1. **To plot a point (pixel) on the screen.**

#include <iostream.h >

#include < conio.h >

#include < graphics.h >

void main()

{

int gdriver = DETECT, gmode;

initgraph(&gdriver, &gmode, "c:\\tc\\bgi");

putpixel(100,100,15);

getch();

}

**2. To draw a straight line using DDA Algorithm**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<graphics.h>

void main()

{

int i,steps,x1,x2,y1,y2;

float x,y,xinc,yinc,dx,dy;

char msg[86];

int gdriver = DETECT,gmode,errorcode;

clrscr();

initgraph(&gdriver,&gmode,"f:\\tc");

printf("\n Enter the co ordinates ");

scanf("%d%d%d%d",&x1,&x2,&y1,&y2);

cleardevice();

outtextxy(200,4,"Draw Using DDA");

line(x1,x2,y1,y2);

dx = x2 - x1;

dy = y2 - y1;

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

steps = abs(dx);

else

steps = abs(dy);

xinc = (float)dx/steps ;

yinc = (float)dy/steps ;

y = y1;

x = x1;

putpixel(ceil(x),ceil(y),20);

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

{

x += xinc ;

y += yinc ;

putpixel(x,y,2);

delay(45);

}

getch();

}

**3. To draw a straight line using Bresenham’s Algorithm.**

#include<stdio.h>

#include<math.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int x1,x2,y1,y2;

int gd=DETECT,gm;

void linebres(int,int,int,int); printf("Enter the two end points:");

scanf("%d%d%d%d",&x1,&x2,&y1,&y2);

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

cleardevice();

linebres(x1,y1,x2,y2);

getch();

line(x1,y1,x2,y2);

getch();

closegraph();

}

void linebres(int x1,int y1,int x2,int y2)

{

int dx=abs(x1-x2),dy=abs(y1-y2); int p,x,y,i,xend,yend;

if(dx!=0)

{

p=2\*dy-dx; if(x1>x2)

{

x=x2;

y=y2;

xend=x1;

}

else

{

x=x1;

y=y1;

xend=x2;

}

putpixel(x,y,2);

for(i=x;i<xend;i++)

{

x+=1;

if(p<0)

p+=2\*dy; else

p+=2\*(dy-dx);

}

putpixel(x,y,2);

}

else

{

p=2\*dx-dy; if(y1>y2)

{

x=x2;

y=y2;

yend=y2;

}

putpixel(x,y,2);

for(i=y;i<yend;i++)

{

y+=1;

if(p<0)

p+=2\*dx; else

{

x+=1;

p+=2\*(dx-dy);

}

putpixel(x,y,2);

}

}

}

**4. Implementation of mid-point circle generating Algorithm**

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void bcircle(int xcent,int ycent,int x,int y);

void main()

{

int gd=DETECT,gm;

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

int x=0,y,d,r,xcent=300,ycent=300;

cout<<"Enter the radius";

cin>>r;

y=r; d=5/4-r;

while(x<y)

{

bcircle(xcent,ycent,x,y);

if(d<0)

{

x=x+1;

d=d+2\*x+1;

}

else

{

x=x+1;

y=y-1;

d=d+2\*(x-y)+1;

}

}

Getch();

7

}

void bcircle(int xcent,int ycent,int x,int y)

{

putpixel(xcent-x,ycent-y,6);

putpixel(xcent-y,ycent-x,12);

putpixel(xcent+y,ycent-x,24);

putpixel(xcent+x,ycent-y,14);

putpixel(xcent+x,ycent+y,13);

putpixel(xcent+y,ycent+x,9);

putpixel(xcent-y,ycent+x,27);

putpixel(xcent-x,ycent+y,5);

getch();

closegraph();

}

**5. Implementation of ellipse generating Algorithm**

#include<stdio.h>

#include<conio.h>

#include<math.h>

#include<graphics.h>

main()

{

int gd=DETECT,gm; int xcenter,ycenter,rx,ry;

int p,x,y,px,py,rx1,ry1,rx2,ry2; initgraph(&gd,&gm,"c:\\tc\\bgi"); printf("Enter The Radius Value:\n"); scanf("%d%d",&rx,&ry);

printf("Enter The xcenter and ycenter Values:\n"); scanf("%d%d",&xcenter,&ycenter);

ry1=ry\*ry;

rx1=rx\*rx;

ry2=2\*ry1;

rx2=2\*rx1;

/\* REGION 1 \*/

x=0;

y=ry;

plotpoints(xcenter,ycenter,x,y); p=(ry1-rx1\*ry+(0.25\*rx1)); px=0;

py=rx2\*y;

while(px<py)

{

x=x+1;

px=px+ry2;

if(p>=0) y=y-1;

py=py-rx2;

if(p<0)

p=p+ry1+px;

else

p=p+ry1+px-py; plotpoints(xcenter,ycenter,x,y);

/\* REGION 2\*/

p=(ry1\*(x+0.5)\*(x+0.5)+rx1\*(y-1)\*(y-1)-rx1\*ry1); while(y>0)

{

y=y-1;

py=py-rx2;

if(p<=0)

{

x=x+1;

px=px+ry2;

}

if(p>0)

p=p+rx1-py;

else

p=p+rx1-py+px; plotpoints(xcenter,ycenter,x,y);

}

}

getch();

return(0);

}

int plotpoints(int xcenter,int ycenter,int x,int y)

{

putpixel(xcenter+x,ycenter+y,6); putpixel(xcenter-x,ycenter+y,6); putpixel(xcenter+x,ycenter-y,6); putpixel(xcenter-x,ycenter-y,6);

}

**6. To translate an object with translation parameters in X and Y directions.**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm; initgraph(&gd,&gm,"c:\\tc\\bgi");

int a[10][2],tx,ty,n;

cout<<"enter the number of vertices:" ; cin>>n;

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

{

cout<<"enter the coordinates:"; cin>>a[i][0]>>a[i][1]>>a[i+1][0]>>a[i+1][1]);

}

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

{

line(a[i][0],a[i][1],a[(i+1)%n][0],a[(i+1)%n][1]);

}

cout<<enter the tranformation vertex tx,ty:\n"; cin>>tx>>ty;

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

{

a[i][0]=a[i][0]+tx;

a[i][1]=a[i][1]+ty;

}

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

{

line(a[i][0],a[i][1],a[(i+1)%n][0],a[(i+1)%n][1]);

}

getch();

closegraph();

}

**7. To scale an object with scaling factors along X and Y directions**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm; initgraph(&gd,&gm,"c:\\tc\\bgi");

int a[10][2],Sx,Sy,n;

cout<<"enter the number of vertices:" ; cin>>n;

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

{

cout<<"enter the coordinates:"; cin>>a[i][0]>>a[i][1]>>a[i+1][0]>>a[i+1][1]);

}

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

{

line(a[i][0],a[i][1],a[(i+1)%n][0],a[(i+1)%n][1]);

}

cout<<enter the Scaling factors Sx ,Sy:\n"; cin>>tx>>ty;

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

{

a[i][0]=a[i][0]\*Sx;

a[i][1]=a[i][1]\*Sy;

}

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

{

line(a[i][0],a[i][1],a[(i+1)%n][0],a[(i+1)%n][1]);

}

getch();

closegraph();

}

**8. To rotate an object with a certain angle about origin.**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm; initgraph(&gd,&gm,"c:\\tc\\bgi");

int a[10][2],Sx,Sy,n;

cout<<"enter the number of vertices:" ; cin>>n;

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

{

cout<<"enter the coordinates:"; cin>>a[i][0]>>a[i][1]>>a[i+1][0]>>a[i+1][1]);

}

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

{

line(a[i][0],a[i][1],a[(i+1)%n][0],a[(i+1)%n][1]);

}

cout<<enter the rotating angle:"; cin>>y;

k=(y\*3.14)/180; for(i=0;i<=n;i++)

{

a[i][0]=(a[i][0]\*cos(k)-(a[i][1])\*sin(k); a[i][1]=(a[i][0])\*sin(k)-(a[i][1]\*cos(k);

}

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

{

line(a[i][0],a[i][1],a[(i+1)%n][0],a[(i+1)%n][1]);

}

getch();

closegraph(); }

**9. Perform the rotation of an object with certain angle about an arbitrary point**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm; initgraph(&gd,&gm,"c:\\tc\\bgi");

int a[10][2],Sx,Sy,n;

cout<<"enter the number of vertices:" ; cin>>n;

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

{

cout<<"enter the coordinates:"; cin>>a[i][0]>>a[i][1]>>a[i+1][0]>>a[i+1][1]);

}

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

{

line(a[i][0],a[i][1],a[(i+1)%n][0],a[(i+1)%n][1]);

}

cout<<enter the rotating angle:"; cin>>y;

k=(y\*3.14)/180; for(i=0;i<=n;i++)

{

a[i][0]=fx+(a[i][0]-fx)\*cos(k)-(a[i][1]-fy)\*sin(k); a[i][1]=fy+(a[i][0]-fx)\*sin(k)-(a[i][1]-fy)\*cos(k);

}

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

{

line(a[i][0],a[i][1],a[(i+1)%n][0],a[(i+1)%n][1]);

}

getch();

closegraph();

}

**10. To perform composite transformations of an object.**

#include<iostream.h>

#include<conio.h>

#include<math.h>

#include<graphics.h>

#include<stdlib.h>

void main()

{

int gd,gm,n,i,xa[10],ya[10],op,tx,ty,xa1[10],ya1[10],theta,xf,yf,rx,ry, sx,sy,shx,shy,xref,yref;

char d; gd=DETECT;

initgraph(&gd,&gm,"c:\\tc\\bgi"); cout<<"enter the no of points"; cin>>n;

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

{

cout<<"enter the coordinates"<<i+1; cin>>xa[i]>>ya[i];

}

do

{

cout<<"menu";

cout<<"\n1.translation\n2.rotation\n3.scaling\n4.shearing\n5.reflection\n6.exit";

cin>>op;

switch(op)

{

case 1:

cout<<"before translation"; for(i=0;i<n;i++)

{

line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);

}

cout<<"enter the translation vector"; cin>>tx>>ty;

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

{

xa[i]=xa[i]+tx;

ya[i]=ya[i]+ty;

}

cleardevice();

cout<<"after translation"; for(i=0;i<n;i++)

{

line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);

}

getch();

cleardevice();

break; case 2:

cout<<"enter the rotation angle"; cin>>theta; theta=(theta\*3.14)/180; cout<<"enter the reference points"; cin>>xf>>yf;

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

{

xa1[i]=xf+(xa[i]-xf)\*cos(theta)-(ya[i]-yf)\*sin(theta); ya[i]=yf+(xa[i]-xf)\*sin(theta)-(ya[i]-yf)\*cos(theta);

xa[i]=xa1[i];

}

cleardevice(); cout<<"before rotation"; for(i=0;i<n;i++)

{

line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);

}

cout<<"after rotation"; for(i=0;i<n;i++)

{

line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);

}

getch();

cleardevice();

break;

case 3:

cout<<"enter the scaling factor"; cin>>sx>>sy;

cout<<"enter the reference point"; cin>>rx>>ry;

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

{

xa[i]=xa[i]\*sx+rx\*(1-sx); ya[i]=ya[i]\*sy+ry\*(1-sy);

}

cleardevice(); cout<<"before scaling"; for(i=0;i<n;i++)

{

line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);

}

cout<<"after scaling";

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

{

line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);

}

getch();

cleardevice();

break;

case 4:

cout<<"enter the shear value"; cin>>shx>>shy;

cout<<"enter the reference point"; cin>>xref>>yref;

cout<<"enter the shear direction x or y"; cin>>d;

if(d=='x')

{

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

{

xa[i]=xa[i]+shx\*(ya[i]-yref); ya[i]=ya[i];

}

}

cleardevice(); cout<<"before shearing"; for(i=0;i<n;i++)

{

line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);

}

cout<<"after shearing"; for(i=0;i<n;i++)

{

line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);

}

getch();

cleardevice();

break;

case 5:

cout<<"before reflection"; for(i=0;i<n;i++)

{

line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);

}

cout<<"after reflection"; for(i=0;i<n;i++)

{

line(ya[i],xa[i],ya[(i+1)%n],xa[(i+1)%n]);

}

getch();

cleardevice();

break;

case 6:

exit(0);

break;

}

}while(op!=6);

}**11. To perform the reflection of an object about major axis**

# include <iostream.h>

# include <conio.h>

# include <graphics.h>

# include <math.h>

char IncFlag;

int PolygonPoints[3][2] = {{10,100},{110,100},{110,200}};

void PolyLine()

{

int iCnt;

cleardevice();

line(0,240,640,240);

line(320,0,320,480);

for (iCnt=0; iCnt<3; iCnt++)

{

line(PolygonPoints[iCnt][0],PolygonPoints[iCnt][1], PolygonPoints[(iCnt+1)%3][0],PolygonPoints[(iCnt+1)%3][1]);

}

}

void Reflect()

{

float Angle;

int iCnt;

int Tx,Ty;

cout<<endl;

for (iCnt=0; iCnt<3; iCnt++)

PolygonPoints[iCnt][1] = (480 - PolygonPoints[iCnt][1]);

}

void main()

{

18

int gDriver = DETECT, gMode;

int iCnt;

initgraph(&gDriver, &gMode, "C:\\TC\\BGI");

for (iCnt=0; iCnt<3; iCnt++)

{

PolygonPoints[iCnt][0] += 320; PolygonPoints[iCnt][1] = 240 - PolygonPoints[iCnt][1];

}

PolyLine();

getch();

Reflect();

PolyLine();

getch();

}

**12. To clip line segments against windows using Cohen Sutherland Algorithm**

#include<iostream.h>

#include<conio.h>

#include<math.h>

#include<graphics.h>

int pixels[2][4];

float xn,yn,xn1,yn1,m;

int xmin,ymin,xmax,ymax,x1,y1,x2,y2;

//int ch,n;

void encode(int x1,int y1,int x2,int y2)

{

int i,j,f=1;

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

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

pixels[i][j]=0;

if(y1>ymax)

pixels[0][0]=1;

if(y1<ymin)

pixels[0][1]=1;

if(x1>xmax)

pixels[0][2]=1;

if(x1<xmin)

pixels[0][3]=1;

if(y2>ymax)

pixels[1][0]=1;

if(y2<ymin)

pixels[1][1]=1;

if(x2>xmax)

pixels[1][2]=1;

if(x2<xmin)

pixels[1][3]=1;

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

{

if((pixels[0][j]==0)&& (pixels[1][j]==0))

continue;

if((pixels[0][j]==1)&& (pixels[1][j]==1))

{

f=2;

break;

}

f=3;

}

switch(f)

{

case 1:

cleardevice();

rectangle(xmin,ymin,xmax,ymax);

line(x1,y1,x2,y2);

break;

case 2:

cout<<"\nThe line is completely outside";

break;

case 3:

m=(y2-y1)/(x2-x1);

xn=x1;xn1=x2;

yn=y1;yn1=y2;

if(pixels[0][0]==1)

{

xn=x1+(ymax-y1)/m;

yn=ymax;

}

if(pixels[0][1]==1)

{

xn=x1+(ymin-y1)/m;

yn=ymin;

}

if(pixels[0][2]==1)

{

yn=y1+(xmax-x1)\*m;

xn=xmax;

}

if(pixels[0][3]==1)

{

yn=y1+(xmin-x1)\*m;

xn=xmin;

}

if(pixels[1][0]==1)

{

xn1=x2+(ymax-y2)/m;

yn1=ymax;

}

if(pixels[1][1]==1)

{

xn1=x2+(ymin-y2)/m;

yn1=ymin;

}

if(pixels[1][2]==1)

{

yn1=y2+(xmax-x2)\*m;

xn1=xmax;

}

if(pixels[1][3]==1)

{

yn1=y2+(xmin-x2)\*m;

xn1=xmin;

}

cleardevice();

rectangle(xmin,ymin,xmax,ymax);

line(xn,yn,xn1,yn1);

break;

}

}

void main()

{

int gd=DETECT,gm,i,j;

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

cout<<"\nEnter the co-ordinates for the window:";

cin>>xmin>>ymin>>xmax>>ymax;

rectangle(xmin,ymin,xmax,ymax);

cout<<"\nenter the line co-ordinates:";

cin>>x1>>y1>>x2>>y2;

line(x1,y1,x2,y2);

//sleep(5);

encode(x1,y1,x2,y2);

cout<<"\nLine After Clipping";

getch();

//closegraph();

}

**13. Perform the polygon clipping against windows using Sutherland Hodgeman technique**

#include <iostream.h>

#include <conio.h>

#include <graphics.h>

int xl,yl,xh,yh,poly[100],n;

void left\_clip(){

int temp[100],i,j=0,count=0,x1,y1,x2,y2;

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

x1 = poly[i];

y1 = poly[i+1];

x2 = poly[i+2];

y2 = poly[i+3];

if(x1<xl && x2<xl){

//both points outside. Do not store any vertices

}else if(x1>xl && x2>xl){

//both points inside. Store second vertex

temp[j] = x2;

temp[j+1] = y2;

j+=2;

count++;

}else if(x1<xl && x2>xl){

//outside to inside. Store intersection n second vertex

int x=xl;

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

temp[j]=x;

temp[j+1]=y;

temp[j+2]=x2;

temp[j+3]=y2;

j+=4;

count+=2;

}else{

//inside to outside. Store intersection only

int x=xl;

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

temp[j] = x;

temp[j+1] = y;

j+=2;

count++;

}

}

n=count;

//store 1st vertex as last

temp[j]=temp[0];

temp[j+1]=temp[1];

for(i=0;i<2\*(n+1);i++)

poly[i]=temp[i];

}

void right\_clip(){

int temp[100],i,j=0,count=0,x1,y1,x2,y2;

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

x1 = poly[i];

y1 = poly[i+1];

x2 = poly[i+2];

y2 = poly[i+3];

if(x1>xh && x2>xh){

//both points outside. Do not store any vertices

}else if(x1<xh && x2<xh){

//both points inside. Store second vertex

temp[j] = x2;

temp[j+1] = y2;

j+=2;

count++;

}else if(x1>xh && x2<xh){

//outside to inside. Store intersection n second vertex

int x=xh;

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

temp[j]=x;

temp[j+1]=y;

temp[j+2]=x2;

temp[j+3]=y2;

j+=4;

count+=2;

}else{

//inside to outside. Store intersection only

int x=xh;

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

temp[j] = x;

temp[j+1] = y;

j+=2;

count++;

}

}

n=count;

//store 1st vertex as last

temp[j]=temp[0];

temp[j+1]=temp[1];

for(i=0;i<2\*(n+1);i++)

poly[i]=temp[i];

}

void bottom\_clip(){

int temp[100],i,j=0,count=0,x1,y1,x2,y2;

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

x1 = poly[i];

y1 = poly[i+1];

x2 = poly[i+2];

y2 = poly[i+3];

if(y1>yh && y2>yh){

//both points outside. Do not store any vertices

}else if(y1<yh && y2<yh){

//both points inside. Store second vertex

temp[j] = x2;

temp[j+1] = y2;

j+=2;

count++;

}else if(y1>yh && y2<yh){

//outside to inside. Store intersection n second vertex

int x= x1 + (yh-y1)/((float)(y2-y1)/(x2-x1));

int y= yl;

temp[j]=x;

temp[j+1]=y;

temp[j+2]=x2;

temp[j+3]=y2;

j+=4;

count+=2;

}else{

//inside to outside. Store intersection only

int x= x1 + (yh-y1)/((float)(y2-y1)/(x2-x1));

int y= yl;

temp[j] = x;

temp[j+1] = y;

j+=2;

count++;

}

}

n=count;

//store 1st vertex as last

temp[j]=temp[0];

temp[j+1]=temp[1];

for(i=0;i<2\*(n+1);i++)

poly[i]=temp[i];

}

void top\_clip(){

int temp[100],i,j=0,count=0,x1,y1,x2,y2;

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

x1 = poly[i];

y1 = poly[i+1];

x2 = poly[i+2];

y2 = poly[i+3];

if(y1<yl && y2<yl){

//both points outside. Do not store any vertices

}else if(y1>yl && y2>yl){

//both points inside. Store second vertex

temp[j] = x2;

temp[j+1] = y2;

j+=2;

count++;

}else if(y1<yl && y2>yl){

//outside to inside. Store intersection n second vertex

int x= x1 + (yl-y1)/((float)(y2-y1)/(x2-x1));

int y= yl;

temp[j]=x;

temp[j+1]=y;

temp[j+2]=x2;

temp[j+3]=y2;

j+=4;

count+=2;

}else{

//inside to outside. Store intersection only

int x= x1 + (yl-y1)/((float)(y2-y1)/(x2-x1));

int y= yl;

temp[j] = x;

temp[j+1] = y;

j+=2;

count++;

}

}

n=count;

//store 1st vertex as last

temp[j]=temp[0];

temp[j+1]=temp[1];

for(i=0;i<2\*(n+1);i++)

poly[i]=temp[i];

}

void main(){

int gdriver = DETECT,gmode;

initgraph(&gdriver,&gmode,"C:\\TC\\BGi");

int i;

//setcolor(BLUE);

cout<<"Enter bottom left and top right co-ordinates of window: ";

cin>>xl>>yl>>xh>>yh;

rectangle(xl,yl,xh,yh);

cout<<"Enter the no. of vertices: ";

cin>>n;

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

cout<<"Enter co-ordinates of vertex "<<(i/2+1)<<": ";

cin>>poly[i]>>poly[i+1];

}

//store 1st vertex as last

poly[2\*n] = poly[0];

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

drawpoly(n+1,poly);

getch();

left\_clip();

cout<<n<<" ";

right\_clip();

cout<<n<<" ";

//bottom\_clip();

cout<<n<<" ";

//top\_clip();

cout<<n<<" ";

cleardevice();

rectangle(xl,yl,xh,yh);

cout<<"After clipping:";

setcolor(YELLOW);

//cout<<n;

drawpoly(n+1,poly);

getch();

closegraph();

}

**15. Implementation of flood-fill and boundary-fill algorithms**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<dos.h>

void b\_fill(int,int,int,int);

void flood\_fill4(int,int,int,int);

void main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\tc\\bgi");

rectangle(50,50,150,150);

b\_fill(51,51,GREEN,WHITE);

rectangle(150,150,250,250);

b\_fill(151,151,RED,WHITE);

rectangle(250,250,350,350);

b\_fill(251,251,BLUE,WHITE);

rectangle(250,200,300,250);

flood\_fill4(251,201,4,0); // 4(red) - newColor 0(black) - oldColor

getch();

}

void b\_fill(int x,int y,int newc,int boundc)

{

int g;

g=getpixel(x,y);

if((g!=boundc)&&(g!=newc))

{

putpixel(x,y,newc);

delay(1);

b\_fill(x+1,y,newc,boundc);

//b\_fill(x-1,y,newc,boundc);

b\_fill(x,y-1,newc,boundc);

b\_fill(x,y+1,newc,boundc);

}

}

void flood\_fill4(int x,int y,int newColor,int oldColor)

{

int c;

c=getpixel(x,y);

if(c==oldColor)

{

setcolor(newColor);

putpixel (x,y,newColor);

delay(10);

flood\_fill4(x+1,y,newColor,oldColor);

flood\_fill4(x,y+1,newColor,oldColor);

flood\_fill4(x-1,y,newColor,oldColor);

flood\_fill4(x,y-1,newColor,oldColor);

}

}