1) Write C++ program to draw a concave polygon and fill it with desired colour using scan fill algorithm. Apply the concept of inheritance.

PROGRAM

```
#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
{
```

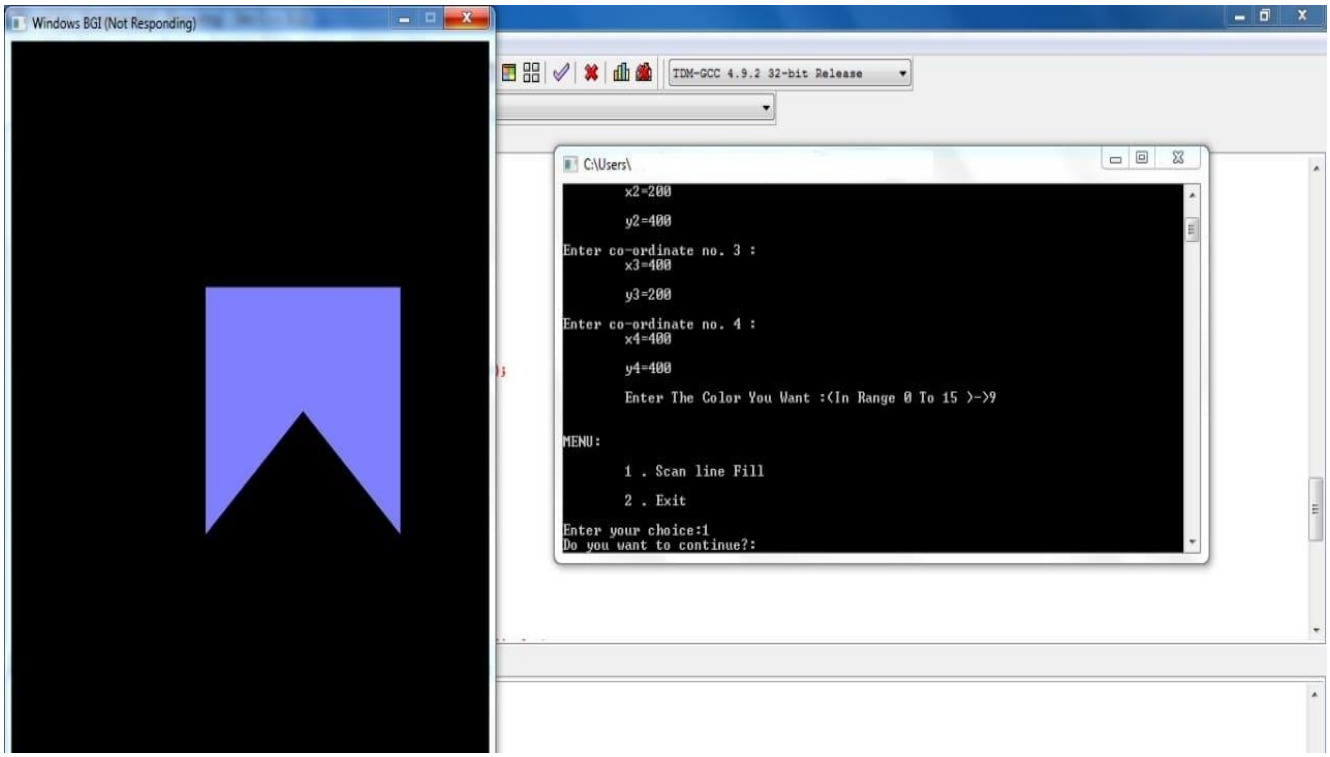
```
 int temp,j,i;
 for(i=0;i<v;i++)
 {
 line(p[i].x,p[i].y,p[i+1].x,p[i+1].y);
 }
 delay(100);
 for(i=0; i<c;i+=2)
 {
 delay(100);
 line(inter[i],z,inter[i+1],z);
 }
}
int main() //main
{
 int cl, gd=DETECT, gm;
 initgraph(&gd,&gm,NULL);
 cleardevice();
 poly x;
 x.read();
 x.calcs();
 cleardevice();
 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

```
Number of Vertices : 4

Cordinates 1st :
x1= 200
y1= 200

Cordinates 2st :
x2= 200
y2= 400

Cordinates 3st :
x3= 400
```

y3= 200

Cordinates 4st :
x4= 400
y4= 400



2) Write C++ program to implement Cohen Southerland line clipping algorithm.

PROGRAM

```
#include<iostream>
#include<stdlib.h>
#include<math.h>
#include<graphics.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,NULL);
	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);
```

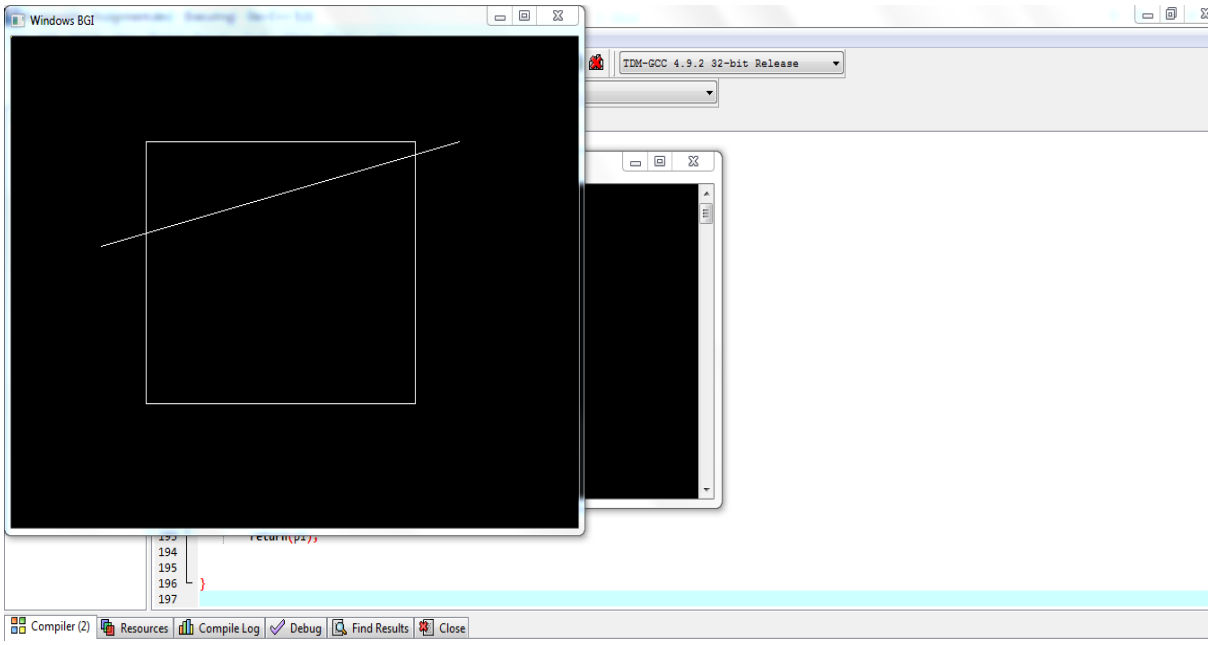
```
	return(2);
}

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);
```

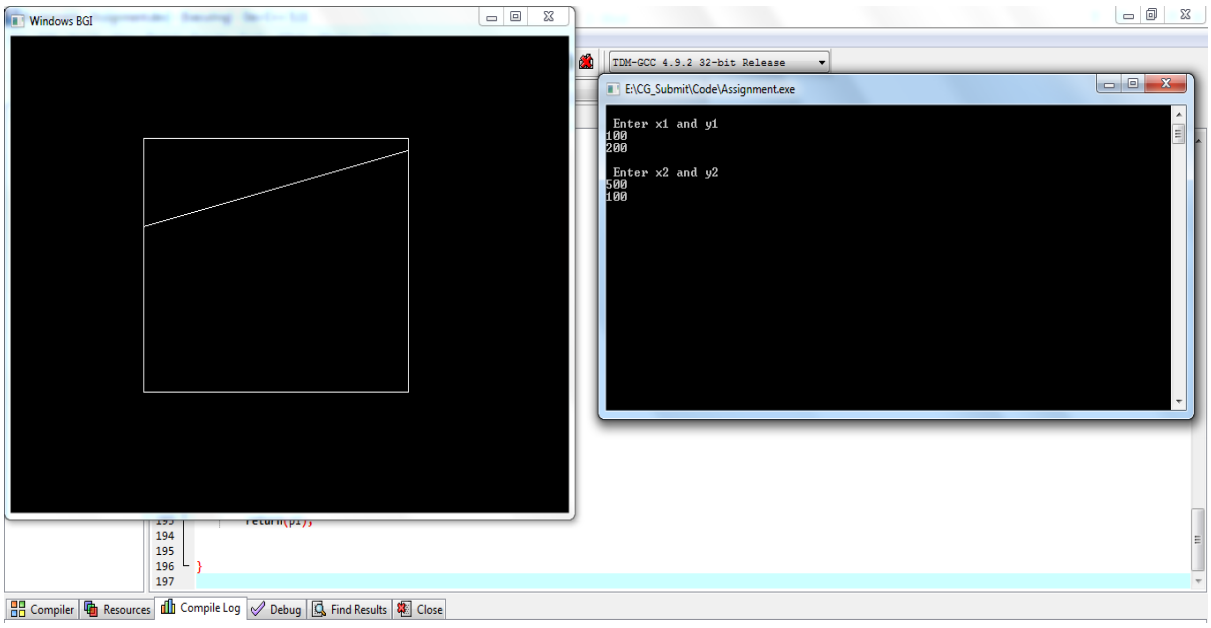
}

OUTPUT

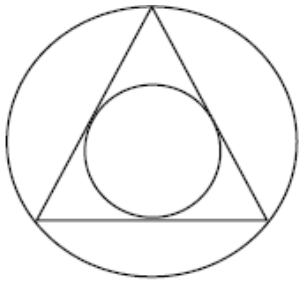
X1 , Y1:
100
200

X2, Y2 :
500
100





3) Write C++ program to draw the following pattern. Use DDA line and Bresenham's circle drawing algorithm. Apply the concept of encapsulation.



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()
{
```

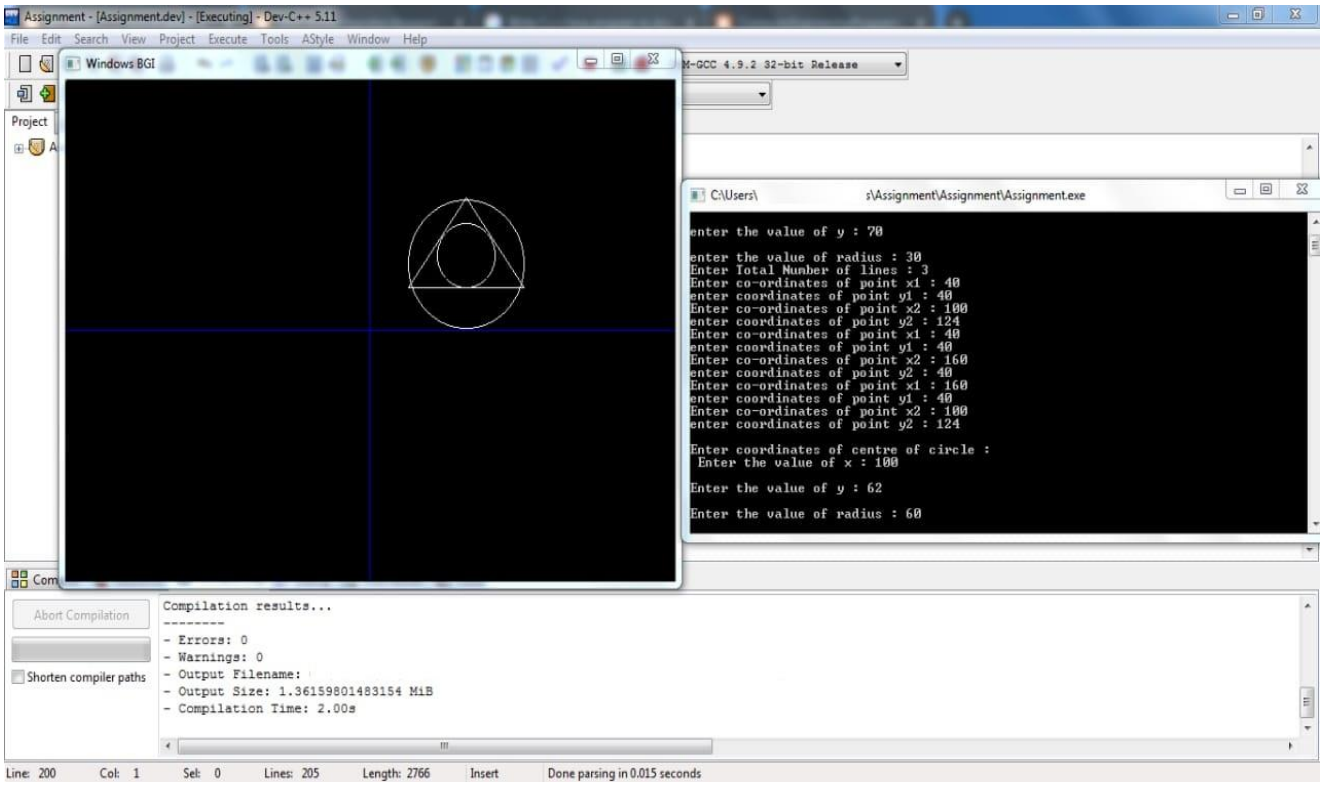
```
int gd=DETECT, gm;
initgraph(&gd, &gm, NULL);
int x,y,r, x1, x2, y1, y2, xmax, ymax, xmid, ymid, n, i;
dcircle c;
cout<<"\nenter coordinates of centre of circle : ";
cout<<"\n enter the value of x : ";
cin>>x;
cout<<"\nenter the value of y : ";
cin>>y;
cout<<"\nenter the value of radius : ";
cin>>r;
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;
}
```

OUTPUT

Value Of X : 100
Value Of Y : 70
Value Of R : 30

Next Inputs In Image Given Below.



4) Write C++ program to draw 2-D object and perform transformation [Scaling, Translation & Rotation]. Apply the concept of operator overloading.

PROGRAM

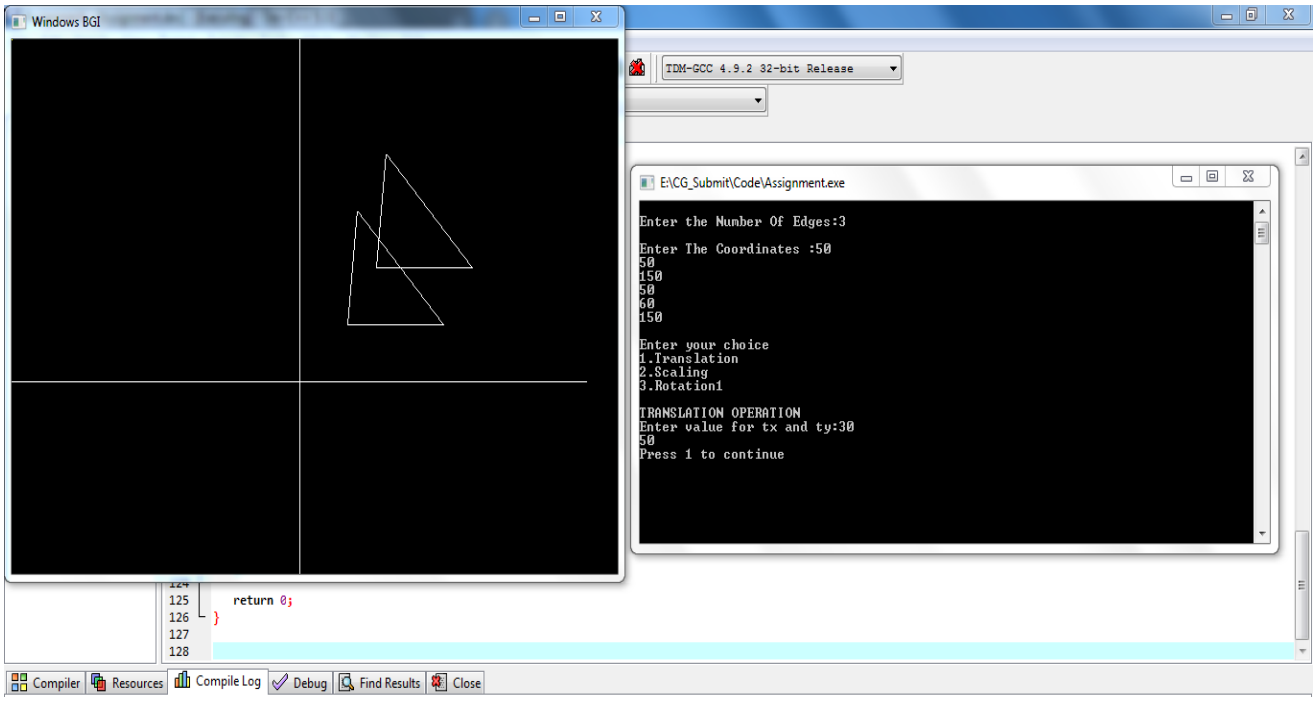
```
#include<iostream>
#include<graphics.h>
#include<math.h>
using namespace std;
class transform
{
        public:
                int m,a[20][20],c[20][20];
                int i,j,k;
                public:

                void object();
                void accept();
                void operator *(float b[20][20])
                {
                        for(int i=0;i<m;i++)
                        {
                                for(int j=0;j<m;j++)
                                {
                                        c[i][j]=0;
                                        for(int k=0;k<m;k++)
                                        {
                                                c[i][j]=c[i][j]+(a[i][k]*b[k][j]);
                                        }
                                }
                        }
                }
};
void transform::object()
{
         int gd,gm;
        gd=DETECT;
        initgraph(&gd,&gm,NULL);
  line(300,0,300,600);
  line(0,300,600,300);
   for( i=0;i<m-1;i++)
   {
        line(300+a[i][0],300-a[i][1],300+a[i+1][0],300-a[i+1][1]);
```

```
        }
        line(300+a[0][0],300-a[0][1],300+a[i][0],300-a[i][1]);
        for( i=0;i<m-1;i++)
        {

                line(300+c[i][0],300-c[i][1],300+c[i+1][0],300-c[i+1][1]);
        }
        line(300+c[0][0],300-c[0][1],300+c[i][0],300-c[i][1]);
        int temp;
        cout << "Press 1 to continue";
        cin >> temp;
        closegraph();
}
void transform::accept()
{
cout<<"\n";
 cout<<"Enter the Number Of Edges:";
   cin>>m;
   cout<<"\nEnter The Coordinates :";
   for(int i=0;i<m;i++)
   {
        for(int j=0;j<3;j++)
        {
                if(j>=2)
                a[i][j]=1;
                else
                cin>>a[i][j];
                }
        }
}
int main()
{
        int ch,tx,ty,sx,sy;
        float deg,theta,b[20][20];
        transform t;
        t.accept();

           cout<<"\nEnter your choice";
           cout<<"\n1.Translation"
                "\n2.Scaling"
                        "\n3.Rotation";
                        cin>>ch;
                switch(ch)
                {
                case 1: cout<<"\nTRANSLATION OPERATION\n";
```

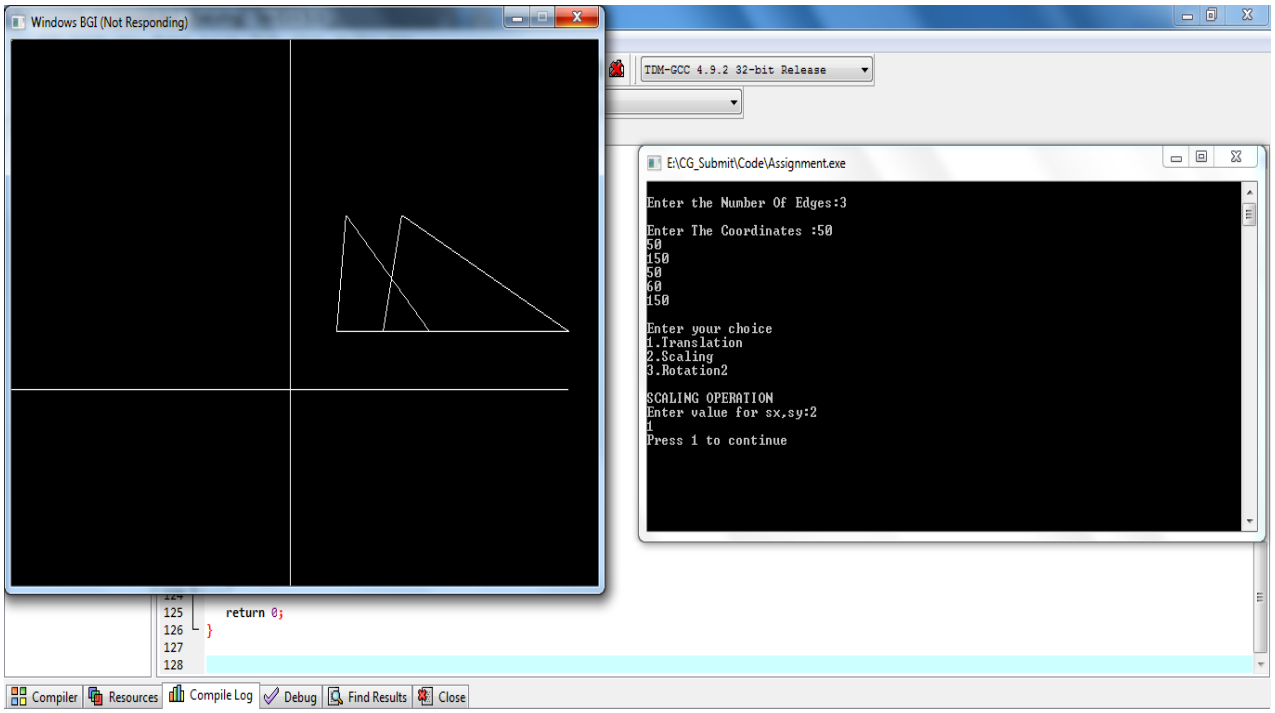
```
                cout<<"Enter value for tx and ty:";
                cin>>tx>>ty;
                b[0][0]=b[2][2]=b[1][1]=1;
                        b[0][1]=b[0][2]=b[1][0]=b[1][2]=0;
                        b[2][0]=tx;
                        b[2][1]=ty;
                        t * b;


                        t.object();
                        break;
        case 2: cout<<"\nSCALING OPERATION\n";
                cout<<"Enter value for sx,sy:";
                cin>>sx>>sy;
                b[0][0]=sx;
                b[1][1]=sy;
                b[0][1]=b[0][2]=b[1][0]=b[1][2]=0;
                b[2][0]=b[2][1]=0;
                        b[2][2] = 1;
                        t * b;
                        t.object();
                        break;
            case 3: cout<<"\nROTATION OPERATION\n";
                cout<<"Enter value for angle:";
                cin>>deg;
                        theta=deg*(3.14/100);
                        b[0][0]=b[1][1]=cos(theta);
                        b[0][1]=sin(theta);
                        b[1][0]=sin(-theta);
                        b[0][2]=b[1][2]=b[2][0]=b[2][1]=0;
                        b[2][2]=1;
                        t * b;
                        t.object();
                        break;
            default:
               cout<<"\nInvalid choice";

            }

  getch();

  return 0;
}
```

OUTPUT

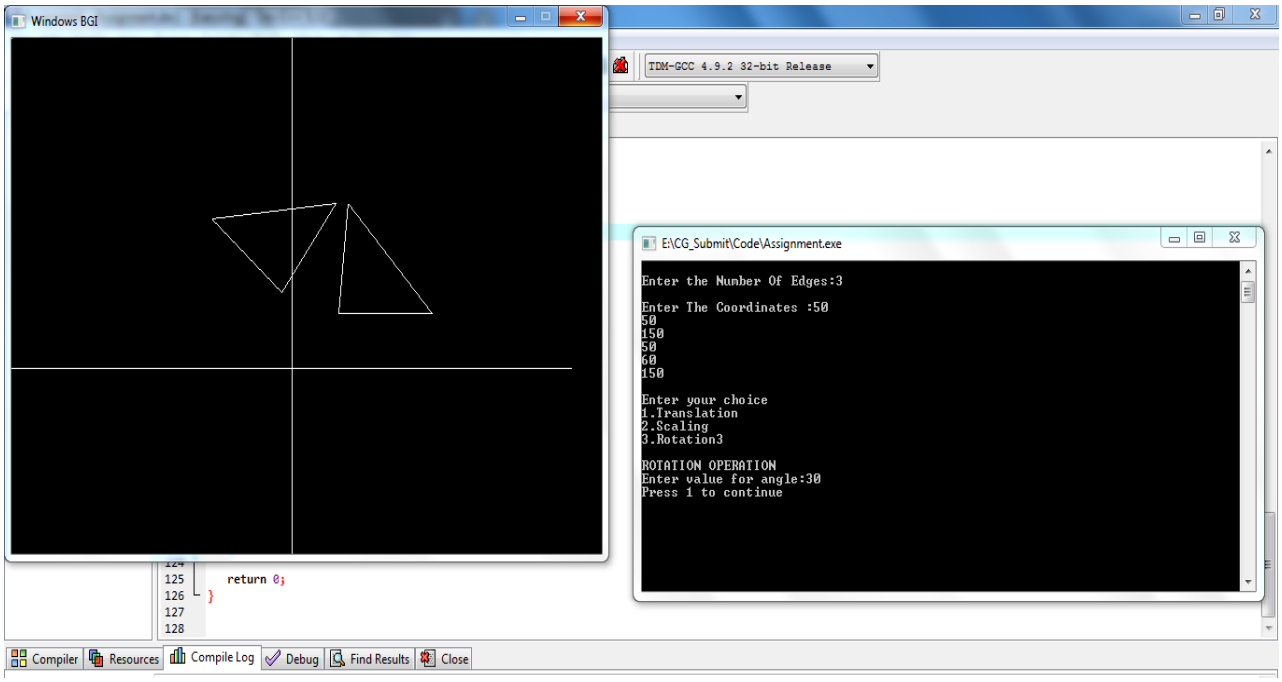
For Tranlation :



For Scaling :



For Rotation :



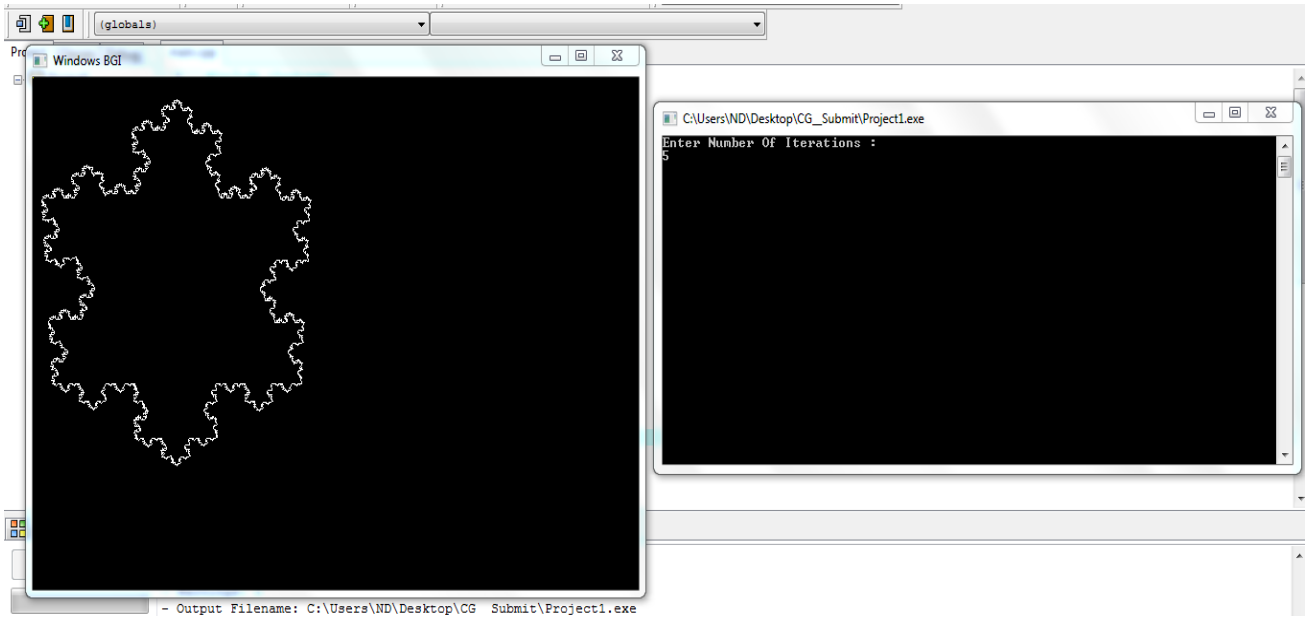
5) Write C++ Program To Generate Fractal Patterns By Using Koch Curves

PROGRAM

```
#include<iostream>
#include<math.h>
#include<graphics.h>
using namespace std;
class kochCurve
{
public:
void koch(int it,int x1,int y1,int x5,int y5)
{
int x2,y2,x3,y3,x4,y4;
int dx,dy;
if (it==0)
{
line(x1,y1,x5,y5);
}
else
{
delay(10);
dx=(x5-x1)/3;
dy=(y5-y1)/3;
x2=x1+dx;
y2=y1+dy;
x3=(int)(0.5*(x1+x5)+sqrt(3)*(y1-y5)/6);
y3=(int)(0.5*(y1+y5)+sqrt(3)*(x5-x1)/6);
x4=2*dx+x1;
y4=2*dy+y1;
koch(it-1,x1,y1,x2,y2);
koch(it-1,x2,y2,x3,y3);
koch(it-1,x3,y3,x4,y4);
koch(it-1,x4,y4,x5,y5);
}
}
};
int main()
{
kochCurve k;
int it;
cout<<"Enter Number Of Iterations : "<<endl;
cin>>it;
int gd=DETECT,gm;
```

```
initgraph(&gd,&gm,NULL);
k.koch(it,150,20,20,280);
k.koch(it,280,280,150,20);
k.koch(it,20,280,280,280);
getch();
closegraph();
return 0;
}
```

OUTPUT



6) Write C++ program to draw 3-D cube and perform following transformations on it using OpenGL i) Translation ii) Scaling iii) Rotation about an axis (X/Y/Z)

PROGRAM

```
#include<iostream>
#include<math.h>
#include<GL/glut.h>
using namespace std;
typedef float Matrix4 [4][4];
Matrix4 theMatrix;
static GLfloat input[8][3]=
{
 {40,40,-50},{90,40,-50},{90,90,-50},{40,90,-50},
 {30,30,0},{80,30,0},{80,80,0},{30,80,0}
};
float output[8][3];
float tx,ty,tz;
float sx,sy,sz;
float angle;
int choice,choiceRot;
void setIdentityM(Matrix4 m)
{
for(int i=0;i<4;i++)
 for(int j=0;j<4;j++)
 m[i][j]=(i==j);
}
void translate(int tx,int ty,int tz)
{
for(int i=0;i<8;i++)
{
output[i][0]=input[i][0]+tx;
output[i][1]=input[i][1]+ty;
output[i][2]=input[i][2]+tz;
}
}
void scale(int sx,int sy,int sz)
{
 theMatrix[0][0]=sx;
 theMatrix[1][1]=sy;
 theMatrix[2][2]=sz;
}
void RotateX(float angle) //Parallel to x
```

```
{
angle = angle*3.142/180;
 theMatrix[1][1] = cos(angle);
theMatrix[1][2] = -sin(angle);
theMatrix[2][1] = sin(angle);
theMatrix[2][2] = cos(angle);
}
void RotateY(float angle) //parallel to y
{
angle = angle*3.14/180;
theMatrix[0][0] = cos(angle);
theMatrix[0][2] = -sin(angle);
theMatrix[2][0] = sin(angle);
theMatrix[2][2] = cos(angle);
}
void RotateZ(float angle) //parallel to z
{
angle = angle*3.14/180;
theMatrix[0][0] = cos(angle);
theMatrix[0][1] = sin(angle);
theMatrix[1][0] = -sin(angle);
theMatrix[1][1] = cos(angle);
}
void multiplyM()
{
//We Don't require 4th row and column in scaling and rotation
//[8][3]=[8][3]*[3][3] //4th not used
for(int i=0;i<8;i++)
{
 for(int j=0;j<3;j++)
 {
 output[i][j]=0;
 for(int k=0;k<3;k++)
 {
 output[i][j]=output[i][j]+input[i][k]*theMatrix[k][j];
 }
 }
}
}
void Axes(void)
{
glColor3f (0.0, 0.0, 0.0); // Set the color to BLACK
glBegin(GL_LINES); // Plotting X-Axis
glVertex2s(-1000 ,0);
glVertex2s( 1000 ,0);
```

```
glEnd();
glBegin(GL_LINES); // Plotting Y-Axis
glVertex2s(0 ,-1000);
glVertex2s(0 , 1000);
glEnd();
}
void draw(float a[8][3])
{
 glBegin(GL_QUADS);
 glColor3f(0.7,0.4,0.5); //behind
 glVertex3fv(a[0]);
 glVertex3fv(a[1]);
 glVertex3fv(a[2]);
 glVertex3fv(a[3]);
 glColor3f(0.8,0.2,0.4); //bottom
 glVertex3fv(a[0]);
 glVertex3fv(a[1]);
 glVertex3fv(a[5]);
 glVertex3fv(a[4]);
 glColor3f(0.3,0.6,0.7); //left
 glVertex3fv(a[0]);
 glVertex3fv(a[4]);
 glVertex3fv(a[7]);
 glVertex3fv(a[3]);
 glColor3f(0.2,0.8,0.2); //right
glVertex3fv(a[1]);
glVertex3fv(a[2]);
glVertex3fv(a[6]);
glVertex3fv(a[5]);
glColor3f(0.7,0.7,0.2); //up
glVertex3fv(a[2]);
glVertex3fv(a[3]);
glVertex3fv(a[7]);
glVertex3fv(a[6]);
glColor3f(1.0,0.1,0.1);
glVertex3fv(a[4]);
glVertex3fv(a[5]);
glVertex3fv(a[6]);
glVertex3fv(a[7]);
glEnd();
}
void init()
{
 glClearColor(1.0,1.0,1.0,1.0); //set backgrond color to white
 glOrtho(-454.0,454.0,-250.0,250.0,-250.0,250.0);
```

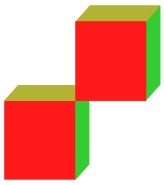
```
 // Set the no. of Co-ordinates along X & Y axes and their gappings
 glEnable(GL_DEPTH_TEST);
 // To Render the surfaces Properly according to their depths
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
Axes();
glColor3f(1.0,0.0,0.0);
draw(input);
setIdentityM(theMatrix);
switch(choice)
{
case 1:
 translate(tx,ty,tz);
 break;
case 2:
 scale(sx,sy,sz);
multiplyM();
 break;
case 3:
 switch (choiceRot) {
 case 1:
 RotateX(angle);
 break;
 case 2: RotateY(angle);
 break;
 case 3:
 RotateZ(angle);
 break;
 default:
 break;
 }
multiplyM();
 break;
}
draw(output);
glFlush();
}
int main(int argc, char** argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
 glutInitWindowSize(1362,750);
 glutInitWindowPosition(0,0);
```

```
 glutCreateWindow("3D TRANSFORMATIONS");
 init();
 cout<<"Enter your choice number:\n1.Translation\n2.Scaling\n3.Rotation\n=>";
 cin>>choice;
 switch (choice) {
 case 1:
 cout<<"\nEnter Tx,Ty &Tz: \n";
 cin>>tx>>ty>>tz;
 break;
 case 2:
 cout<<"\nEnter Sx,Sy & Sz: \n";
 cin>>sx>>sy>>sz;
 break;
 case 3:
 cout<<"Enter your choice for Rotation about axis:\n1.parallel to X-axis."
 <<"(y& z)\n2.parallel to Y-axis.(x& z)\n3.parallel to Z-axis."
 <<"(x& y)\n =>";
 cin>>choiceRot;
 switch (choiceRot) {
 case 1:
 cout<<"\nENter Rotation angle: ";
 cin>>angle;
 break;
 case 2:
 cout<<"\nENter Rotation angle: ";
 cin>>angle;
 break;
 case 3:
 cout<<"\nENter Rotation angle: ";
 cin>>angle;
 break;
 default:
 break;
 }
 break;
 default:
 break;
 }
 glutDisplayFunc(display);
 glutMainLoop();
return 0;
}
```

OUTPUT

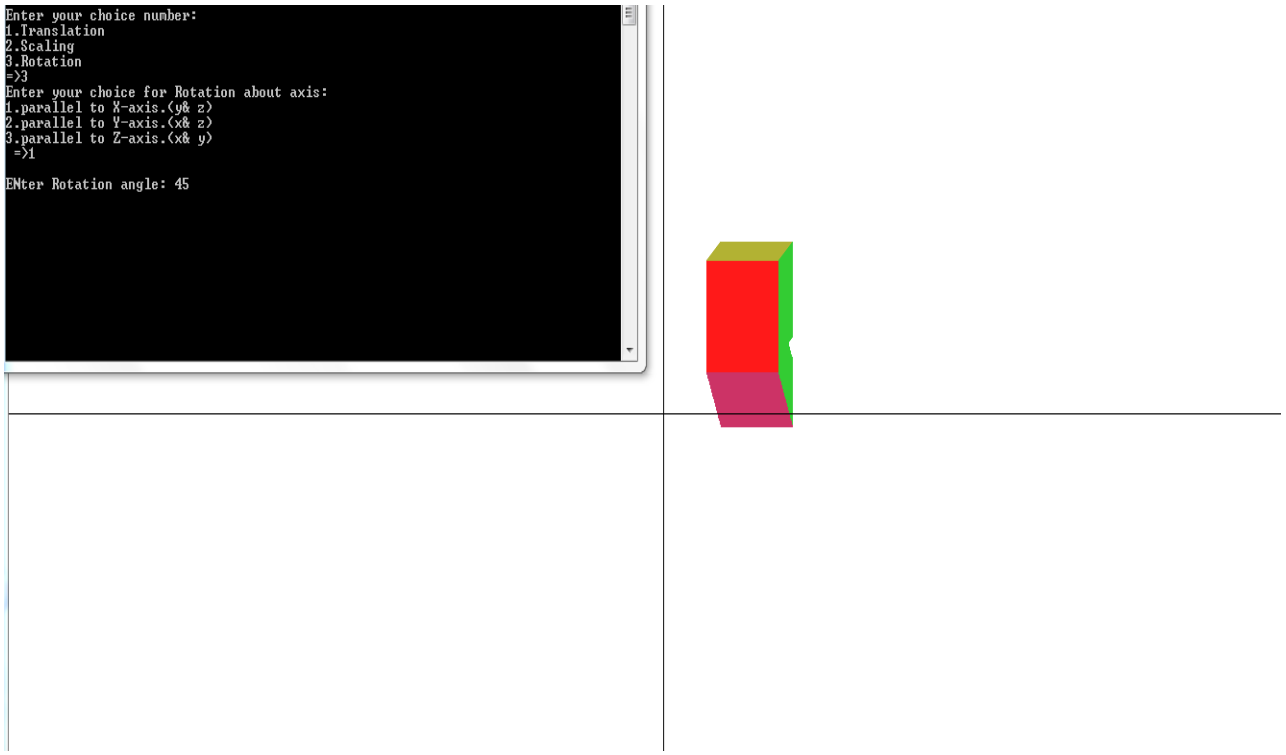
1) Run Command in terminal >>> sudo apt-get install freeglut-dev
2) Compile your Program by using following command>>>
   g++ filename.cpp –lGLU –lGL –lglut
3) Run your Program by using command>>> ./a.out

1. Translation :



2. Scaling :

3. Rotation:


Enter your choice number:
1.Translation
2.Scaling
3.Rotation
=>3
Enter your choice for Rotation about axis:
1.parallel to X-axis.(y& z)
2.parallel to Y-axis.(x& z)
3.parallel to Z-axis.(x& y)
=>1
ENter Rotation angle: 45

7) Write C++ program to draw man walking in the rain with an umbrella. Apply the concept of polymorphism

PROGRAM

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

using namespace std;
class walkingman
{
int rhx,rhy;
public:
void draw(int,int);
void draw(int);
};
void walkingman::draw(int i)
{
line(20,380,580,380);
if(i%2)
{
line(25+i,380,35+i,340);
line(45+i,380,35+i,340);
line(35+i,310,25+i,330);
delay(20);
}
else
{
line(35+i,340,35+i,310);
line(35+i,310,40+i,330);
delay(20);
}
line(35+i,340,35+i,310);
circle(35+i,300,10);
line(35+i,310,50+i,330);
line(50+i,330,50+i,280);
line(15+i,280,85+i,280);
arc(50+i,280,0,180,35);
arc(55+i,330,180,360,5);
}
void walkingman::draw(int x,int y)
{
int j;
rhx=x;
rhy=y;
for
(j=0;j<100;j++)
{
outtextxy(rand()%rhx,rand()%(rhy-50),"|");
setcolor(WHITE);
```

```
}
}
int main()
{
int gd=DETECT,gm;
int rhx,rhy,j,i;
walkingman obj;
initgraph(&gd,&gm,NULL);
for(i=0;i<500;i++)
{
obj.draw(i);
rhx=getmaxx();
rhy=getmaxy();
obj.draw(rhx,rhy);
delay(150);
cleardevice();
}
getch();
}
```

OUTPUT