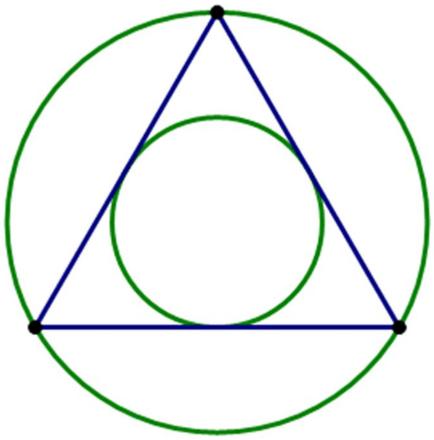
COMPUTER GRAPHICS

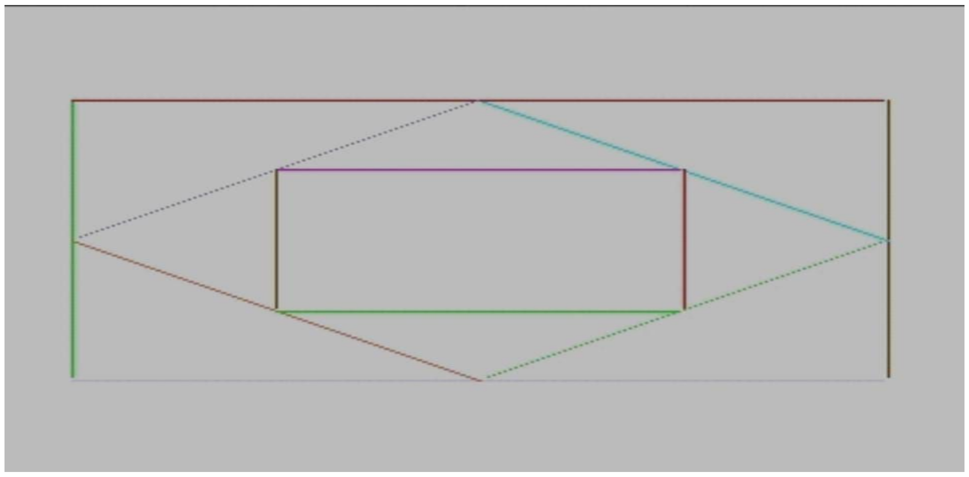
GROUP-A EXPERIMENT NUMBER:-1(A)

- ***Experiment Name:-*** *DDA Line and Bresenham's Circle Algorithm.*
- ***Aim:-*** *Write C++ program to draw the following pattern, using DDA Line and Bresenham's Circle Algorithm. Apply the concept of encapsulation.*

Pattern 1-



Pattern 2-



- ***Objective:-*** *Understanding the DDA Line and Bresenham's Circle Algorithm, And apply to make the pattern.*

- ***Theory:-*** *DDA Line Algorithm-*

The Digital Differential Analyzer (DDA) generates lines from their differential equations. The equation of a straight line is

$$\frac{dy}{dx} = \frac{\Delta y}{\Delta x}$$

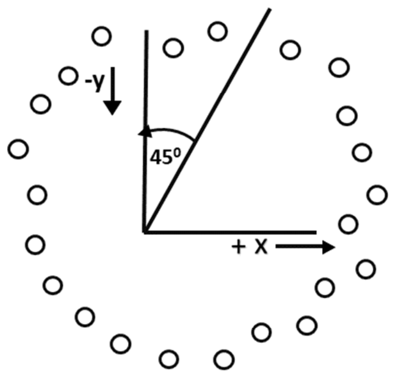
The DDA works on the principle that we simultaneously increment x and y by small steps proportional to the first derivatives of x and y. In this case of a straight line, the first derivatives are constant and are proportional to Δx and Δy . Therefore, we could generate a line by incrementing x and y by $\epsilon\ \Delta x$ and $\epsilon\ \Delta y$, where ϵ is some small quantity. There are two ways to generate points

1. By rounding to the nearest integer after each incremental step, after rounding we display dots at the resultant x and y.

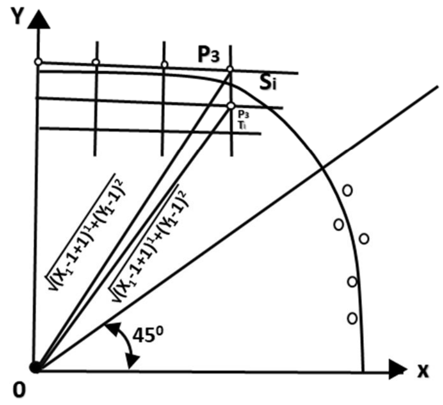
2. An alternative to rounding the use of arithmetic overflow: x and y are kept in registers that have two parts, integer and fractional. The incrementing values, which are both less than unity, are repeatedly added to the fractional parts and whenever the results overflows, the corresponding integer part is incremented. The integer parts of the x and y registers are used in plotting the line. In the case of the symmetrical DDA, we choose $\varepsilon=2^{-n}$,where $2^{n-1} \leq max(|\Delta x|, |\Delta y|) < 2^{\pi}$

Bresenham's Circle Algorithm-

Scan-Converting a circle using Bresenham's algorithm works as follows: Points are generated from 90° to 45°, moves will be made only in the +x & -y directions as shown in fig:



The best approximation of the true circle will be described by those pixels in the raster that falls the least distance from the true circle. We want to generate the points from



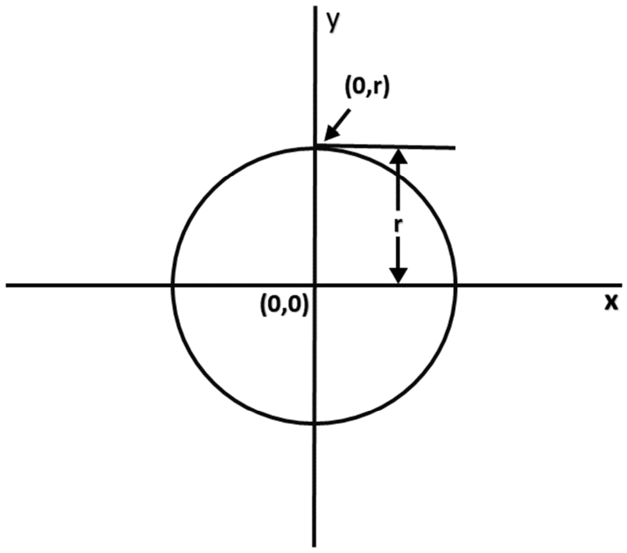
90° to 45°. Assume that the last scan-converted pixel is P_1 as shown in fig. Each new point closest to the true circle can be found by taking either of two actions.

1. *Move in the x-direction one unit or*
2. *Move in the x- direction one unit & move in the negative y-direction one unit.*

Let D (S_i) is the distance from the origin to the true circle squared minus the distance to point P_3 squared. D (T_i) is the distance from the origin to the true circle squared minus the distance to point P_2 squared. Therefore, the following expressions arise.

$$D(S_i)=(x_{i-1}+1)^2+y_{i-1}^2-r^2$$
$$D(T_i)=(x_{i-1}+1)^2+(y_{i-1}-1)^2-r^2$$

Since D (S_i) will always be +ve & D (T_i) will always be -ve, a decision variable d may be defined as follows:



$d_i=D(S_i)+D(T_i)$

Therefore,
$d_i=(x_{i-1}+1)^2+y_{i-1}^2-r^2+(x_{i-1}+1)^2+(y_{i-1}-1)^2-r^2$

From this equation, we can drive initial values of d_i as

If it is assumed that the circle is centered at the origin, then at the first step $x = 0$ & $y = r$.

Therefore,

$d_i=(0+1)^2+r^2 -r^2+(0+1)^2+(r-1)^2-r^2$

$=1+1+r^2-2r+1-r^2$

$= 3 - 2r$

Thereafter, if d_i<0,then only x is incremented.

$x_{i+1}=x_{i+1}$ $\quad d_{i+1}=d_i+ 4x_i+6$

& if $d_i \geq 0$, then x & y are incremented

$x_{i+1}=x_{i+1}$ $\quad y_{i+1} =y_i+ 1$

$d_{i+1}=d_i+ 4 (x_i-y_i)+10$

- ## **Algorithm:-** DDA Line Drawing Algorithm-

```
Step1: Start Algorithm

Step2: Declare $x_1,y_1,x_2,y_2,dx,dy,x,y$ as integer variables.

Step3: Enter value of $x_1,y_1,x_2,y_2$.

Step4: Calculate $dx = x_2-x_1$

Step5: Calculate $dy = y_2-y_1$

Step6: If ABS (dx) > ABS (dy)
        Then step = abs (dx)
        Else

Step7: $x_{inc}=dx/step$
       $y_{inc}=dy/step$
       assign $x = x_1$
       assign $y = y_1$

Step8: Set pixel (x, y)

Step9: $x = x + x_{inc}$
       $y = y + y_{inc}$
       Set pixels (Round (x), Round (y))

Step10: Repeat step 9 until $x = x_2$

Step11: End Algorithm
```

Bresenham's Circle Algorithm-

```
Step1: Start Algorithm

Step2: Declare p, q, x, y, r, d variables
        p, q are coordinates of the center of the circle
        r is the radius of the circle

Step3: Enter the value of r

Step4: Calculate d = 3 - 2r

Step5: Initialize       x=0
        &nbsy= r

Step6: Check if the whole circle is scan converted
        If x > = y
        Stop

Step7: Plot eight points by using concepts of eight-way symmetry. The center is at
(       (p,q).Current active pixel is (x, y).
         putpixel (x+p, y+q)
         putpixel (y+p, x+q)
         putpixel (-y+p, x+q)
         putpixel (-x+p, y+q)
         putpixel (-x+p, -y+q)
         putpixel (-y+p, -x+q)
         putpixel (y+p, -x+q)
         putpixel (x+p, -y-q)

Step8: Find location of next pixels to be scanned
         If d < 0
         then d = d + 4x + 6
          increment x = x + 1
         If d ≥ 0
         then d = d + 4 (x - y) + 10
          increment x = x + 1
          decrement y = y - 1

Step9: Go to step 6

Step10: Stop Algorithm
```

- **Program 1 (For Pattern 1):-**

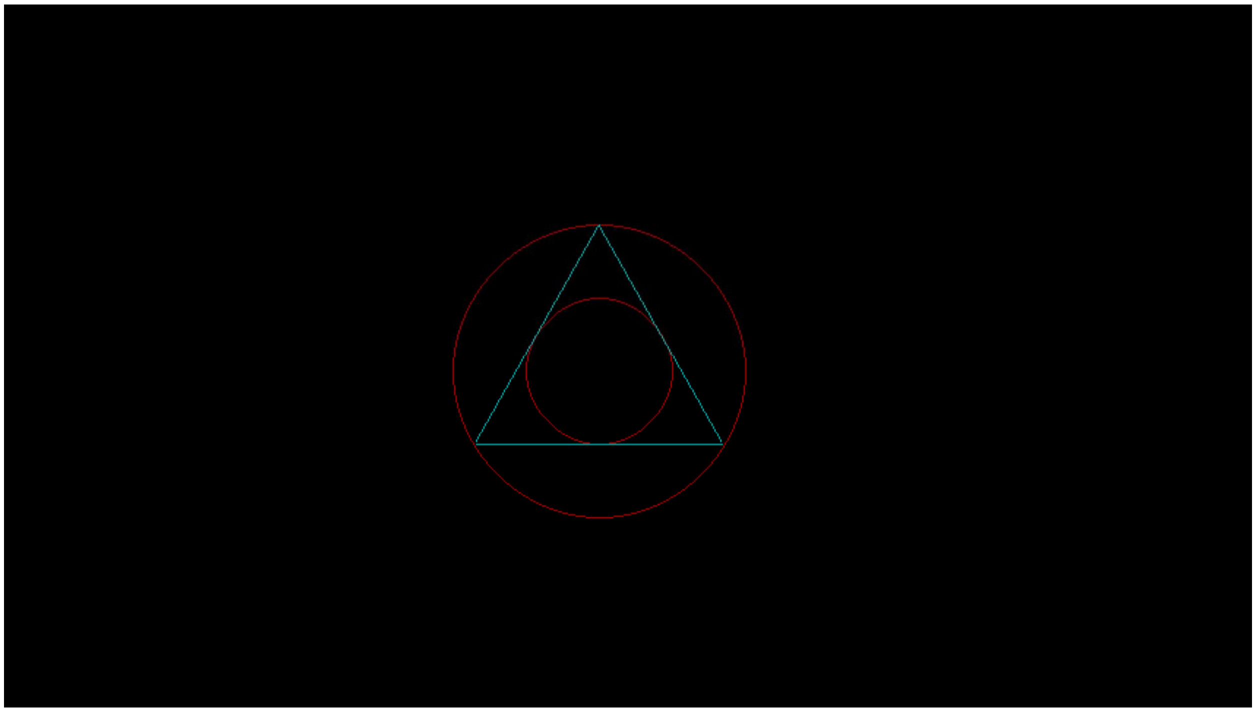
```
#include<graphics.h>
#include<conio.h>
#include<math.h>
#include<iostream.h>
int sign(int x)
{
   if(x>0)
        return 1;
   else if(x<0)
        return -1;
   else
        return 0;
}

void dda(int x1,int y1,int x2,int y2)
{
   float x,y,i,dx,dy,l;
   if(x1==x2 && y1==y2)
   {
        putpixel(x1,y1,3);
   }
```

```
else
{
    dx=abs(x2-x1);
    dy=abs(y2-y1);
    if(dx>dy)
        l=dx;
    else
        l=dy;
    dx=(x2-x1)/l;
    dy=(y2-y1)/l;
    x=x1+0.5*sign(dx);
    y=y1+0.5*sign(dy);
    i=1;
    while(i<l)
    {
        putpixel(x,y,3);
        x=x+dx;
        y=y+dy;
        i++;
    }
}
```

```
}

void show(int x1,int y1,int x,int y)
{
        putpixel(x1+x,y1+y,4);
        putpixel(x1-x,y1+y,4);
        putpixel(x1+x,y1-y,4);
        putpixel(x1-x,y1-y,4);
        putpixel(x1+y,y1+x,4);
        putpixel(x1-y,y1+x,4);
        putpixel(x1+y,y1-x,4);
        putpixel(x1-y,y1-x,4);
}

void b_circle(int x1,int y1,int r)
{
   int d;
   d=3-2*r;
   int x=0,y=r;

   show(x1,y1,x,y);
```

```
    while(y>=x)
    {
        x++;
        if(d>0)
        {
            y--;
            d=d+4*(x-y)+10;
        }
        else
        {
            d=d+4*x+6;
        }
        show(x1,y1,x,y);
    }
}


void main()
{
    clrscr();
    int x1,x2,y1,y2,col;
    int gd=DETECT,gm;
    initgraph(&gd,&gm,"C:\\turboc3\\bgi");

    b_circle(300,250,100);
    b_circle(300,250,50);

    dda(300,150,385,300);
    dda(300,150,215,300);
    dda(385,300,215,300);



    getch();
    closegraph();
}
```

- ***Output:-***



- ***Program 2 (For Pattern 2):-***

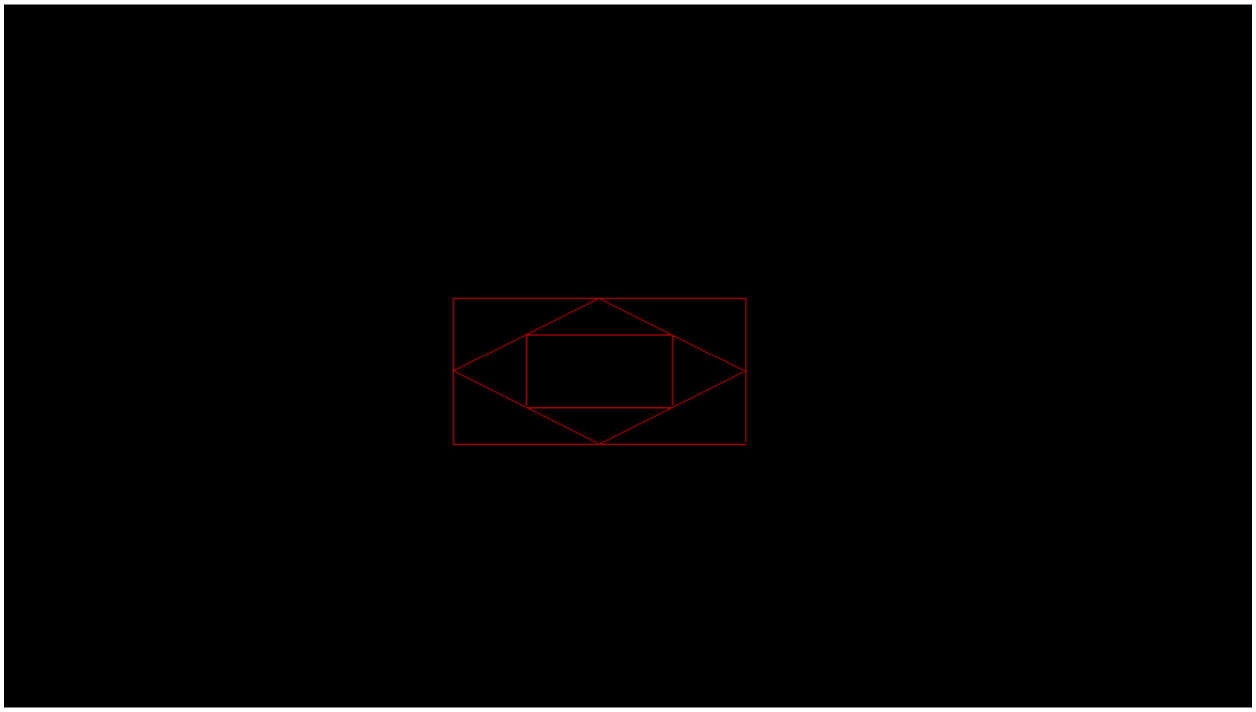
```
#include<graphics.h>
#include<conio.h>
#include<math.h>
#include<iostream.h>
int sign(int x)
{
 if(x>0)
   return 1;
 else if(x<0)
    return -1;
 else
    return 0;
}
void dda(int x1,int y1,int x2,int y2)
```

```
{
 float x,y,l,i,dx,dy;
 if(x1==x2 && y1==y2)
 {
   putpixel(x1,y1,4);
 }
 else
 {
   dx=abs(x2-x1);
   dy=abs(y2-y1);
   if(dx>=dy)
     l=dx;
   else
      l=dy;
   dx=(x2-x1)/l;
   dy=(y2-y1)/l;
   x=x1+0.5*sign(dx);
   y=y1+0.5*sign(dy);
   i=1;
   while(i<l)
   {
     putpixel(x,y,4);
      x=x+dx;
      y=y+dy;
      i++;
    }
 }
}
```

```
void bla(int x1,int y1,int x2,int y2)
{
 float dx,dy,x,y,e,i;
 if(x1==x2 && y1==y2)
    putpixel(x1,y1,4);
 else
 {
    dx=abs(x2-x1);
    dy=abs(y2-y1);
    x=x1;
    y=y1;
    putpixel(x,y,4);
    e=2*dy-dx;
    i=1;
    while(i<=dx)
    {
      while(e>=0)
      {
         y=y+1;
         e=e-2*dx;
       }
       x=x+1;
       e=e+2*dy;
       putpixel(x,y,4);
       i=i+1;
    }
 }
}
```

```
void main()
{
 clrscr();
 int gd=DETECT,gm;
 initgraph(&gd,&gm,"C:\\turboc3\\bgi");

 bla(200,300,400,300);
 dda(200,300,200,200);
 bla(200,200,400,200);
 dda(400,200,400,300);

 dda(200,250,300,200);
 dda(200,250,300,300);
 bla(300,200,400,250);
 dda(300,300,400,250);

 bla(250,225,150,225);
 dda(250,225,250,275);
 dda(350,275,250,275);
 dda(350,225,350,275);

 getch();
 closegraph();
}
```

- ***Output:-***



- ***Conclusion:-*** *Thus, we have studied DDA and Bresenham's Line drawing algorithm and we have implemented them to draw the above patters.*

COMPUTER GRAPHICS

GROUP-A	EXPERIMENT NUMBER:-2

- ***Experiment Name:-*** *Scan Fill Algorithm*
- ***Aim:-*** *Write a c++ program to draw a concave polygon and fill it with desired color using scan fill algorithm.*
- ***Objective:-***
  1. *To learn to draw a concave polygon in c++.*
  2. *To understand and implement scan fill algorithm.*
- ***Theory:-***

Scan Line Polygon Fill Algorithm:-

This algorithm lines interior points of a polygon on the scan line and these points are done on or off according to requirement. The polygon is filled with various colors by coloring various pixels. Scanline filling is basically filling up of polygons using horizontal lines orscanlines. The purpose of the SLPF algorithm is to fill (color) the interior pixels of a polygon given only the vertices of the figure. To understand Scanline,think ofthe image being drawn by a single pen starting from bottom left, continuing to the right, plotting only points where there is a point present in the image, and when the line is complete, start from the next line and continue. This algorithmworks by intersecting scanline withpolygon edges and fillsthe polygon between pairs of intersections.

Special cases of polygon vertices:-

1. If both lines intersecting at the vertex are on the same side of the scanline, consider it as two points.
2. If lines intersecting at the vertex are at opposite sides of the scanline, consider it as only one point.

- ***Algorithm:-***

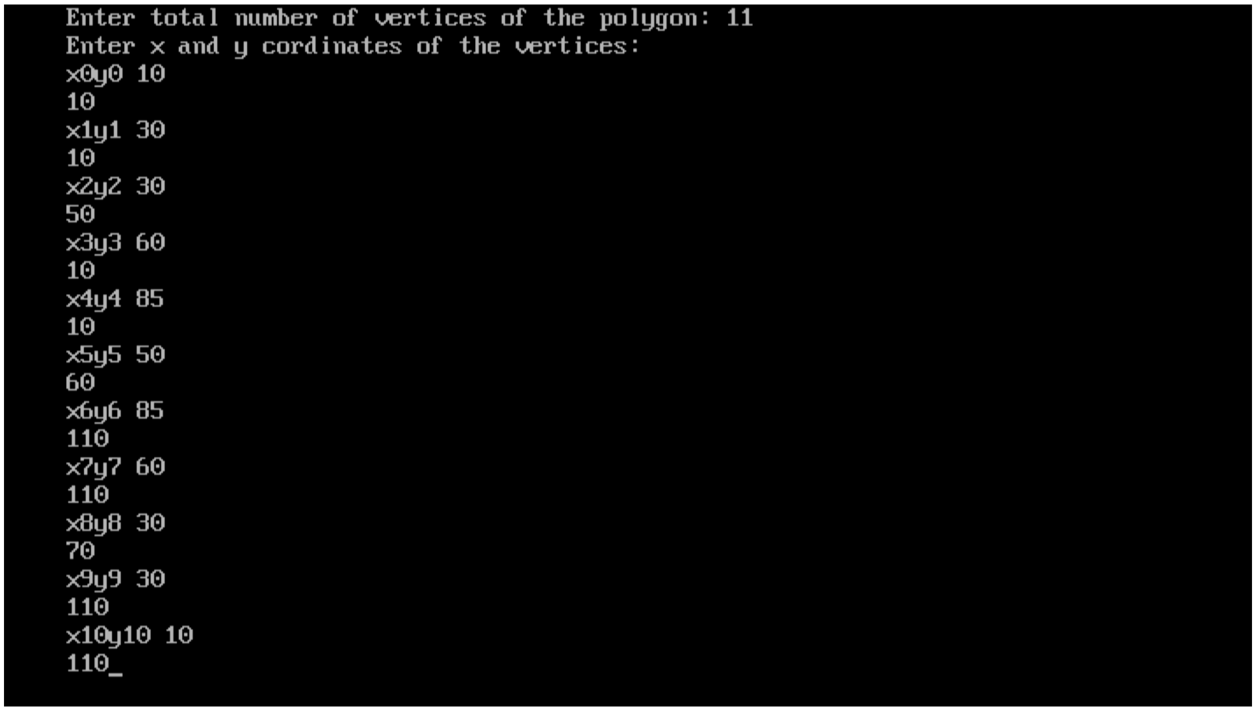
  1. We will process the polygon edge after edge, and store in the edge Table.
  2. Storing is done by storing the edge in the same scanline edge tuple as the lowermost point's y-coordinate value of the edge.
  3. After addition of any edge in an edge tuple, the tuple is sorted using insertion sort, according to its x of y min value.
  4. After the whole polygon is added to the edge table, the figure is now filled. 5. Filling is started from the first scanline at the bottom, and continued till the top.
  6. Now the active edge table is taken and the following things are repeated for each scanline:
  I . Copy all edge buckets of the designated scanline to the active edge tuple
  II . Perform an in sertion sort according to the x of y min values
  III . Remove all edge buckets whose y max is equal or greater than the scanline
  IV . Fill up pairs of edges in active tuple, if any vertex is got, follow these instructions: o If both lines intersecting at the vertex are on the same side of the scanline, consider it astwo points. o If lines intersecting at the vertex are at opposite sides of the scanline, consider it as only one point. v. Update the x of y min by adding slopeinverse for each bucket.

- ***Program:-***

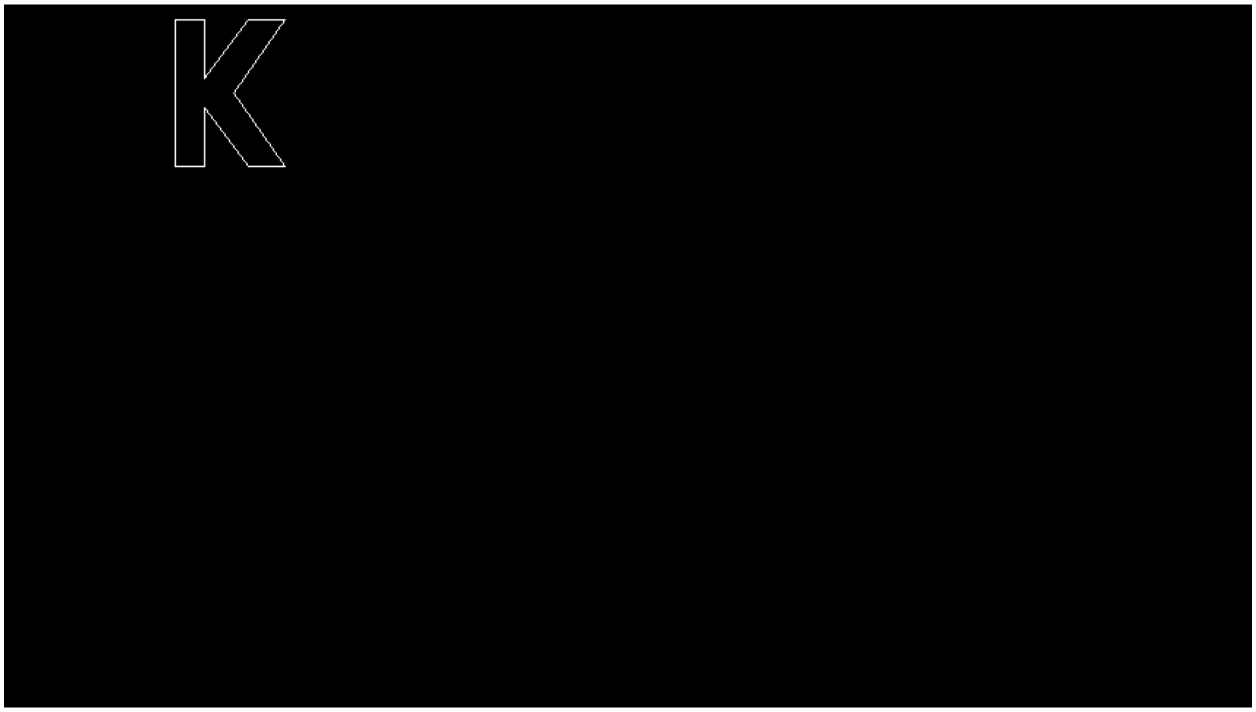
```
#include<graphics.h>
#include<conio.h>
#include<iostream.h>
#include<dos.h>
void main()
{
 clrscr();
 int gd=DETECT,gm,dx,dy,x,y,temp,n,i,j,k;
 int p[20][2],xi[20];
 float slope[20];
  cout<<"Enter total number of vertices of the polygon: ";
  cin>>n;
  cout<<"Enter x and y cordinates of the vertices: "<<endl;
 for(i=0;i<n;i++)
  {
cout<<"x"<<i<<"y"<<i<<" ";
cin>>p[i][0]>>p[i][1];
  }
  p[n][0]=p[0][0];
  p[n][1]=p[0][1];
  initgraph(&gd,&gm,"C:\\turboc3\\bgi");
 for(i=0;i<n;i++)
  {
line(p[i][0],p[i][1],p[i+1][0],p[i+1][1]);
  }
  getch();
 for(i=0;i<n;i++)
  {
dx=p[i+1][0]-p[i][0];
dy=p[i+1][1]-p[i][1];
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=480;y>0;y--)
 {
k=0;
for(i=0;i<n;i++)
{
 if(((p[i][1]<=y)&&(p[i+1][1]>y))||((p[i][1]>y)&&(p[i+1][1]<=y)))
 {
xi[k]=(int)(p[i][0]+slope[i]*(y-p[i][1]));
k++;
 }
}
for(j=0;j<k-1;j++)
 for(i=0;i<k-1;i++)
 {
if(xi[i]>xi[i+1])
{
 temp=xi[i];
 xi[i]=xi[i+1];
 xi[i+1]=temp;
}
 }
 setcolor(11);
for(i=0;i<k;i+=2)
{
 line(xi[i],y,xi[i+1],y);
 delay(20);
}
 }
```

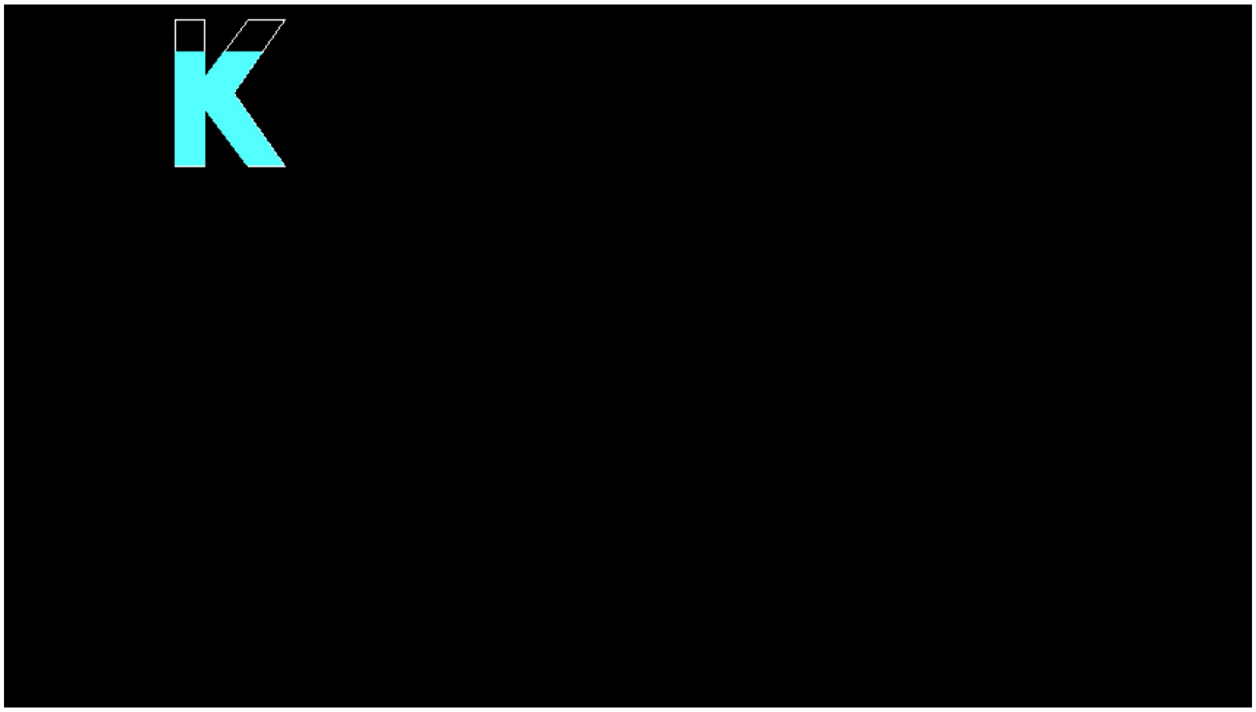
```
getch();
closegraph();
}
```

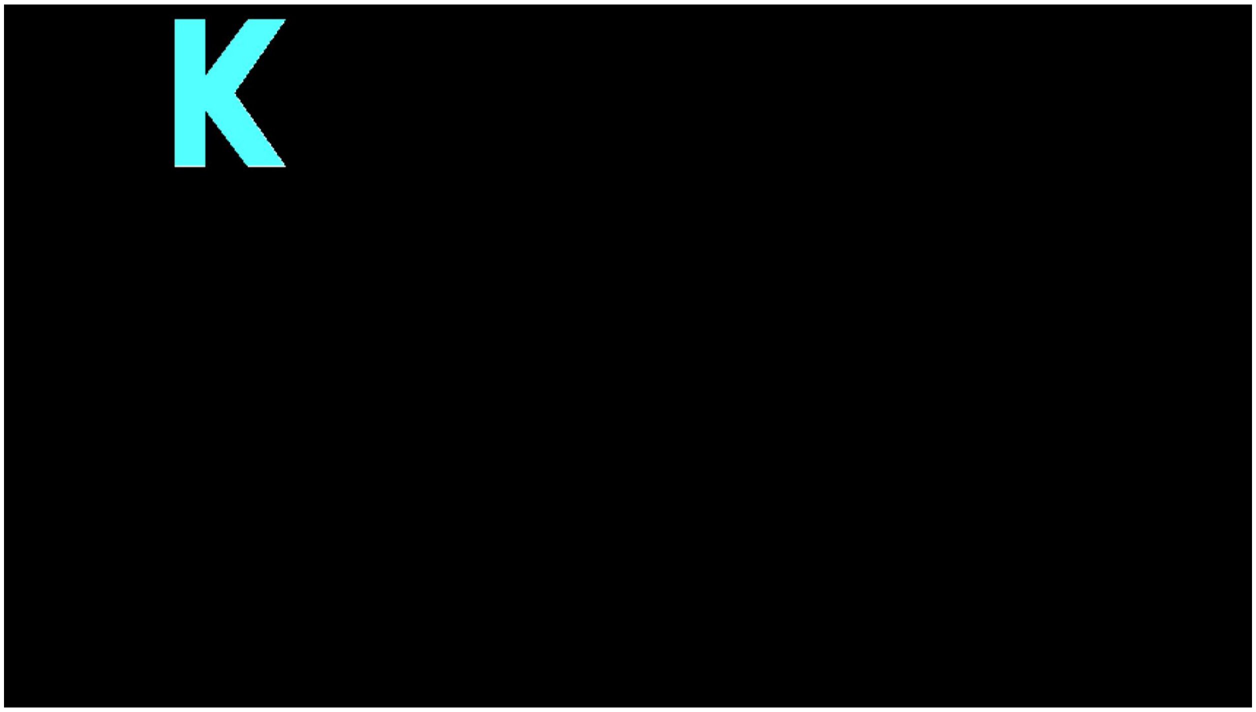
- ***Output:-***



```
Enter total number of vertices of the polygon: 11
Enter x and y cordinates of the vertices:
x0y0 10
10
x1y1 30
10
x2y2 30
50
x3y3 60
10
x4y4 85
10
x5y5 50
60
x6y6 85
110
x7y7 60
110
x8y8 30
70
x9y9 30
110
x10y10 10
110_
```







- ***Conclusion:-*** *Hence, we have filled a concave polygon using scan fill algorithm.*

COMPUTER GRAPHICS

GROUP-A EXPERIMENT NUMBER:-3

- ***Experiment Name:-*** *Cohen Southerland Line Clipping Algorithm*
- ***Aim:-*** *Write a C++ program to implement Cohen Southerland Line Clipping Algorithm.*
- ***Objective:-*** *To study and implement Cohen Southerland Line Clipping Algorithm.*
- ***Theory:-***
- ***Algorithm:-***

  Step1 : Calculate positions of both endpoints of the line
  Step2 : Perform OR operation on both of these end-points
  Step3 : If the OR operation gives 0000 Then line is considered to be visible else Perform AND operation on both endpoints If And ≠ 0000 then the line is invisible else And=0000 Line is considered the clipped case.
  Step4:If a line is clipped case, find an intersection with boundaries of the window m=(y2-y1)(x2-x1)

  (a) If bit 1 is "1" line intersects with left boundary of rectangle window y3=y1+m(x-X1) where X = Xwmin where Xwminis the minimum value of X co-ordinate of window

  (b) If bit 2 is "1" line intersect with right boundary y3=y1+m(X-X1) where X = Xwmax where X more is maximum value of X co-ordinate of the window

  (c) If bit 3 is "1" line intersects with bottom boundary X3=X1+(y-y1)/m where y = ywmin ywmin is the minimum value of Y co-ordinate of the window

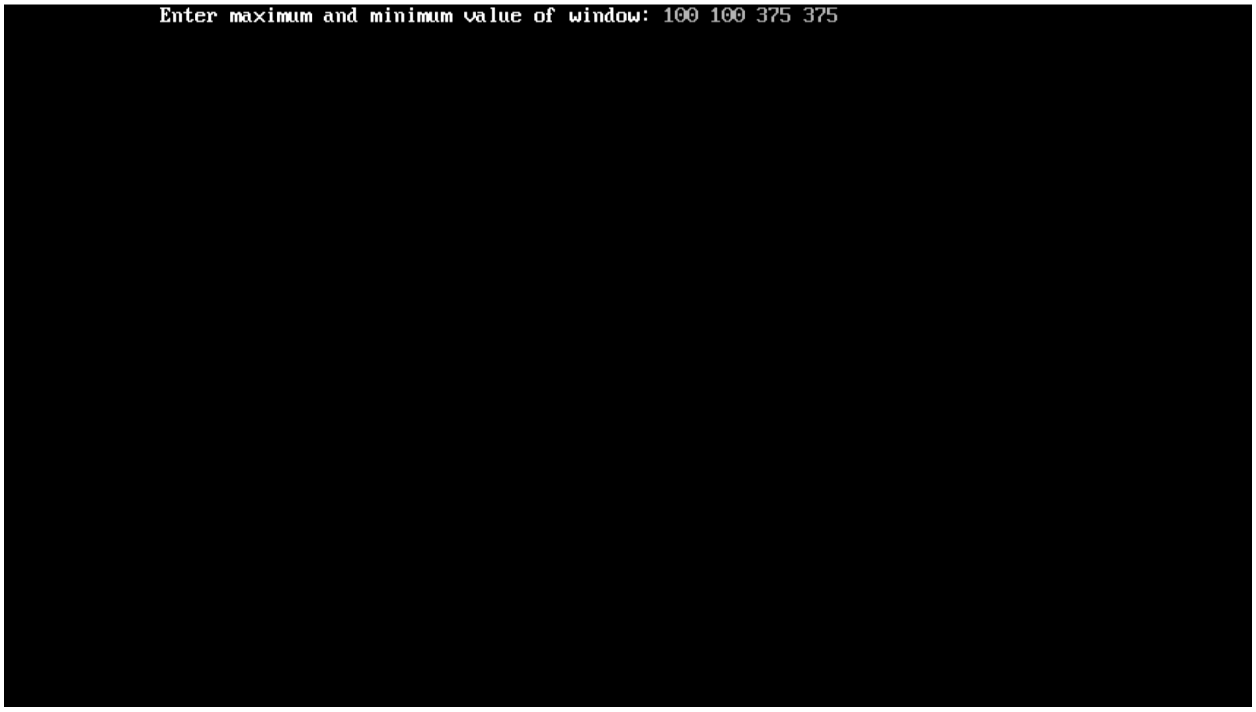
(d) If bit 4 is "1" line intersects with the top boundary X3=X1+(y-y1)/m where y = ywmax ywmax is the maximum value of Y co-ordinate of the window

- **Program:-**

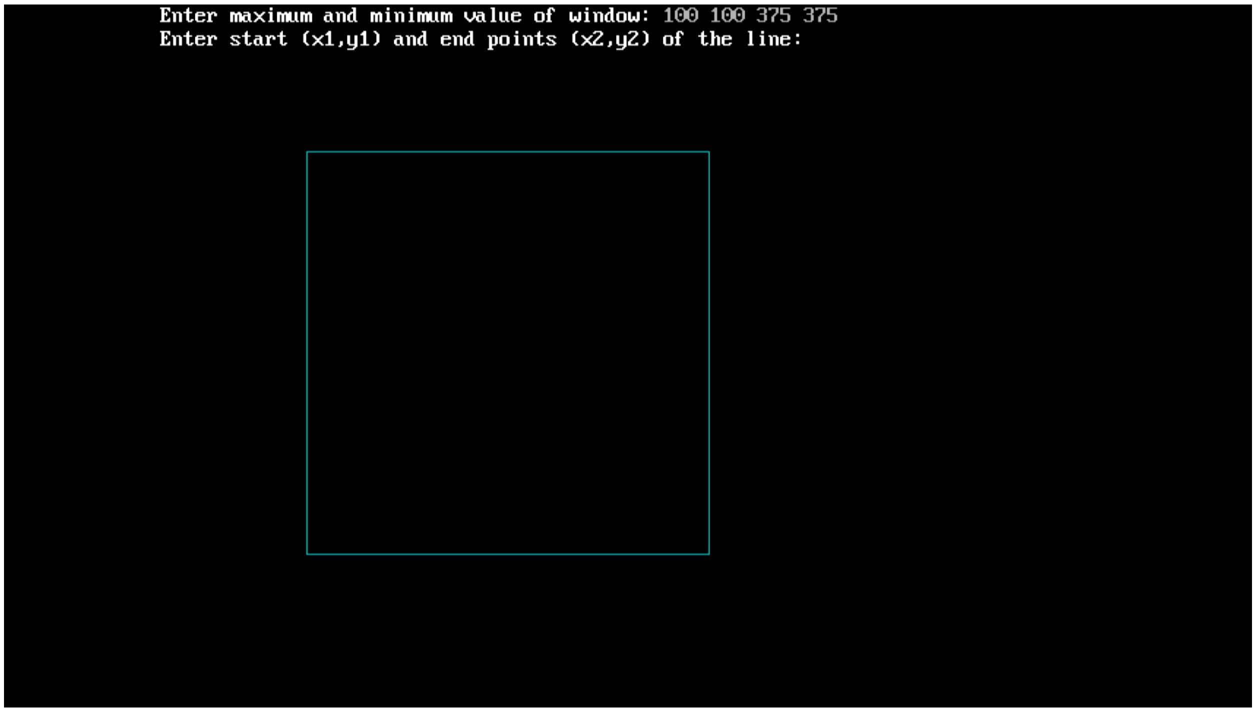
```
 #include<graphics.h>
 #include<iostream.h>
 #include<conio.h>
 #include<stdlib.h>
static int LEFT=1,RIGHT=2,BOTTOM=4,TOP=8,xmax,ymax,xmin,ymin;
int find_code(int x,int y)
{
 int code=0;
 if(y>ymax)
code|=TOP;
 if(y<ymin)
code|=BOTTOM;
 if(x>xmax)
code|=RIGHT;
 if(x<xmin)
code|=LEFT;
 return code;
}
void main()
{
 clrscr();
 int x1,y1,x2,y2;
 int gd=DETECT,gm;
 initgraph(&gd,&gm,"C:\\turboc3\\bgi");
 setcolor(CYAN);
 cout<<"Enter maximum and minimum value of window: ";
 cin>>xmin>>ymin>>xmax>>ymax;
 rectangle(xmin,ymin,xmax,ymax);
 cout<<"Enter start (x1,y1) and end points (x2,y2) of the line: ";
 cin>>x1>>y1>>x2>>y2;
 line(x1,y1,x2,y2);
 getch();
 int ocode1=find_code(x1,y1),ocode2=find_code(x2,y2);
 int accept=0;
 while(1)
 {
float m=(float)(y2-y1)/(x2-x1);
if(ocode1==0 && ocode2==0)
{
 accept=1;
 break;
}
else if((ocode1&ocode2)!=0)
{
 break;
}
else
{
 int x,y;
 int temp;
 if(ocode1==0)
 {
temp=ocode2;
 }
 else
 {
temp=ocode1;
 }
 if(temp&TOP)
 {
x=x1+(ymax-y1)/m;
y=ymax;
 }
 else if(temp&BOTTOM)
 {
x=x1+(ymin-y1)/m;
y=ymin;
 }
 else if(temp&LEFT)
 {
x=xmin;
y=y1+m*(xmin-x1);
 }
 else if(temp&RIGHT)
 {
x=xmax;
y=y1+m*(xmax-x1);
 }
 if(temp==ocode1)
 {
x1=x;
y1=y;
ocode1=find_code(x1,y1);
 }
 else
 {
x2=x;
y2=y;
ocode2=find_code(x2,y2);
```

```
 }
}
 }
 setcolor(RED);
 cout<<"After clipping";
 if(accept)
 {
line(x1,y1,x2,y2);
rectangle(xmin,ymin,xmax,ymax);
 }
 getch();
 closegraph();
}
```

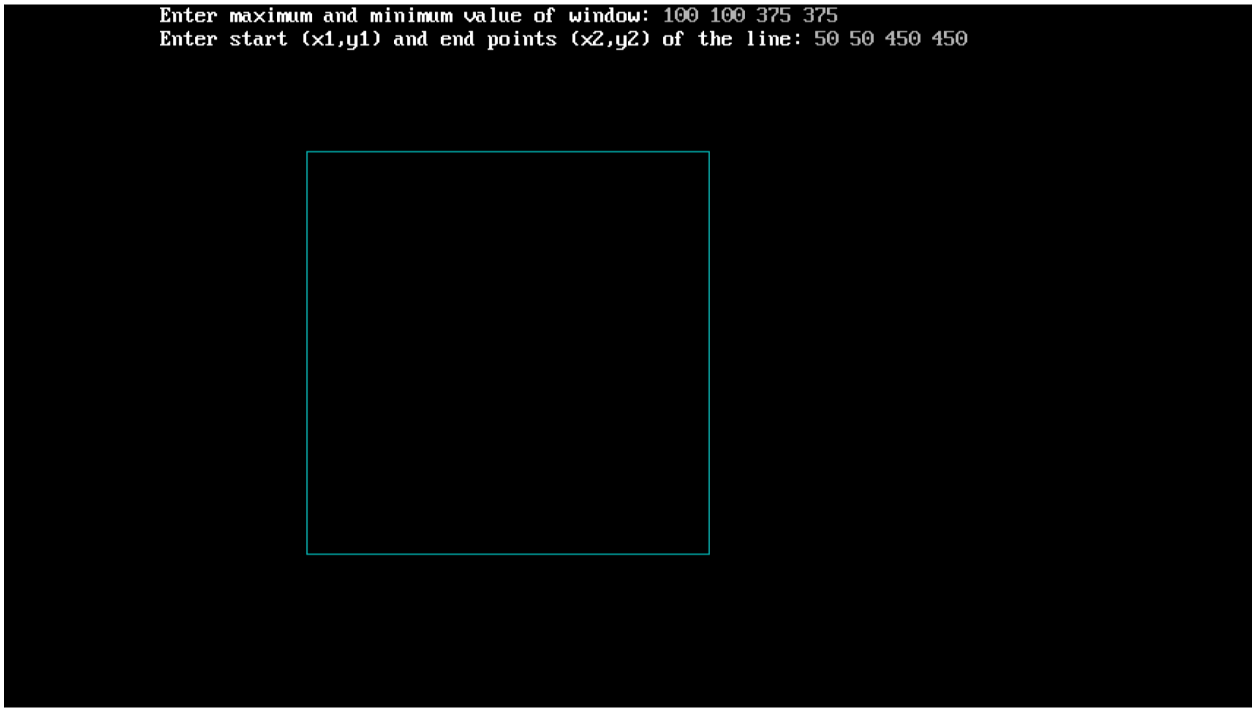
- ***Output:-***



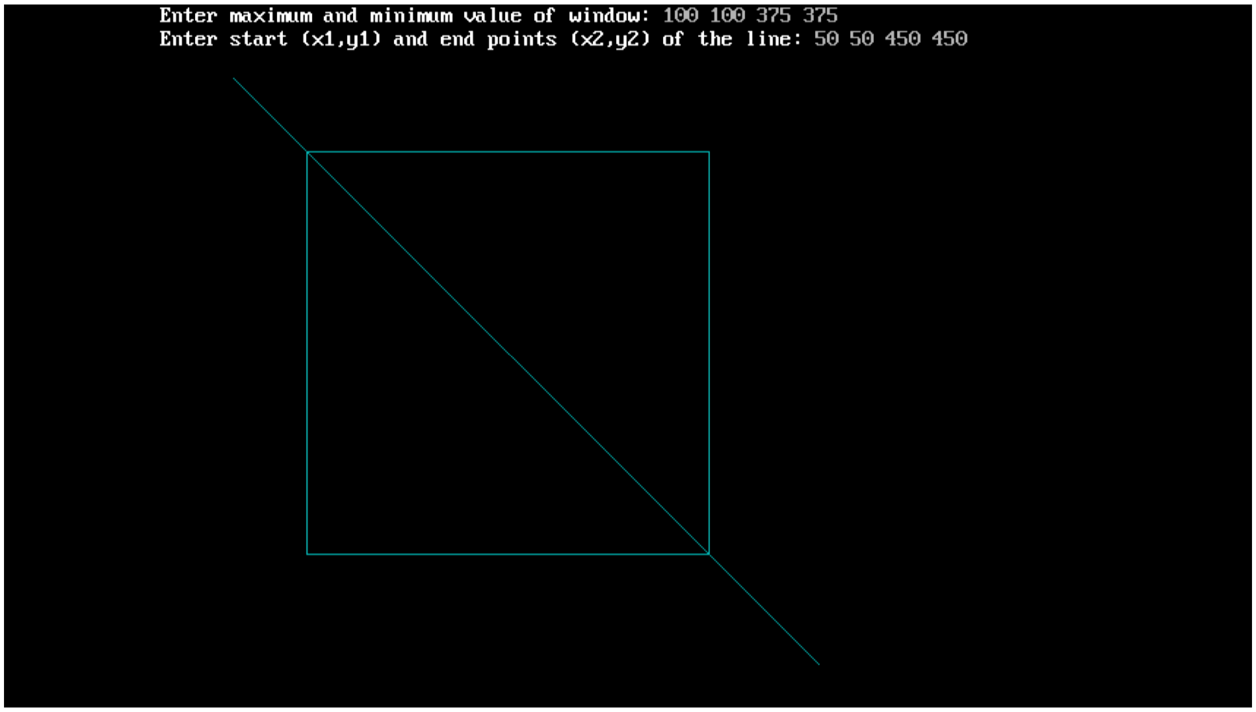
```
Enter maximum and minimum value of window: 100 100 375 375
```



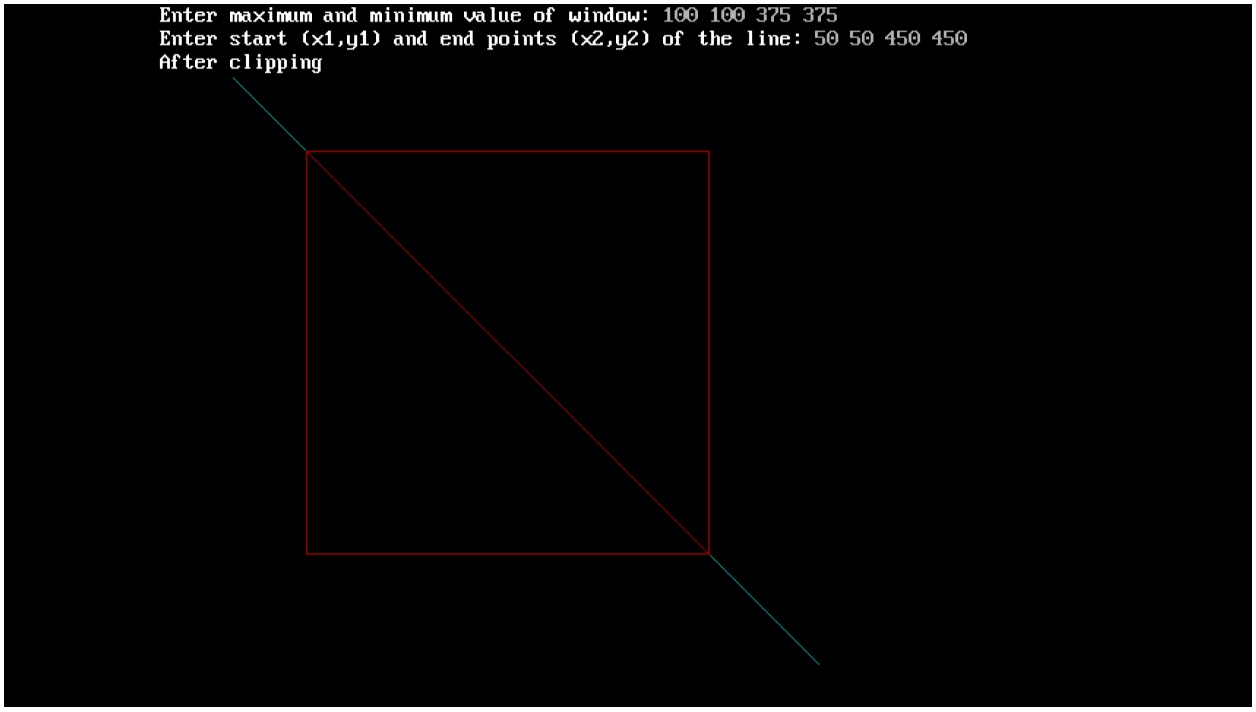
```
Enter maximum and minimum value of window: 100 100 375 375
Enter start (x1,y1) and end points (x2,y2) of the line:
```



```
Enter maximum and minimum value of window: 100 100 375 375
Enter start (x1,y1) and end points (x2,y2) of the line: 50 50 450 450
```



```
Enter maximum and minimum value of window: 100 100 375 375
Enter start (x1,y1) and end points (x2,y2) of the line: 50 50 450 450
```



```
Enter maximum and minimum value of window: 100 100 375 375
Enter start (x1,y1) and end points (x2,y2) of the line: 50 50 450 450
After clipping
```

- ***Conclusion:-*** *Hence, we have implemented Cohen Southerland Line clipping algorithm.*

COMPUTER GRAPHICS

GROUP-B **EXPERIMENT NUMBER:-4**

- ***Experiment Name:-*** *2D Transformation*
- ***Aim:-*** *Write a C++ program to draw a 2D object and perform the following transformations:-*

  1. Scaling
  2. Translation
  3. Rotation

- ***Objective:-*** *1. To understand 2D transformation.*

  2. To implement scaling on an object.
  3. To implement translation on an object.
  4. To implement Rotation on an object.

- ***Theory:-***

2D Transformation: *Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, etc. When a transformation takes place on a 2D plane, it is called 2D transformation.*

Translation:- *A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate (tx, ty) to the original coordinate X,YX,Y to get the new coordinate X′,Y′X′,Y′. X' = X + tx Y' = Y + ty The pair (tx, ty) is called the translation vector or shift vector.*

Rotation:- *In rotation, we rotate the object at particular angle θ thetatheta from its origin. From the following figure, we can see that the point PX,YX,Y is located at angle φ from the horizontal X coordinate with distance r from the origin. Let us suppose you want to rotate it at the angle θ. After rotating it to a new location, you will get a new point P' X′,Y′X′,Y′. Using standard trigonometric the original coordinate of point PX,YX,Y can be represented as*

X=rcosϕ......(1)X=rcosϕ .. (1)

$Y=r\sin\phi......(2)Y=r\sin\phi...(2)$

Same way we can represent the point P' X′, Y′X′, Y′ as −

$x'=r\cos(\phi+\theta)=r\cos\phi\cos\theta-r\sin\phi\sin\theta.......(3)x'=r\cos(\phi+\theta)=r\cos\phi\cos\theta-r\sin\phi\sin\theta.\ (3)$

$y'=r\sin(\phi+\theta)=r\cos\phi\sin\theta+r\sin\phi\cos\theta.......(4)y'=r\sin(\phi+\theta)=r\cos\phi\sin\theta+r\sin\phi\cos\theta\ (4)$

Substituting equation 11 & 22 in 33 & 44 respectively, we will get

$x'=x\cos\theta-y\sin\theta x'=x\cos\theta-y\sin\theta$

$y'=x\sin\theta+y\cos\theta y'=x\sin\theta+y\cos\theta$

Representing the above equation in matrix form,

$[X'Y']=[XY][\cos\theta-\sin\theta\sin\theta\cos\theta]$

OR

$[X'Y']=[XY][\cos\theta\sin\theta-\sin\theta\cos\theta]$

OR

P' = P .

R Where R is the rotation matrix $R=[\cos\theta-\sin\theta\sin\theta\cos\theta]R=[\cos\theta\sin\theta-\sin\theta\cos\theta]$

The rotation angle can be positive and negative. For positive rotation angle, we can use the above rotation matrix. However, for negative angle rotation, the matrix will change as shown below –

$R=[\cos(-\theta)-\sin(-\theta)\sin(-\theta)\cos(-\theta)]$

$R=[\cos(-\theta)\sin(-\theta)-\sin(-\theta)\cos(-\theta)]$

$=[\cos\theta\sin\theta-\sin\theta\cos\theta](\because\cos(-\theta)=\cos\theta and \sin(-\theta)=-\sin\theta)$

Scaling :-To change the size of an object, scaling transformation is used. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result. Let us assume that the original coordinates are X,YX,Y, the scaling factors are (SX, SY), and the produced coordinates are X′,Y′X′,Y′. This can be mathematically represented as shown below –

X' = X . SX and Y' = Y . SY

The scaling factor SX, SY scales the object in X and Y direction respectively. The above equations can also be represented in matrix form as below –

(X′Y′)=(XY)[Sx00Sy](X′Y′)=(XY)[Sx00Sy]
OR
P' = P . S

If we provide valuesless than 1 to the scaling factor S, then we can reduce the size of the object. If we provide values greater than 1, then we can increase the size of the object. P

- ***Program:-***

```
#include<iostream.h>
#include<math.h>
#include<graphics.h>
#include<conio.h>
class translation
{
int val;
public:
void setvalue(int temp)
{
val=temp;
}
int disp()
{
return(val);
}
translation operator+(translation o)
{
translation t;
t.val=val+o.val;
return(t);
}
};
class scale
{
int val;
public:
```

```
void setvalue(int temp){ val=temp;
}
int disp()
{
return(val);
}
scale operator*(scale o)
{
scale s;
s.val=val*o.val;
return(s);
}
};
class rotation
{
public:
float b[3][3],a[3][3];
int x1,y1,x2,y2,x3,y3;
rotation()
{
x1=100,y1=100,x2=300,y2=100,x3=150,y3=50;
}
rotation ret()
{
rotation t1;
t1.b[0][0]=x1;
t1.b[0][1]=y1;
t1.b[0][2]=0;
t1.b[1][0]=x2;
t1.b[1][1]=y2;
t1.b[1][2]=0;
t1.b[2][0]=x3;
t1.b[2][1]=y3;
t1.b[2][2]=1;
return(t1);
}
```

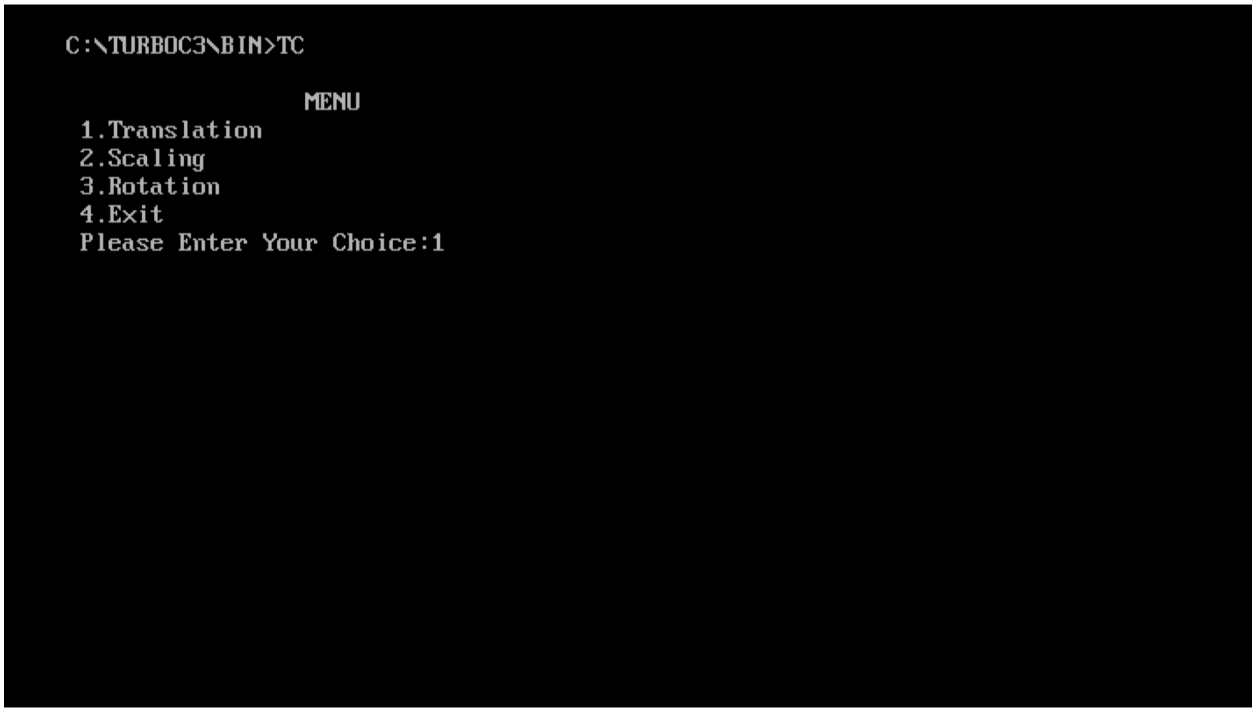
```
rotation ret1()
{
rotation t2;
float t;
float a1,a2,a3;
t=45;
t=t*3.14/180;
a1=cos(t);
a2=sin(t);
a3=-sin(t);
t2.a[0][0]=a1;
t2.a[0][1]=a2;
t2.a[0][2]=0;
t2.a[1][0]=a3;
t2.a[1][1]=a1;
t2.a[1][2]=0;
t2.a[2][0]=0;
t2.a[2][1]=0;
t2.a[2][2]=1;
return(t2);
}
rotation operator*(rotation o)
{
rotation t;
t.b[0][0]=((b[0][0]*o.a[0][0])+(b[0][1]*o.a[1][0])+(b[0][2]*o.a[2][0]));
t.b[0][1]=((b[0][0]*o.a[0][1])+(b[0][1]*o.a[1][1])+(b[0][2]*o.a[2][1]));
t.b[0][2]=((b[0][0]*o.a[0][2])+(b[0][1]*o.a[1][2])+(b[0][2]*o.a[2][2]));
t.b[1][0]=((b[1][0]*o.a[0][0])+(b[1][1]*o.a[1][0])+(b[1][2]*o.a[2][0]));
t.b[1][1]=((b[1][0]*o.a[0][1])+(b[1][1]*o.a[1][1])+(b[1][2]*o.a[2][1]));
t.b[1][2]=((b[1][0]*o.a[0][2])+(b[1][1]*o.a[1][2])+(b[1][2]*o.a[2][2]));
t.b[2][0]=((b[2][0]*o.a[0][0])+(b[2][1]*o.a[1][0])+(b[2][2]*o.a[2][0]));
t.b[2][1]=((b[2][0]*o.a[0][0])+(b[2][1]*o.a[1][0])+(b[2][2]*o.a[2][0]));
t.b[2][2]=((b[2][0]*o.a[0][0])+(b[2][1]*o.a[1][0])+(b[2][2]*o.a[2][0]));
return(t);
}
};
```

```
int main()
{
int gd=DETECT,gm=0;
int ch,flag=0;
int x1=100,y1=100,x2=300,y2=100,x3=150,y3=50,tx=50,ty=50; int
a1=100,b1=100,a2=300,b2=100,a3=150,b3=50,sx=2,sy=3;
translation t1,t2,t3,t4,t5,t6,t7,t8;
rotation r1,r2,r3,r4;
scale s1,s2,s3,s4,s5,s6,s7,s8;
do
{
cout<<"\n\t\t MENU";
cout<<"\n 1.Translation \n 2.Scaling \n 3.Rotation\n 4.Exit \n Please Enter Your
Choice:";
cin>>ch;
switch(ch)
{
case 1:
{
initgraph(&gd,&gm,"C://turboc3//bgi");
t1.setvalue(x1);
t2.setvalue(y1);
t3.setvalue(x2);
t4.setvalue(y2);
t5.setvalue(x3);
t6.setvalue(y3);
line(x1,y1,x2,y2);
line(x2,y2,x3,y3);
line(x1,y1,x3,y3);
t7.setvalue(tx);
t8.setvalue(ty);
setcolor(GREEN);
t1=t1+t7;
t2=t2+t8;
t3=t3+t7;
t4=t4+t8;
```

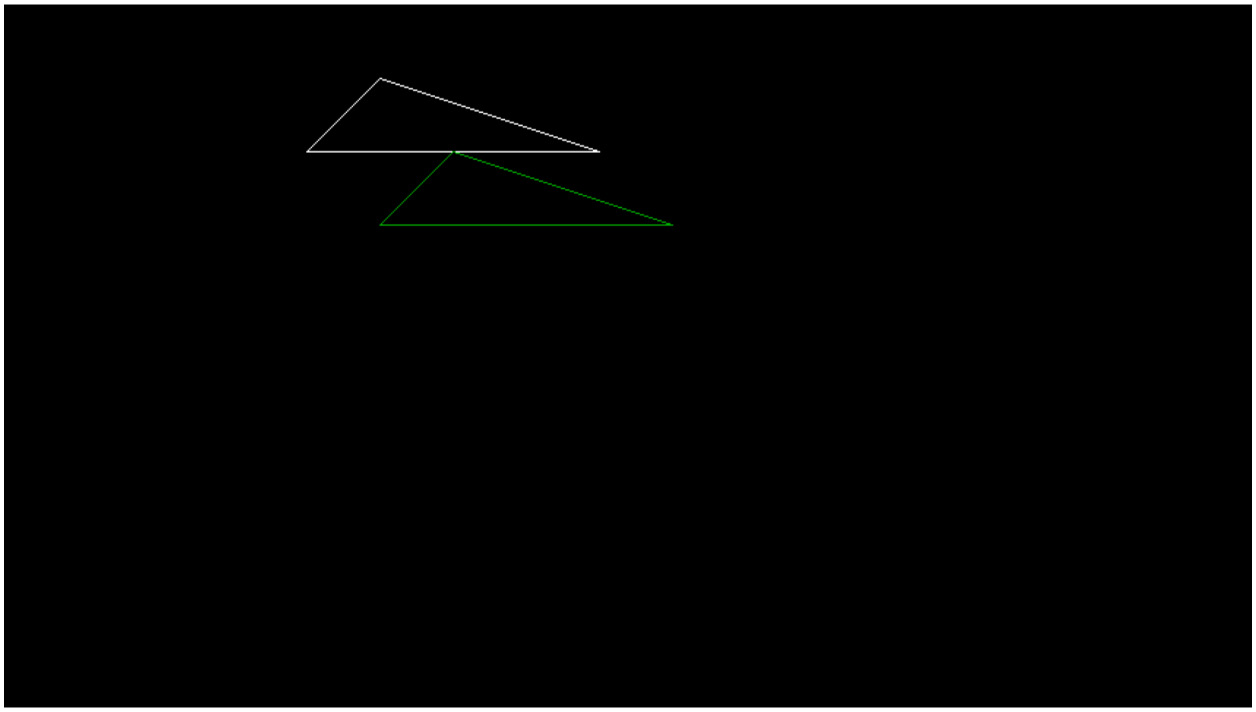
```
t5=t5+t7;
t6=t6+t8;
line(t1.disp(),t2.disp(),t3.disp(),t4.disp());
line(t3.disp(),t4.disp(),t5.disp(),t6.disp());
line(t1.disp(),t2.disp(),t5.disp(),t6.disp());
break;
}
case 2:
{
initgraph(&gd,&gm,"C://turboc3//bgi");
s1.setvalue(a1);
s2.setvalue(b1);
s3.setvalue(a2);
s4.setvalue(b2);
s5.setvalue(a3);
s6.setvalue(b3);
line(a1,b1,a2,b2);
line(a2,b2,a3,b3);
line(a1,b1,a3,b3);
s7.setvalue(sx); s8.setvalue(sy);
setcolor(GREEN);
s1=s1*s7;
s2=s2*s8;
s3=s3*s7;
s4=s4*s8;
s5=s5*s7;
s6=s6*s8;
line(s1.disp(),s2.disp(),s3.disp(),s4.disp());
line(s3.disp(),s4.disp(),s5.disp(),s6.disp());
line(s1.disp(),s2.disp(),s5.disp(),s6.disp());
break;
}
case 3:
{
initgraph(&gd,&gm,"C://turboc3//bgi");
r2=r1.ret();
```

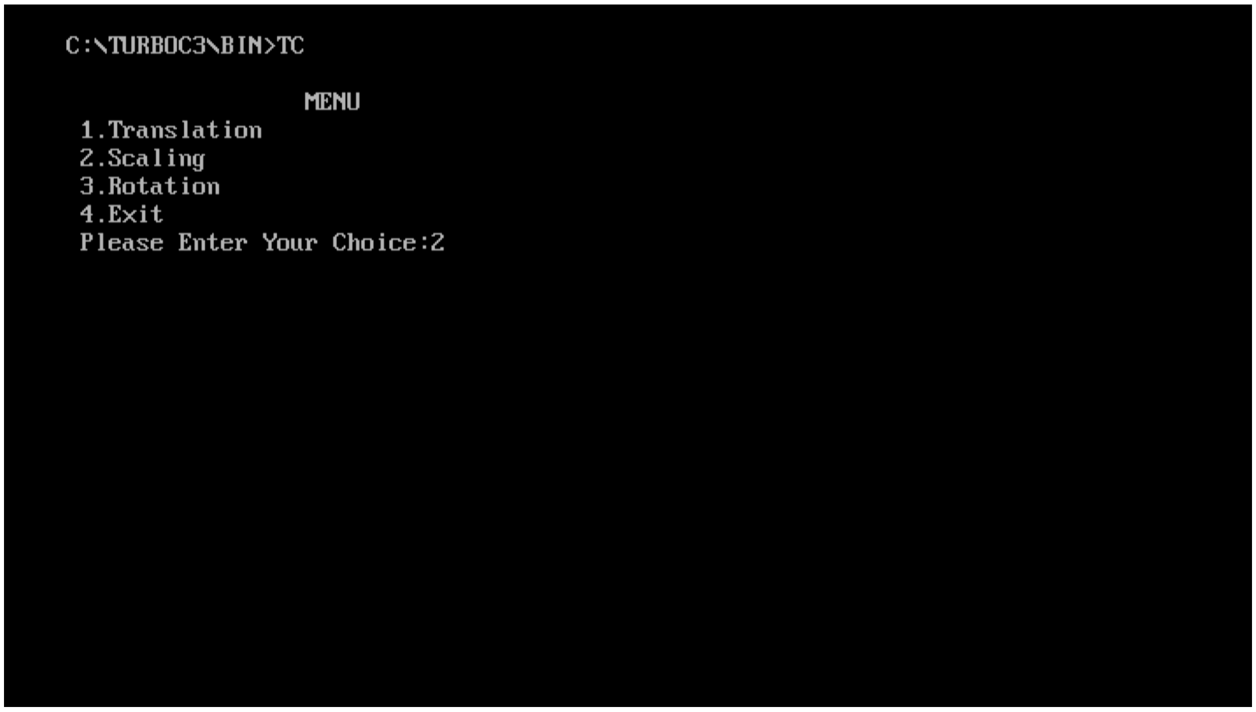
```
r3=r1.ret1();
int nx1,ny1,nx2,ny2,nx3,ny3; r4=r2*r3;
nx1=r4.b[0][0];
ny1=r4.b[0][1];
nx2=r4.b[1][0];
ny2=r4.b[1][1];
nx3=r4.b[2][0];
ny3=r4.b[2][1];
line(nx1,ny1,nx2,ny2); line(nx2,ny2,nx3,ny3); line(nx3,ny3,nx1,ny1);
break;
}
case 4:
{
flag=1;
cout<<"\n\t Thank YOU";
break;
}
default:
 cout<<"\n\t INVALID";
 break;
}
getch();
closegraph();
}while(flag==0);
return 0;
}
```

- ***Output:-***

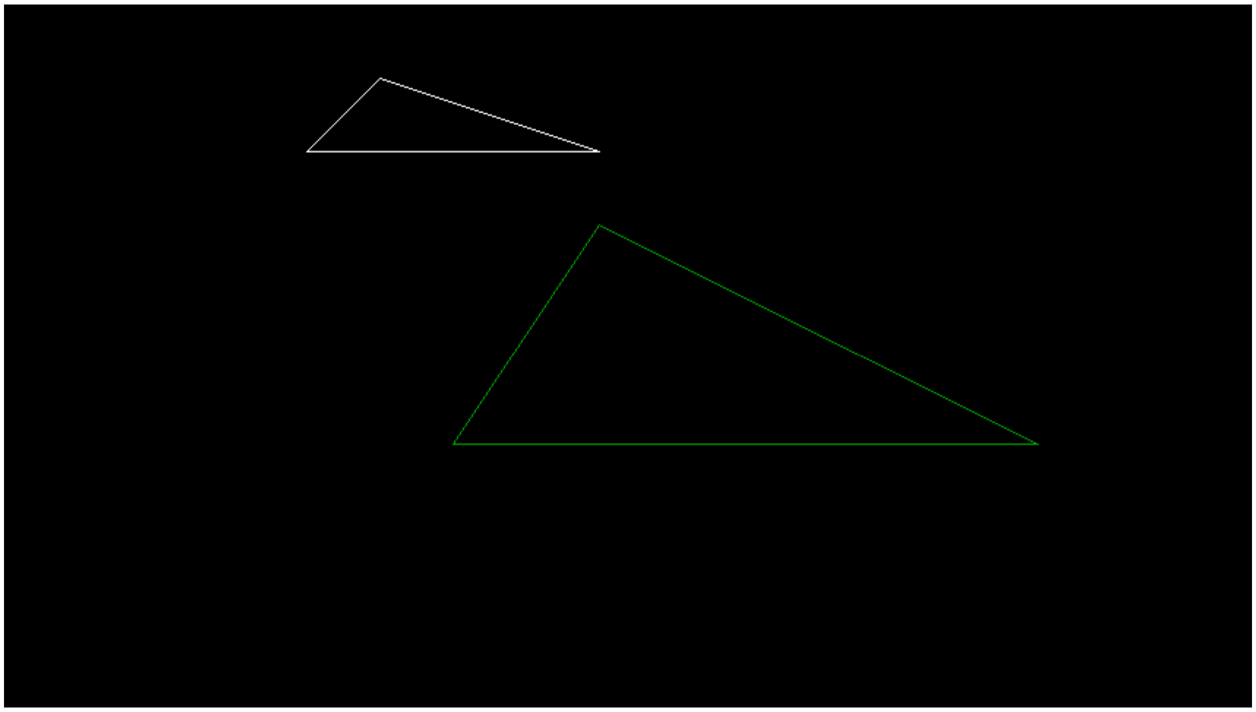


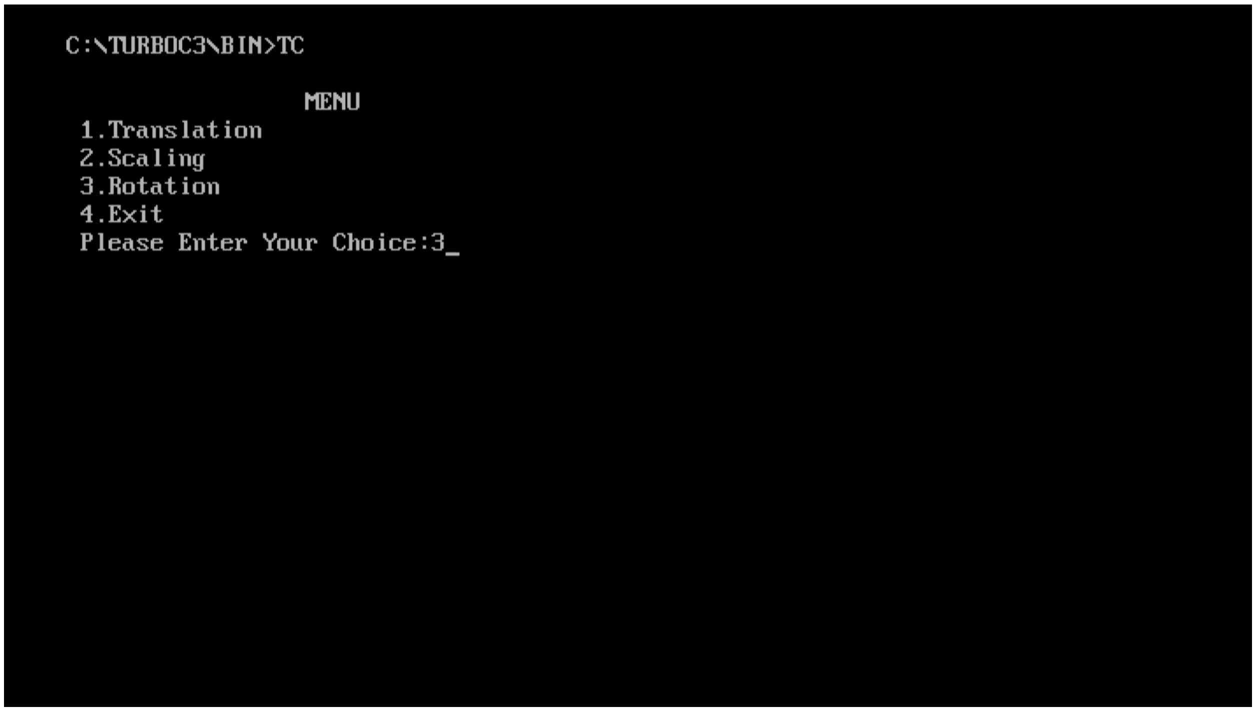
```
C:\TURBOC3\BIN>TC

                  MENU
 1.Translation
 2.Scaling
 3.Rotation
 4.Exit
 Please Enter Your Choice:1
```



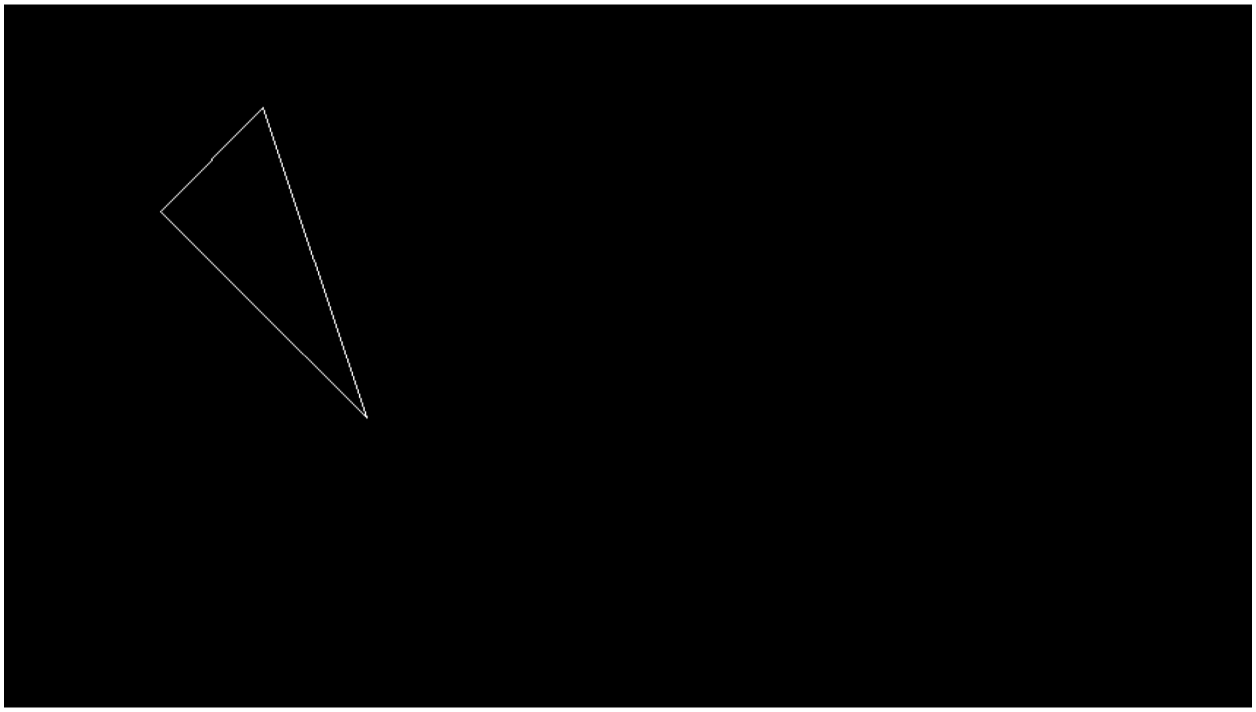


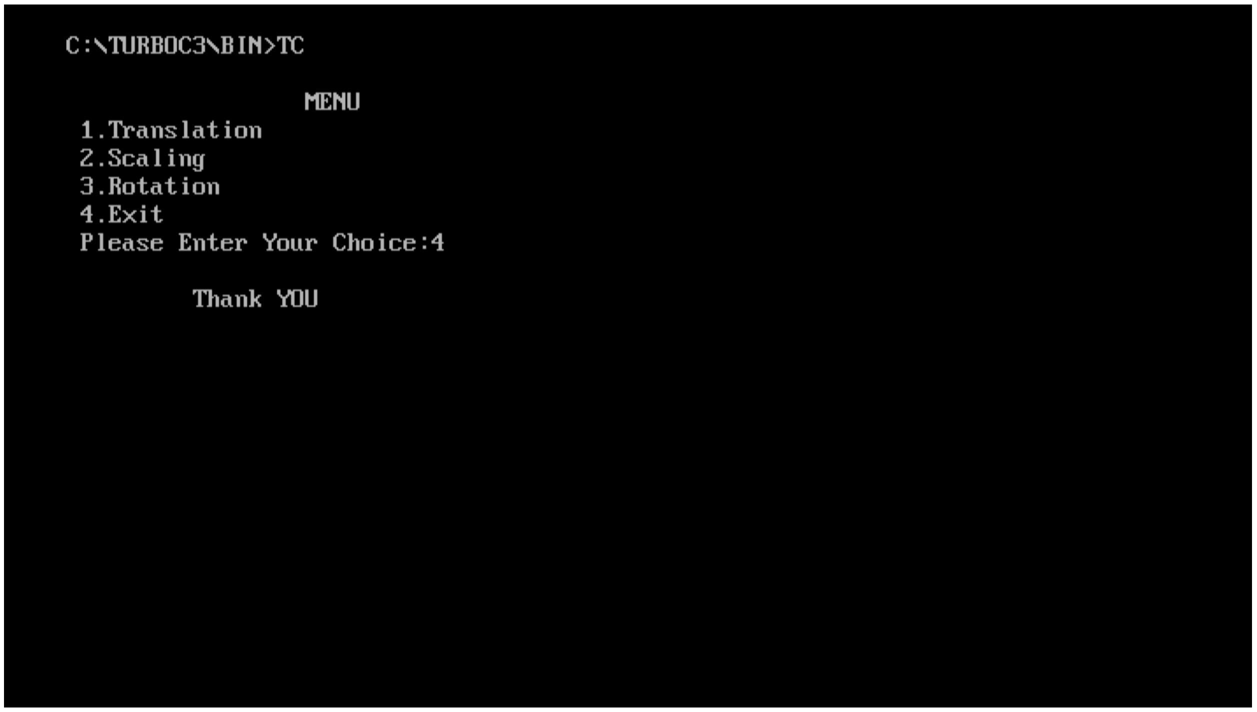
```
C:\TURBOC3\BIN>TC

                 MENU
 1.Translation
 2.Scaling
 3.Rotation
 4.Exit
 Please Enter Your Choice:2
```





```
C:\TURBOC3\BIN>TC

                 MENU
 1.Translation
 2.Scaling
 3.Rotation
 4.Exit
 Please Enter Your Choice:3_
```





```
C:\TURBOC3\BIN>TC

                 MENU
 1.Translation
 2.Scaling
 3.Rotation
 4.Exit
 Please Enter Your Choice:4

         Thank YOU
```

- ***Conclusion:-*** *Hence, we have studied and implemented 2D transformations like translation, scaling and rotation on a 2D object.*

COMPUTER GRAPHICS

GROUP-B EXPERIMENT NUMBER:-5

- ***Experiment Name:-*** *Hilbert Curve*
- ***Aim:-*** *Write a C++ program to generate Hilbert curve using concepts of fractals*
- ***Objective:-***
  1. *To understand the concept of fractals.*
  2. *To generate Hilbert curve using the concepts of fractals.*
- ***Theory:-*** *A rough, jagged and random figure is known as a fractal. Example- tress, mountains, rivers and so on. Fractals are very complex pictures generated by a computer from a single formula. They are created using iterations. This means one formula is repeated with slightly different values over and over again, taking into account the results from the previous iteration.*

Fractals can be classified as:

- *Self similar – These fractals have parts those are scaled down versions of entire object.*
- *Self affine – These fractals have parts those are formed with different scaling parameters in different co-ordinate directions.*
- *Invariant – In these fractals, non linear transformation is used.*

Fractals are used in many areas such as –

- *Astronomy − For analyzing galaxies, rings of Saturn, etc.*
- *Biology/Chemistry − For depicting bacteria cultures, Chemicalreactions, human anatomy, molecules, plants,*
- *Others − For depicting clouds, coastline and borderlines, data compression, diffusion, economy, fractal art, fractal music, landscapes,special effect, etc.*

Hilbert Curve

A Hilbert curve is a curve which is formed by connecting a sequence of U-shaped curves arranged and oriented in different directions. These U-shaped curves are placed at a certain step size distance apart.

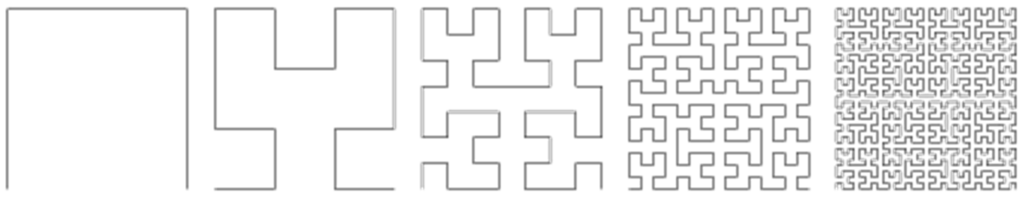
Construction

Hilbert curve can be constructed by the following successive approximations.

1. Divide a square into four quadrants and then draw the first approximation to the Hilbert curve by connecting centre points of each quadrant.

2. The second approximation to the Hilbert's curve can be drawn by further subdividing each of the quadrants and connecting their centers before moving to next major quadrant.

3. The third approximation subdivides the quadrants again. We can draw third approximation to Hilbert's curve by connecting the centers of finest level of quadrants before stepping into the next level of quadrant.

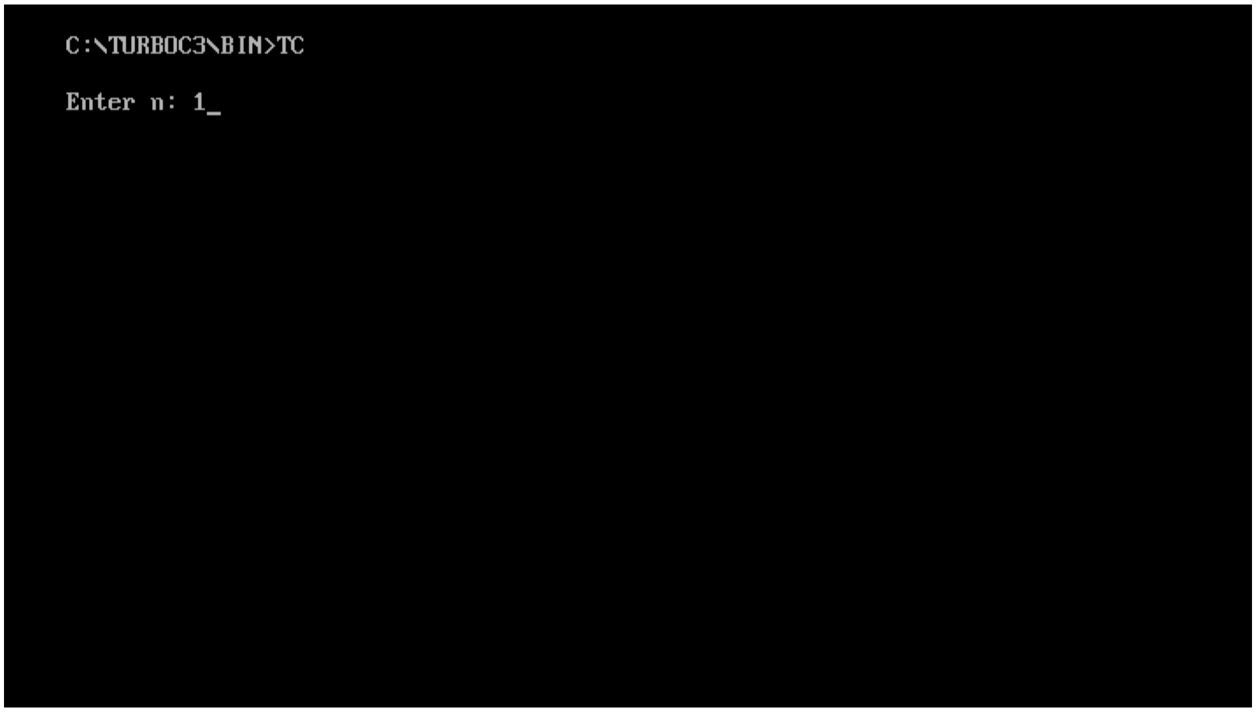

Order1 Order2 Order3 Order5 Order6

- ***Program:-***

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<graphics.h>
#include<math.h>
#include<stdlib.h>
void move(int j,int h,int &x,int &y)
{
 if(j==1)
y-=h;
 else if(j==2)
x+=h;
```

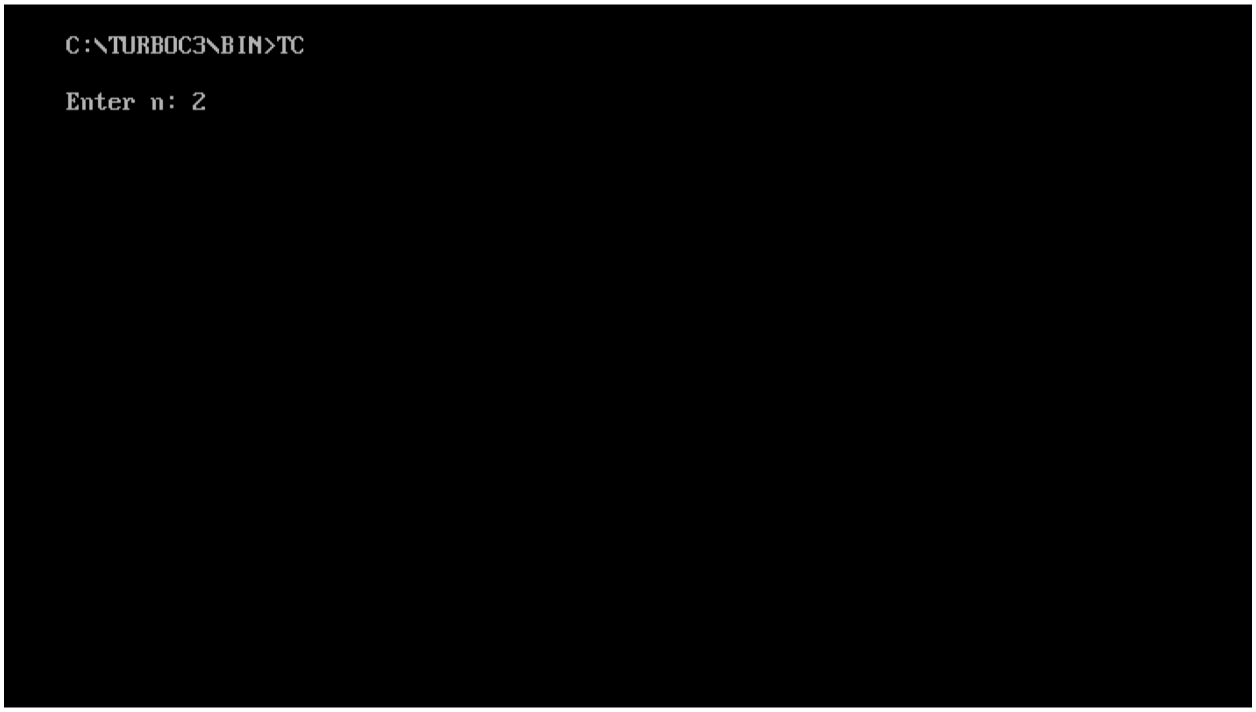
```
 else if(j==3)
y+=h;
 else if(j==4)
x-=h;
 lineto(x,y);
}
void hilbert(int r,int d,int l,int u,int i,int h,int &x,int &y)
{
 if(i>0)
 {
i--;
hilbert(d,r,u,l,i,h,x,y);
move(r,h,x,y);
hilbert(r,d,l,u,i,h,x,y);
move(d,h,x,y);
hilbert(r,d,l,u,i,h,x,y);
move(l,h,x,y);
hilbert(u,l,d,r,i,h,x,y);
 }
}
int main()
{
 int n,x1,y1;
 int x0=20,y0=50,x,y,h=10,r=2,d=3,l=4,u=1;
 cout<<endl<<"Enter n: ";
 cin>>n;
 x=x0;
 y=y0;
 int gd=DETECT,gm;
 initgraph(&gd,&gm,"C://turboc3//bgi");
 moveto(x,y);
 hilbert(r,d,l,u,n,h,x,y);
 getch();
 closegraph();
 return 0;
}
```

- ***Output:-***

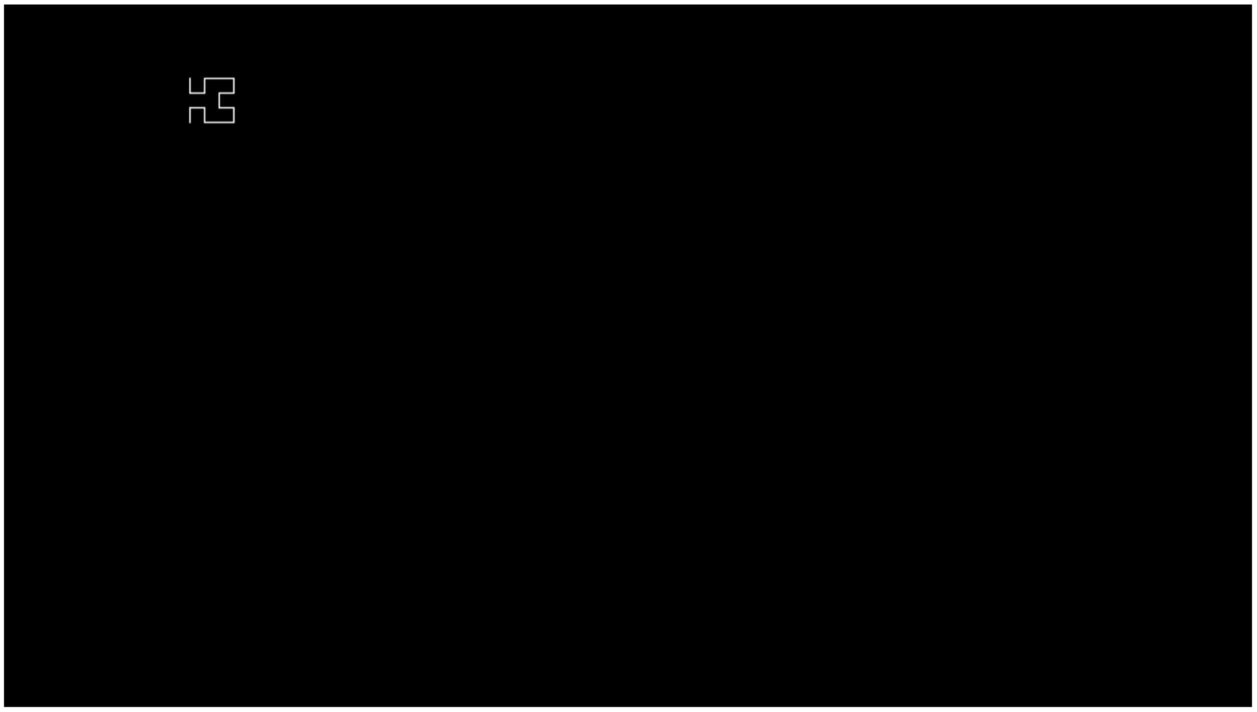


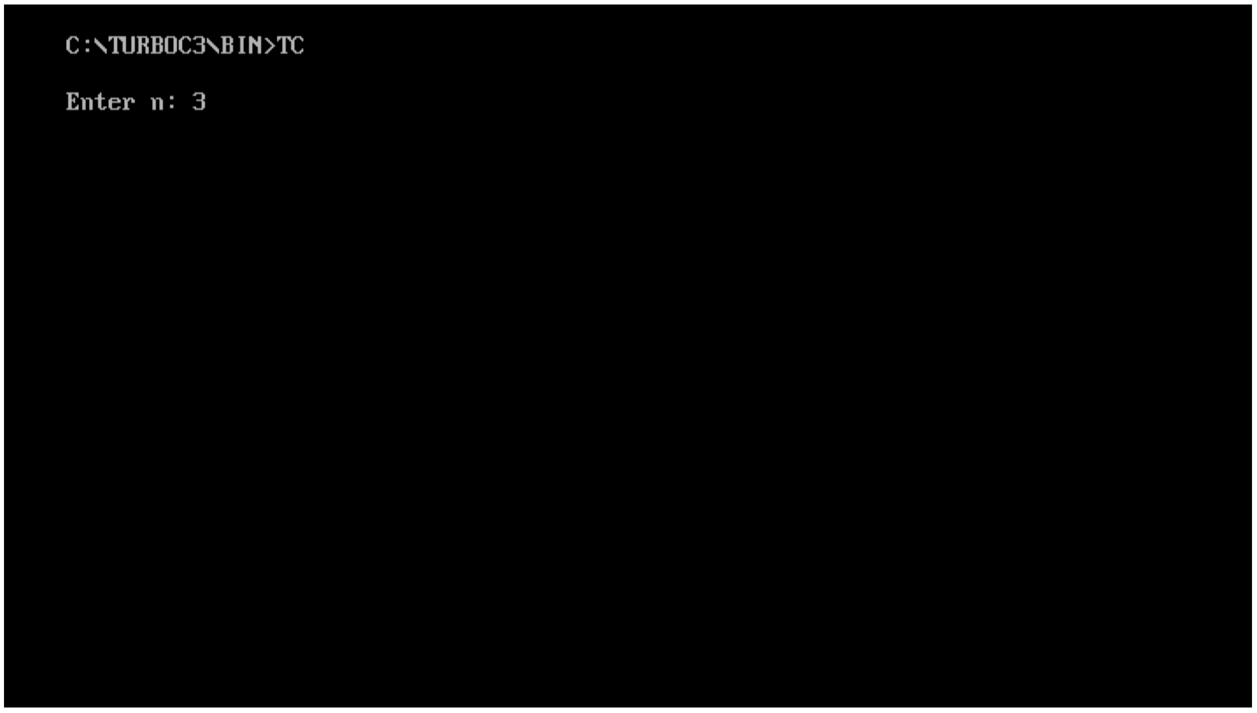
```
C:\TURBOC3\BIN>TC

Enter n: 1_
```

```
      ⊐
```

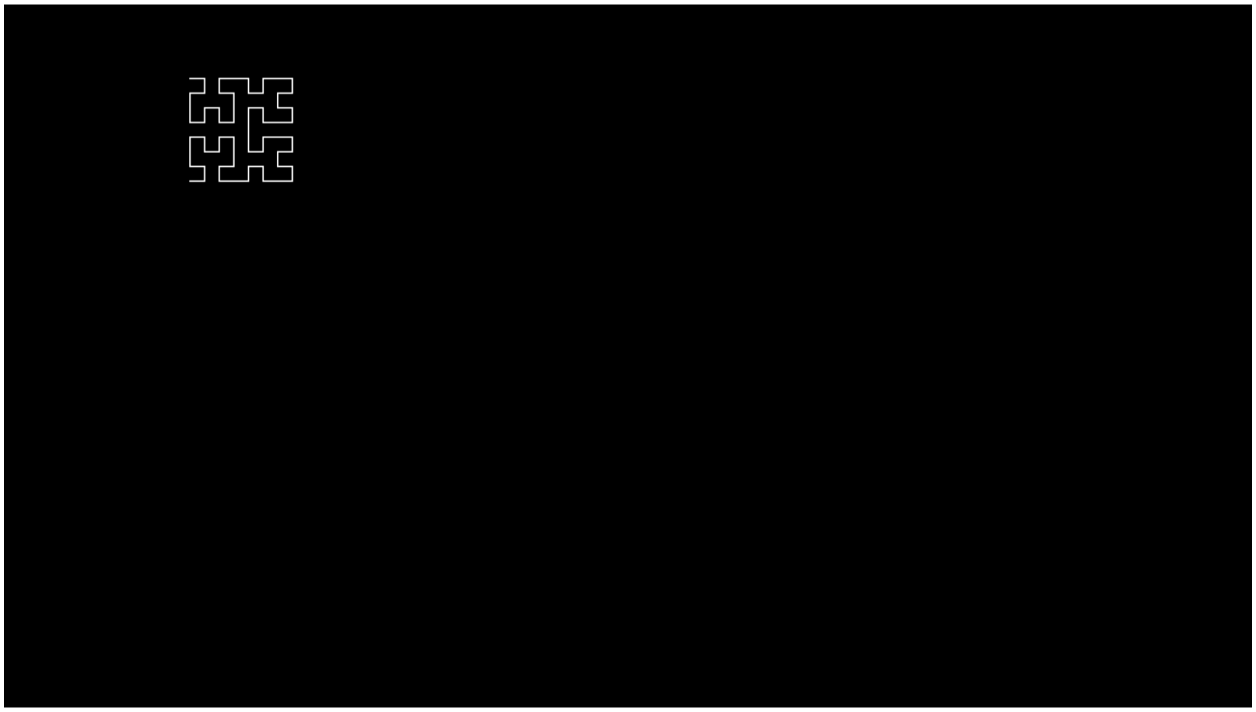


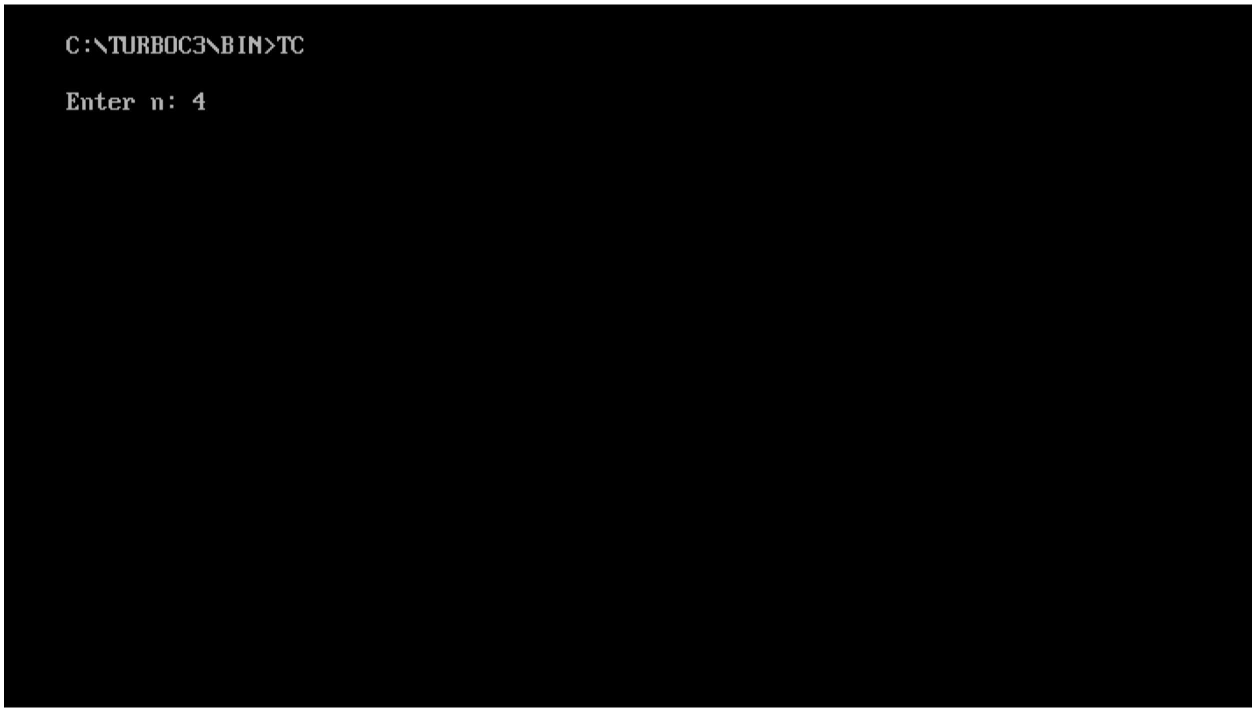
```
C:\TURBOC3\BIN>TC

Enter n: 2
```



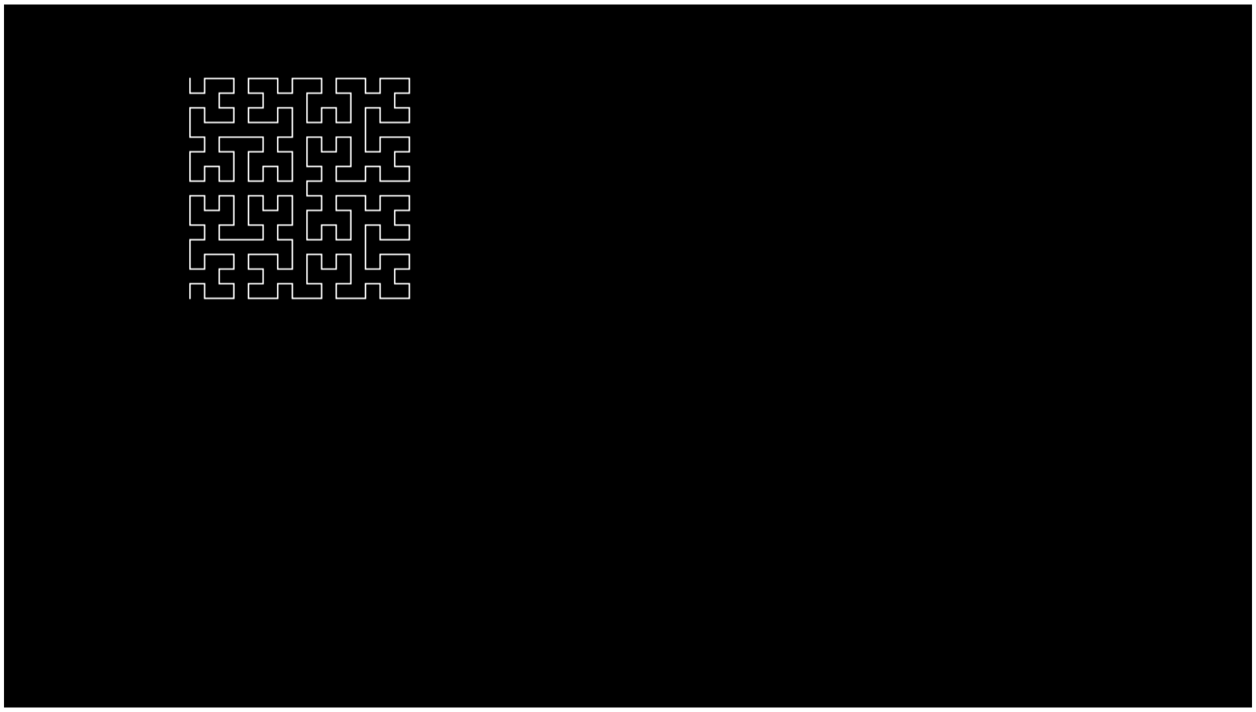


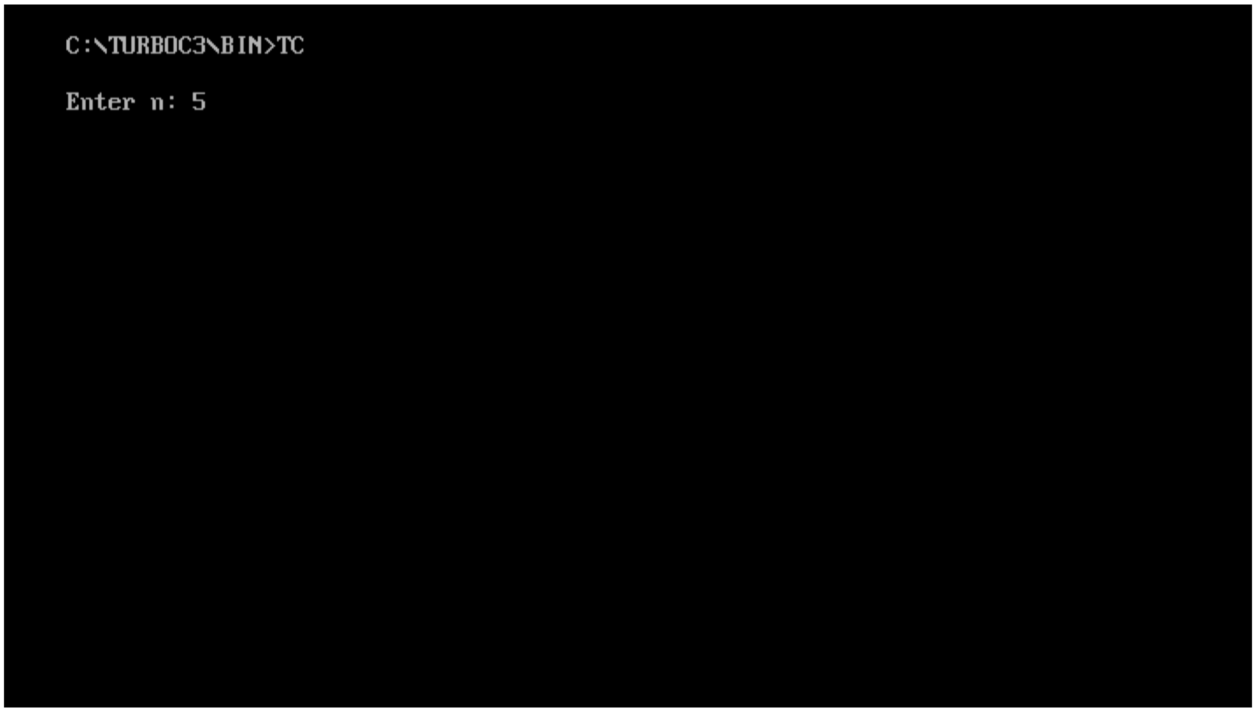
```
C:\TURBOC3\BIN>TC

Enter n: 3
```



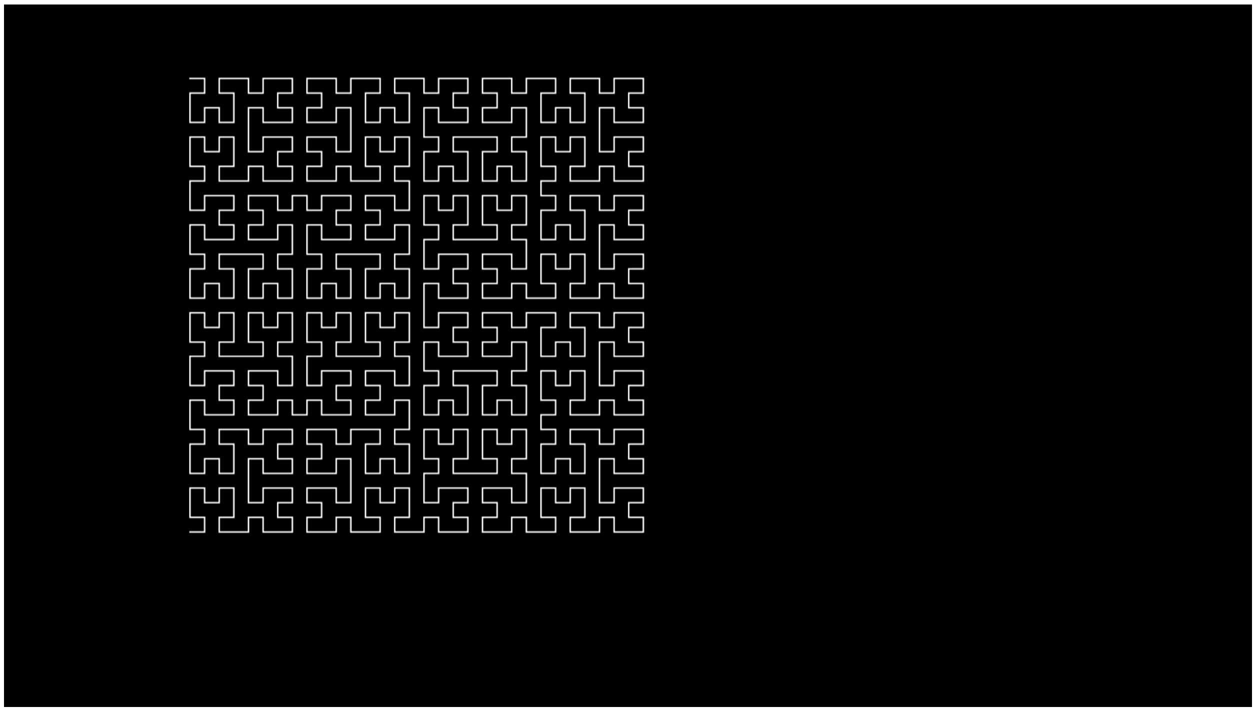


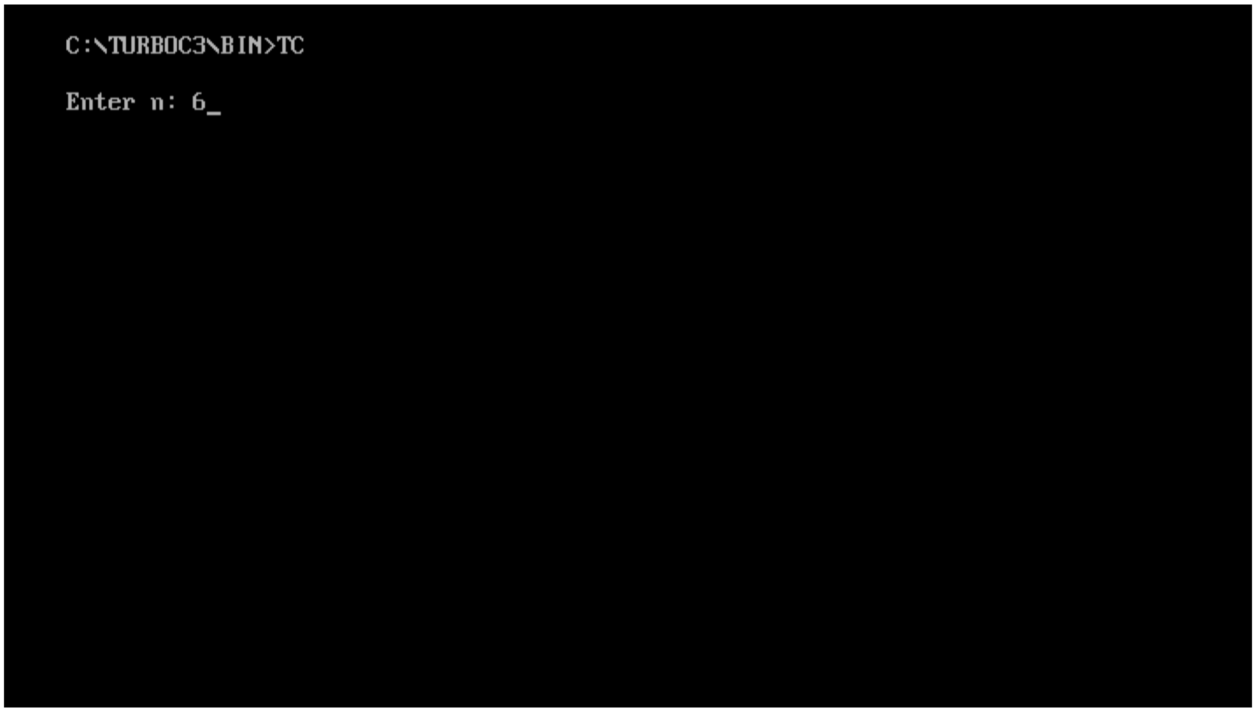
```
C:\TURBOC3\BIN>TC

Enter n: 4
```



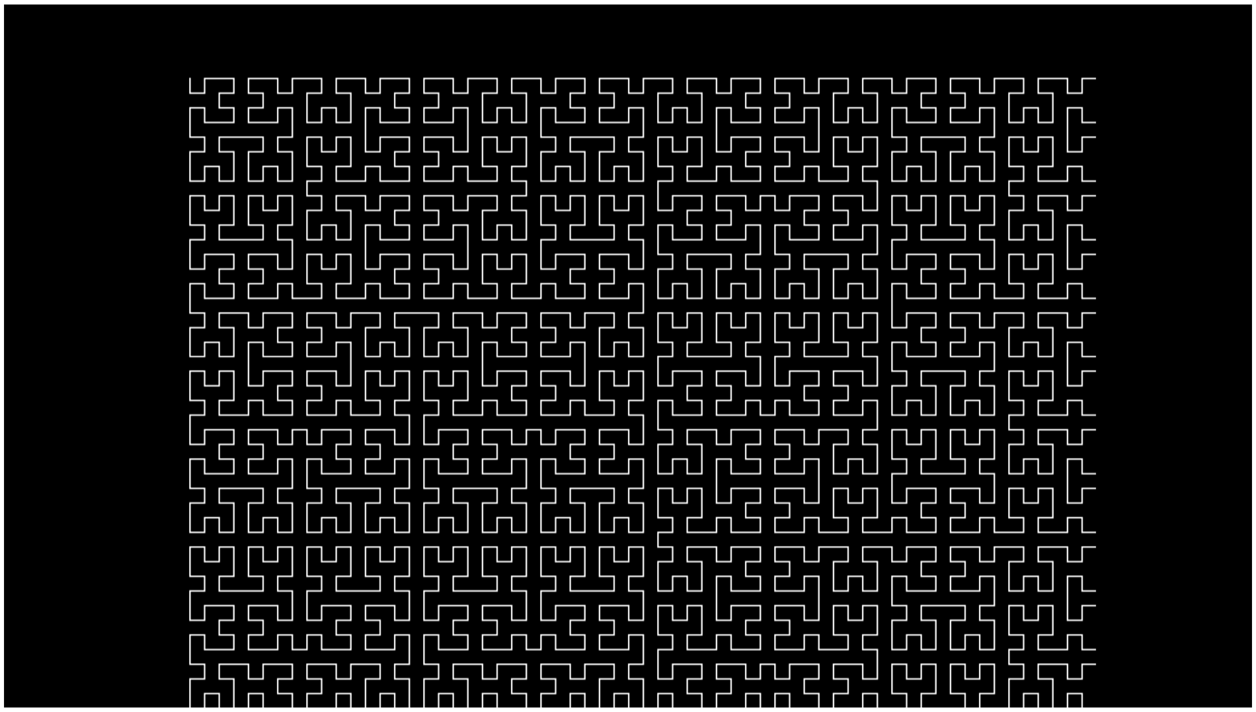


```
C:\TURBOC3\BIN>TC

Enter n: 5
```





```
C:\TURBOC3\BIN>TC

Enter n: 6_
```



- ***Conclusion:-*** *Hence, we have implemented Hilbert curve using the concept of fractals.*

COMPUTER GRAPHICS

GROUP-B EXPERIMENT NUMBER:-6

- ***Experiment Name:-*** *Simple Animation*
- ***Aim:-*** *Write C++ program to simulate any one of or similar scene*
  a) Clock with pendulum OR
  b) National Flag hoisting OR
  c) Vehicle/boat locomotion OR
  d) Water drop falling into the water and generated waves after impact
- ***Objective:-*** *To make animation with basic computer graphics concept.*
- ***Theory:-***

  Computer Animation:- *Animation means giving life to any object in computer graphics. It has the power of injecting energy and emotions into the most seemingly inanimate objects. Computer-assisted animation and computer-generated animation are two categories of computer animation. It can be presented via film or video. The basic idea behind animation is to play back the recorded images at the rates fast enough to fool the human eye into interpreting them as continuous motion. Animation can make a series of dead images come alive. Animation can be used in many areas like entertainment, computer aided-design, scientific visualization, training, education, e-commerce, and computer art.*

  Graphic Animation:- *Computer animation is the art of creating moving images via the use of computers.It is a subfield of computer graphics and animation. Increasingly it is created by means of 3D computer graphics, though 2D computer graphics are still widely used for low bandwidth and faster real-time rendering needs. To create the illusion of movement, an image is displayed on the computer screen then quickly replaced by a new image that is similar to the previous image, but shifted slightly. This technique is* *identical to how the illusion of movement is achieved with television and motion pictures.*

- ***Program:-***

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

int i, j = 0;
char st1[] = "Vande Mataram";
float w = 0.0, h = 0.0, x, y, z;
void filcircle(float x, float y, float r)
{
        float angle = 0;
        glBegin(GL_TRIANGLE_FAN);
        while (angle < 360)
        {
                glVertex2f(x + sin(angle) * r, y + cos(angle) * r);
                angle += 1.0;
        }
        glEnd();

}



void curves2(float x1, float y1, float x2, float y2, float a1, float a2, float b1, float b2)
{
        GLfloat cp[4][3] = { {x1,y1},{a1,a2},{b1,b2},{x2,y2} };
        glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, *cp);
        glEnable(GL_MAP1_VERTEX_3);
        GLint k;
        float c = 0.3;

        glLineWidth(2);
        glBegin(GL_LINE_STRIP);
        for (k = 0; k <= 50; k++)
        {

                glEvalCoord1f(GLfloat(k) / 50.0);
        }
        glEnd();

        glColor3f(0.15, .3, 0.65);
        glBegin(GL_POINTS);
        for (k = 0; k < 4; k++)
                glVertex2fv(&cp[k][0]);
        glEnd();
}


void drawline(float x0, float y0, float x1, float y1)
{

        glBegin(GL_LINES);
        glVertex2f(x0, y0);
        glVertex2f(x1, y1);
        glEnd();
        glFlush();
}





void drawrect(float xmin, float xmax, float ymin, float ymax)
```

```
{
	glBegin(GL_QUADS);
	glVertex2f(xmin, ymin);
	glVertex2f(xmin, ymax);
	glVertex2f(xmax, ymax);
	glVertex2f(xmax, ymin);
	glEnd();
	glFlush();
}

void draw_pixels(int cx, int cy)
{
	glPointSize(5.0);
	glBegin(GL_POINTS);
	glVertex2i(cx, cy);
	glEnd();
	glFlush();
}

void plotpixels(int h, int k, int x, int y)
{
	draw_pixels(x + h, y + k);
	draw_pixels(-x + h, y + k);
	draw_pixels(x + h, -y + k);
	draw_pixels(-x + h, -y + k);
	draw_pixels(y + h, x + k);
	draw_pixels(-y + h, x + k);
	draw_pixels(y + h, -x + k);
	draw_pixels(-y + h, -x + k);
}

void drawcircle(int h, int k, int r)
{
	int d = 1 - r, x = 0, y = r;
	while (y > x)
	{
        plotpixels(h, k, x, y);
        if (d < 0)
            d += 2 * x + 3;
        else
        {
            d += 2 * (x - y) + 5;
            --y;
        }
        ++x;
    }
    plotpixels(h, k, x, y);
}




void init()
{
    glClearColor(0.15, .3, .65, 1);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

}

void sun()
{

    //drawing a sun
    glColor3f(1, 1, 0);
    glLineWidth(2);
    glBegin(GL_LINES);
    glVertex2f(60, 63);
    glVertex2f(100, 63);
    glVertex2f(80, 50);
    glVertex2f(80, 80);
    glVertex2f(70, 77);
```

```
        glVertex2f(90, 50);
        glVertex2f(70, 50);
        glVertex2f(90, 77);
        glVertex2f(65, 70);
        glVertex2f(95, 55);
        glVertex2f(65, 55);
        glVertex2f(95, 70);
        glEnd();
        float angle = 0;
        glColor3f(1, .7, 0);
        glBegin(GL_TRIANGLE_FAN);
        while (angle < 360)
        {
                glVertex2f(80 + sin(angle) * 10, 65 + cos(angle) * 10);
                angle += 1.0;
        }
        glEnd();


}




void draw_flag()
{



        //steps
        glColor3f(1.5, 1.5, 1.5);
        drawrect(-100, -60, -75, -70);
        drawrect(-95, -65, -70, -65);

        //pole
```

```
            glColor3f(0.0, 0.0, 0.0);
            glLineWidth(8.0);
            drawline(-80, -65, -80, 55);
            //rope
            glColor3f(1.0, 1.0, 1.0);
            glLineWidth(0.2);
            curves2(-80, -40, -80, 55, -83, -30, -83, 40);
            //folded flag
            glColor3f(1.0, 1.0, 0.0);
            drawrect(-80, -75, -42, -38);
            //tierope
            glColor3f(1.0, 1.0, 1.0);
            glLineWidth(0.2);
            drawline(-80, -40, -75, -40);

            //hook
            glColor3f(0.0, 0.0, 0.0);
            draw_pixels(-80, 55);
}



void draw_people()
{
            int j = 0;
            float k = 0;
            for (int i = 0; i < 3; i++)
            {

                        //person head
                        glColor3f(0.75, 0.75, 1);
                        filcircle(40 + j, -30, 6);

                        //neck
                        drawrect(38 + j, 42 + j, -40, -35);
                        //eye
```

```
			glColor3f(0, 0, 0);
			draw_pixels(38 + j, -28);
			draw_pixels(42 + j, -28);
			glLineWidth(2.0);
			drawline(38 + j, -32.5, 42 + j, -32.5);
			//body
			glColor3f(1 + k, 0.1 + k, 0.2 + k);
			drawrect(33 + j, 47 + j, -60, -40);
			//hands
			glColor3f(0 + k, 0.75 + k, 1 + k);
			drawrect(30 + j, 33 + j, -50, -40);
			drawrect(47 + j, 50 + j, -50, -40);
			glColor3f(0.75, 0.75, 1);
			drawrect(30 + j, 33 + j, -65, -50);
			drawrect(47 + j, 50 + j, -65, -50);
			//legs
			glColor3f(0 + k, .75 + k, 1 + k);
			drawrect(35 + j, 40 + j, -75, -60);
			drawrect(40 + j, 45 + j, -75, -60);
			glColor3f(0, 0, 0);
			drawline(40 + j, -75, 40 + j, -60);
			j = j + 25;
			k += .3;
		}

}


void printc(int x, int y, char st[])
{
		char* p = st;
		float i = 0;
		while (*p != '\0')
		{
			glRasterPos2i(x + i, y);
			glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *p);
```

```
                i += 5;
                p++;
        }
        glFlush();
}



void reshape(int w, int h)
{
        glViewport(0, 0, (GLsizei)w, (GLsizei)h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        if (w <= h)
                gluOrtho2D(-100.0, 100.0, -80.0 * (GLfloat)h / (GLfloat)w,
                        80.0 * (GLfloat)h / (GLfloat)w);
        else
                gluOrtho2D(-100.0 * (GLfloat)w / (GLfloat)h,
                        100.0 * (GLfloat)w / (GLfloat)h, -80.0, 80.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();

        glutPostRedisplay();

}



void salute()
{
        int j = 0, i;
        float k = .3;
        for (i = 0; i < 2; i++)
        {
                glColor3f(.15, 0.3, 0.65);
                drawrect(55 + j, 58 + j, -50, -40);
                drawrect(55 + j, 58 + j, -65, -50);
                //person second
                glColor3f(0 + k, 0.75 + k, 1 + k);
                glBegin(GL_QUADS);
                glVertex2f(51 + j, -40);
                glVertex2f(58 + j, -40);
                glVertex2f(58 + j, -43);
                glVertex2f(51 + j, -43);
                glEnd();
                glFlush();
                glColor3f(0.75, 0.75, 1);
                glBegin(GL_QUADS);
                glVertex2f(51 + j, -40);
                glVertex2f(54 + j, -40);
                glVertex2f(61 + j, -34);
                glVertex2f(59 + j, -32);
                glEnd();
                glFlush();
                j = j + 25;
                k += .3;
        }

        glColor3f(1, 1, 1);
        printc(17, 40, st1);


}



void flaghoist()
{
        //flag
        glColor3f(1, 0.25, 0);
        drawrect(-80, -40, 49, 56);
        glColor3f(1, 1, 1);
        drawrect(-80, -40, 42, 49);
        glColor3f(0, 1, 0);
        drawrect(-80, -40, 35, 42);
        glColor3f(0, 0, 1);
        drawcircle(-60, 45.5, 3.5);


        drawline(-60, 45.5, -56.5, 45.5);
        drawline(-60, 45.5, -60, 48.5);
        drawline(-60, 45.5, -60, 41.5);
        drawline(-60, 45.5, -63.5, 45.5);
        drawline(-60, 45.5, -58, 47);
        drawline(-60, 45.5, -62, 47);
        drawline(-60, 45.5, -58, 42.5);
        drawline(-60, 45.5, -62, 42.5);



        salute();
}



void move_flag()
{
        //hand movement
        glColor3f(0.15, 0.3, 0.65);
        drawrect(-70, -67, -65, -50);
        glColor3f(0.75, 0.75, 1);
        glBegin(GL_QUADS);
        glVertex2f(-77, -40);
        glVertex2f(-78, -42);
        glVertex2f(-70, -47);
        glVertex2f(-70, -50);
        glEnd();
        glFlush();
```

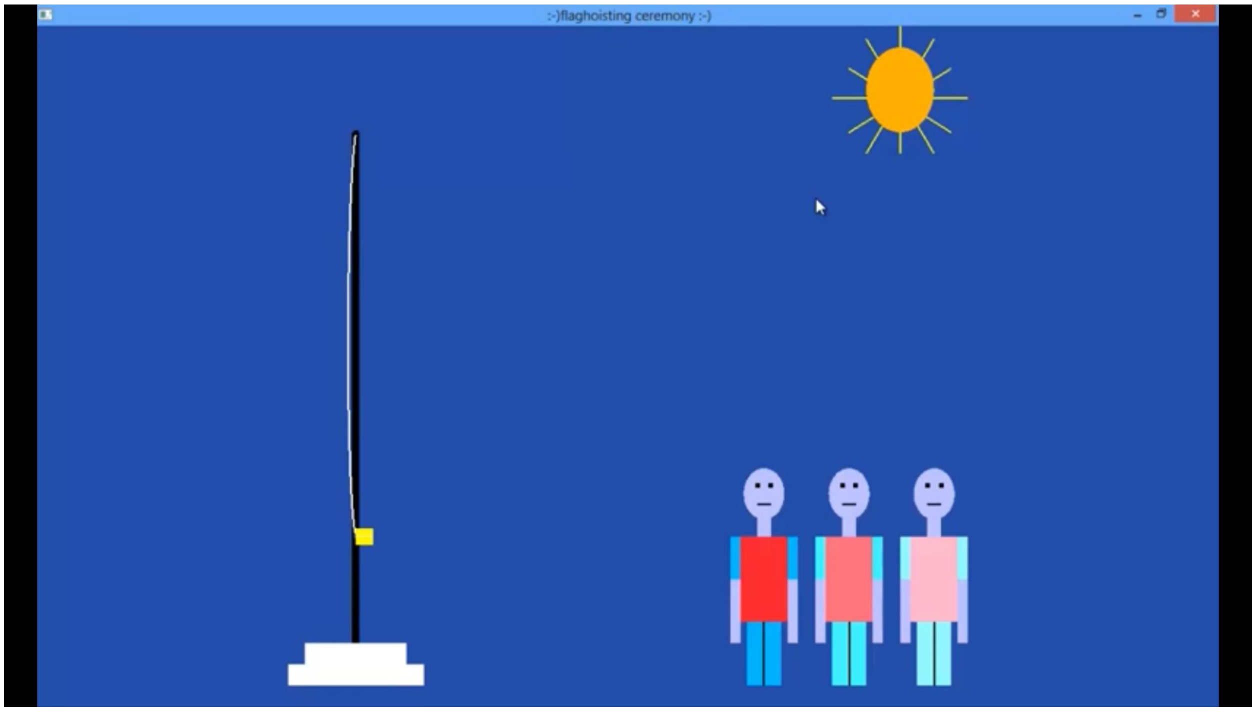
```
//flag movement
float i = 0, j = 0;
while (i < 80.5)
{
        glColor3f(0.15, 0.3, 0.65);
        drawrect(-80, -75, -42 + i, -38 + i);
        i = i + .15;
        j = i;
        glColor3f(1.0, 1.0, 0.0);
        drawrect(-80, -75, -42 + j, -38 + j);

}
glColor3f(.15, 0.3, 0.65);
curves2(-80, -40, -80, 55, -83, -30, -83, 40);
//pole
glColor3f(0.0, 0.0, 0.0);
glLineWidth(8.0);
drawline(-80, -65, -80, 55);

glColor3f(1.0, 1.0, 1.0);
glLineWidth(0.2);
curves2(-75, -50, -80, 55, -65, -35, -95, 40);


//untie rope

glBegin(GL_LINES);
glVertex2f(-78.50, -50);
glVertex2f(-74, -43);
glEnd();
glFlush();
glBegin(GL_LINES);
glVertex2f(-80, -40);
glVertex2f(-74, -43);
glEnd();
```

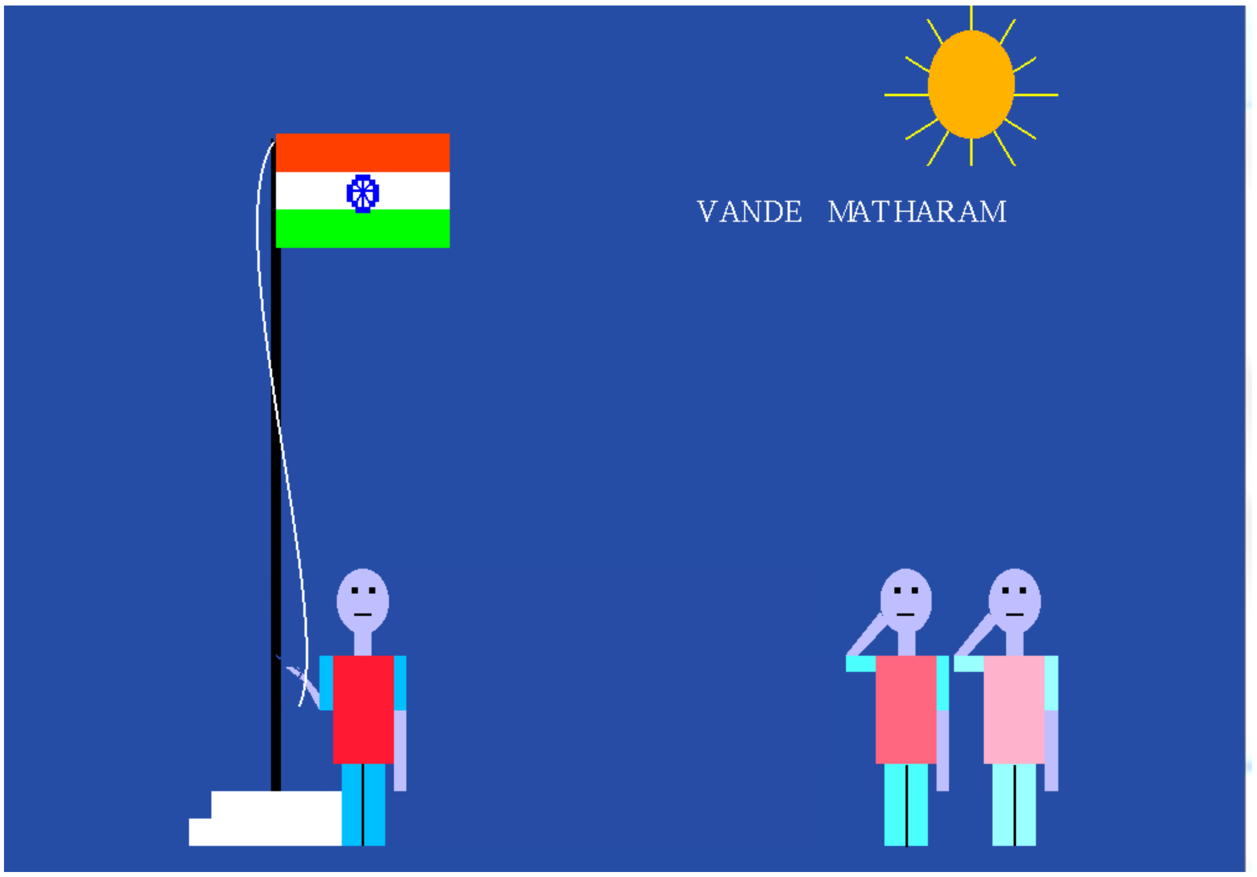
```
	glFlush();
	flaghoist();
}


void move_person()
{

	int i = 0;
	while (i < 100)
	{

		//1st person head
		glColor3f(0.15, .3, .65);
		filcircle(40 - i, -30, 6);

		//neck
		drawrect(38 - i, 42 - i, -40, -35);
		//eye

		draw_pixels(38 - i, -28);
		draw_pixels(42 - i, -28);

		drawline(38 - i, -32.5, 42 - i, -32.5);
		//body
		drawrect(33 - i, 47 - i, -60, -40);
		//hands
		drawrect(30 - i, 33 - i, -50, -40);
		drawrect(47 - i, 50 - i, -50, -40);
		drawrect(30 - i, 33 - i, -65, -50);
		drawrect(47 - i, 50 - i, -65, -50);
		//legs
		drawrect(35 - i, 40 - i, -75, -60);
		drawrect(40 - i, 45 - i, -75, -60);
		drawline(40 - i, -75, 40 - i, -60);
```

```
                i = i + 5;
                j = i;
                //1st person head
                glColor3f(0.75, 0.75, 1);
                filcircle(40 - j, -30, 6);

                //neck
                drawrect(38 - j, 42 - j, -40, -35);

                //eye
                glColor3f(0, 0, 0);
                draw_pixels(38 - j, -28);
                draw_pixels(42 - j, -28);
                glLineWidth(2.0);
                drawline(38 - j, -32.5, 42 - j, -32.5);
                //body
                glColor3f(1, 0.1, 0.2);
                drawrect(33 - j, 47 - j, -60, -40);
                //hands
                glColor3f(0, 0.75, 1);
                drawrect(30 - j, 33 - j, -50, -40);
                drawrect(47 - j, 50 - j, -50, -40);
                glColor3f(0.75, 0.75, 1);
                drawrect(30 - j, 33 - j, -65, -50);

                drawrect(47 - j, 50 - j, -65, -50);
                //legs
                glColor3f(0, .75, 1);
                drawrect(35 - j, 40 - j, -75, -60);
                drawrect(40 - j, 45 - j, -75, -60);
                glColor3f(0, 0, 0);
                drawline(40 - j, -75, 40 - j, -60);
        }
        move_flag();
}
```

```
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        sun();
        draw_flag();
        draw_people();
}




void keys(unsigned char key, int x, int y)
{
        if (key == 's' || key == 'S')
                move_person();
        if (key == 27)
                exit(0);
}

int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
        glutInitWindowSize(1000, 700);
        glutInitWindowPosition(0, 0);
        glutCreateWindow("Flag Hoisting Ceremony");
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        init();
        glutKeyboardFunc(keys);
        glutMainLoop();
}
```

- ***Output:-***





- ***Conclusion:-****We have done a simple animation using basic computer graphics concepts.*