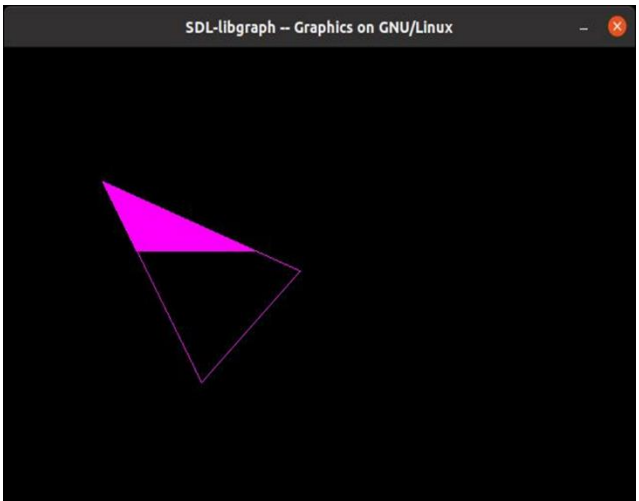
Practical No.:1 Write C++ program to draw a concave polygon and fill it with desired color using scan fill algorithm. Apply the concept of inheritance.

```
#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\t SCAN_FILL ALGORITHM";
    cout<<"\n Enter the no of vertices of polygon:";
    cin>>v;
    if(v>2)
    {
        for(i=0;i<v; i++)
        {
            cout<<"\nEnter the 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()
{ //MAX,MIN
    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)
{
    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()
{
    int cl;
    int gd = DETECT,gm;
    initgraph(&gd,&gm,NULL);
    //initwindow(500,600);
    //cleardevice();
    poly x;
    x.read();
    x.calcs();
    //cleardevice();
    cout<<"\n\tEnter the colour u want:(0-15)->"; //Selecting colour
    cin>>cl;
    setcolor(cl);
    x.display();
    closegraph();
    getch();
    return 0;
}
```



Practical No.: 2 Write C++ program to implement Cohen Southerland line clipping algorithm.

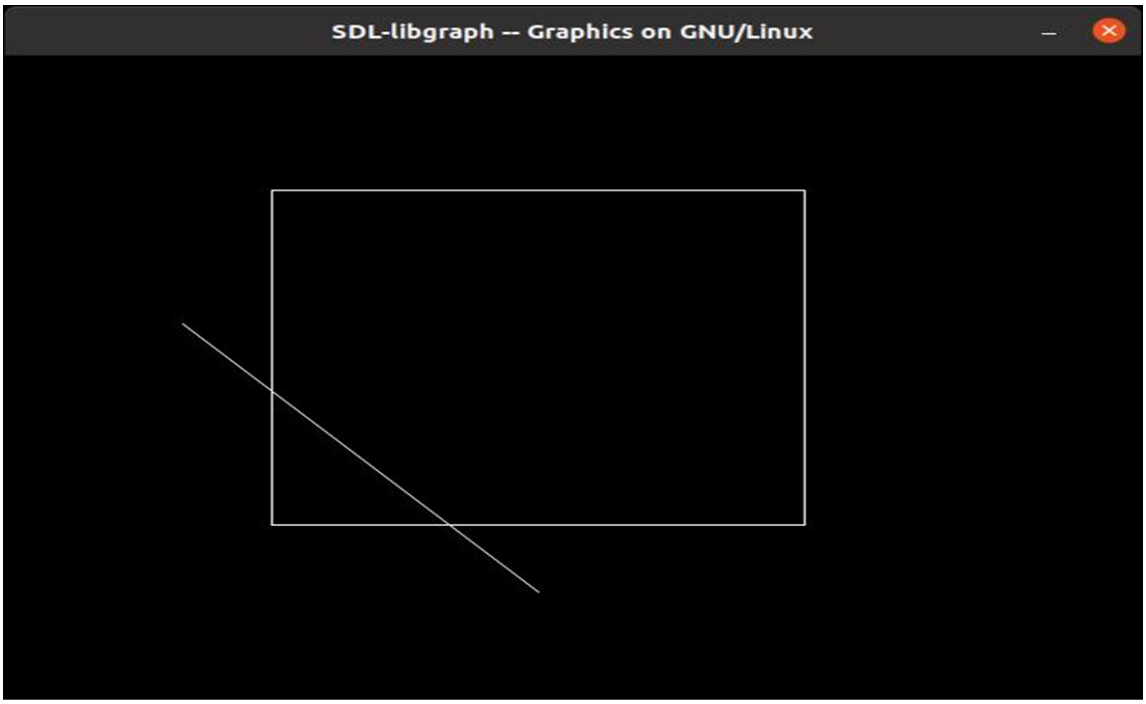
```
#include<iostream>
#include<stdlib.h>
#include<math.h>
#include<graphics.h>
//#include<dos.h>
using namespace std;
class Coordinate
{
    public:
        int x,y;
        char code[4];
};
class Lineclip
{
    public:
        Coordinate PT;
        void drawwindow();
        void drawline(Coordinate 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,NULL);
    lc.drawwindow();
    delay(10000);

    lc.drawline (p1,p2);
    delay(10000);
    cleardevice();
    delay(10000);
```

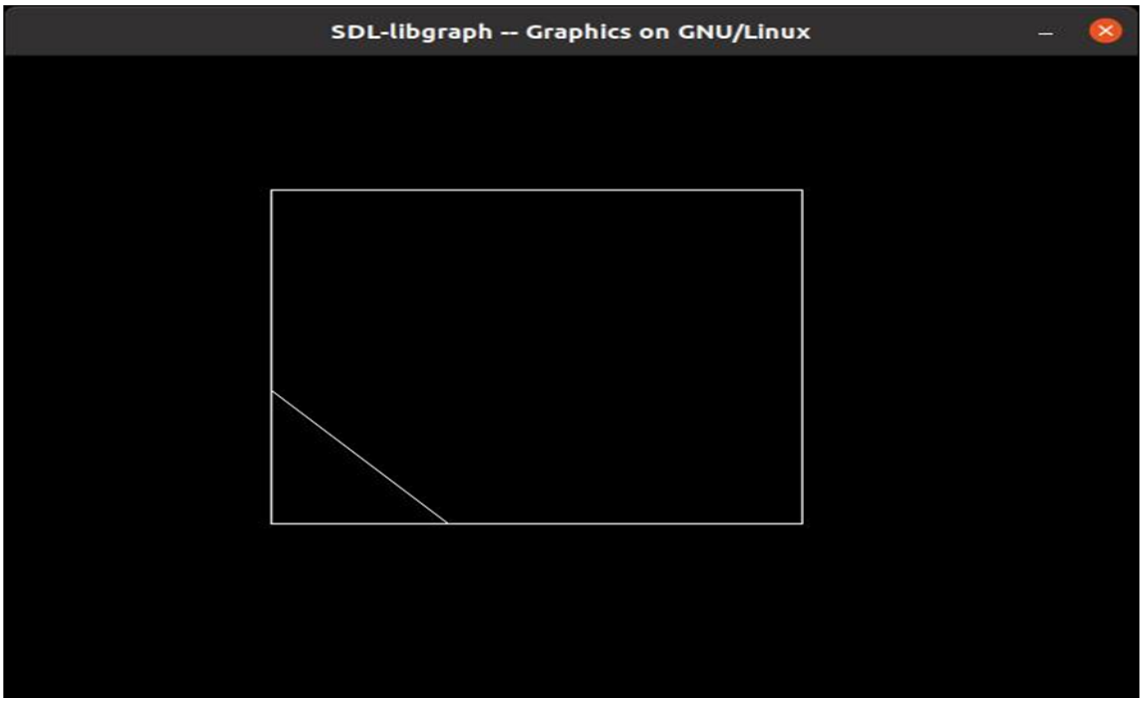
```
    p1=lc.setcode(p1);
    p2=lc.setcode(p2);
    v=lc.visibility(p1,p2);
    delay(10000);

    switch(v)
    {
        case 0: lc.drawwindow();
                delay(10000);
                lc.drawline(p1,p2);
                break;
        case 1:lc.drawwindow();
                delay(10000);
                break;
        case 2:p3=lc.resetendpt(p1,p2);
               p4=lc.resetendpt(p2,p1);
               lc.drawwindow();
               delay(10000);
               lc.drawline(p3,p4);
               break;
     }
     delay(10000);
     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'))
```

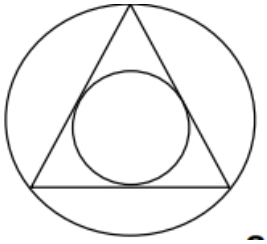
```
         flag='0';
    }

    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);


        }
}
```





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



```
#include<iostream>
#include<graphics.h>
#include<stdlib.h>
#include<math.h>
using namespace std;
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,x0,y0;
    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 draw1(int color)
{
    float x,y,dx,dy,length;
    int i;
    pt::setcolor(color);
    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);
        //delay(500);
        pt::draw();
        x=x+dx;
        y=y+dy;
        i=i+1;
    }
    pt::setco(x,y);
    pt::draw();
}
void draw1(int colour,int xo,int yo)
{
    float x,y,dx,dy,e,temp;
    int i,gd,gm,s1,s2,ex;
```

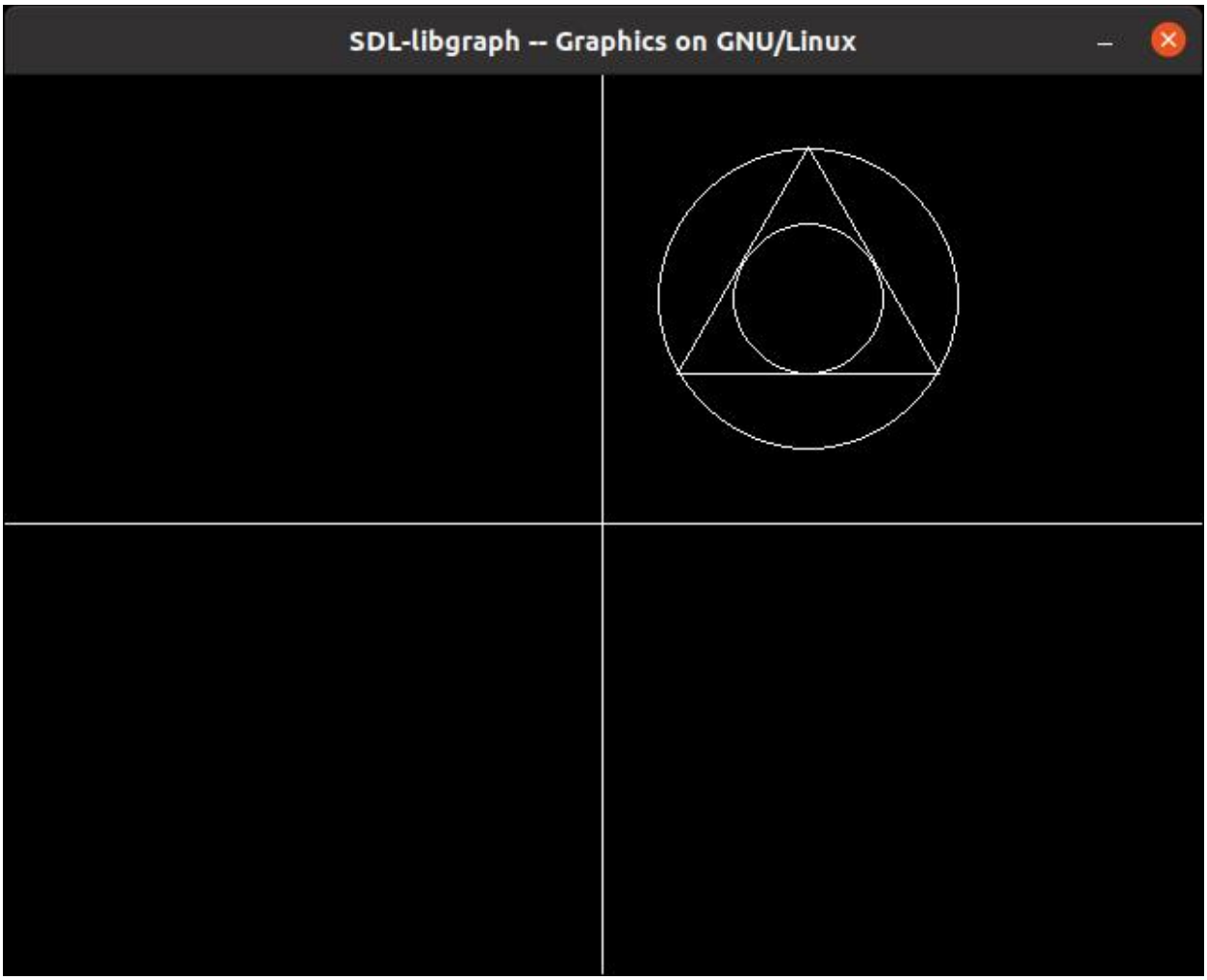
```
pt::setcolor(colour);
dx=abs(x2-xco);
dy=abs(y2-yco);
x=xco;
y=yco;
putpixel(x+xo,yo-y,15);
if(x2>xco)
{
    s1=1;
}
if(x2==xco)
{
    s1=0;
}
if(x2<xco)
{
    s1=-1;
}
if(y2>yco)
{
    s2=1;
}
if(y2==yco)
{
    s2=0;
}
if(y2<yco)
{
    s2=-1;
}
if(dy>dx)
{
    temp=dx;
    dx=dy;
    dy=temp;
    ex=1;
}
else
{
    ex=0;
}
e=2*dy-dx;
i=1;
do
{
    while(e>=0)
```

```
        {
            if(ex==1)
            {
                x=x+s1;
            }
            else
            {
                y=y+s2;
            }
            e=e-2*dx;
        }
        if(ex==1)
        {
            y=y+s2;
        }
        else
        {
            x=x+s1;
        }
        e=e+2*dy;
        //delay(500);
        putpixel(x+xo,yo-y,15);
        i=i+1;
    }while(i<=dx);
}


void setline1(int x,int y,int xx,int yy,int xm,int ym)
{
    pt::setco(x,y);
    x2=xx;
    y2=yy;
    x0=xm;y0=ym;
}

void drawl(int x1, int y1, int r)//bresanhams
{
    int i, x, y;
    float d;
    x=0, y=r;
    d = 3 - 2*r;            //decision variable

    do
    {

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

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


                if(d<=0)
                {
                x = x + 1;
                d = d + (4*x) + 6;
                }
                else
                {
                x = x + 1;
                y = y - 1;
                d = d + (4*x-4*y) + 10;
                }

            }while(x<=y);
        }



};
int main()
{
    int gd=DETECT,gm=VGAMAX;
    int x1,x2,y1,y2,x3,y3,xmax,ymax,xmid,ymid;
    float xc,yc,r,r1,yd;
    pt p1;
    p1.setco(5000,5000);
    p1.setcolor(8);
    dline l;
    cout<<"\nEnter three points for triangle:";
    cout<<"\nEnter the value of x1:";
    cin>>x1;
    cout<<"\nEnter the value of y1:";
    cin>>y1;
    cout<<"\nEnter the value of x2:";
    cin>>x2;
    cout<<"\nEnter the value of y2:";
    cin>>y2;
    /*cout<<"\nEnter the value of x3:";
```

```
    cin>>x3;
    cout<<"\nEnter the value of y3:";
    cin>>y3;*/

    x3=(x2+x1)/2;

    yd=sqrt(((x2-x1)*(x2-x1))+((y2-y1)*(y2-y1)))/2;

    //yd=(x2-x1)/2;

    y3=(yd*sqrt(3))+y1;

    xc=(x1+x2+x3)/3; yc=(y1+y2+y3)/3;
    r=sqrt(((x2-xc)*(x2-xc))+((y2-yc)*(y2-yc)));

    r1=r/2;

    initgraph(&gd,&gm,NULL);
    xmax=getmaxx();
    ymax=getmaxy();
    xmid=xmax/2;
    ymid=ymax/2;
    line(xmid,0,xmid,ymax);
    line(0,ymid,xmax,ymid);

    l.setline(x1,y1,x2,y2);//first line
    l.draw1(15,xmid,ymid);

    l.setline(x2,y2,x3,y3);//second line
    l.draw1(15,xmid,ymid);

    l.setline(x3,y3,x1,y1);//third line
    l.draw1(15,xmid,ymid);


    l.setline1(x1,y1,x2,y2,xmid,ymid);//Big circle
    l.drawl(xc,yc,r);


    l.setline1(x1,y1,x2,y2,xmid,ymid);//small circle
    l.drawl(xc,yc,r1);

    delay(500000);
    closegraph();
    return 0;
}
```

```
/******
output:
akshay@akshay-pc:~$ g++ A7.cpp -lgraph
akshay@akshay-pc:~$ ./a.out

Enter three points for triangle:
Enter the value of x1:40

Enter the value of y1:70

Enter the value of x2:180

Enter the value of y2:70
**/
```



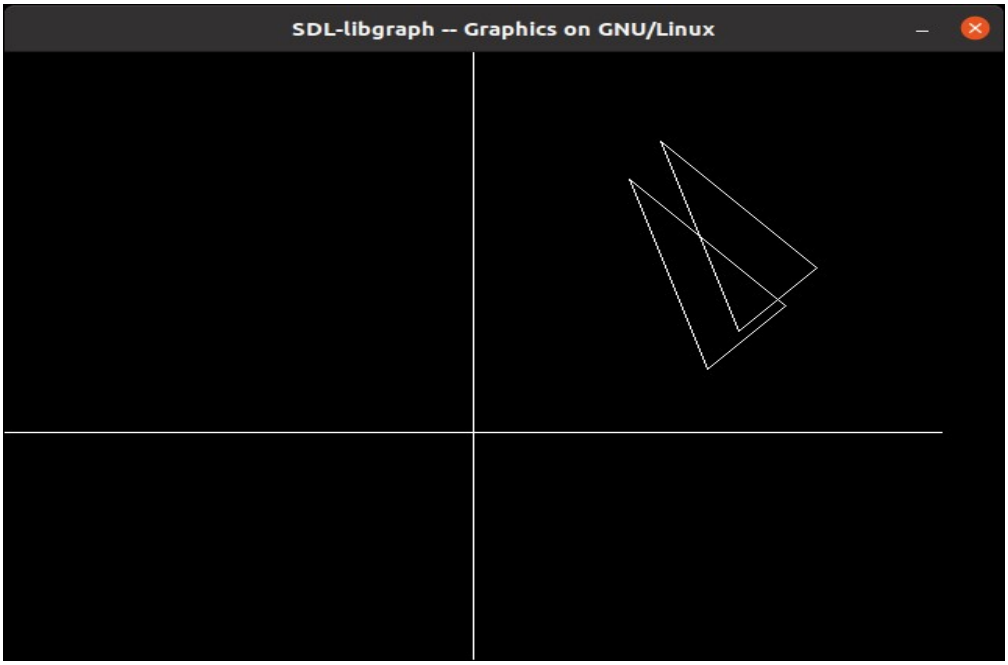
Q.4.Write C++ program to draw 2-D object and perform following basic transformations, Scaling b) Translation c) Rotation. Apply the concept of operator overloading.

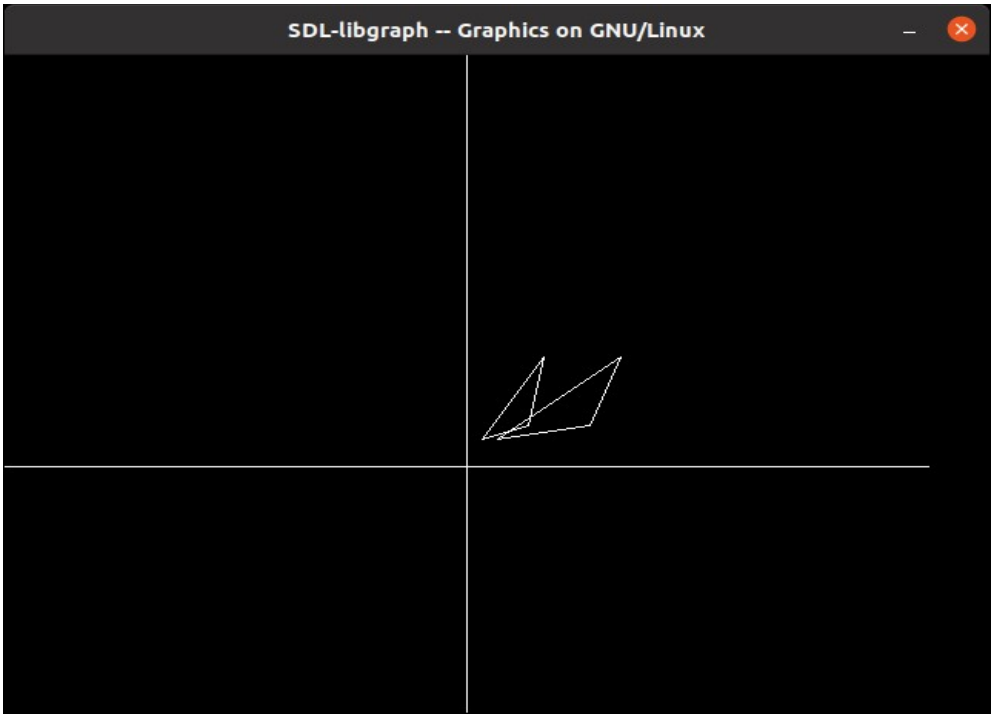
```
#include<iostream>
#include<graphics.h>
#include<math.h>
using namespace std;
class transform
{
    public:
        int m,a[20][20],c[20][20];
        int i,j,k;
        public:

        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++)
    {
```

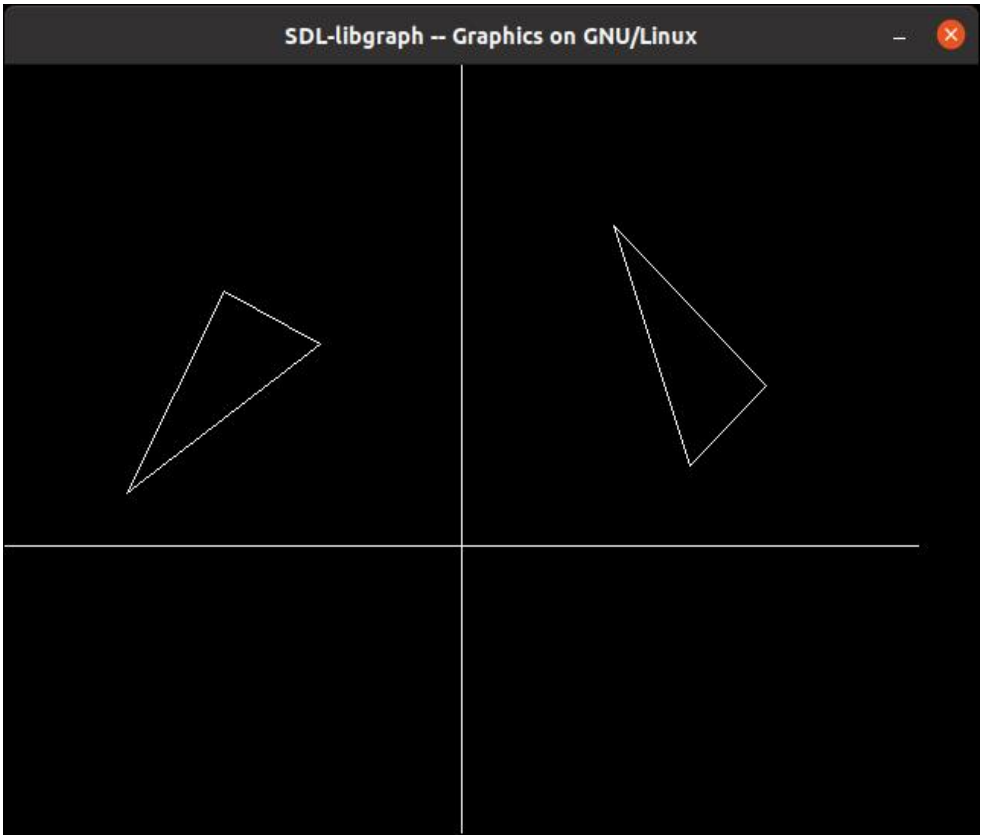
```
        line(300+a[i][0],300-a[i][1],300+a[i+1][0],300-a[i+1][1]);
    }
    line(300+a[0][0],300-a[0][1],300+a[i][0],300-a[i][1]);
    for( i=0;i<m-1;i++)
    {

        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<<"\n\t\bMenu";
        cout<<"\n1.Translation\n2.Scaling\n3.Rotation\nEnter your
choice :";
              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:


SDL-libgraph -- Graphics on GNU/Linux

2. 
SDL-libgraph -- Graphics on GNU/Linux

3. 

Q.5. Write C++ program to generate Hilbert curve using concept of fractals

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

using namespace std;

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);
}
0
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=50,y0=150,x,y,h=10,r=2,d=3,l=4,u=1;

cout<<"\nGive the value of n: ";
cin>>n;
```

```
x=x0;y=y0;
int gm,gd=DETECT;
initgraph(&gd,&gm,NULL);
moveto(x,y);
hilbert(r,d,l,u,n,h,x,y);
delay(10000);

closegraph();

return 0;
}
```

Output::

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

```
#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++)
```

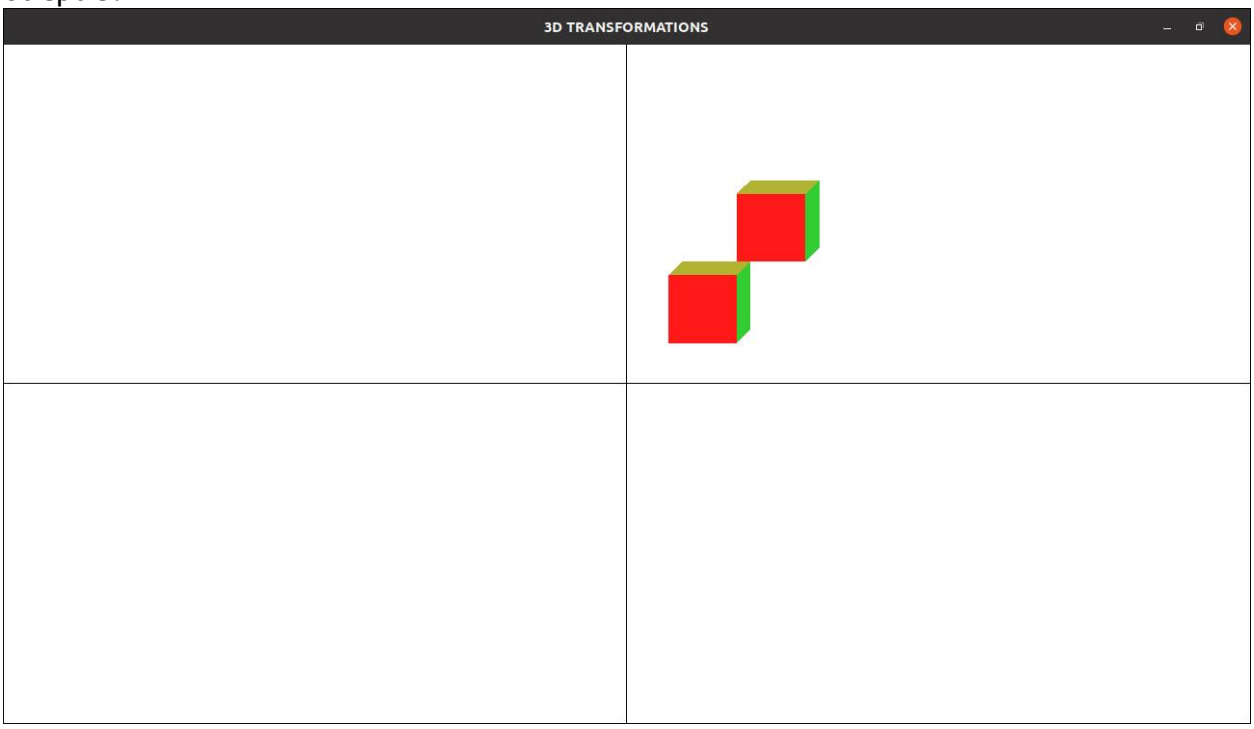
```
        {
            output[i][j]=output[i][j]+input[i][k]*theMatrix[k][j];
        }
    }
}

}
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]);
```

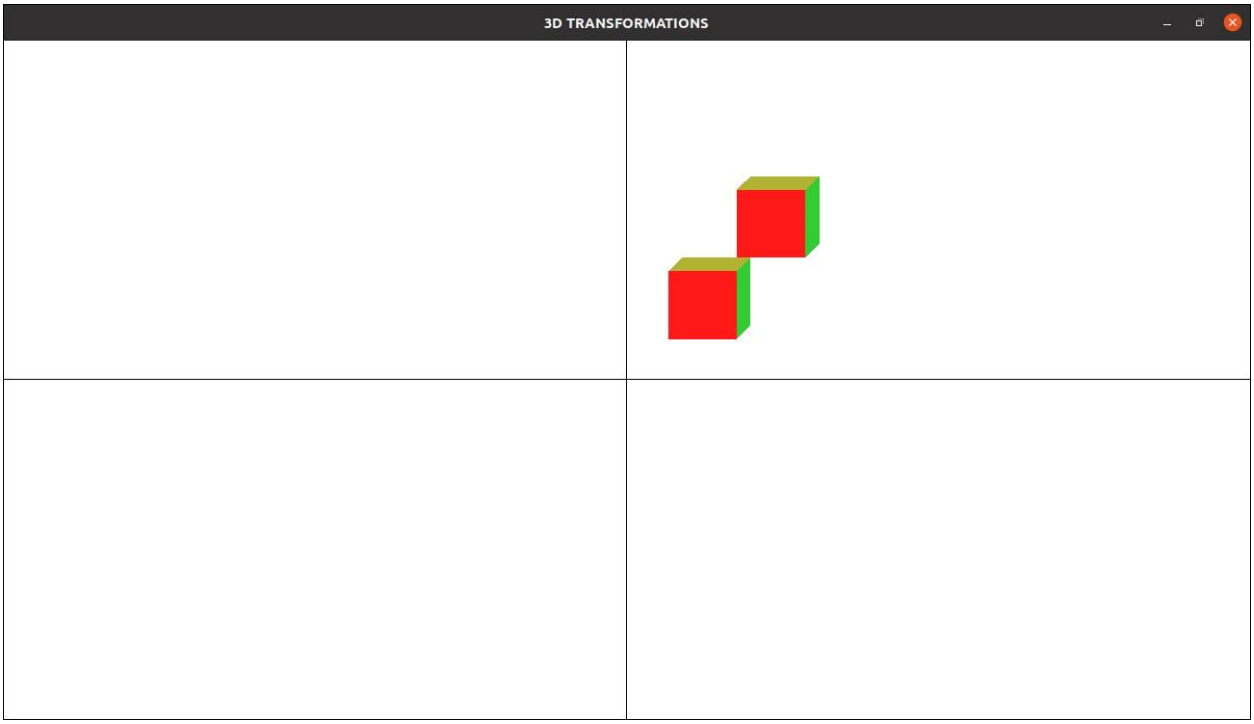
```
 glVertex3fv(a[6]);
 glVertex3fv(a[5]);

 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 =>";
```

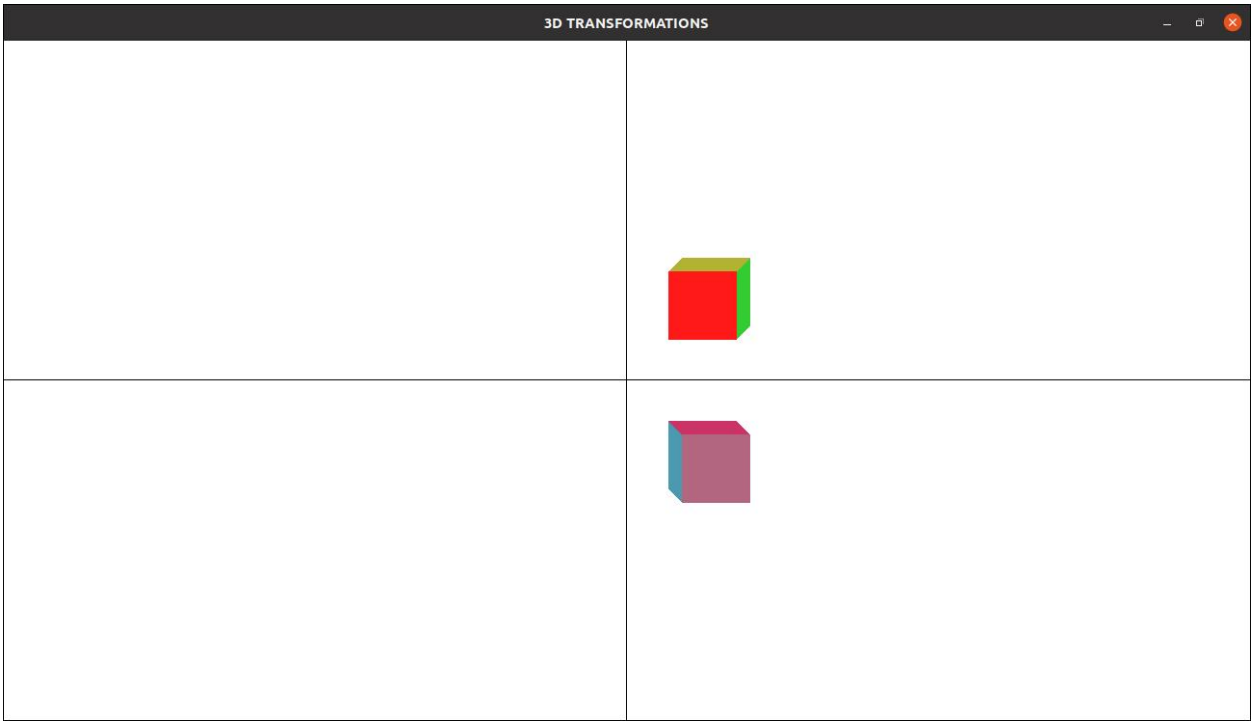
```
        cin>>choiceRot;
        switch (choiceRot) {
        case 1:
            cout<<"\nENter Rotation angle: ";
            cin>>angle;
            break;
        case 2:
            cout<<"\nENter Rotation angle: ";
            cin>>angle;
            break;
        case 3:
            cout<<"\nENter Rotation angle: ";
            cin>>angle;
            break;
        default:
            break;
        }
        break;
    default:
        break;
    }
    glutDisplayFunc(display);
    glutMainLoop();
return 0;
}
```

Output:



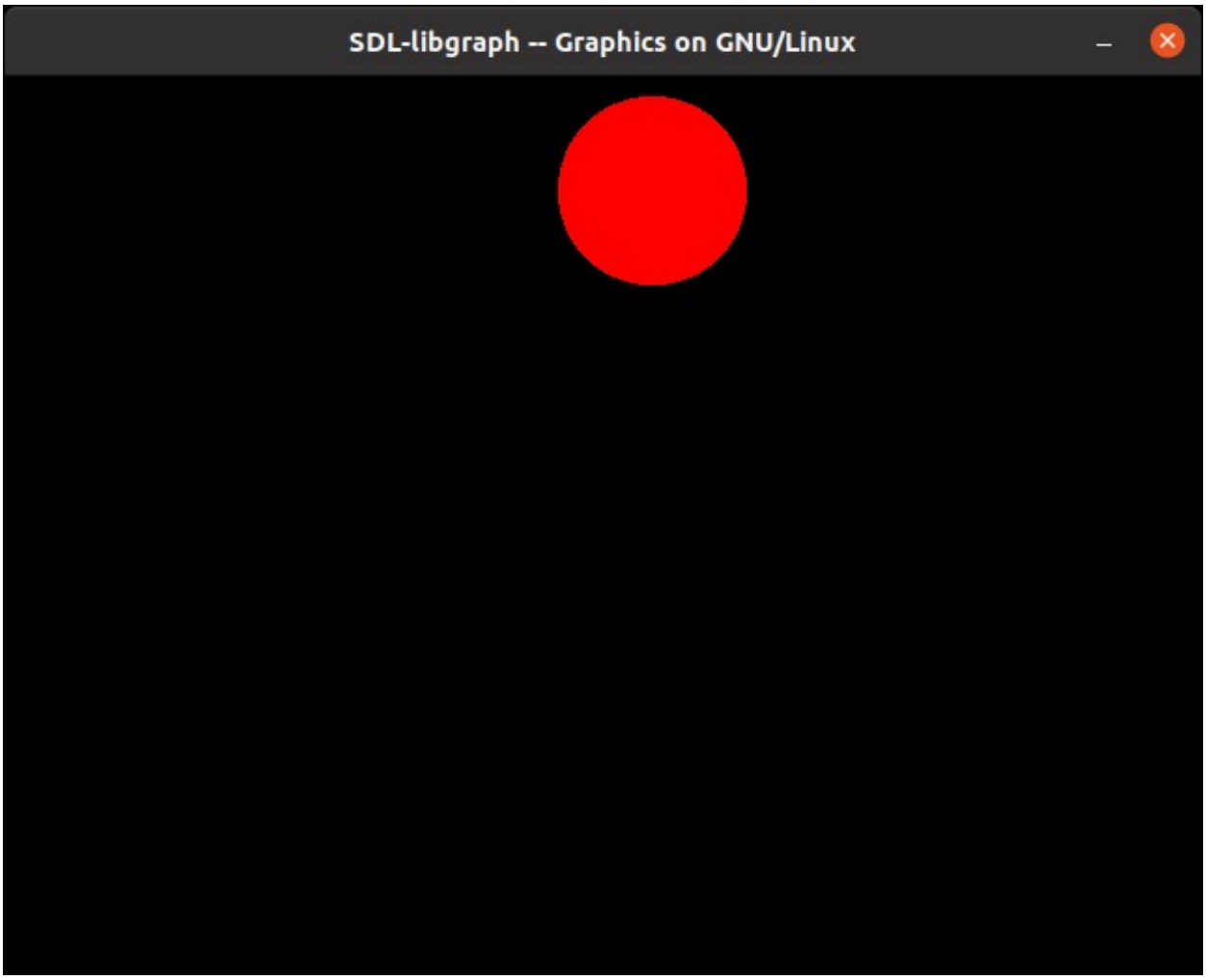


3.


3D TRANSFORMATIONS

7.Write a C++ program to implement bouncing ball using sine wave form. Apply the concept of polymorphism

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

int main()
{

    int gd=DETECT,gm;
    int x,y,i;

    initgraph(&gd,&gm,NULL);

    x=getmaxx()/2;
    y=getmaxx()/2;

    for(i=0;i<1000;i++)
    {
        cleardevice();

        x=x+1;
        y=200+50*tan(i*3.142/180);

        setcolor(RED);
        circle(x,y,50);
        floodfill(x,y,RED);

        delay(10);
    }
    getch();
    closegraph();
    return 0;
}
```


SDL-libgraph -- Graphics on GNU/Linux