**/\*WAP to implement Cohen Sutherland Algorithm in Cpp.\*/**

**#include <iostream>**

**#include <cmath>**

**#include <graphics.h>**

**#define pi 3.141592654**

**using namespace std;**

**int i,plot\_count;**

**float m,x\_1,y\_1,x\_2,y\_2,xe1,ye1,xe2,ye2;**

**int b1,b2,b3,b4,b5,b6,b7,b8;**

**void create\_graphics()**

**{**

**initwindow(1366,768);**

**setlinestyle(0,0,1);**

**for(i=0; i<=1366; i++)**

**{**

**line(0,i,1366,i);**

**}**

**setcolor(LIGHTBLUE);**

**line(0,384,1365,384);**

**line(683,0,683,1365);**

**setcolor(LIGHTGRAY);**

**for(i=633; 0<=i; i=i-50)//633=683-50**

**{**

**line(i,0,i,768);**

**}**

**for(i=733; i<=1366; i=i+50)//733=683+50**

**{**

**line(i,0,i,768);**

**}**

**for(i=334; 0<=i; i=i-50)//334=384-50**

**{**

**line(0,i,1366,i);**

**}**

**for(i=434; i<=1365; i=i+50)//434=384+50**

**{**

**line(0,i,1365,i);**

**}**

**setcolor(BLUE);**

**setlinestyle(0,0,3);**

**line(0,183,1366,183);//horizontal +ve**

**line(0,583,1366,583);//horizontal -ve**

**line(982,0,982,763);//vertcal +ve**

**line(382,0,382,763);//vertcal -ve**

**}**

**void sutherland()**

**{**

**if (x\_1<-6) b1=1;**

**else b1=0;**

**if (6<x\_1) b2=1;**

**else b2=0;**

**if (y\_1<-4) b3=1;**

**else b3=0;**

**if (4<y\_1) b4=1;**

**else b4=0;**

**if (x\_2<-6) b5=1;**

**else b5=0;**

**if (6<x\_2) b6=1;**

**else b6=0;**

**if (y\_2<-4) b7=1;**

**else b7=0;**

**if (4<y\_2) b8=1;**

**else b8=0;**

**if (b1==0 && b2==0 && b3==0 && b4==0 && b5==0 && b6==0 && b7==0 && b8==0)**

**{**

**setcolor(GREEN);**

**line(682+x\_1\*50,383-50\*y\_1,682+x\_2\*50,383-50\*y\_2);**

**}**

**else if (b1\*b5!=0 || b2\*b6!=0 || b3\*b7!=0 || b4\*b8!=0)**

**{**

**setcolor(RED);**

**line(682+x\_1\*50,383-50\*y\_1,682+x\_2\*50,383-50\*y\_2);**

**}**

**else**

**{**

**m=(y\_2-y\_1)/(x\_2-x\_1);**

**setcolor(RED);**

**line(682+x\_1\*50,383-50\*y\_1,682+x\_2\*50,383-50\*y\_2);**

**if (b1==1)**

**{**

**xe1=-6;**

**ye1=y\_1+m\*(xe1-x\_1);**

**}**

**if (b2==1)**

**{**

**xe1=6;**

**ye1=y\_1+m\*(xe1-x\_1);**

**}**

**if (b3==1)**

**{**

**ye1=-4;**

**xe1=x\_1+(ye1-y\_1)/m;**

**}**

**if (b4==1)**

**{**

**ye1=4;**

**xe1=x\_1+(ye1-y\_1)/m;**

**}**

**if (b5==1)**

**{**

**xe2=-6;**

**ye2=y\_2+m\*(xe2-x\_2);**

**}**

**if (b6==1)**

**{**

**xe2=6;**

**ye2=y\_2+m\*(xe2-x\_2);**

**}**

**if (b7==1)**

**{**

**ye2=-4;**

**xe2=x\_2+(ye2-y\_2)/m;**

**}**

**if (b8==1)**

**{**

**ye2=4;**

**xe2=x\_2+(ye2-y\_2)/m;**

**}**

**setcolor(CYAN);**

**line(682+xe1\*50,383-50\*ye1,682+xe2\*50,383-50\*ye2);**

**cout<<endl<<" xe1 = "<<xe1<<" ye1 = "<<ye1<<" x// C++ program to implement Cohen Sutherland algorithm**

**// for line clipping.**

**// including libraries**

**#include <bits/stdc++.h>**

**#include <graphics.h>**

**using namespace std;**

**// Global Variables**

**int xmin, xmax, ymin, ymax;**

**// Lines where co-ordinates are (x1, y1) and (x2, y2)**

**struct lines**

**{**

**int x1, y1, x2, y2;**

**};**

**// This will return the sign required.**

**int sign(int x)**

**{**

**if (x > 0)**

**return 1;**

**else**

**return 0;**

**}**

**// CohenSutherLand LineClipping Algorith As Described in theory.**

**// This will clip the lines as per window boundaries.**

**void clip(struct lines mylines)**

**{**

**// arrays will store bits**

**// Here bits implies initial Point whereas bite implies end points**

**int bits[4], bite[4], i, var;**

**// setting color of graphics to be RED**

**setcolor(RED);**

**// Finding Bits**

**bits[0] = sign(xmin - mylines.x1);**

**bite[0] = sign(xmin - mylines.x2);**

**bits[1] = sign(mylines.x1 - xmax);**

**bite[1] = sign(mylines.x2 - xmax);**

**bits[2] = sign(ymin - mylines.y1);**

**bite[2] = sign(ymin - mylines.y2);**

**bits[3] = sign(mylines.y1 - ymax);**

**bite[3] = sign(mylines.y2 - ymax);**

**// initial will used for initial coordinates and end for final**

**string initial = "", end = "", temp = "";**

**// convert bits to string**

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

**{**

**if (bits[i] == 0)**

**initial += '0';**

**else**

**initial += '1';**

**}**

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

**{**

**if (bite[i] == 0)**

**end += '0';**

**else**

**end += '1';**

**}**

**// finding slope of line y=mx+c as (y-y1)=m(x-x1)+c**

**// where m is slope m=dy/dx;**

**float m = (mylines.y2 - mylines.y1) / (float)(mylines.x2 - mylines.x1);**

**float c = mylines.y1 - m \* mylines.x1;**

**// if both points are inside the Accept the line and draw**

**if (initial == end && end == "0000")**

**{**

**// inbuild function to draw the line from(x1, y1) to (x2, y2)**

**line(mylines.x1, mylines.y1, mylines.x2, mylines.y2);**

**return;**

**}**

**// this will contain cases where line maybe totally outside for partially inside**

**else**

**{**

**// taking bitwise end of every value**

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

**{**

**int val = (bits[i] & bite[i]);**

**if (val == 0)**

**temp += '0';**

**else**

**temp += '1';**

**}**

**// as per algo if AND is not 0000 means line is completely outside hene draw nothing and retrurn**

**if (temp != "0000")**

**return;**

**// Here contain cases of partial inside or outside**

**// So check for every boundary one by one**

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

**{**

**// if boths bit are same hence we cannot find any intersection with boundary so continue**

**if (bits[i] == bite[i])**

**continue;**

**// Otherwise there exist a intersection**

**// Case when initial point is in left xmin**

**if (i == 0 && bits[i] == 1)**

**{**

**var = round(m \* xmin + c);**

**mylines.y1 = var;**

**mylines.x1 = xmin;**

**}**

**// Case when final point is in left xmin**

**if (i == 0 && bite[i] == 1)**

**{**

**var = round(m \* xmin + c);**

**mylines.y2 = var;**

**mylines.x2 = xmin;**

**}**

**// Case when initial point is in right of xmax**

**if (i == 1 && bits[i] == 1)**

**{**

**var = round(m \* xmax + c);**

**mylines.y1 = var;**

**mylines.x1 = xmax;**

**}**

**// Case when final point is in right of xmax**

**if (i == 1 && bite[i] == 1)**

**{**

**var = round(m \* xmax + c);**

**mylines.y2 = var;**

**mylines.x2 = xmax;**

**}**

**// Case when initial point is in top of ymin**

**if (i == 2 && bits[i] == 1)**

**{**

**var = round((float)(ymin - c) / m);**

**mylines.y1 = ymin;**

**mylines.x1 = var;**

**}**

**// Case when final point is in top of ymin**

**if (i == 2 && bite[i] == 1)**

**{**

**var = round((float)(ymin - c) / m);**

**mylines.y2 = ymin;**

**mylines.x2 = var;**

**}**

**// Case when initial point is in bottom of ymax**

**if (i == 3 && bits[i] == 1)**

**{**

**var = round((float)(ymax - c) / m);**

**mylines.y1 = ymax;**

**mylines.x1 = var;**

**}**

**// Case when final point is in bottom of ymax**

**if (i == 3 && bite[i] == 1)**

**{**

**var = round((float)(ymax - c) / m);**

**mylines.y2 = ymax;**

**mylines.x2 = var;**

**}**

**// Updating Bits at every point**

**bits[0] = sign(xmin - mylines.x1);**

**bite[0] = sign(xmin - mylines.x2);**

**bits[1] = sign(mylines.x1 - xmax);**

**bite[1] = sign(mylines.x2 - xmax);**

**bits[2] = sign(ymin - mylines.y1);**

**bite[2] = sign(ymin - mylines.y2);**

**bits[3] = sign(mylines.y1 - ymax);**

**bite[3] = sign(mylines.y2 - ymax);**

**} // end of for loop**

**// Initialize initial and end to NULL**

**initial = "", end = "";**

**// Updating strings again by bit**

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

**{**

**if (bits[i] == 0)**

**initial += '0';**

**else**

**initial += '1';**

**}**

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

**{**

**if (bite[i] == 0)**

**end += '0';**

**else**

**end += '1';**

**}**

**// If now both points lie inside or on boundary then simply draw the updated line**

**if (initial == end && end == "0000")**

**{**

**line(mylines.x1, mylines.y1, mylines.x2, mylines.y2);**

**return;**

**}**

**// else line was completely outside hence rejected**

**else**

**return;**

**}**

**}**

**// Driver Function**

**int main()**

**{**

**int gd = DETECT, gm;**

**// Setting values of Clipping window**

**xmin = 40;**

**xmax = 100;**

**ymin = 40;**

**ymax = 80;**

**// initialize the graph**

**initgraph(&gd, &gm, NULL);**

**// Drawing Window using Lines**

**line(xmin, ymin, xmax, ymin);**

**line(xmax, ymin, xmax, ymax);**

**line(xmax, ymax, xmin, ymax);**

**line(xmin, ymax, xmin, ymin);**

**// Assume 4 lines to be clipped**

**struct lines mylines[4];**

**// Setting the coordinated of 4 lines**

**mylines[0].x1 = 30;**

**mylines[0].y1 = 65;**

**mylines[0].x2 = 55;**

**mylines[0].y2 = 30;**

**mylines[1].x1 = 60;**

**mylines[1].y1 = 20;**

**mylines[1].x2 = 100;**

**mylines[1].y2 = 90;**

**mylines[2].x1 = 60;**

**mylines[2].y1 = 100;**

**mylines[2].x2 = 80;**

**mylines[2].y2 = 70;**

**mylines[3].x1 = 85;**

**mylines[3].y1 = 50;**

**mylines[3].x2 = 120;**

**mylines[3].y2 = 75;**

**// Drawing Initial Lines without clipping**

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

**{**

**line(mylines[i].x1, mylines[i].y1,**

**mylines[i].x2, mylines[i].y2);**

**delay(1000);**

**}**

**// Drawing clipped Line**

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

**{**

**// Calling clip() which in term clip the line as per window and draw it**

**clip(mylines[i]);**

**delay(1000);**

**}**

**delay(4000);**

**getch();**

**// For Closing the graph.**

**closegraph();**

**return 0;**

**}**

**e2 = "<<xe2<<" ye2 = "<<ye2;**

**cout<<endl<<" b1 = "<<b1<<" b2 = "<<b2<<" b3 = "<<b3<<" b4 = "<<b4;**

**cout<<endl<<" b5 = "<<b5<<" b6 = "<<b6<<" b7 = "<<b7<<" b8 = "<<b8;**

**}**

**}**

**int main()**

**{**

**while(1)**

**{**

**cout<<"\n\n\n\t\t\t\t\tCohen Sutherland Algorithm";**

**cout<<"\n\n\n\t\t\t\t(-13.66,0),(13.66,0),(0,-7.66),(0,7.66) ";**

**cout<<"\n\n\n\t\t\t\t\t(-6,4),(-6,-4),(6,-4),(6,4)";**

**cout<<"\n\n\n\t\t Enter coordinate of two points: ";**

**cout<<"\n\n\t\t Enter x\_1: ";**

**cin>>x\_1;**

**cout<<"\n\n\t\t Enter y\_1: ";**

**cin>>y\_1;**

**cout<<"\n\n\t\t Enter x\_2: ";**

**cin>>x\_2;**

**cout<<"\n\n\t\t Enter y\_2: ";**

**cin>>y\_2;**

**create\_graphics();**

**sutherland();**

**getch();**

**closegraph();**

**}**

**return 0;**

**}**

**/\*WAP to implement Cohen Sutherland Algorithm in Cpp.\*/**

**#include<GL/gl.h>**

**#include<GL/glu.h>**

**#include<GL/glut.h>**

**#include<iostream>**

**#include<vector>**

**#include<math.h>**

**#define PI 3.14159265358979323846**

**using namespace std;**

**void display(); //display function**

**void reshape(int,int); //reshape the viewport**

**void timer(int); //for displaying no of frames in a sec**

**void getinfo(); //info from user**

**void getdata();//2 points**

**float xmin,xmax,ymin,ymax;**

**struct point**

**{**

**float x;**

**float y;**

**int code=0;**

**};**

**void giveCode(point&);**

**void clipping(point,point);**

**int num;//number of lines**

**point p1,p2; //2 points for line**

**point sp1,sp2; // the point that lies in viewport**

**vector<point> orgPoint;**

**vector<point> clipPoint;**

**void init()**

**{**

**glClearColor(0.1,0.1,0.1,1.0); //background color**

**}**

**int main(int argc, char\*\* argv)**

**{**

**getinfo();**

**glutInit(&argc,argv);**

**glutInitDisplayMode(GLUT\_RGB|GLUT\_DOUBLE);**

**glutInitWindowSize(1000,700);**

**glutInitWindowPosition(0,0);**

**glutCreateWindow("Clipping");**

**glutReshapeFunc(reshape);**

**glutDisplayFunc(display);**

**glutSetKeyRepeat(GLUT\_KEY\_REPEAT\_OFF);**

**glutTimerFunc(0,timer,0);**

**init();**

**glutMainLoop();**

**return 0;**

**}**

**void display()**

**{**

**glClear(GL\_COLOR\_BUFFER\_BIT);**

**glLoadIdentity();**

**glColor3f(1,0,0);**

**glBegin(GL\_LINES);**

**glVertex2f(xmin,ymin);**

**glVertex2f(xmax,ymin);**

**glVertex2f(xmax,ymin);**

**glVertex2f(xmax,ymax);**

**glVertex2f(xmax,ymax);**

**glVertex2f(xmin,ymax);**

**glVertex2f(xmin,ymax);**

**glVertex2f(xmin,ymin);**

**glEnd();**

**glColor3f(0,0,1);**

**glBegin(GL\_LINES);**

**//glVertex2f(p1.x,p1.y);**

**//glVertex2f(p2.x,p2.y);**

**for(int i=0; i<orgPoint.size(); i++)**

**{**

**glVertex2f(orgPoint.at(i).x,orgPoint.at(i).y);**

**}**

**glEnd();**

**glColor3f(0,1,0);**

**glBegin(GL\_LINES);**

**//glVertex2f(sp1.x,sp1.y);**

**//glVertex2f(sp2.x,sp2.y);**

**for(int i=0; i<clipPoint.size(); i++)**

**{**

**glVertex2f(clipPoint.at(i).x,clipPoint.at(i).y);**

**}**

**glEnd();**

**glutSwapBuffers();**

**}**

**void reshape(int w,int h)**

**{**

**glViewport(0,0,w,h);**

**glMatrixMode(GL\_PROJECTION);**

**glLoadIdentity();**

**//gluOrtho2D(-250,250,-250,250);**

**gluOrtho2D(0,1000,0,700);**

**glMatrixMode(GL\_MODELVIEW);**

**}**

**void timer(int)**

**{**

**glutPostRedisplay();**

**glutTimerFunc(1000/30,timer,0);**

**}**

**void getinfo()**

**{**

**cout<<"\n\n";**

**cout<<"\tEnter following:\n";**

**cout<<"\t\tXmin: ";**

**cin>>xmin;**

**cout<<"\t\tXmax: ";**

**cin>>xmax;**

**cout<<"\t\tYmin: ";**

**cin>>ymin;**

**cout<<"\t\tYmax: ";**

**cin>>ymax;**

**cout<<"\n\t\tnumber of lines: ";**

**cin>>num;**

**for(int i=0; i<num; i++)**

**{**

**cout<<"\n LINE "<<i+1<<"#::"<<endl;**

**getdata();**

**}**

**}**

**void getdata()**

**{**

**p1.code=0;**

**p2.code=0;**

**cout<<"\n\t Enter the line details:\n";**

**cout<<"\t\t X1= ";**

**cin>>p1.x;**

**cout<<"\t\t Y1= ";**

**cin>>p1.y;**

**cout<<"\t\t X2= ";**

**cin>>p2.x;**

**cout<<"\t\t Y2= ";**

**cin>>p2.y;**

**giveCode(p1);**

**giveCode(p2);**

**orgPoint.push\_back(p1);**

**orgPoint.push\_back(p2);**

**clipping(p1,p2);**

**}**

**void giveCode(point &A)**

**{**

**A.code=0;**

**if(A.x<xmin)**

**A.code+=1; //1=0001**

**if(A.x>xmax)**

**A.code+=2; //2=0010**

**if(A.y<ymin)**

**A.code+=4; //4=0100**

**if(A.y>ymax)**

**A.code+=8; //8=1000**

**}**

**// clipping code**

**void clipping(point A, point B)**

**{**

**float m=(A.y-B.y)/(A.x-B.x);**

**if((A.code|B.code) == false )**

**{**

**cout<<"This line lies completely inside."<<endl;**

**sp1.x=A.x;**

**sp1.y=A.y;**

**sp2.x=B.x;**

**sp2.y=B.y;**

**clipPoint.push\_back(sp1);**

**clipPoint.push\_back(sp2);**

**return;**

**}**

**else if(A.code & B.code)**

**{**

**cout<<"Line lies completely outside."<<endl;**

**return;**

**}**

**else**

**{**

**point check;**

**if(A.code==0)**

**{**

**check.x=B.x;**

**check.y=B.y;**

**check.code=B.code;**

**}**

**else**

**{**

**check.x=A.x;**

**check.y=A.y;**

**check.code=A.code;**

**}**

**if(check.code&1)**

**{**

**check.y=check.y+ m\*(xmin-check.x);**

**check.x=xmin;**

**}**

**if(check.code&2)**

**{**

**check.y=check.y+ m\*(xmax-check.x);**

**check.x=xmax;**

**}**

**if(check.code&4)**

**{**

**check.x=check.x+(ymin-check.y)/m;**

**check.y=ymin;**

**}**

**if(check.code&8)**

**{**

**check.x=check.x+ (ymax-check.y)/m;**

**check.y=ymax;**

**}**

**if(A.code==0)**

**{**

**B.x=check.x;**

**B.y=check.y;**

**B.code=check.code;**

**}**

**else**

**{**

**A.x=check.x;**

**A.y=check.y;**

**A.code=check.code;**

**}**

**sp1.x=A.x;**

**sp1.y=A.y;**

**sp2.x=B.x;**

**sp2.y=B.y;**

**giveCode(sp1);**

**giveCode(sp2);**

**cout<<"The New points: "<<endl;**

**cout<<"("<<sp1.x<<","<<sp1.y<<")\t";**

**cout<<"("<<sp2.x<<" "<<sp2.y<<")"<<endl;**

**clipping(sp1,sp2);**

**}**

**return;**

**}**