Search
ENG
IN
10:07 BHARTI
23-11-2024

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

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

using namespace std;

int main() {
    int window1 = initwindow(800, 800);
    bar3d(270, 200, 370, 300, 50, 5); // Initial 3D bar

    int ch;
    cout << "Select Your Choice for 3D Transformation\n";
    cout << "1. Translate\n2. Scale\n3. Rotation along x-axis\n4. Shearing\n";
    cin >> ch;
    cleardevice(); // Clear the screen for transformation

    switch (ch) {
        case 1: {
            int tx, ty;
            cout << "Enter the translation factor for x,y axis: ";
            cin >> tx >> ty;
            bar3d(270 + tx, 200 + ty, 370 + tx, 300 + ty, 50, 5); // Translated bar
            delay(7000);
            cleardevice();
            outtextxy(10, 20, "Parallel projection side view");
            bar3d(0, 200 + ty, 0, 300 + ty, 50, 5); // Side view after translation
            delay(7000);
            break;
        }
        case 2: {
            int sx, sy;
            cout << "Enter the scaling factor for x,y axis: ";
            cin >> sx >> sy;
            bar3d(270 * sx, 200 * sy, 370 * sx, 300 * sy, 50, 5); // Scaled bar
            delay(7000);
            cleardevice();
            outtextxy(10, 20, "Parallel projection side view");
            bar3d(0, 200 * sy, 0, 300 * sy, 50, 5); // Side view after scaling
            delay(7000);
            break;
        }
        case 4: {
            int shx, shy;
            cout << "Enter the shearing factor for x,y axis: ";
            cin >> shx >> shy;
            bar3d(270 + (shx * 270),
0 + (shy * 270),
0 + (shx * 370),
```

```
0 + (shy * 300),
 + (shx * 50),
; // Sheared bar
            delay(7000);
            break;
        }
        case 3: {
            int ang;
            cout << "Enter the rotation angle: ";
            cin >> ang;

            // Convert angle from degrees to radians
            ang = (ang * M_PI) / 180;

            // Calculate new coordinates after rotation
            int x1 = static_cast<int>(200 * cos(ang) - 50 * sin(ang));
            int y1 = static_cast<int>(50 * cos(ang) + 200 * sin(ang));
            int x2 = static_cast<int>(300 * cos(ang) - 500 * sin(ang));
            int y2 = static_cast<int>(50 * cos(ang) + 300 * sin(ang));

            bar3d(x1, y1, x2, y2, 50, 5); // Rotated bar
            delay(7000);
            break;
        }
        default:
            cout << "Invalid choice!" << endl; // Handle invalid input
    }

    return 0;
}
```

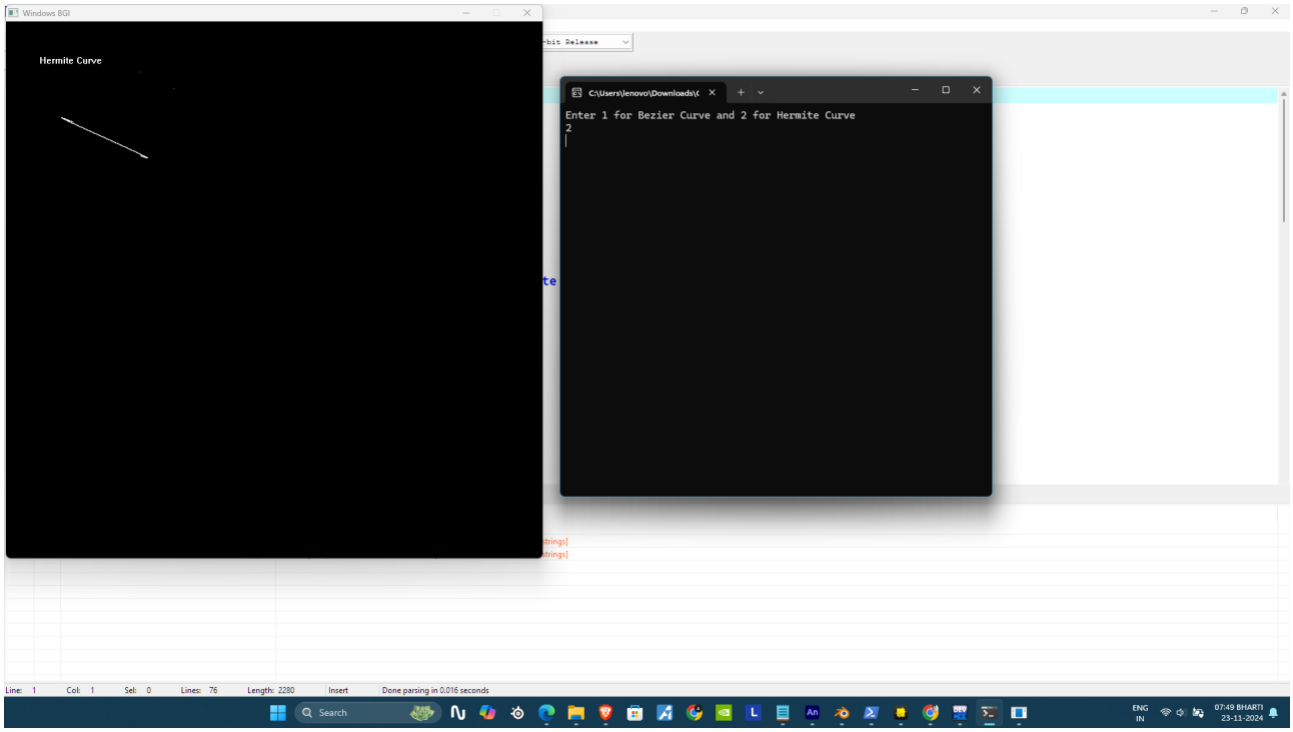


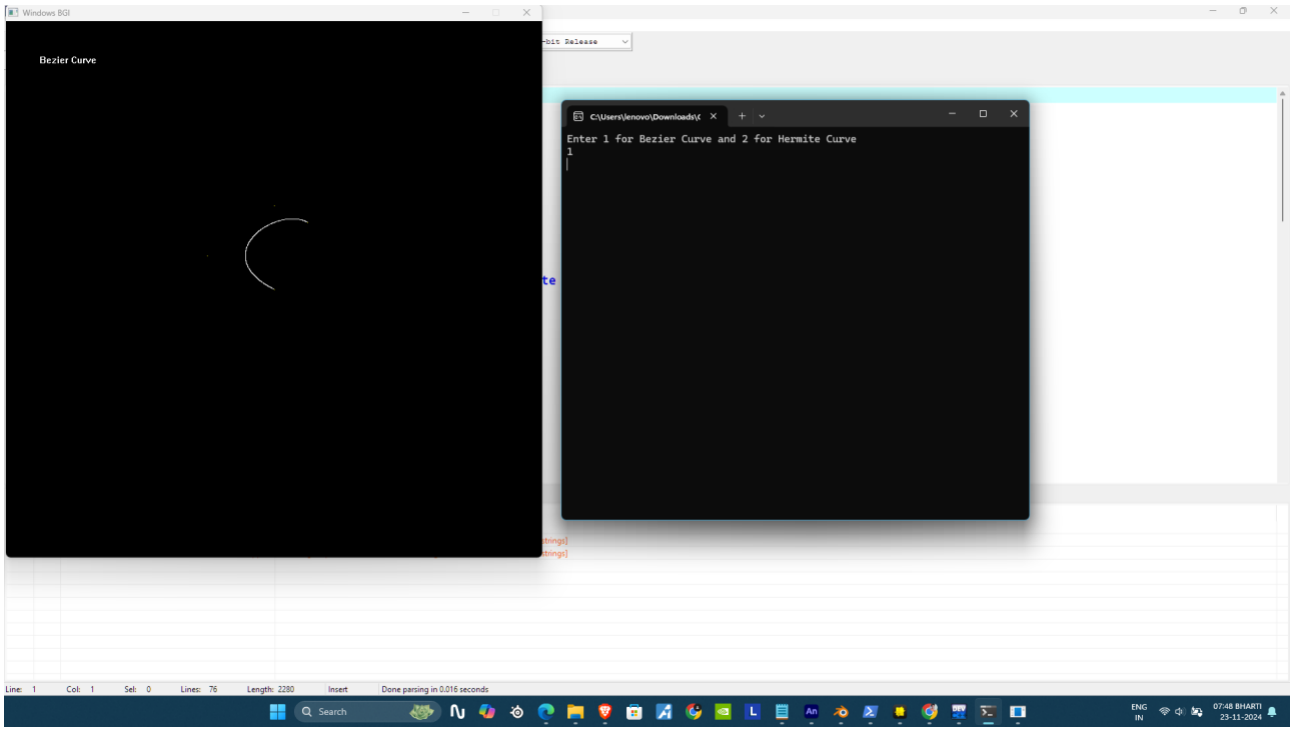


Q8. Write a program to draw Hermite/ Bezier curve.

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

using namespace std;

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

    cout << "Enter 1 for Bezier Curve and 2 for Hermite Curve" << endl;
    cin >> ch;

    switch (ch) {
        case 1: {
            // Bezier Curve points
            int x[4] = {400, 300, 400, 450};
            int y[4] = {400, 350, 275, 300};

            outtextxy(50, 50, "Bezier Curve");
            for (t = 0; t <= 1; t += 0.0005) {
                xt = pow(1 - t, 3) * x[0] +
* t * pow(1 - t, 2) * x[1] +
* pow(t, 2) * (1 - t) * x[2] +
```

```
                    pow(t, 3) * x[3];

                yt = pow(1 - t, 3) * y[0] +
* t * pow(1 - t, 2) * y[1] +
* pow(t, 2) * (1 - t) * y[2] +
                    pow(t, 3) * y[3];

                putpixel(xt, yt, WHITE);
            }

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

            outtextxy(50, 50, "Hermite Curve");
            for (t = 0; t <= 1; 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);
            }

            // Draw control points
            for (i = 0; i < 4; i++) {
                putpixel(x1[i], y1[i], YELLOW);
            }
            delay(9000);
            break;
        }
        default:
            cout << "Invalid choice!" << endl;
    }

    return 0;
}
```


Windows BGI
Hermite Curve
C:\Users\lenovo\Downloads\(
Enter 1 for Bezier Curve and 2 for Hermite Curve
2
Line: 1 Col: 1 Sel: 0 Lines: 76 Length: 2280 Insert Done parsing in 0.016 seconds
Search
ENG
IN
07:49 BHARTI
23-11-2024


Windows BGI
Bezier Curve
C:\Users\lenovo\Downloads\(
Enter 1 for Bezier Curve and 2 for Hermite Curve
1
Line: 1 Col: 1 Sel: 0 Lines: 76 Length: 2280 Insert Done parsing in 0.016 seconds
Search
ENG
IN
07:48 BHARTI
23-11-2024