CG All Practicals ~ By HK_OFFICIAL_

Title	A concave polygon filling using scan fill algorithm
Aim/Problem Statement	Write C++ program to draw a concave polygon and fill it with desired color using scan fill algorithm. Apply the concept of inheritance.
CO Mapped	CO3
Pre -re quisite	1. Basic programming skills of C++ 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC
Learning Objective	To understand and implement scanline polygon fill algorithm.

Program :

```
#include <conio.h>

#include <iostream>

#include <graphics.h>

#include <stdlib.h>

using namespace std;


class point {

public:

  int x, y;

};


class poly {

private:

  point p[20];

  int inter[20], x, y;

  int v, xmin, ymin, xmax, ymax;


public:

  int c;

  void read();
  void calcs();
  void display();
  void ints(float);
  void sort(int);
};

void poly::read() {
  int i;
  cout << "\n Scan Fill Algorithm ";
  cout << "\n Enter Number Of Vertices Of Polygon: ";
  cin >> v;
  if (v > 2) {
    for (i = 0; i < v; i++) { // ACCEPT THE VERTICES
      cout << "\nEnter co-ordinate no. " << i + 1 << " : ";
      cout << "\n\tx" << (i + 1) << "=";
      cin >> p[i].x;
      cout << "\n\ty" << (i + 1) << "=";
      cin >> p[i].y;
    }
    p[i].x = p[0].x;
    p[i].y = p[0].y;
    xmin = xmax = p[0].x;
    ymin = ymax = p[0].y;
  } else {
    cout << "\n Enter valid no. of vertices.";
  }
}
```

```
void poly::calcs() {
  for (int i = 0; i < v; i++) {
    if (xmin > p[i].x)
      xmin = p[i].x;
    if (xmax < p[i].x)
      xmax = p[i].x;
    if (ymin > p[i].y)
      ymin = p[i].y;
    if (ymax < p[i].y)
      ymax = p[i].y;
  }
}

void poly::display() {
  int ch1;
  char ch = 'y';
  float s, s2;
  do {
    cout << "\n\nMENU:";
    cout << "\n\n\t1 . Scan line Fill ";
    cout << "\n\n\t2 . Exit ";
    cout << "\n\nEnter your choice:";
    cin >> ch1;
    switch (ch1) {
    case 1:
      s = ymin + 0.01;
      delay(100);
      cleardevice();
```

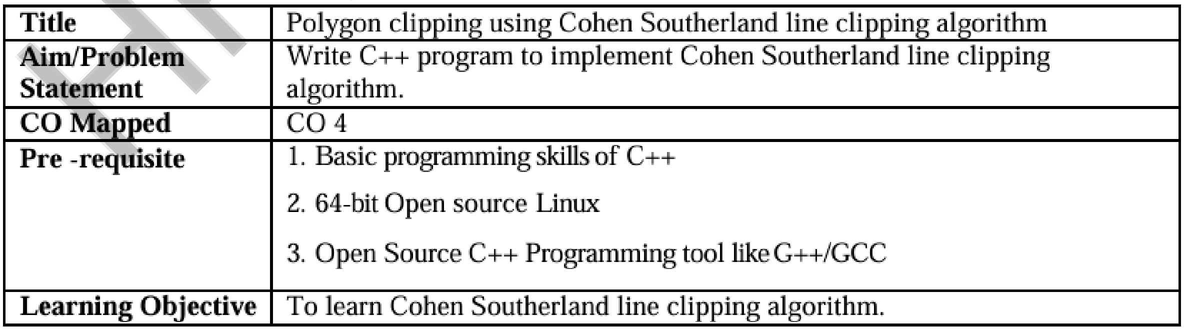
```
            while (s <= ymax) {
                ints(s);
                sort(s);
                s++;
            }
            break;
        case 2:
            exit(0);
        }
        cout << "Do you want to continue?: ";
        cin >> ch;
    } while (ch == 'y' || ch == 'Y');
}

void poly::ints(float z) {
    int x1, x2, y1, y2, temp;
    c = 0;
    for (int i = 0; i < v; i++) {
        x1 = p[i].x;
        y1 = p[i].y;
        x2 = p[i + 1].x;
        y2 = p[i + 1].y;
        if (y2 < y1) {
            temp = x1;
            x1 = x2;
            x2 = temp;
            temp = y1;
            y1 = y2;
            y2 = temp;
        }
        if (z <= y2 && z >= y1) {
            if ((y1 - y2) == 0)
                x = x1;
            else {
                x = ((x2 - x1) * (z - y1)) / (y2 - y1);
                x = x + x1;
            }
            if (x <= xmax && x >= xmin)
                inter[c++] = x;
        }
    }
}

void poly::sort(int z) { // sorting
    int temp, j, i;
    for (i = 0; i < v; i++) {
        line(p[i].x, p[i].y, p[i + 1].x, p[i + 1].y);
    }
    delay(100);
    for (i = 0; i < c; i += 2) {
        delay(100);
        line(inter[i], z, inter[i + 1], z);
    }
}

int main() { // main
```

```
 int cl;
 initwindow(500, 600);
 cleardevice();
 poly x;
 x.read();
 x.calcs();
 cleardevice();
 cout << "\n\tEnter The Color You Want :(In Range 0 To 15 )->"; // selecting color
 cin >> cl;
 setcolor(cl);
 x.display();

 closegraph(); // closing graph
 getch();
 return 0;
}
```

Output -----

```
 Scan Fill Algorithm
 Enter Number Of Vertices Of Polygon: 4

Enter co-ordinate no. 1 :
     x1=100
```

```
    y1=100

Enter co-ordinate no. 2 :
    x2=200

    y2=100

Enter co-ordinate no. 3 :
    x3=200

    y3=200

Enter co-ordinate no. 4 :
    x4=100

    y4=200

    Enter The Color You Want :(In Range 0 To 15 )->12
```



Title	Polygon clipping using Cohen Southerland line clipping algorithm
Aim/Problem Statement	Write C++ program to implement Cohen Southerland line clipping algorithm.
CO Mapped	CO 4
Pre -requisite	1. Basic programming skills of C++ 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC
Learning Objective	To learn Cohen Southerland line clipping algorithm.

Program :

```
#include<iostream>
```

```
#include<stdlib.h>

#include<math.h>

#include<graphics.h>

#include<dos.h>

using namespace std;

class Coordinate

{

    public:

        int x,y;

        char code[4];

};

class Lineclip

{

    public:

        Coordinate PT;

        void drawwindow();

        void drawline(Coordinate p1,Coordinate p2);

        Coordinate setcode(Coordinate p);

        int visibility(Coordinate p1,Coordinate p2);

        Coordinate resetendpt(Coordinate p1,Coordinate p2);

};

int main()

{

    Lineclip lc;

    int gd = DETECT,v,gm;

    Coordinate p1,p2,p3,p4,ptemp;


    cout<<"\n Enter x1 and y1\n";
```

```
cin>>p1.x>>p1.y;

cout<<"\n Enter x2 and y2\n";

cin>>p2.x>>p2.y;


initgraph(&gd,&gm,"");

lc.drawwindow();

delay(2000);


lc.drawline (p1,p2);

delay(2000);

cleardevice();


delay(2000);

p1=lc.setcode(p1);

p2=lc.setcode(p2);

v=lc.visibility(p1,p2);

delay(2000);


switch(v)

{

        case 0: lc.drawwindow();

                        delay(2000);

                        lc.drawline(p1,p2);

                        break;

  case 1:lc.drawwindow();

        delay(2000);

        break;

  case 2:p3=lc.resetendpt(p1,p2);
```

```
                        p4=lc.resetendpt(p2,p1);

                        lc.drawwindow();

                        delay(2000);

                        lc.drawline(p3,p4);

                        break;


  }

   delay(2000);

   closegraph();

}



void Lineclip::drawwindow()

{

        line(150,100,450,100);

        line(450,100,450,350);

        line(450,350,150,350);

        line(150,350,150,100);


}


void Lineclip::drawline(Coordinate p1,Coordinate p2)

{

        line(p1.x,p1.y,p2.x,p2.y);

}


Coordinate Lineclip::setcode(Coordinate p)
```

```
{
        Coordinate ptemp;



        if(p.y<100)
          ptemp.code[0]='1';
        else
          ptemp.code[0]='0';

        if(p.y>350)
                ptemp.code[1]='1';
        else
                ptemp.code[1]='0';



        if(p.x>450)
                ptemp.code[2]='1';
        else
                ptemp.code[2]='0';

        if(p.x<150)
                ptemp.code[3]='1';
        else
                ptemp.code[3]='0';



        ptemp.x=p.x;
        ptemp.y=p.y;
        return(ptemp);

};



int Lineclip:: visibility(Coordinate p1,Coordinate p2)
{
        int i,flag=0;

        for(i=0;i<4;i++)
        {
                if(p1.code[i]!='0' || (p2.code[i]=='1'))
                 flag='0';
        }

        if(flag==0)
         return(0);

                for(i=0;i<4;i++)
        {
                if(p1.code[i]==p2.code[i] && (p2.code[i]=='1'))
                 flag='0';
        }

        if(flag==0)
                return(1);
```

```
    return(2);
}

Coordinate Lineclip::resetendpt(Coordinate p1,Coordinate p2)
{
    Coordinate temp;
    int x,y,i;
    float m,k;



    if(p1.code[3]=='1')
        x=150;
    if(p1.code[2]=='1')
        x=450;
    if((p1.code[3]=='1') || (p1.code[2])=='1')
    {



        m=(float)(p2.y-p1.y)/(p2.x-p1.x);
        k=(p1.y+(m*(x-p1.x)));
        temp.y=k;
        temp.x=x;

        for(i=0;i<4;i++)
            temp.code[i]=p1.code[i];

     if(temp.y<=350 && temp.y>=100)
```

```
        return (temp);
    }



    if(p1.code[0]=='1')
        y=100;
    if(p1.code[1]=='1')
        y=350;
    if((p1.code[1]=='1') || (p1.code[1]=='1'))
    {
        m=(float)(p2.y-p1.y)/(p2.x-p1.x);
        k=(float)p1.x+(float)(y-p1.y)/m;
        temp.x=k;
        temp.y=y;

        for(i=0;i<4;i++)
            temp.code[i]=p1.code[i];

        return(temp);


    }
    else
        return(p1);


}
```

Input :

X1 , Y1:

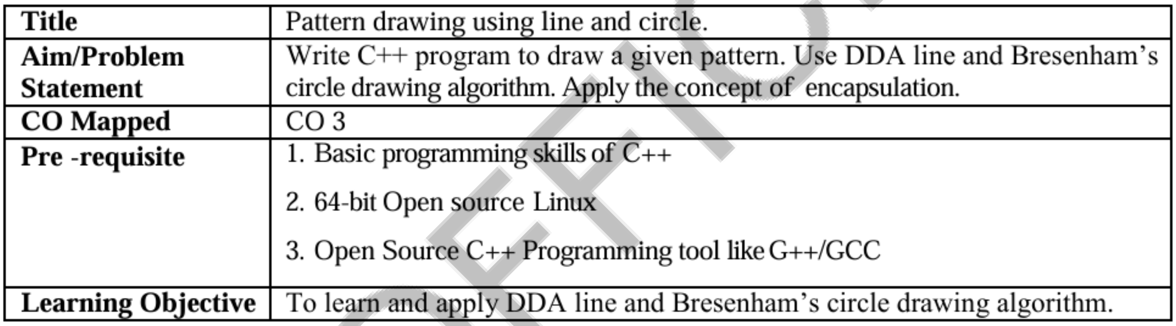
100

200

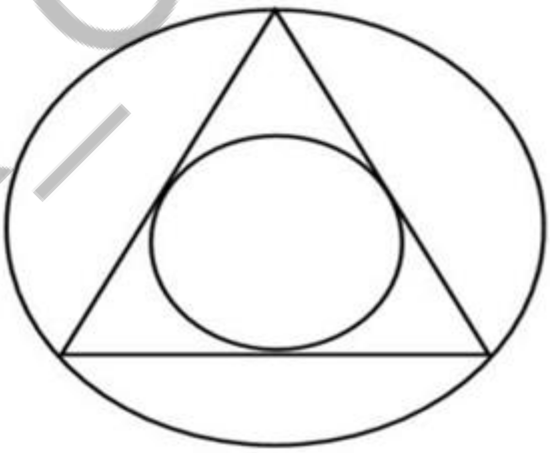
X2, Y2 :

500

100



Title	Pattern drawing using line and circle.
Aim/Problem Statement	Write C++ program to draw a given pattern. Use DDA line and Bresenham's circle drawing algorithm. Apply the concept of encapsulation.
CO Mapped	CO 3
Pre -requisite	1. Basic programming skills of C++ 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC
Learning Objective	To learn and apply DDA line and Bresenham's circle drawing algorithm.



Program :

```
#include <iostream>

# include <graphics.h>
# include <stdlib.h>
using namespace std;

class dcircle
{
private: int x0, y0;
public:
dcircle()
{
x0=0;
y0=0;
}
void setoff(int xx, int yy)
{
x0=xx;
y0=yy;
}
void drawc(int x1, int y1, int r)
{
float d;
int x,y;
x=0;
y=r;
d=3-2*r;
do
{
putpixel(x1+x0+x, y0+y-y1, 15);
putpixel(x1+x0+y, y0+x-y1,15);
putpixel(x1+x0+y, y0-x-y1,15);

putpixel(x1+x0+x,y0-y-y1,15);

putpixel(x1+x0-x,y0-y-y1,15);

putpixel(x1+x0-y, y0-x-y1,15);

putpixel(x1+x0-y, y0+x-y1,15);

putpixel(x1+x0-x, y0+y-y1,15);

if (d<=0)

{

d = d+4*x+6;

}

else

{

d=d+4*(x-y)+10;

y=y-1;

}

x=x+1;

}

while(x<y);

}

};


class pt

{

protected: int xco, yco,color;

public:

pt()

{

xco=0,yco=0,color=15;
```

```
}
void setco(int x, int y)
{
xco=x;
yco=y;
}
void setcolor(int c)
{
color=c;
}
void draw()
{
putpixel(xco,yco,color);
}
};
class dline:public pt
{
private: int x2, y2;
public:
dline():pt()
{
x2=0;
y2=0;
}
void setline(int x, int y, int xx, int yy)
{
pt::setco(x,y);
x2=xx;
```

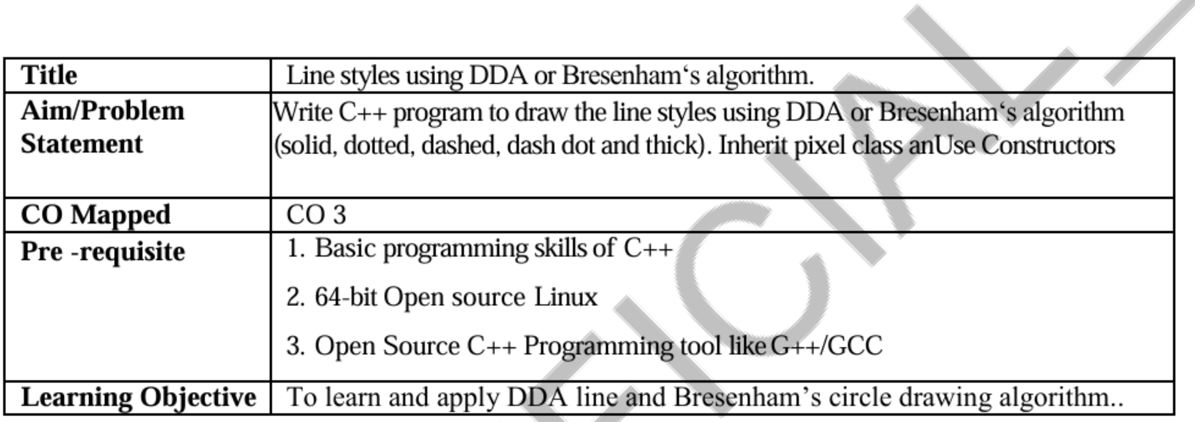
```
y2=yy;
}
void drawl( int colour)
{
float x,y,dx,dy,length;
int i;
pt::setcolor(colour);
dx= abs(x2-xco);
dy=abs(y2-yco);
if(dx>=dy)
{
length= dx;
}
else
{
length= dy;
}
dx=(x2-xco)/length;
dy=(y2-yco)/length;
x=xco+0.5;
y=yco+0.5;
i=1;
while(i<=length)
{
pt::setco(x,y);
pt::draw();
x=x+dx;
y=y+dy;
i=i+1;

}

pt::setco(x,y);

pt::draw();

}

};

int main()

{

int gd=DETECT, gm;

initgraph(&gd, &gm, NULL);

int x,y,r, x1, x2, y1, y2, xmax, ymax, xmid, ymid, n, i;

dcircle c;

cout<<"\nenter coordinates of centre of circle : ";

cout<<"\n enter the value of x : ";

cin>>x;

cout<<"\nenter the value of y : ";

cin>>y;

cout<<"\nenter the value of radius : ";

cin>>r;

xmax= getmaxx();

ymax=getmaxy();

xmid=xmax/2;

ymid=ymax/2;

setcolor(1);

c.setoff(xmid,ymid);

line(xmid, 0, xmid, ymax);

line(0,ymid,xmax,ymid);

setcolor(15);
```

```
c.drawc(x,y,r);
pt p1;
p1.setco(100,100);
p1.setcolor(14);
dline l;
l.setline(x1+xmid, ymid-y1, x2+xmid, ymid-y2);
cout<<"Enter Total Number of lines : ";
cin>>n;
for(i=0;i<n;i++)
{
cout<<"Enter co-ordinates of point x1 : ";
cin>>x1;
cout<<"enter coordinates of point y1 : ";
cin>>y1;
cout<<"Enter co-ordinates of point x2 : ";
cin>>x2;
cout<<"enter coordinates of point y2 : ";
cin>>y2;
l.setline(x1+xmid, ymid-y1, x2+xmid, ymid-y2);
l.drawl(15);
}
cout<<"\nEnter coordinates of centre of circle : ";
cout<<"\n Enter the value of x : ";
cin>>x;
cout<<"\nEnter the value of y : ";
cin>>y;
cout<<"\nEnter the value of radius : ";
cin>>r;
```

```
setcolor(5);
c.drawc(x,y,r);
getch();
delay(200);
closegraph();
return 0;
}
```

```
Input :
enter coordinates of centre of circle :
 enter the value of x : 100


enter the value of y : 70


enter the value of radius : 30
Enter Total Number of lines : 3
Enter co-ordinates of point x1 : 40
enter coordinates of point y1 : 40
Enter co-ordinates of point x2 : 100
enter coordinates of point y2 : 124
Enter co-ordinates of point x1 : 40
enter coordinates of point y1 : 40
Enter co-ordinates of point x2 : 160
enter coordinates of point y2 : 40
Enter co-ordinates of point x1 : 160
enter coordinates of point y1 : 40
Enter co-ordinates of point x2 : 100
enter coordinates of point y2 : 124
```

```
Enter coordinates of centre of circle :

 Enter the value of x : 100


Enter the value of y : 62


Enter the value of radius : 60
```



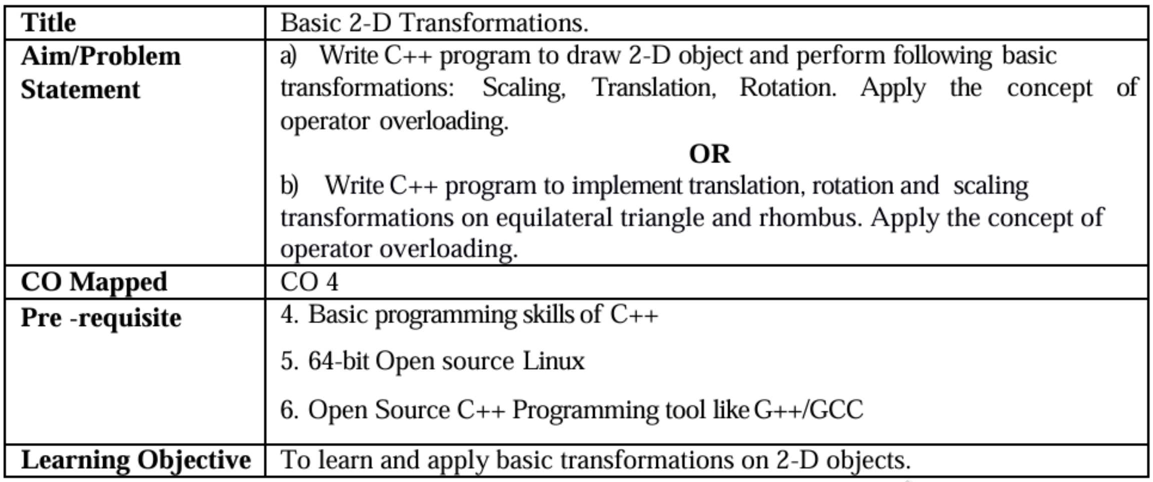
Title	Line styles using DDA or Bresenham‘s algorithm.
Aim/Problem Statement	Write C++ program to draw the line styles using DDA or Bresenham‘s algorithm (solid, dotted, dashed, dash dot and thick). Inherit pixel class anUse Constructors
CO Mapped	CO 3
Pre -requisite	1. Basic programming skills of C++ 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC
Learning Objective	To learn and apply DDA line and Bresenham’s circle drawing algorithm..

Program :

```
#include <iostream>

#include <graphics.h>

#include <math.h>

using namespace std;


class pixel {

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

public:

   void dda(int x1, int y1, int x2, int y2, int col, int ch);

};


void pixel::dda(int x1, int y1, int x2, int y2, int col, int ch) {

   float x, y, dx, dy;
```

```
int len;

dx = x2 - x1;
dy = y2 - y1;

if (abs(dx) > abs(dy)) {
   len = abs(dx);
} else {
   len = abs(dy);
}

dx = (x2 - x1) / (float)len;
dy = (y2 - y1) / (float)len;

x = x1;
y = y1;

putpixel(x, y, col);

int i;
int count = 4;
switch (ch) {
   case 1:  // Normal Line
      i = 1;
      while (i <= len) {
         x = x + dx;
         y = y + dy;
         putpixel(x, y, col);
      i++;
    }
    break;

  case 2:  // Dotted Line
    i = 1;
    while (i <= len) {
      x = x + dx;
      y = y + dy;
      if (i % 3 == 0) {  // Dots every third pixel
        putpixel(x, y, col);
      }
      i++;
    }
    break;

  case 3:  // Dashed Line
    i = 1;
    while (i <= len) {
      x = x + dx;
      y = y + dy;
      if (i % 5 < 3) {  // Dashes of length 3
        putpixel(x, y, col);
      }
      i++;
    }
    break;
    case 4:  // Dash Dot Dash Line
      i = 1;
      while (i <= len) {
        x = x + dx;
        y = y + dy;
        if (i % 10 < 4 || (i % 10 == 8)) {  // Dash for 4 pixels, then dot
          putpixel(x, y, col);
        }
        i++;
      }
      break;

    case 5:  // Thick Line
      i = 1;
      while (i <= len) {
        x = x + dx;
        y = y + dy;
        putpixel(x, y, col);
        putpixel(x, y + 1, col);  // Adding a second line for thickness
        i++;
      }
      break;

    default:
      cout << "Invalid choice\n";
      break;
  }
}
```

```
int main() {
  pixel p;
  int gd = DETECT, gm, ch;
  initgraph(&gd, &gm, NULL);

  do {
    cout << "\n1. Normal Line\n2. Dotted Line\n3. Dashed Line\n4. Dash Dot Dash Line\n5. Thick Line\nEnter your choice: ";
    cin >> ch;

    switch (ch) {
      case 1: p.dda(100, 100, 500, 100, 1, 1); break;
      case 2: p.dda(100, 130, 500, 130, 2, 2); break;
      case 3: p.dda(100, 160, 500, 160, 3, 3); break;
      case 4: p.dda(100, 190, 500, 190, 4, 4); break;
      case 5: p.dda(100, 220, 500, 220, 5, 5); break;
      case 6: cout << "Exiting...\n"; break;
      default: cout << "Invalid choice\n"; break;
    }

  } while (ch != 6);

  delay(10000);
  closegraph();

  return 0;
}
```



Title	Basic 2-D Transformations.
Aim/Problem Statement	a) Write C++ program to draw 2-D object and perform following basic transformations: Scaling, Translation, Rotation. Apply the concept of operator overloading. **OR** b) Write C++ program to implement translation, rotation and scaling transformations on equilateral triangle and rhombus. Apply the concept of operator overloading.
CO Mapped	CO 4
Pre -requisite	4. Basic programming skills of C++ 5. 64-bit Open source Linux 6. Open Source C++ Programming tool like G++/GCC
Learning Objective	To learn and apply basic transformations on 2-D objects.

Program :

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

using namespace std;

class Transformation {
public:
  int x1, x2, y1, y2;
  void accept() {
    cout << "Enter coordinate x1 : ";
    cin >> x1;

    cout << "Enter coordinate y1 : ";
    cin >> y1;

    cout << "Enter coordinate x2 : ";
    cin >> x2;
    cout << "Enter coordinate y2 : ";
    cin >> y2;

    line(x1, y1, x2, y2);
  }

  void translate(){
  int tx, ty;
  cout << "Enter coordinate for point x : ";
  cin >> tx;
  cout << "Enter coordinate for point y : ";
  cin >> ty;

  line(x1+tx, y1+ty, x2+tx, y2+ty);
  }

  void scaling(){
  int sx, sy;
  cout << "Enter coordinate for point x : ";
  cin >> sx;
  cout << "Enter coordinate for point y : ";
  cin >> sy;

  line(x1*sx, y1*sy, x2*sx, y2*sy);
  }

  void rotation() {
  int Rx1, Ry1, Rx2, Ry2;
```

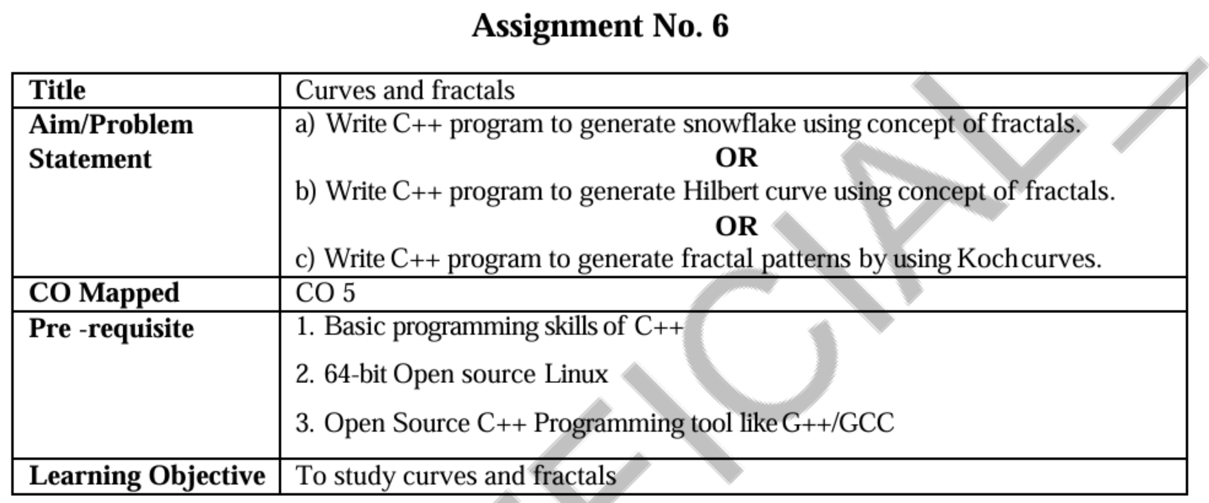
```
double s, c, angle;

cout << "Enter the angle to rotate the line : ";
cin >> angle;

// Convert the angle from degrees to radians
c = cos(angle * 3.14 / 180);
s = sin(angle * 3.14 / 180);

// Find the midpoint of the line
int mx = (x1 + x2) / 2;
int my = (y1 + y2) / 2;

// Translate the line to the origin (midpoint)
int tx1 = x1 - mx;
int ty1 = y1 - my;
int tx2 = x2 - mx;
int ty2 = y2 - my;

// Apply the rotation matrix
Rx1 = floor(tx1 * c - ty1 * s);
Ry1 = floor(tx1 * s + ty1 * c);
Rx2 = floor(tx2 * c - ty2 * s);
Ry2 = floor(tx2 * s + ty2 * c);

// Translate back to the original position
Rx1 += mx;
Ry1 += my;
```

```
  Rx2 += mx;
  Ry2 += my;

  // Draw the rotated line
  line(Rx1, Ry1, Rx2, Ry2);
}

};

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

  Transformation t;
  int ch;
  char q;

  t.accept();
  do {
    cout << "Menu : ";
    cout << "\n1. Translate";
    cout << "\n2. Scale";
    cout << "\n3. Rotate";
    cout << "\n4. Exit"<<endl;
    cout << "Enter your choice : ";

    cin >> ch;
  switch(ch){
  case 1 :
    t.translate();
    break;
  case 2 :
    t.scaling();
    break;
  case 3 :
    t.rotation();
    break;
  case 4 :
    exit(0);

  default :
    cout << "Invalid choice...";
    break;
    }

    cout << "Do you want to continue? (y/n): ";
    cin >> q;
  } while(q == 'y' || q == 'Y');

  getch();
  closegraph();
  return 0;
}
```

input :

```
Enter coordinate x1 : 50

Enter coordinate y1 : 50

Enter coordinate x2 : 200

Enter coordinate y2 : 200


Menu :

1. Translate

2. Scale

3. Rotate

4. Exit

Enter your choice : 1

Enter coordinate for point x : 30

Enter coordinate for point y : 40

Do you want to continue? (y/n): y


Menu :

1. Translate

2. Scale

3. Rotate

4. Exit

Enter your choice : 2

Enter coordinate for point x : 2

Enter coordinate for point y : 2

Do you want to continue? (y/n): y


Menu :

1. Translate

2. Scale
```

```
3. Rotate

4. Exit

Enter your choice : 3

Enter the angle to rotate the line : 45

Do you want to continue? (y/n): n
```



Assignment No. 6

Title	Curves and fractals
Aim/Problem Statement	a) Write C++ program to generate snowflake using concept of fractals. **OR** b) Write C++ program to generate Hilbert curve using concept of fractals. **OR** c) Write C++ program to generate fractal patterns by using Koch curves.
CO Mapped	CO 5
Pre -requisite	1. Basic programming skills of C++ 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC
Learning Objective	To study curves and fractals

Program : (Snowflake)

```
// Write C++ program to generate fractal patterns by using Koch curves.

#include<iostream>

#include<graphics.h>

#include<math.h>

using namespace std;

void snow(int x1, int y1, int x2, int y2, int it)

{

float angle = 60*M_PI/180;

int x3 = (2*x1+x2)/3;

int y3 = (2*y1+y2)/3;

int x4 = (x1+2*x2)/3;

int y4 = (y1+2*y2)/3;
int x = x3+(x4-x3)*cos(angle)+(y4-y3)*sin(angle);
int y = y3-(x4-x3)*sin(angle)+(y4-y3)*cos(angle);
if(it > 0)
{
snow(x1, y1, x3, y3, it-1);
snow(x3, y3, x, y, it-1);
snow(x, y, x4, y4, it-1);
snow(x4, y4, x2, y2, it-1);
}
else
{
line(x1, y1, x3, y3);
line(x3, y3, x, y);
line(x, y, x4, y4);
line(x4, y4, x2, y2);
}
}
int main()
{
int gd = DETECT,gm;
initgraph(&gd, &gm, NULL);
int x1 = 150, y1 = 100, x2 = 350, y2 = 100;
snow(x1, y1, x2, y2,2);
snow(250,350,150,100,2);
snow(350,100,250,350,2);
getch();
return 0;
}
```

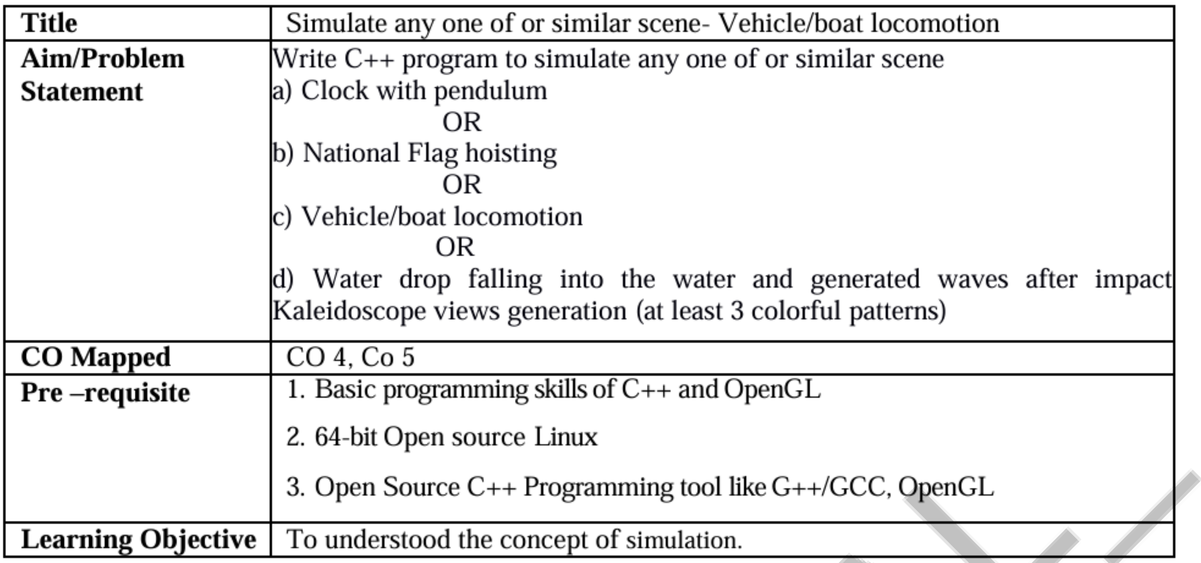
OR

```
#include <iostream>

#include <graphics.h>

#include <cmath>


using namespace std;


class KochCurve {

public:

   // Function to draw the Koch curve recursively

   void drawKochCurve(int x1, int y1, int x2, int y2, int iteration) {

      if (iteration == 0) {

         // Draw the line when no more iterations are left

         line(x1, y1, x2, y2);

      } else {

         // Calculate the points to divide the line into 3 equal parts

         int dx = x2 - x1;

         int dy = y2 - y1;


         // Divide the line into three parts

         int x3 = x1 + dx / 3;

         int y3 = y1 + dy / 3;


         int x4 = x1 + 2 * dx / 3;

         int y4 = y1 + 2 * dy / 3;


         // Calculate the peak of the equilateral triangle
```

```
        int x5 = x3 + (x4 - x3) / 2 - (y4 - y3) * sqrt(3) / 2;
        int y5 = y3 + (y4 - y3) / 2 + (x4 - x3) * sqrt(3) / 2;

        // Recursively draw the smaller Koch curves
        drawKochCurve(x1, y1, x3, y3, iteration - 1); // First segment
        drawKochCurve(x3, y3, x5, y5, iteration - 1); // Triangle peak
        drawKochCurve(x5, y5, x4, y4, iteration - 1); // Third segment
        drawKochCurve(x4, y4, x2, y2, iteration - 1); // Final segment
    }
}

// Function to draw the full Koch snowflake (starting with an equilateral triangle)
void drawKochSnowflake(int x1, int y1, int x2, int y2, int iteration) {
    // First side of the triangle
    drawKochCurve(x1, y1, x2, y2, iteration);

    // Second side of the triangle
    int x3 = (x1 + x2) / 2 + (y1 - y2) * sqrt(3) / 2;
    int y3 = (y1 + y2) / 2 - (x2 - x1) * sqrt(3) / 2;
    drawKochCurve(x2, y2, x3, y3, iteration);

    // Third side of the triangle
    x3 = (x1 + x2) / 2 + (y1 - y2) * sqrt(3) / 2;
    y3 = (y1 + y2) / 2 - (x2 - x1) * sqrt(3) / 2;
    drawKochCurve(x3, y3, x1, y1, iteration);
}

// Function to calculate the bounding box for the triangle to center it
```

```
  void centerKochSnowflake(int &x1, int &y1, int &x2, int &y2, int &x3, int &y3) {
    // Adjust starting points to center the Koch snowflake in the screen
    int screenWidth = getmaxx();
    int screenHeight = getmaxy();

    // Calculate the center of the screen
    int centerX = screenWidth / 2;
    int centerY = screenHeight / 2;

    // Adjust the side length and calculate the coordinates of the triangle
    int sideLength = 300; // You can adjust this value based on your desired size of the snowflake
    x1 = centerX - sideLength / 2;
    y1 = centerY + sideLength / 2;

    x2 = centerX + sideLength / 2;
    y2 = y1;

    x3 = centerX;
    y3 = centerY - static_cast<int>(sideLength * sqrt(3) / 2); // Height of equilateral triangle
  }
};

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

  int x1, y1, x2, y2, x3, y3;
```

```
    int iterations;

    // Create object of KochCurve class
    KochCurve k;

    // Calculate the coordinates to center the Koch snowflake
    k.centerKochSnowflake(x1, y1, x2, y2, x3, y3);

    // Get user input for the number of iterations
    cout << "Enter the number of iterations for Koch Curve: ";
    cin >> iterations;

    // Draw the Koch snowflake fractal
    k.drawKochSnowflake(x1, y1, x2, y2, iterations); // Draw first side of triangle

    // Wait for user input before closing the graphics window
    getch();
    closegraph();
    return 0;
}
```



Title	Simulate any one of or similar scene- Vehicle/boat locomotion
Aim/Problem Statement	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 Kaleidoscope views generation (at least 3 colorful patterns)
CO Mapped	CO 4, Co 5
Pre –requisite	1. Basic programming skills of C++ and OpenGL 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC, OpenGL
Learning Objective	To understood the concept of simulation.

Program :

```
#include<iostream>

#include<graphics.h>

int main() {

   int gd=DETECT,gm;

   int i,maxx,midy;


   /* initialize graphic mode */

   initgraph(&gd,&gm,NULL);

   /* maximum pixel in horizontal axis */

   maxx=getmaxx();

   /* mid pixel in vertical axis */

   midy=getmaxy()/2;


   for(i=0;i<maxx-150;i=i+5)

{

      /* clears screen */

      cleardevice();
```

```
/* draw a white road */
setcolor(WHITE);
line(0,midy+37,maxx,midy+37);

/* Draw Car */
setcolor(YELLOW);
//setfillstyle(SOLID_FILL, RED);

line(i,midy+23,i,midy);
line(i,midy,40+i,midy-20);
line(40+i,midy-20,80+i,midy-20);
line(80+i,midy-20,100+i,midy);
line(100+i,midy,120+i,midy);
line(120+i,midy,120+i,midy+23);
line(0+i,midy+23, 18 + i, midy + 23);
arc(30+i,midy+23,0,180,12);
line(42+i,midy+23,78+i,midy+23);
arc(90+i,midy+23,0,180,12);
line(102+i,midy+23,120+i,midy+23);
line(28+i,midy,43+i,midy-15);
line(43+i,midy-15,57+i,midy-15);
line(57+i,midy-15,57+i,midy);
line(57+i,midy,28+i,midy);
line(62+i,midy-15,77+i,midy-15);
line(77+i,midy-15,92+i,midy);
line(92+i,midy,62+i,midy);
line(62+i,midy,62+i,midy-15);
```

```
    /*  Draw Wheels */
    circle(30 + i, midy + 25, 9);
circle(90 + i, midy + 25, 9);
/* Add delay  */
delay(100);
}
getch();
closegraph();
return 0;
}
```