**1. Write a Program to implement DDA Line drawing algorithm. (dda)**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void main(){

int x,y,x1,y1,x2,y2,dx,dy,step,xinc,yinc,i,gd=DETECT,gm;

clrscr();

initgraph(&gd,&gm,"c:\\turboc3\\bgi");

printf("ENTER START CO-ORDINATES:\n");

scanf("%d%d",&x1,&y1);

printf("ENTER END CO-ORDINATES:\n");

scanf("%d%d",&x2,&y2);

dx=x2-x1;

dy=y2-y1;

if(abs(dx)>=abs(dy))

step=abs(dx);

else

step=abs(dy);

xinc=dx/step;

yinc=dy/step;

x=x1;

y=y1;

putpixel(x,y,4);

for(i=0;i<step;i++){

x=x+xinc;

y=y+yinc;

putpixel(x,y,4);

}

getch();

closegraph();

}

**2. Write a Program to implement Bresenham’s Line drawing algorithm. (bline)**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<stdlib.h>

void bline(int x1,int y1,int x2,int y2){

int dx=abs(x2-x1);

int dy=abs(y2-y1);

int sx=(x1 < x2)?1:-1;

int sy=(y1 < y2)?1:-1;

int err=dx-dy;

int e2;

while(x1!=x2 || y1!=y2){

putpixel(x1,y1,WHITE);

e2=2\*err;

if(e2 > -dy){

err-=dy;

x1+=sx;

}

if(e2<dx){

err+=dx;

y1+=sy;

}

}

}

void main(){

int x1,y1,x2,y2;

int gd=DETECT,gm;

initgraph(&gd,&gm,"c://turboc3//bgi");

while(1){

printf("enter x1,y1,x2,y2:");

if(scanf("%d,%d,%d,%d",&x1,&y1,&x2,&y2)!=4){

printf("invalid input");

break;

}

bline(x1,y1,x2,y2);

}

getch();

closegraph();

return 0;

}

**3. Write a Program to implement Bresenham’s Circle drawing algorithm. (bcircle)**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void cir(int x,int y,int r)

{

int i=0,j=r,p=3-2\*r;

while(i<=j)

{

putpixel(x+i,y+j,WHITE);

putpixel(x-i,y-j,WHITE);

putpixel(x+j,y+i,WHITE);

putpixel(x-j,y-i,WHITE);

putpixel(x-i,y+j,WHITE);

putpixel(x+i,y-j,WHITE);

putpixel(x-j,y+i,WHITE);

putpixel(x+j,y-i,WHITE);

if(p<0)

{

i++;

p=p+4\*i+6;

}

else

{

i++;

j--;

p=p+4\*(i-j)+10;

}

}

}

int main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C://TURBOC3//BGI");

cir(250, 110, 50);

cir(300, 120, 50);

cir(350, 130, 50);

cir(275, 150, 50);

cir(325, 170, 50);

cir(300, 180, 50);

getch();

closegraph();

return 0;

}

**4. Write a Program to implement Mid-point Circle drawing algorithm. (mcircle)**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void midpointCircle(int x, int y, int radius);

void midpointCircle(int x, int y, int radius) {

int p = 1 - radius;

int i = 0;

int j = radius;

while (i <= j) {

putpixel(x + i, y + j, WHITE);

putpixel(x + j, y + i, WHITE);

putpixel(x - j, y + i, WHITE);

putpixel(x + j, y - i, WHITE);

putpixel(x - i, y - j, WHITE);

putpixel(x - j, y - i, WHITE);

putpixel(x + i, y - j, WHITE);

putpixel(x - i, y + j, WHITE);

if (p < 0) {

i++;

p += 2 \* i + 1;

}

else {

i++;

j--;

p += 2 \* (i - j) + 1;

}

}

}

int main() {

int gd=DETECT,gm;

initgraph(&gd,&gm,(char\*)"C:\\Turboc3\\BGI");

midpointCircle(200,300,50);

delay(5000);

closegraph();

return 0;

}

**5. Write a Program to implement Scan line fill algorithm**

#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <math.h>

int vertices[][2] = {{100, 100}, {150, 50}, {200, 100}, {175, 125}, {125, 125}};

int n = sizeof(vertices) / sizeof(vertices[0]);

// DDA Line Algorithm

void ddaLine(int x1, int y1, int x2, int y2) {

int k;

int dx = x2 - x1;

int dy = y2 - y1;

int steps;

float xIncrement, yIncrement;

float x = x1, y = y1;

if (abs(dx) > abs(dy)) {

steps = abs(dx);

} else {

steps = abs(dy);

}

xIncrement = dx / (float) steps;

yIncrement = dy / (float) steps;

putpixel(x, y, WHITE);

for (k = 0; k < steps; k++) {

x += xIncrement;

y += yIncrement;

putpixel(x,y, WHITE);

}

}

//scanline function

void scanLineFill(int sides, int \*points, int fill\_color) {

int i, j, x, y, count, temp, xi, x1, y1, x2, y2;

for (y = 0; y < getmaxy(); y++) {

count = 0;

for (i = 0, j = sides \* 2 - 2; i < sides \* 2; j = i, i += 2) {

y1 = points[i + 1], y2 = points[j + 1];

x1 = points[i], x2 = points[j];

if ((y1 <= y && y2 > y) || (y2 <= y && y1 > y)) {

xi = (int)((float)(y - y1) / (y2 - y1) \* (x2 - x1) + x1);

if (count == 0) {

count++;

x = xi;

} else {

count = 0;

if (x > xi) {

temp = x;

x = xi;

xi = temp;

}

for (; x <= xi; x++) {

putpixel(x, y, fill\_color);

}

}

}

}

}

}

int main() {

int i,gd = DETECT, gm;

initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

// Draw the polygon using DDA Line Algorithm

for ( i = 0; i < n; i++) {

int x1 = vertices[i][0];

int y1 = vertices[i][1];

int x2 = vertices[(i + 1) % n][0];

int y2 = vertices[(i + 1) % n][1];

ddaLine(x1, y1, x2, y2);

}

// Perform Fence Fill inside the polygon

scanLineFill(n, vertices, YELLOW); // A point inside the polygon

getch();

closegraph();

return 0;

}

**6. Write a Program to implement Boundary fill.**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void bfill(int x,int y,int newcolor,int boundarycolor){

int current=getpixel(x,y);

if(current!=newcolor && current!=boundarycolor){

putpixel(x,y,newcolor);

bfill(x+1,y,newcolor,boundarycolor);

bfill(x,y+1,newcolor,boundarycolor);

bfill(x-1,y,newcolor,boundarycolor);

bfill(x,y-1,newcolor,boundarycolor);

}

}

void main(){

int x,y,current,newcolor,boundarycolor;

int gd=DETECT,gm;

initgraph(&gd,&gm,"c://turboc3//bgi");

printf("BOUNDARY FILL:");

rectangle(50,50,100,100);

delay(300);

bfill(60,60,YELLOW,WHITE);

delay(3000);

getch();

closegraph();

return 0;

}

**7. Write a Program to implement Cohen Sutherland line clipping algorithm. (l25)**

#include<graphics.h>

#include<conio.h>

#include<stdio.h>

#include<math.h>

void main()

{

int rcode\_begin[4]={0,0,0,0},rcode\_end[4]={0,0,0,0},region\_code[4];

int W\_xmax,W\_ymax,W\_xmin,W\_ymin,flag=0;

float slope;

int x,y,x1,y1,i, xc,yc;

int gr=DETECT,gm;

initgraph(&gr,&gm,"C:\\TURBOC3\\BGI");

printf("\n\*\*\* Cohen Sutherlsnd Line Clipping algorithm \*\*\*\*");

printf("\n Now, enter XMin, YMin =");

scanf("%d %d",&W\_xmin,&W\_ymin);

printf("\n First enter XMax, YMax =");

scanf("%d %d",&W\_xmax,&W\_ymax);

printf("\n Please enter intial point x and y= ");

scanf("%d %d",&x,&y);

printf("\n Now, enter final point x1 and y1= ");

scanf("%d %d",&x1,&y1);

cleardevice();

rectangle(W\_xmin,W\_ymin,W\_xmax,W\_ymax);

line(x,y,x1,y1);

line(0,0,600,0);

line(0,0,0,600);

if(y>W\_ymax) {

rcode\_begin[0]=1; // Top

flag=1 ;

}

if(y<W\_ymin) {

rcode\_begin[1]=1; // Bottom

flag=1;

}

if(x>W\_xmax) {

rcode\_begin[2]=1; // Right

flag=1;

}

if(x<W\_xmin) {

rcode\_begin[3]=1; //Left

flag=1;

}

//end point of Line

if(y1>W\_ymax){

rcode\_end[0]=1; // Top

flag=1;

}

if(y1<W\_ymin) {

rcode\_end[1]=1; // Bottom

flag=1;

}

if(x1>W\_xmax){

rcode\_end[2]=1; // Right

flag=1;

}

if(x1<W\_xmin){

rcode\_end[3]=1; //Left

flag=1;

}

if(flag==0)

{

printf("No need of clipping as it is already in window");

}

flag=1;

for(i=0;i<4;i++){

region\_code[i]= rcode\_begin[i] && rcode\_end[i] ;

if(region\_code[i]==1)

flag=0;

}

if(flag==0)

{

printf("\n Line is completely outside the window");

}

else{

slope=(float)(y1-y)/(x1-x);

if(rcode\_begin[2]==0 && rcode\_begin[3]==1) //left

{

y=y+(float) (W\_xmin-x)\*slope ;

x=W\_xmin;

}

if(rcode\_begin[2]==1 && rcode\_begin[3]==0) // right

{

y=y+(float) (W\_xmax-x)\*slope ;

x=W\_xmax;

}

if(rcode\_begin[0]==1 && rcode\_begin[1]==0) // top

{

x=x+(float) (W\_ymax-y)/slope ;

y=W\_ymax;

}

if(rcode\_begin[0]==0 && rcode\_begin[1]==1) // bottom

{

x=x+(float) (W\_ymin-y)/slope ;

y=W\_ymin;

}

// end points

if(rcode\_end[2]==0 && rcode\_end[3]==1) //left

{

y1=y+(float) (W\_xmin-x1)\*slope ;

x1=W\_xmin;

}

if(rcode\_end[2]==1 && rcode\_end[3]==0) // right

{

y1=y1+(float) (W\_xmax-x1)\*slope ;

x1=W\_xmax;

}

if(rcode\_end[0]==1 && rcode\_end[1]==0) // top

{

x1=x1+(float) (W\_ymax-y1)/slope ;

y1=W\_ymax;

}

if(rcode\_end[0]==0 && rcode\_end[1]==1) // bottom

{

x1=x1+(float) (W\_ymin-y1)/slope ;

y1=W\_ymin;

}

}

delay(1000);

clearviewport();

rectangle(W\_xmin,W\_ymin,W\_xmax,W\_ymax);

line(0,0,600,0);

line(0,0,0,600);

setcolor(RED);

line(x,y,x1,y1);

getch();

closegraph();

}

**8. Write a Program to implement midpoint line clipping algorithm.**

#include<stdio.h>

#include<math.h>

#include<graphics.h>

#include<dos.h>

typedef struct coordinate

{

int x,y;

char code[4];

}PT;

void drawwindow();

void drawline(PT p1,PT p2);

PT setcode(PT p);

int visibility(PT p1, PT p2);

PT resetendpt(PT p1, PT p2);

void main()

{

int gd=DETECT,v,gm;

PT p1,p2,p3,p4,temp;

printf("\nEnter x1 and y1\n");

scanf("%d %d", &p1.x, &p1.y);

printf("\nEnter x2 and y2\n");

scanf("%d %d", &p2.x, &p2.y);

initgraph(&gd,&gm,"C://TURBOC3//BGI");

drawwindow();

delay(500);

drawline(p1,p2);

delay(500);

cleardevice();

delay(500);

p1=setcode(p1);

p2=setcode(p2);

v=visibility(p1,p2);

delay(500);

switch(v)

{

case 0: drawwindow();

delay(500);

drawline(p1,p2);

break;

case 1: drawwindow();

delay(500);

break;

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

p4=resetendpt(p2,p1);

drawwindow();

delay(500);

drawline(p3,p4);

break;

}

delay(5000);

closegraph();

}

void drawwindow()

{

line(150,100,450,100);

line(450,100,450,350);

line(450,350,150,350);

line(150,350,150,100);

}

void drawline(PT p1,PT p2)

{

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

}

PT setcode(PT p)

{

PT 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 visibility(PT p1,PT p2)

{

int i,flag=0;

for(i=0;i<4;i++)

{

if((p1.code[i]!='0') || (p2.code[i]!='0'))

flag=1;

}

if(flag==0)

return(0);

for(i=0;i<4;i++)

{

if((p1.code[i]==p2.code[i]) && (p1.code[i]=='1'))

flag='0';

}

if(flag==0)

return(1);

return(2);

}

PT resetendpt(PT p1, PT p2)

{

PT temp;

int x,y,i;

float m,k;

if(p1.code[3]=='1')

x=150;

if(p1.code[2]=='1')

x=150;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[0]=='1') || (p1.code[1]=='1'))

{

m=(float)(p2.y-p1.y)/(p2.y-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);

}

//200 200 , 500 500

**9. Write a Program to implement Sutherland-Hodgeman Polygon clipping algorithm. (l27)**

#include<stdio.h>

#include<graphics.h>

#include<conio.h>

#include<stdlib.h>

int main()

{

int gd,gm,n,\*x,i,k=0;

//window coordinates int wx1=220,wy1=140,wx2=420,wy2=140,wx3=420,wy3=340,wx4=220,wy4=340;

int w[]= {220,140,420,140,420,340,220,340,220,140};//array for drawing window

detectgraph(&gd,&gm);

initgraph(&gd,&gm,"c:\\turboc3\\bgi"); //initializing graphics

printf("Window:-");

setcolor(RED); //red colored window

drawpoly(5,w); //window drawn

printf("Enter the no. of vertices of polygon: ");

scanf("%d",&n);

x = malloc(n\*2+1);

printf("Enter the coordinates of points:\n");

k=0;

for(i=0;i<n\*2;i+=2) //reading vertices of polygon

{

printf("(x%d,y%d): ",k,k);

scanf("%d,%d",&x[i],&x[i+1]);

k++;

}

x[n\*2]=x[0]; //assigning the coordinates of first vertex to last additional vertex for drawpoly method.

x[n\*2+1]=x[1];

setcolor(WHITE);

drawpoly(n+1,x);

printf("\nPress a button to clip a polygon..");

getch();

setcolor(RED);

drawpoly(5,w);

setfillstyle(SOLID\_FILL,BLACK);

floodfill(2,2,RED);

gotoxy(1,1); //bringing cursor at starting position

printf("\nThis is the clipped polygon..");

getch();

cleardevice();

closegraph();

return 0;

}

**10)Write a program to implement of different 2D transformations:**

**a) Translate a polygon by 2 units.**

**b)Rotate a polygon about 90 degree.**

#include <stdio.h>

#include <graphics.h>

#include <math.h>

int xt = 0, yt = 0;

int points[] = {300,100, 400,300, 200,300}; // Declare points array for triangle

// Drawing polygon (triangle in this case)

void drawPolygon(int sides, int \*points) {

int i;

for (i = 0; i < sides \* 2; i += 2) {

line(points[i], points[i + 1], points[(i + 2) % (sides \* 2)], points[(i + 3) % (sides \* 2)]);

}

}

// Rotate point function

void rotatePoint(int \*x, int \*y, int cx, int cy, float theta) {

float radian = theta \* (M\_PI / 180);

int new\_x = cx + (\*x - cx) \* cos(radian) - (\*y - cy) \* sin(radian);

int new\_y = cy + (\*x - cx) \* sin(radian) + (\*y - cy) \* cos(radian);

\*x = new\_x;

\*y = new\_y;

}

int main() {

int gd = DETECT, gm, i;

int choice;

float angle;

initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

// Draw the object as it is

drawPolygon(3, points);

delay(2000);

printf("Choose an operation:\n");

printf("1. Translation\n");

printf("2. Rotation\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter translation values (x y): ");

scanf("%d %d", &xt, &yt);

break;

case 2:

printf("Enter rotation angle (in degrees): ");

scanf("%f", &angle);

angle \*= -1; // Reverse angle for clockwise rotation

break;

default:

printf("Invalid choice\n");

return 0;

}

cleardevice();

// Apply the chosen transformation to the object and redraw it

switch (choice) {

case 1:

for (i = 0; i < 6; i += 2) {

points[i] += xt;

points[i + 1] += yt;

}

drawPolygon(3, points);

break;

case 2: {

int cx = (points[0] + points[2] + points[4]) / 3;

int cy = (points[1] + points[3] + points[5]) / 3;

for (i = 0; i < 6; i += 2) {

rotatePoint(&points[i], &points[i + 1], cx, cy, angle);

}

drawPolygon(3, points);

break;

}

}

getch();

closegraph();

return 0;

}

**11) Write a program to implement of different 2D transformations:**

**a)Scale a polygon by 3 units**

**b)Reflect a polygon about x.**

#include<graphics.h>

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int points[6] = {100, 100, 200, 100, 150, 50};

void drawPolygon(int \*points, int sides, int color) {

int i,x1,x2,y1,y2;

setcolor(color);

for (i = 0; i < sides; i++) {

x1 = points[2 \* i];

y1 = points[2 \* i + 1];

x2 = points[2 \* ((i + 1) % sides)];

y2 = points[2 \* ((i + 1) % sides) + 1];

line(x1, y1, x2, y2);

}

}

void scalePolygon(int \*points, int sides, float scale) {

int i;

for (i = 0; i < sides \* 2; i++) {

points[i] = points[i] \* scale;

}

}

void reflectPolygon(int \*points, int sides, char axis) {

int i;

int max\_x = getmaxx() / 2;

int max\_y = getmaxy() / 2;

// Translate to center

for (i = 0; i < sides \* 2; i += 2) {

points[i] -= max\_x;

}

for (i = 1; i < sides \* 2; i += 2) {

points[i] -= max\_y;

}

if (axis == 'x' || axis == 'X') {

for (i = 1; i < sides \* 2; i += 2) {

points[i] = -points[i];

}

} else if (axis == 'y' || axis == 'Y') {

for (i = 0; i < sides \* 2; i += 2) {

points[i] = -points[i];

}

}

// Translate back to positive coordinates

for (i = 0; i < sides \* 2; i += 2) {

points[i] += max\_x;

}

for (i = 1; i < sides \* 2; i += 2) {

points[i] += max\_y;

}

}

int main() {

int i,gd = DETECT, gm;

char axis;

int sides = 3;

int scaledPoints[6];

initgraph(&gd, &gm, (char\*)"c://turboc3//bgi");

// Draw original polygon

drawPolygon(points, sides, WHITE);

getch();

cleardevice();

// Scale polygon by 3 units

for ( i = 0; i < 6; i++) {

scaledPoints[i] = points[i];

}

scalePolygon(scaledPoints, sides, 3);

// Draw scaled polygon

drawPolygon(scaledPoints, sides, GREEN);

getch();

cleardevice();

// Reflect polygon about x or y axis

printf("Enter the axis of reflection (x or y): ");

scanf(" %c", &axis);

reflectPolygon(scaledPoints, sides, axis);

// Draw reflected polygon

drawPolygon(scaledPoints, sides, RED);

getch();

closegraph();

return 0;

}

**12)Write a program to implement of different 3D transformations:**

**a) Translate a polygon by 2 units.**

**b)Rotate a polygon about 90 degree.**

**13) Write a program to implement of different 3D transformations:**

**a)Scale a polygon by 3 units**

**b)Reflect a polygon about z.**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

int maxx,maxy,midx,midy; //global variables

void axis()

{

getch();

cleardevice();

line(midx,0,midx,maxy); // horizontal line I.e x axis

line(0,midy,maxx,midy); //vertical line I.e y axis

}

void main()

{

int gd,gm,x,y,z,ang,x1,x2,y1,y2;

detectgraph(&gd,&gm);

initgraph(&gd,&gm,"C://TURBOC3//BGI");

setfillstyle(3,25);

maxx = getmaxx();

maxy = getmaxy();

midx=maxx/2;

midy=maxy/2;

outtextxy(100,100,"ORIGINAL OBJECT");

line(midx,0,midx,maxy);

line(0,midy,maxx,midy);

bar3d(midx+100,midy-20,midx+60,midy-90,20,5);

axis();

outtextxy(100,20,"SCALING");

printf("\nEnter the Scaling Factor: ");

scanf("%d%d%d", &x,&y,&z);

bar3d(midx+100,midy-20,midx+60,midy-90,20,5);

bar3d(midx+(x\*100),midy-(y\*20),midx+(x\*60),midy-(y\*90),20\*z,5);

axis();

outtextxy(100,20,"REFLECTION");

printf("\n Enter the Reflection angle: ");

scanf("%d",&ang);

x1=100\*cos(ang\*3.14/180)-20\*sin(ang\*3.14/180);

y1=100\*sin(ang\*3.14/180)+20\*sin(ang\*3.14/180);

x2=60\*cos(ang\*3.14/180)-90\*sin(ang\*3.14/180);

y2=60\*sin(ang\*3.14/180)+90\*sin(ang\*3.14/180);

axis();

printf("\n After reflection about z-axis\n");

bar3d(midx+100,midy-20,midx+60,midy-90,20,5);

bar3d(midx+x1,midy-y1,midx+x2,midy-y2,20,5);

axis();

closegraph();

}

**14. Write a Program to draw a Hilbert curve upto ‘n’ iterations. (l30)**

#include <graphics.h>

#include <stdio.h>

#include <math.h>

#include <conio.h>

#include <dos.h>

// Function to move to the next point based on the direction

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);

}

// Recursive function to draw the Hilbert Curve

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);

delay(100);

hilbert(r, d, l, u, i, h, x, y);

move(d, h, x, y);

delay(100);

hilbert(r, d, l, u, i, h, x, y);

move(l, h, x, y);

delay(100);

hilbert(u, l, d, r, i, h, x, y);

}

}

int main() {

int n, x1, y1;

int x0 = 50, y0 = 150, x, y, h = 10;

int r = 2, d = 3, l = 4, u = 1;

printf("Give the value of n: ");

scanf("%d", &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);

delay(100);

getch();

closegraph();

return 0;

}