Q1. Write a program to implement DDA and Bresenham’s line drawing algorithm.

#include <bits/stdc++.h>

using namespace std;

class IncrementalAlgo{

float sx,sy,tx,ty, slope;

vector<vector<float>> points;

public:

void get();

void incremental();

void print();

};

void IncrementalAlgo::get(){

cout<<"\nEnter starting coordinate (x,y) : "; cin>>sx>>sy;

cout<<"Enter ending coordinate (x,y) : "; cin>>tx>>ty;

slope = (ty-sy)/(tx-sx);

}

void IncrementalAlgo::incremental(){

float x=sx,y=sy;

points.push\_back({x,y});

do{

if(slope<1){

x = x + 1;

y = y + slope;

}

else{

x = x + 1/slope;

y = y + 1;

}

points.push\_back({x,y});

}while(x!=tx && y!=ty);

}

void IncrementalAlgo::print(){

int len = points.size();

for(int i=0;i<len;i++){

cout<<"("<<round(points[i][0])<<","<<round(points[i][1])<<")--->";

}

}

int main(){

IncrementalAlgo obj;

obj.get();

obj.incremental();

obj.print();

}

/\* OUTPUT

Enter starting coordinate (x,y) : 5 6

Enter ending coordinate (x,y) : 9 12

(5,6)--->(6,7)--->(6,8)--->(7,9)--->(8,10)--->(8,11)--->(9,12)--->

--------------------------------

Process exited after 8.383 seconds with return value 0

Press any key to continue . . .

\*/

#include <bits/stdc++.h>

using namespace std;

class midPointLine{

float sx,sy,tx,ty,dy,dx, p;

vector<vector<float>> points;

public:

void get();

void bresenham();

void print();

};

void midPointLine::get(){

cout<<"\nEnter starting coordinate (x,y) : "; cin>>sx>>sy;

cout<<"Enter ending coordinate (x,y) : "; cin>>tx>>ty;

dy = ty-sy; dx = tx-sx;

p = 2\*dy-dx;

}

void midPointLine::bresenham(){

float x=sx,y=sy;

points.push\_back({x,y});

do{

if(p<0){

x = x + 1;

p += 2\*dy ;

}

else{

x = x + 1;

y = y + 1;

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

}

points.push\_back({x,y});

}while(x!=tx && y!=ty);

}

void midPointLine::print(){

int len = points.size();

cout<<endl;

for(int i=0;i<len;i++){

cout<<"("<<round(points[i][0])<<","<<round(points[i][1])<<")--->";

}

}

int main(){

midPointLine obj;

obj.get();

obj.bresenham();

obj.print();

}

/\*

OUTPUT

Enter starting coordinate (x,y) : 9 18

Enter ending coordinate (x,y) : 14 22

(9,18)--->(10,19)--->(11,20)--->(12,20)--->(13,21)--->(14,22)--->

--------------------------------

Process exited after 7.495 seconds with return value 0

Press any key to continue . . .

\*/

Q2. Write a program to implement mid-point circle drawing algorithm.

#include <bits/stdc++.h>

using namespace std;

class Circle{

float r,dFactor;

vector<vector<float>> points;

public:

void get();

void circle();

void print();

};

void Circle::get(){

cout<<"Enter radius : "; cin>>r;

dFactor = 1.25 - r;

}

void Circle::circle(){

float x=0,y=r;

points.push\_back({x,y});

while(y>x){

if(dFactor<0){

dFactor += 2\*x+3;

x++;

}

else{

dFactor += 2\*(x-y)+5;

y--; x++;

}

points.push\_back({x,y});

points.push\_back({x,-y});

points.push\_back({-x,y});

points.push\_back({-x,-y});

points.push\_back({y,x});

points.push\_back({y,-x});

points.push\_back({-y,x});

points.push\_back({-y,-x});

}

}

void Circle::print(){

int len = points.size();

cout<<len<<endl;

for(int i=0;i<len;i++){

cout<<"("<<round(points[i][0])<<","<<round(points[i][1])<<")--->";

}

}

int main(){

Circle obj;

obj.get();

obj.circle();

obj.print();

}

/\*

OUTPUT

Enter radius : 6

[

(0,6), (1,6), (1,-6), (-1,6), (-1,-6), (6,1), (6,-1), (-6,1), (-6,-1), (2,6), (2,-6), (-2,6), (-2,-6), (6,2), (6,-2), (-6,2), (-6,-2), (3,5), (3,-5), (-3,5), (-3,-5), (5,3), (5,-3), (-5,3), (-5,-3), (4,4), (4,-4), (-4,4), (-4,-4), (4,4), (4,-4), (-4,4), (-4,-4), ]

--------------------------------

Process exited after 1.374 seconds with return value 0

Press any key to continue . . .

\*/

Q4. Write a program to clip a polygon using Sutherland Hodgeman algorithm.

#include<iostream>

#include<conio.h>

#include<graphics.h>

#define round(a) ((int)(a+0.5))

using namespace std;

class polygon

{

int k,n,poly[20];

float xmin,ymin,xmax,ymax,arr[20],polyy[20],m;

int \*\*p1,\*\*p2;

public:

void getPolygon();

void clipl(float,float,float,float);

void clipr(float,float,float,float);

void clipt(float,float,float,float);

void clipb(float,float,float,float);

void polygon\_cliping();

};

void polygon::getPolygon()

{

cout<<"Coordinates of rectangular clip window :\n";

cout<<"\nMinimum value of X-coordinate:-";

cin>>xmin;

cout<<"\nMinimum value of Y-coordinate:-";

cin>>ymin;

cout<<"\nMaximum value of X-coordinate:-";

cin>>xmax;

cout<<"\nMaximum value of Y-coordinate:-";

cin>>ymax;

cout<<"\n\nPolygon to be clipped :\nNumber of sides :";

cin>>n;

p1=new int\*[n];

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

p1[i]=new int[2];

cout<<"\nEnter the coordinate of the Polygon:-";

int i;

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

cin>>polyy[i];

polyy[i]=polyy[0];

polyy[i+1]=polyy[1];

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

poly[i]=round(polyy[i]);

//Save the original points of polygon

int a=0;

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

for(int j=0;j<2;++j)

p1[i][j]=poly[a++];

setcolor(RED);

rectangle(xmin,ymax,xmax,ymin);

cout<<"\n\nUNCLIPPED POLYGON\n";

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

{

for(int j=0;j<2;++j)

cout<<p1[i][j]<<" ";

cout<<"\n";

}

setcolor(WHITE);

fillpoly(n,poly);

}

void polygon::clipl(float x1,float y1,float x2,float y2)

{

if(x2-x1)

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

else

m=100000;

if(x1 >= xmin && x2 >= xmin)

{

arr[k]=x2;

arr[k+1]=y2;

k+=2;

}

if(x1 < xmin && x2 >= xmin)

{

arr[k]=xmin;

arr[k+1]=y1+m\*(xmin-x1);

arr[k+2]=x2;

arr[k+3]=y2;

k+=4;

}

if(x1 >= xmin && x2 < xmin)

{

arr[k]=xmin;

arr[k+1]=y1+m\*(xmin-x1);

k+=2;

}

}

void polygon::clipr(float x1,float y1,float x2,float y2)

{

if(x2-x1)

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

else

m=100000;

if(x1 <= xmax && x2 <= xmax)

{

arr[k]=x2;

arr[k+1]=y2;

k+=2;

}

if(x1 > xmax && x2 <= xmax)

{

arr[k]=xmax;

arr[k+1]=y1+m\*(xmax-x1);

arr[k+2]=x2;

arr[k+3]=y2;

k+=4;

}

if(x1 <= xmax && x2 > xmax)

{

arr[k]=xmax;

arr[k+1]=y1+m\*(xmax-x1);

k+=2;

}

}

void polygon::clipt(float x1,float y1,float x2,float y2)

{

if(y2-y1)

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

else

m=100000;

if(y1 <= ymax && y2 <= ymax)

{

arr[k]=x2;

arr[k+1]=y2;

k+=2;

}

if(y1 > ymax && y2 <= ymax)

{

arr[k]=x1+m\*(ymax-y1);

arr[k+1]=ymax;

arr[k+2]=x2;

arr[k+3]=y2;

k+=4;

}

if(y1 <= ymax && y2 > ymax)

{

arr[k]=x1+m\*(ymax-y1);

arr[k+1]=ymax;

k+=2;

}

}

void polygon::clipb(float x1,float y1,float x2,float y2)

{

if(y2-y1)

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

else

m=100000;

if(y1 >= ymin && y2 >= ymin)

{

arr[k]=x2;

arr[k+1]=y2;

k+=2;

}

if(y1 < ymin && y2 >= ymin)

{

arr[k]=x1+m\*(ymin-y1);

arr[k+1]=ymin;

arr[k+2]=x2;

arr[k+3]=y2;

k+=4;

}

if(y1 >= ymin && y2 < ymin)

{

arr[k]=x1+m\*(ymin-y1);

arr[k+1]=ymin;

k+=2;

}

}

void polygon::polygon\_cliping()

{

//cleardevice();

k=0;

int i;

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

clipl(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);

n=k/2;

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

polyy[i]=arr[i];

polyy[i]=polyy[0];

polyy[i+1]=polyy[1];

k=0;

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

clipt(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);

n=k/2;

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

polyy[i]=arr[i];

polyy[i]=polyy[0];

polyy[i+1]=polyy[1];

k=0;

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

clipr(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);

n=k/2;

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

polyy[i]=arr[i];

polyy[i]=polyy[0];

polyy[i+1]=polyy[1];

k=0;

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

clipb(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);

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

poly[i]=round(arr[i]);

setcolor(RED);

rectangle(xmin,ymax,xmax,ymin);

if(k)

fillpoly(k/2,poly);

cout<<"\n\nCLIPPED POLYGON\n";

//cout<<k;

int size=(k)/2;

p2=new int\*[size];

for(int i=0;i<size;++i)

p2[i]=new int[2];

//Save the Clipped points of polygon

int a=0;

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

for(int j=0;j<2;++j)

p2[i][j]=poly[a++];

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

{

for(int j=0;j<2;++j)

cout<<p2[i][j]<<" ";

cout<<"\n";

}

getch();

closegraph();

}

int main() {

int gdriver=DETECT,gmode;

initgraph(&gdriver,&gmode,"");

polygon p1;

p1.getPolygon();

getch();

cleardevice();

p1.polygon\_cliping();

getch();

return 0;

}

/\* OUTPUT

Coordinates of rectangular clip window :

Minimum value of X-coordinate:-200

Minimum value of Y-coordinate:-200

Maximum value of X-coordinate:-400

Maximum value of Y-coordinate:-400

Polygon to be clipped :

Number of sides :3

Enter the coordinate of the Polygon:-150 300

300 150

300 300

UNCLIPPED POLYGON

150 300

300 150

300 300

CLIPPED POLYGON

200 300

200 250

250 200

300 200

300 300

\*/

Q5. Write a program to fill a polygon using Scan line fill algorithm.

// Write a program to fill a polygon using Scan line fill algorithm.

#include <conio.h>

#include <iostream>

#include <graphics.h>

#include <stdlib.h>

using namespace std;

//Declaration of class point

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 input();

void calcs();

void display();

void ints(float);

void sort(int);

};

void poly::input(){

int i;

cout<<"\nEnter the number of vertices of polygon: ";

cin>>v;

if(v>2){

for(i=0; i<v; i++){ //ACCEPT THE VERTICES

cout<<"\n Coordinates of Vertex "<<i+1<<": ";

cout<<"\n\tx"<<(i+1)<<"=";

cin>>p[i].x;

cout<<"\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 Number of vertices is invalid.";

}

//function for finding min/max of the coordinates

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;

}

}

//DISPLAY FUNCTION

void poly::display(){

float s;

s = ymin + 0.01;

delay(10);

cleardevice();

while(s <= ymax){

ints(s);

sort(s);

s++;

}

cout<<"\nFILLED POLYGON DRAWN!";

}

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 // used to make changes in x. so that we can fill our polygon after certain distance

{

x=((x2-x1)\*(z-y1))/(y2-y1);

x=x+x1;

}

if(x<=xmax && x>=xmin)

inter[c++]=x;

}

}

}

void poly::sort(int z) //SORT FUNCTION

{

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); // used to make hollow outlines of a polygon

}

delay(10);

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

delay(10);

line(inter[i],z,inter[i+1],z); // Used to fill the polygon

}

}

int main(){

cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

cout<<"~~ POLYGON FILLING USING SCAN LINE FILL ALGORITHM ~~\n";

cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

initwindow(500,600);

cleardevice();

poly x;

x.input();

x.calcs();

cleardevice();

x.display();

getch();

return 0;

}

/\*

OUTPUT:

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~~ POLYGON FILLING USING SCAN LINE FILL ALGORITHM ~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Enter the number of vertices of polygon: 3

Coordinates of Vertex 1:

x1=100

y1=200

Coordinates of Vertex 2:

x2=200

y2=300

Coordinates of Vertex 3:

x3=250

y3=150

FILLED POLYGON DRAWN!

\*/

#include <bits/stdc++.h>

using namespace std;

class LineClipping{

int xmax,xmin,ymax,ymin,n;

int r1,r2,r3,r4,r5,r6,r7,r8,r9;

vector<vector<int>> lines;

vector<vector<int>> clipped;

vector<vector<int>>::iterator it;

public:

LineClipping(){

r1=9; r2=8; r3=10;

r4=1; r5=0; r6=2;

r7=5; r8=4; r9=6;

}

void input();

int find\_region(int x,int y);

void clipLine();

void clipCurrLine(int x1,int y1,int x2,int y2);

void display(vector<vector<int>> v);

};

void LineClipping::input(){

//window

cout<<"\nEnter Window info :";

cout<<"\nxmax : "; cin>>xmax;

cout<<"ymax : "; cin>>ymax;

cout<<"xmin : "; cin>>xmin;

cout<<"ymin : "; cin>>ymin;

//lines

cout<<"\nEnter number of lines :";

cin>>n;

int x1,y1,x2,y2;

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

cout<<"\nEnter end point of line "<<i+1<<" (x1 y1 x2 y2) : ";

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

lines.push\_back({x1,y1,x2,y2});

}

cout<<"\n\nOriginal lines :";

cout<<"\n==================\n";

display(lines);

}

int LineClipping::find\_region(int x,int y){

// cout<<endl<<x<<" "<<y;

if(x<xmin && y>ymax)

return r1;

else if(x>=xmin && x<=xmax && y>ymax)

return r2;

else if(x>xmax && y>ymax)

return r3;

else if(x<xmin && y>=ymin && y<=ymax)

return r4;

else if(x>=xmin && x<=xmax && y>=ymin && y<=ymax)

return r5;

else if(x>xmax && y>=ymin && y<=ymax)

return r6;

else if(x<xmin && y<ymin)

return r7;

else if(x>=xmin && x<=xmax && y<ymin)

return r8;

else if(x>xmax && y<ymin)

return r9;

else

return -1;

}

void LineClipping::clipCurrLine(int x1,int y1,int x2,int y2){

vector<int> clip;

int codeStart = find\_region(x1,y1);

int codeEnd = find\_region(x2,y2);

bool flag=false, reject = false;

do{

if(!(codeStart | codeEnd)){

//trivially accepted

flag= true;

}

else if(codeStart & codeEnd){

//trivially reject

flag=true;

reject = true;

}

else{

// pick the point which lies outside the clipping window

int x,y;

int pointOutside = codeStart ? codeStart:codeEnd;

if(pointOutside & 1){

//point lies left of xmin

x = xmin;

y = y1 + (y2-y1)\*(xmin-x1)/(x2-x1);

}

else if(pointOutside & 2){

//point lies right of xmax

x = xmax;

y = y1 + (y2-y1)\*(xmax-x1)/(x2-x1);

}

else if(pointOutside & 4){

//point lies below ymin

y = ymin;

x = x1 + (x2-x1)\*(ymin-y1)/(y2-y1);

}

else if(pointOutside & 8){

//point lies above ymax

y = ymax;

x = x1 + (x2-x1)\*(ymax-y1)/(y2-y1);

}

//update the new point

if(pointOutside == codeStart){

x1 = x; y1 = y;

codeStart = find\_region(x1,y1);

}

else{

x2 = x; y2 = y;

codeEnd = find\_region(x2,y2);

}

}

}while(flag==false);

if(!reject)

clipped.push\_back({x1,y1,x2,y2});

}

void LineClipping::clipLine(){

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

clipCurrLine(lines[i][0],lines[i][1],lines[i][2],lines[i][3]);

}

cout<<"\nClipped lines :";

cout<<"\n==================\n";

display(clipped);

}

void LineClipping::display(vector<vector<int>> v){

int size = v.size();

for(int i=0;i<size;i++){

cout<<"("<<v[i][0]<<","<<v[i][1]<<") <--> ("<<v[i][2]<<","<<v[i][3]<<")"<<endl;

}

}

int main(){

LineClipping obj;

obj.input();

obj.clipLine();

return 0;

}

/\*

Enter Window info :

xmax : 10

ymax : 8

xmin : 4

ymin : 4

Enter number of lines :3

Enter end point of line 1 (x1 y1 x2 y2) : 5 5 7 7

Enter end point of line 2 (x1 y1 x2 y2) : 7 9 11 4

Enter end point of line 3 (x1 y1 x2 y2) : 1 5 4 1

Original lines :

==================

(5,5) <--> (7,7)

(7,9) <--> (11,4)

(1,5) <--> (4,1)

Clipped lines :

==================

(5,5) <--> (7,7)

(7,8) <--> (10,5)

--------------------------------

Process exited after 16.71 seconds with return value 0

Press any key to continue . . .

\*/