COMPUTER GRAPHICS

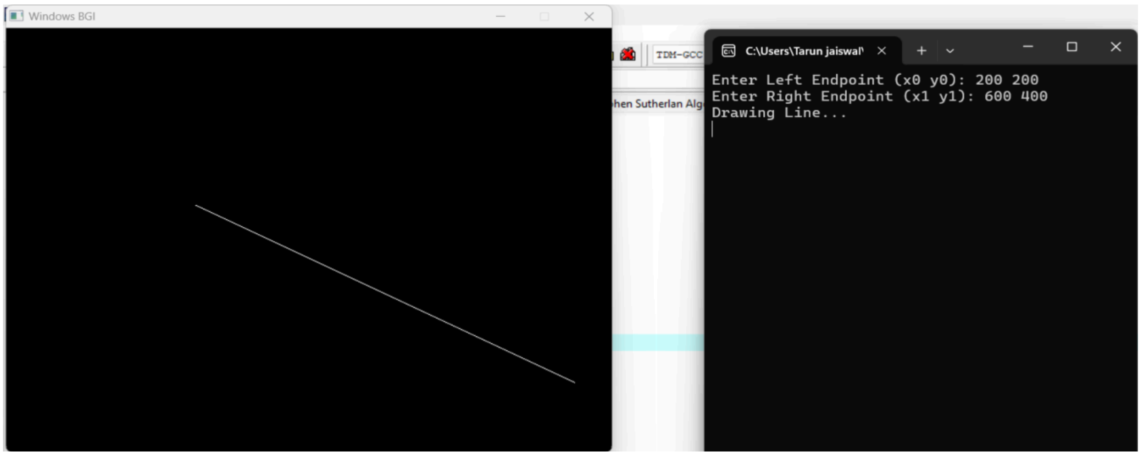
1. Write a program to implement Bresenham's line drawing algorithm.

```
#include <cmath>
#include <cstdlib>
#include <graphics.h>
#include <iostream>
using namespace std;
void bresenhamLine(int x0, int y0, int x1, int y1, int val)
{
 if (x0 == x1 && y0 == y1)
 {
  putpixel(x1, y1, val);
 }
 else
 {
  int dx = x1 - x0;
  int dy = y1 - y0;

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

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

  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);

    while (x < x1)
    {
      if (d <= 0)
      {
        d += del_E;
        x += 1;
      }
      else
      {
        d += del_NE;
        x += 1;
        y += 1;
      }

      putpixel(x, y, val);
    }
  }

  return;
}

int main(void)
{
  int x0, y0, x1, y1;
  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);
    bresenhamLine(x0, y0, x1, y1, WHITE);
    delay(5e3); //5000
    closegraph();

    cout << "Finished..." << endl;
    return 0;
  }
```

<u>Output:-</u>



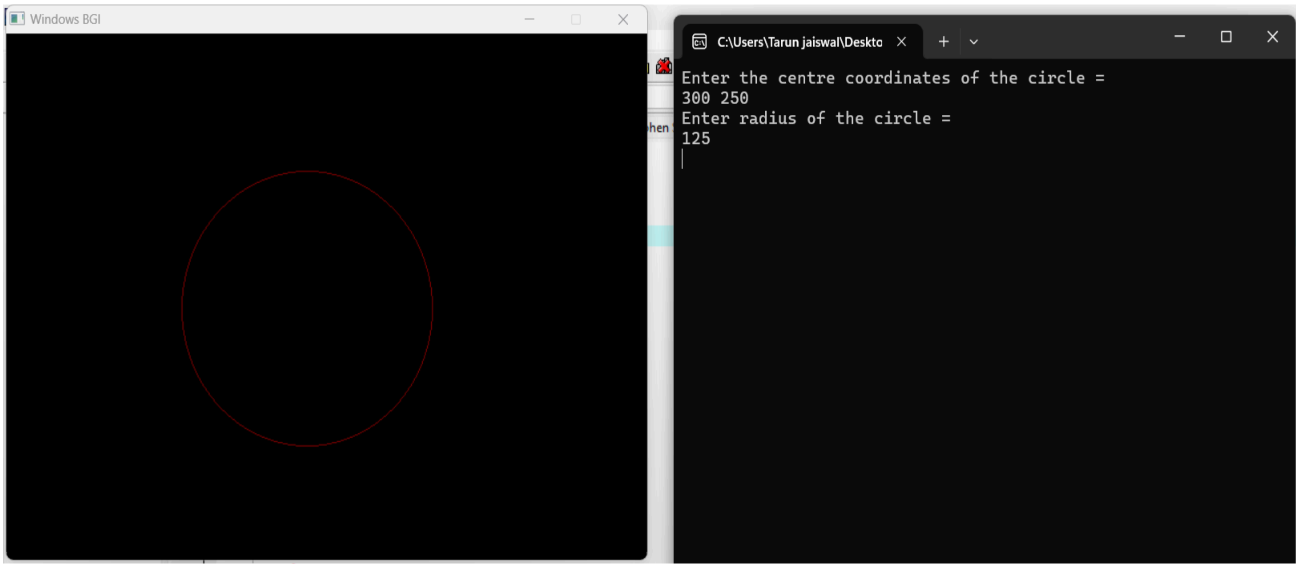
```
Enter Left Endpoint (x0 y0): 200 200
Enter Right Endpoint (x1 y1): 600 400
Drawing Line...
```

2. Write a program to implement a midpoint circle drawing algorithm.

```
#include <iostream>

#include <graphics.h>

using namespace std;

int main(){

    int c,r,xc,yc;
```

```
cout<<"Enter the centre coordinates of the circle =  "<<endl;
cin>>xc>>yc;
cout<<"Enter radius of the circle = "<<endl;
cin>>r;
int x = 0;
int y = r;
int p = 1-r;
int gd = DETECT, gMode;
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);
        putpixel(xc-x, yc+y,4);
        putpixel(xc-y, yc+x,4);
        putpixel(xc-y, yc-x,4);
        if(p<0){
                x =x+1;
                p = p+2*x+1;
                putpixel(x+xc, y+yc,4);
        }
        else{
            x = x+1;

            y = y-1;

            p = p+2*x-2*y+1;

            putpixel(x+xc, y+yc, 4);

        }

    }while(x<=y);

    delay(10000);

    return 0;

}
```

OUTPUT:-



```
Enter the centre coordinates of the circle =
300 250
Enter radius of the circle =
125
```

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

```
#include <iostream>

#include <graphics.h>

using namespace std;
```

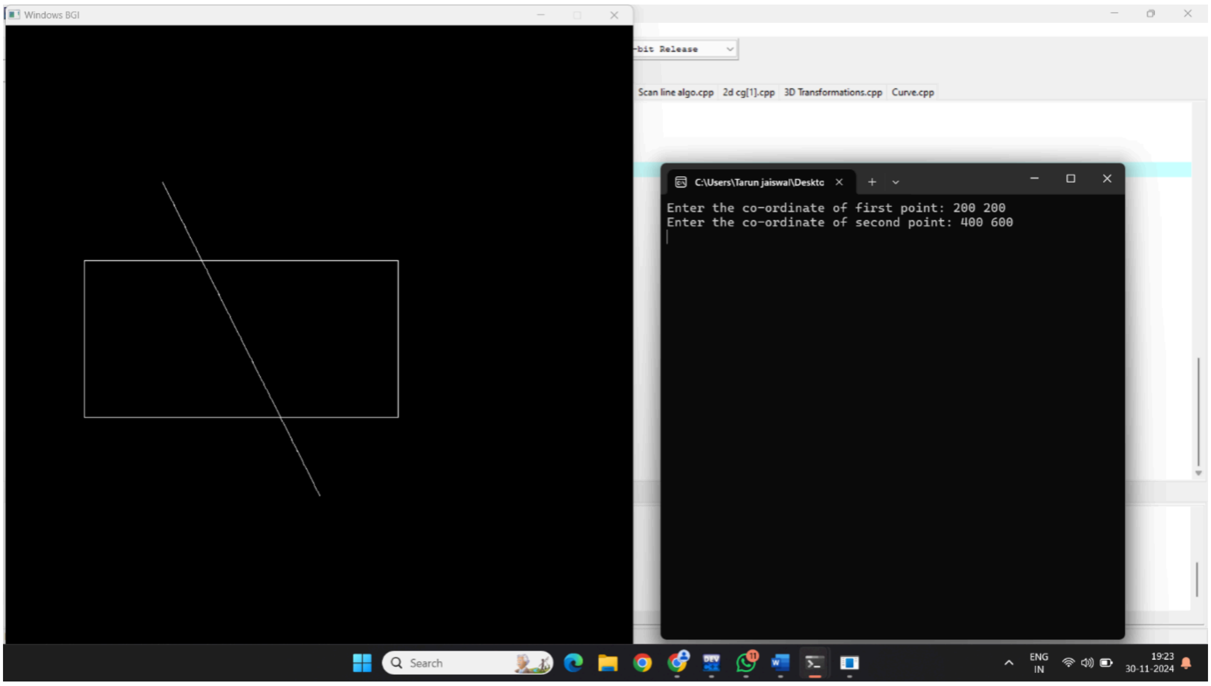
```
int xmin = 100, ymin = 300, xmax = 500, ymax = 500;
const int Left = 1;
const int Right = 2;
const int Top = 8;
const int Bottom = 4;

int computecode (int x, int y) {
    int code = 0;
    if (x < xmin) code |= Left;
    if (y < ymin) code |= Bottom;
    if (x > xmax) code |= Right;
    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);// do-while end
    }
    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;
}
```

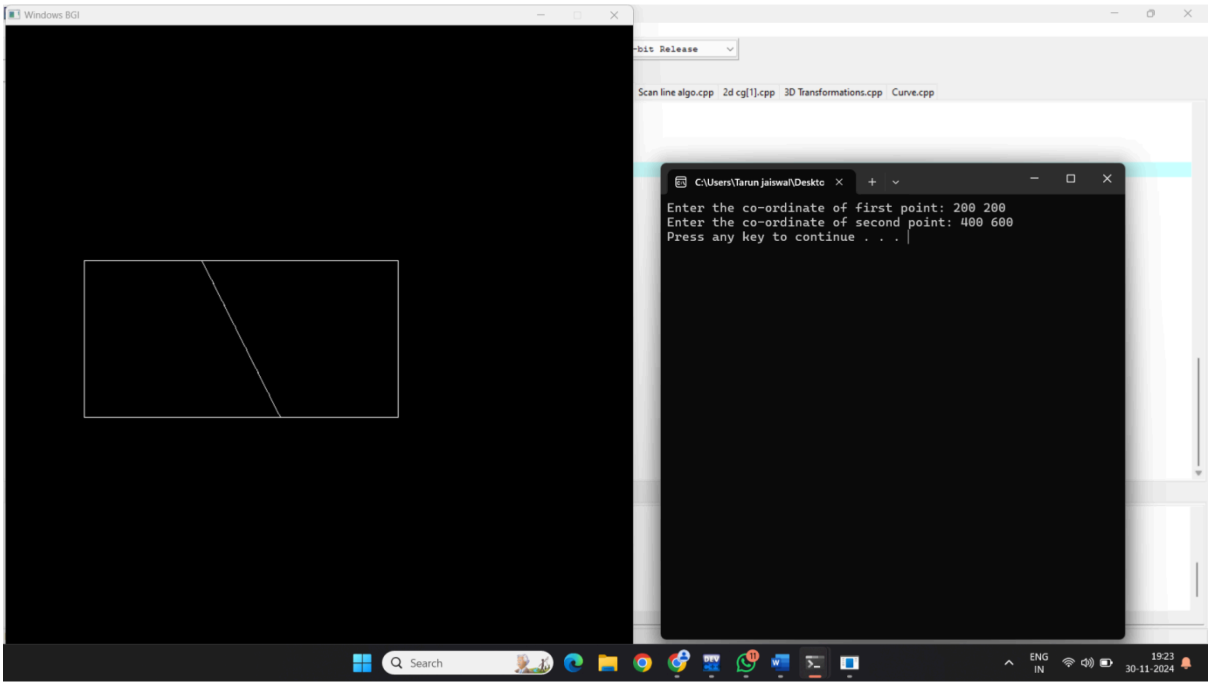
Output:-

Before Cliping:-



```
Enter the co-ordinate of first point: 200 200
Enter the co-ordinate of second point: 400 600
```

After Cliping:-



```
Enter the co-ordinate of first point: 200 200
Enter the co-ordinate of second point: 400 600
Press any key to continue . . .
```

4. Write a program to clip a polygon using Sutherland Hodgemann algorithm.

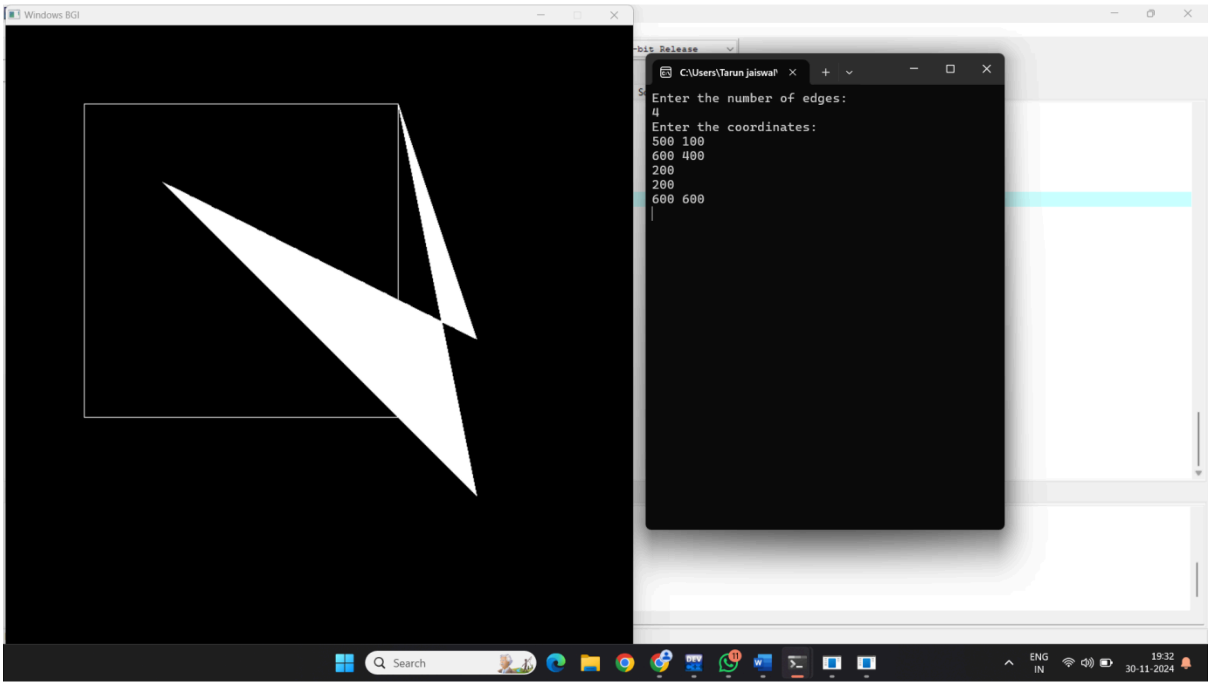
```
#include <iostream>

#include <graphics.h>

using namespace std;

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 >= ymax && y2 < ymin) {
                arr[k] = x1 + m * (ymin - y1);
                arr[k+1] = ymin;
                k += 2;
        }
}
int main() {
        int poly[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>>poly[i];
        poly[i] = poly[0];
        poly[i+1] = poly[1];
        rectangle(xmin, ymax, xmax, ymin);
        fillpoly(n , poly);
        delay(1000);
        cleardevice();
        k = 0;
        for(i =0; i<2*n; i +=2)
        clipLeft(poly[i], poly[i+1], poly[i+2], poly[i+3]);
        n = k/2;
        for(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)
        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);
        if(k)
        fillpoly(k/2,poly);
        system("pause");
        return 0;
 }
```

OUTPUT:-

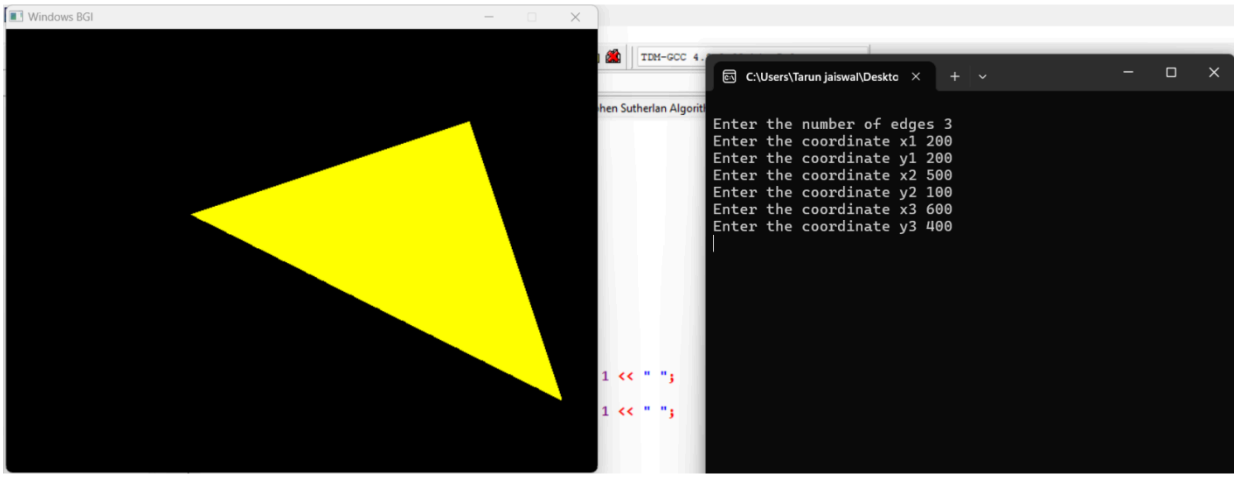


```
Enter the number of edges:
4
Enter the coordinates:
500 100
600 400
200
200
600 600
```

5. 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;

   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];
   }
   a[n][0] = a[0][0];
   a[n][1] = a[0][1];
   initgraph(&gd, &gm, NULL);
   setcolor(YELLOW);
   for (i = 0; i < n; i++)
   {
       line(a[i][0], a[i][1], a[i + 1][0], a[i + 1][1]);
   }
   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;
       if (dx == 0)
           slope[i] = 0.0;
       if ((dy != 0) && (dx != 0))
       {
           slope[i] = (float)dx / dy;
        }
    }
    for (y = 0; y < 400; y++)
    {
        k = 0;
        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++;
            }
        }
        for (j = 0; j < k; j++)
            for (i = 0; i < k; i++)
            {
                if (xi[i] > xi[i + 1])
                {
                    temp = xi[i];
                    xi[i] = xi[i + 1];
                    xi[i + 1] = temp;
                }
            }
        setcolor(YELLOW);
        for (i = 0; i < k; i += 2)
        {
            line(xi[i], y, xi[i + 1] + 1, y);
            temp1 = i;
        }
    }
    delay(7000);
    return 0;
}
```

Output:-



```
Enter the number of edges 3
Enter the coordinate x1 200
Enter the coordinate y1 200
Enter the coordinate x2 500
Enter the coordinate y2 100
Enter the coordinate x3 600
Enter the coordinate y3 400
```

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

```
#include <iostream>
#include <graphics.h>
#include<cmath>
using namespace std;
```

```
int main(){
int tx=2,ty=5;
int window1= initwindow(800,800);
int i,j,k;
float P[2][3];
cout<<"Enter the coordinates of line"<<endl;
for(i=0;i<2;i++){
for(j=0;j<2;j++)
cin>>P[i][j];
P[i][j]=1;
line(P[0][0], P[0][1], P[1][0], P[1][1]);
delay(7000);
float pp[2][3]={0};
int ch;
cout<<"Enter the 2d-transformation"<<endl;
cout<<"1.translation \n 2. shearing \n 3.reflection \n 4.rotation \n S.scaling \n 6.exit"<<endl;
cin>>ch;
switch(ch){
case 1: {
cout<<"Enter the translating factor"<<endl;
cin>>tx>>ty;
int T[3][3]= {{1,0,0},{0,1,0},{tx,ty,1} };
for(i=0;i<2;i++){
```

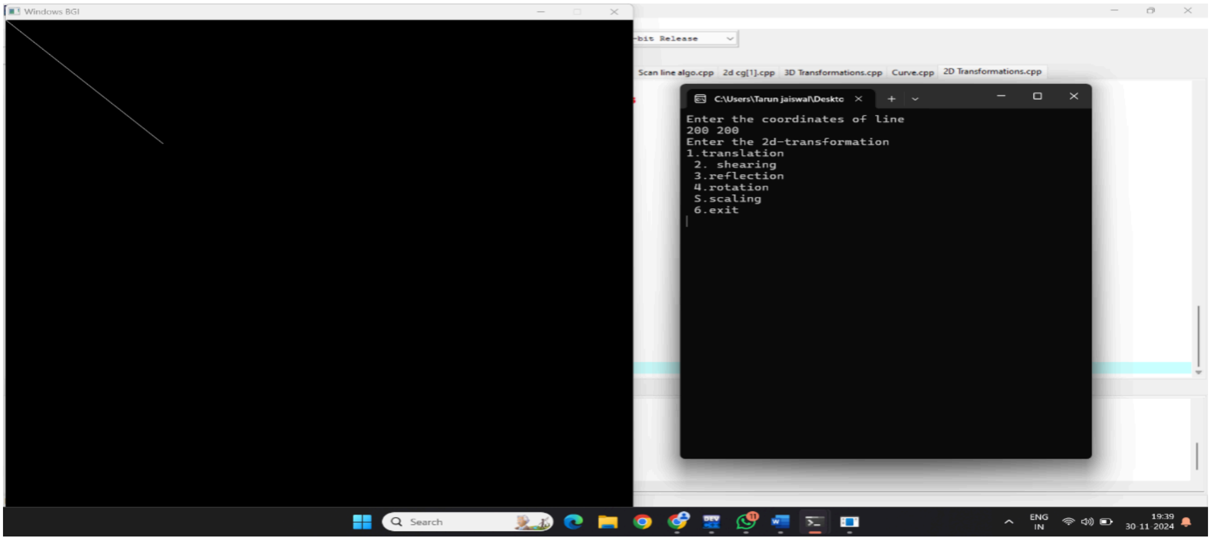
```
for(j=0;j<3;j++)

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:

int sh;

char ax;

cout<<"Enter the shearing axis"<<endl;

cin>>ax;

cout<<"Enter the shearing factor"<<endl;

if(ax=='x'){

cin>>sh;

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

for(i-0;i<2;i++){

for(j-0;j<3;j++)

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");}

if(ax=='y'){

cin>>sh;
```

```
int T[3][3]={{1,sh,0},{0,1,0},{0,0, 1}};
for(i=0;i<2;i++){
for(j=0;j<3;j++)
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:{
int midx,midy, xn1,yn1,xn2,yn2;
char ax;
midx=getmaxx() /2;
midy=getmaxy() /2;
line(0,midy,midx *2,midy);
line(midx,0,midx,midy*2);
cout<<"Enter the axis for reflection"<<endl;
cin>>ax;
if(ax=='y') {
xn1=(midx-P[1][0])+midx;
yn1=P[0][1];
xn2=(midx-P[0][0])+midx;
yn2=P[1][1]; }
if(ax=='x') {
yn1=(midy-P[1][1])-+midy;
xn1=P[0][0];
yn2=(midy-P[0][1 ])+midy;
xn2=P[1][0];
cout<<xn1<<" "<<yn1<<""<<xn2<<" "<<yn2<<endl;
line(xn1,yn1,xn2,yn2);
system("pause");}
break; }
case 4: {

float theta;
cout<<"Enter the theta for rotation"<<endl;
cin>>theta;
float rx;
rx=(theta*3.14)/180;
float T[3][3]={{cos(rx),sin(rx),0},{-sin(rx),cos(rx),0},{0,0,1}};
for(i-0;i<2;i++){
for(j-0;j<3;j++)
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:
int Sx,Sy;
```

```
cout<<"Enter the scaling factor for x-axis"<<endl;
cin>>Sx;
cout<<"Enter the scaling factor for y -axis"<<endl;
cin>>Sy;
int T[3][3]={{Sx,0,1},{0,Sy,1 },{0,0, 1}};
for(i=0;i<2;i++){
for(j=0;j<3;j++)
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;
}}
return 0;}
```

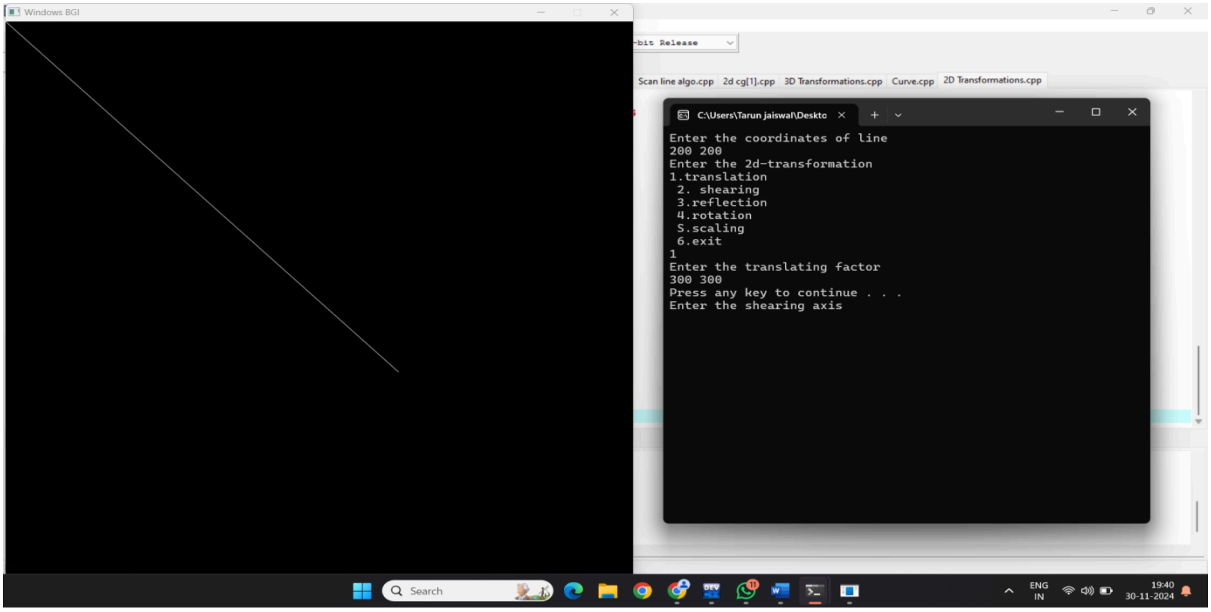
Output:-

Before:-



```
Enter the coordinates of line
200 200
Enter the 2d-transformation
1.translation
 2. shearing
 3.reflection
 4.rotation
 5.scaling
 6.exit
```

After:-



```
Enter the coordinates of line
200 200
Enter the 2d-transformation
1.translation
 2. shearing
 3.reflection
 4.rotation
 5.scaling
 6.exit
1
Enter the translating factor
300 300
Press any key to continue . . .
Enter the shearing axis
```

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

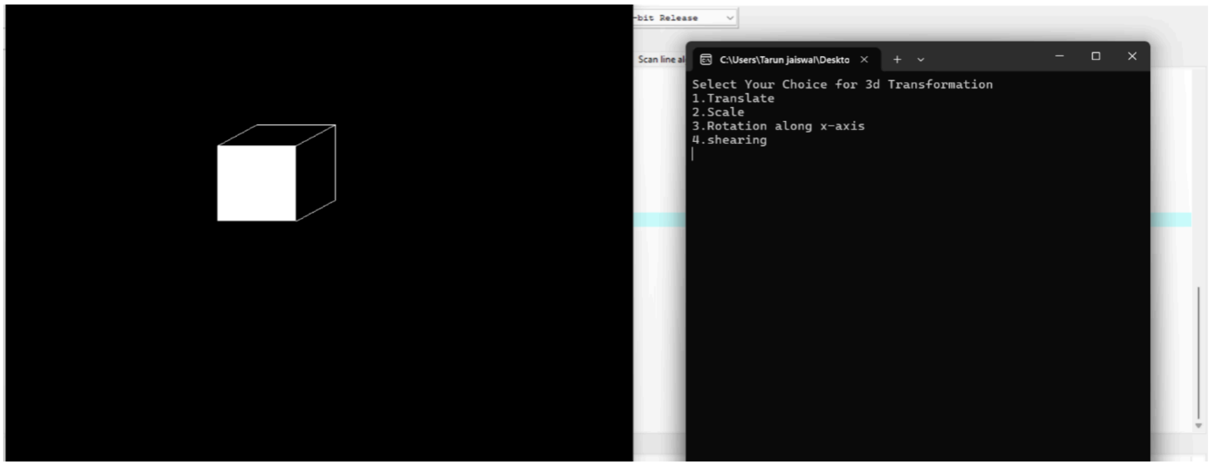
```
#include<iostream>

#include<graphics.h>

#include<cmath>

using namespace std;


int main(){

int window1 = initwindow(800,800);

bar3d(270,200,370,300,50,5);

int ch,i,j,k;


cout<<"Select Your Choice for 3d Transformation\n";

cout<<"1.Translate\n2.Scale\n3.Rotation along x-axis\n4.shearing\n";

cin>>ch;
```

```
cleardevice();
switch(ch){
case 1:{
int tx,ty;
cout<<"Enter the translation factor for x,y axis"<<endl;
cin>>tx>>ty;
bar3d(270+tx,200+ty,370+tx,300+ty,50,5);
delay(7000);
cleardevice();
outtextxy(10,20,"Parallel projection side view");
bar3d(0,200+ty,0,300+ty,50,5);
delay(7000);
delay(7000);
break;
}
case 2:{
int sx,sy;
cout<<"Enter the scaling factor for x,y axis"<<endl;
cin>>sx>>sy;
bar3d(270*sx,200*sy,370*sx,300*sy,50,5);
delay(7000);
cleardevice();
outtextxy(10,20,"Parallel projection side view");
bar3d(0,200*sy,0,300*sy,50,5);
delay(7000);
break;
}
case 4:{
int shx,shy;
cout<<"Enter the shearing factor for x,y axis"<<endl;
cin>>shx>>shy;
bar3d(270,200+(shy*270),370,300+(shy*50),50+(270*shx),5);
delay(7000);
break;
}
case 3:{
int ang;
cout<<"Enter the rotation angle"<<endl;
cin>>ang;
ang=(ang*3.14)/180;
int x1= 200*cos(ang)-50*sin(ang);
int y1= 50*cos(ang)+200*sin(ang);
int x2=300*cos(ang)-500*sin(ang);
int y2= 50*cos(ang)+300*sin(ang);
bar3d(x1,y1,x2,y2,50,5);
delay(7000);
break;
}
```

```
}
return 0;
}
```

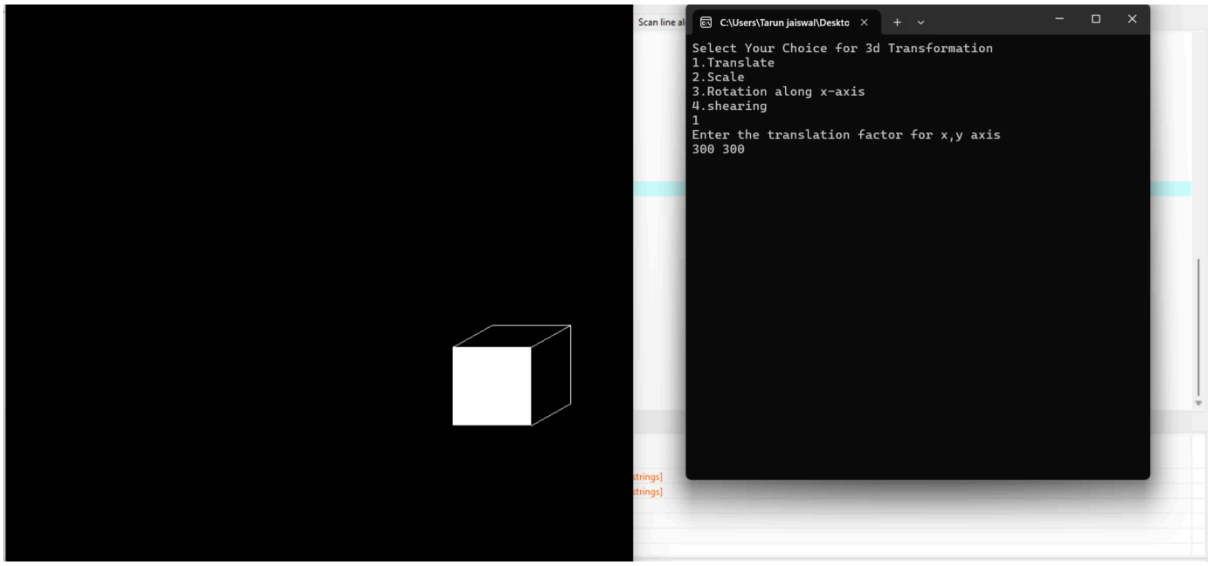
Output:-

Before Transformation:-



```
Select Your Choice for 3d Transformation
1.Translate
2.Scale
3.Rotation along x-axis
4.shearing
|
```

After Transformation:-



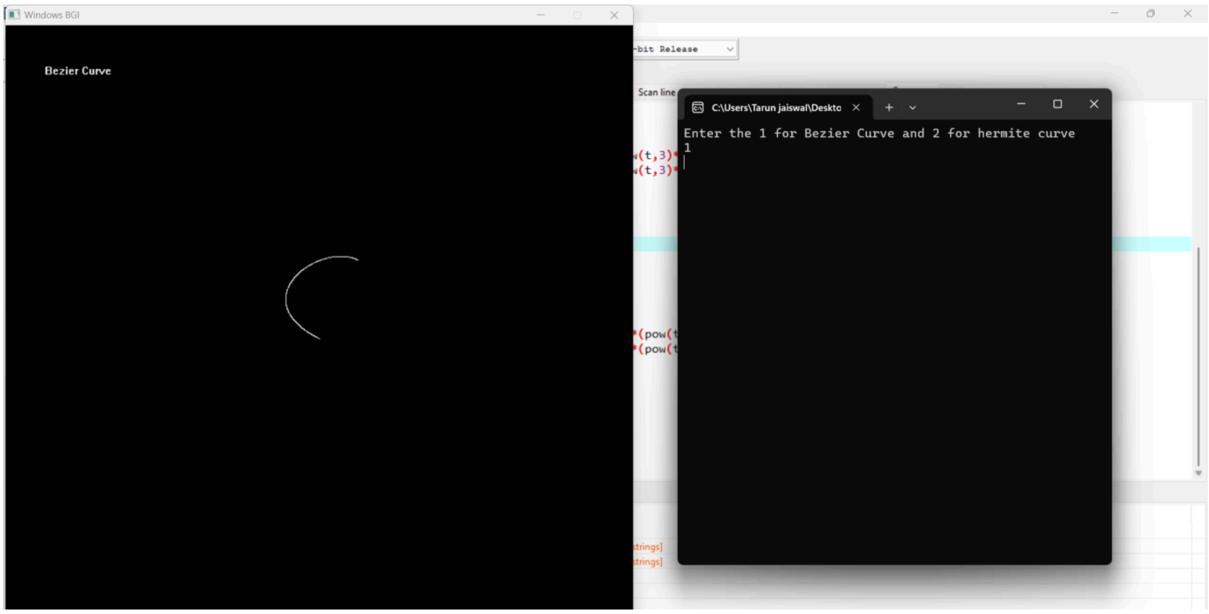
```
Select Your Choice for 3d Transformation
1.Translate
2.Scale
3.Rotation along x-axis
4.shearing
1
Enter the translation factor for x,y axis
300 300
```

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

```
#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};
outtextxy(50,50,"Bezier Curve");
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};
outtextxy(50,50,"Hermite Curve");
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:-



```
Enter the 1 for Bezier Curve and 2 for hermite curve
1
```