```
//Polygon Filling Using Scan Fill Algorithm (Assignment-1)
#include<graphics.h>
#include<iostream>

using namespace std;

void FloodFill(int x, int y, int oldc, int newc)
{
	int current;
	current=getpixel(x,y);
	if(current==oldc)
	{
		putpixel(x,y,newc);
		delay(5);
		FloodFill(x+1,y,oldc,newc);
		FloodFill(x-1,y,oldc,newc);
		FloodFill(x,y+1,oldc,newc);
		FloodFill(x,y-1,oldc,newc);
	}
}

int main()
{
	int x,y,oldc=0,x1,y1,x2,y2;

	int gDriver=DETECT,gmode;
	initgraph(&gDriver,&gmode, NULL);

	 cout<<"Enter the coordinates of rectangle:";
	 cin>>x1>>y1>>x2>>y2;
			setcolor(1);
	line(x1,y1,x2,y1);
	 setcolor(2);
	 line(x2,y1,x2,y2);
	 setcolor(3);
	 line(x2,y2,x1,y2);
	 setcolor(5);
	 line(x1,y2,x1,y1);

	x=(x1+x2)/2;
	y=(y1+y2)/2;

	FloodFill(x,y,oldc,4);

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

```
// Polygon clipping Cohen Sutherland line clipping algorithm (Assignment-2)
#include<iostream>
#include<graphics.h>
typedef unsigned int outcode;
enum{TOP=0x1,BOTTOM=0x2,RIGHT=0x4,LEFT=0x8};
using namespace std;
outcode CompOutCode(double ,double ,double ,double ,double ,double );
void CSLCAD(double x0,double y0,double x1,double y1,double xmin,double xmax,double
ymin,double ymax)
{
 outcode outcode0,outcode1,outcodeout;
 boolean accept=FALSE, done=FALSE;
 outcode0=CompOutCode(x0,y0,xmin,xmax,ymin,ymax);
 outcode1=CompOutCode(x1,y1,xmin,xmax,ymin,ymax);
 cout<<"outcode0="<<outcode0<<endl;
 cout<<"outcode1="<<outcode1<<endl;
 do
 {
 if(outcode0==0 && outcode1==0)
 {
 accept=TRUE;
 done=TRUE;
 }
 else if(outcode0 & outcode1)
 {
 done=TRUE;
 }
 else
 {
 double x,y;
 int ocd=outcode0 ? outcode0:outcode1;
 if(ocd & TOP)
 {
 x=x0+(x1-x0)*(ymax-y0)/(y1-y0);
 y=ymax;
 }
 else if(ocd & BOTTOM)
 {
 x=x0+(x1-x0)*(ymin-y0)/(y1-y0);
 y=ymin;
 }
 else if(ocd & LEFT)
 {
 y=y0+(y1-y0)*(xmin-x0)/(x1-x0);
 x=xmin;
 }
 else
 {
 y=y0+(y1-y0)*(xmax-x0)/(x1-x0);
 x=xmax;
 }
 if(ocd==outcode0)
 {
 x0=x;
y0=y;
outcode0=CompOutCode(x0,y0,xmin,xmax,ymin,ymax);
```

```
 }
 else
 {
 x1=x;
y1=y;
outcode1=CompOutCode(x1,y1,xmin,xmax,ymin,ymax);
 }
 }
 }while(done==FALSE);
 if(accept==TRUE)
 {
 line(x0,y0,x1,y1);
 }
}
outcode CompOutCode(double x,double y,double xmin,double xmax,double ymin,double ymax)
{
 outcode code=0;
 if(y>ymax)
 code|=TOP;
 if(y<ymin)
 code|=BOTTOM;
 if(x>xmax)
 code|=RIGHT;
 if(x<xmin)
 code|=LEFT;
 return code;
}
int main()
{
 string ch;
 double xmin,xmax,ymin,ymax,x0,y0,x1,y1;
 initwindow(500,600);
 cout<<"Enter the bottom co-ordinates of window:";
cin>>xmin;
cout<<"Enter the left coordinates of the window:";
cin>>ymin;
cout<<"Enter the right coordinates of the window:";
cin>>xmax;
cout<<"Enter the top coordinates of the window:";
cin>>ymax;
rectangle(xmin,ymin,xmax,ymax);
cout<<"Enter the coordinates(Terminal Points) of the line: ";
cin>>x0>>y0;
cin>>x1>>y1;
line(x0,y0,x1,y1);
 delay(5000);
 cleardevice();
 CSLCAD(x0,y0,x1,y1,xmin,xmax,ymin,ymax);
 rectangle(xmin,ymin,xmax,ymax);
 delay(50000);
 closegraph();
}
```

```
//Pattern Using Lines and Circles (Assignment-3)
#include<iostream>
#include<graphics.h>
#include<math.h>

using namespace std;

void drawcircle(int xc, int yc, int r)
{
    int d=3-2*r;

    int x=0;
    int y=r;

    while (y >= x)
    {
        putpixel(xc+x,yc+y,15);
        putpixel(xc+y,yc+x,15);
        putpixel(xc+y,yc-x,15);
        putpixel(xc+x,yc-y,15);
        putpixel(xc-x,yc-y,15);
        putpixel(xc-y,yc-x,15);
        putpixel(xc-y,yc+x,15);
        putpixel(xc-x,yc+y,15);
        x++;

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

        delay(10);
    }
}

void drawline(float x1,float y1,float x2,float y2)
{
    float dx,dy,steps,x,y,xinc,yinc;

    dx=abs(x2-x1);
    dy=abs(y2-y1);

    if(dx>dy)
       steps=dx;
    else
       steps=dy;

    xinc=(x2-x1)/steps;
    yinc=(y2-y1)/steps;

    x=x1;
```

```
    y=y1;

    putpixel(round(x),round(y),15);

    for(int k=0;k<steps;k++)
    {
       x=x+xinc;
       y=y+yinc;
         putpixel(round(x),round(y),15);
    }
}


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

    int x,y,r;
    float x1,y1,x2,y2,x3;

    cout<<"ENTER COORDINATES  : ";
    cout<<"X1 : ";
    cin>>x1;
    cout<<"Y1 : ";
    cin>>y1;
    cout<<"X2 : ";
    cin>>x2;
       x3=(x2+x1)/2; // x-coordinate of Third point of tringle
   // x3=(x2-x1)/2+x1;

    y2=y1-sqrt(pow((x2-x1),2)-pow((x2-x1)/2,2));    // y-coordinate of Third point of
tringle

    drawline(x1,y1,x2,y1);
    drawline(x2,y1,x3,y2);
    drawline(x1,y1,x3,y2);

    x=x3;                      // x-coordinate of center of a circle
    y=y2+2*(y1-y2)/3;          // y-coordinate of center of a circle

    r=(y1-y2)/3;              //Radius of inner circle
    drawcircle(x,y,r);

    r=2*(y1-y2)/3;            //Radius of outer circle
    drawcircle(x,y,r);

    delay(50000);
    closegraph();
    return 0;
}
```

```
// Basic 2-D transformation (Assignment-4)
#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\n";
 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;
}
```

```
// Curves & Fractals (Assignment-5)
// Hilbert Curve
#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);
}
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(100000);
closegraph();
return 0;
}
```

```
//Koch Curves (Assignment-5)
//Curves and Fractals
#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;
}
```

```
// 3-D Cube (Assignment-6)
#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;
}
```

```
// Bouncing Ball (Asssignment-7)
#include<iostream>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
int main()
{

	int gd=0,gm,x=20,flag=0,y=200,uplimit=250;
	initgraph(&gd,&gm,"C:\\Tc\\BGI");
	while(!kbhit())
	{
		setcolor(4);
		line(0,400,679,400);
		if(flag==0)
		{
			y+=2;
			x+=1;
			if(y>=385)
			flag=1;
		}
		if(flag==1)
		{
			y-=2;
			x+=1;
			if(y<=uplimit)
			{
				flag=0;
				uplimit+=20;

			}
		}

		setcolor(15);
		fillellipse(x,y,15,15);
		delay(15);
		setcolor(0);
		setfillstyle(1,10);
		fillellipse(x,y,15,15);
		cleardevice();
	}

	getch();
}
```

```
// Man Walked In Rain (Assignment-7)
#include<iostream>
#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
#include<dos.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,"");
```

```
for(i=0;i<500;i++)
{
obj.draw(i);
rhx=getmaxx();
rhy=getmaxy();
obj.draw(rhx,rhy);
delay(150);
cleardevice();
}
getch();
}
```