---------------------------------------------------------------------- Write C++/Java program to implement Cohen-Sutherland line clipping algorithm for given window. Draw line using mouse interfacing to draw polygon

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

#include "lineclipping.h"

#include "ui\_lineclipping.h"

#include<QPainter>

#include "QMouseEvent"

lineclipping::lineclipping(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::lineclipping)

{

ui->setupUi(this);

}

lineclipping::~lineclipping()

{

delete ui;

}

void lineclipping::paintEvent(QPaintEvent \*)

{

if(flag==2)

{

x\_max=ui->xmax->toPlainText().toInt();

y\_max=ui->ymax->toPlainText().toInt();

x\_min=ui->xmin->toPlainText().toInt();

y\_min=ui->ymin->toPlainText().toInt();

draw(x\_max,y\_max,x\_max,y\_min);

draw(x\_min,y\_min,x\_max,y\_min);

draw(x\_min,y\_min,x\_min,y\_max);

draw(x\_min,y\_max,x\_max,y\_max);

draw(x[0],y[0],x[1],y[1]);

}

if(flag==3)

{

x\_max=ui->xmax->toPlainText().toInt();

y\_max=ui->ymax->toPlainText().toInt();

x\_min=ui->xmin->toPlainText().toInt();

y\_min=ui->ymin->toPlainText().toInt();

draw(x\_max,y\_max,x\_max,y\_min);

draw(x\_min,y\_min,x\_max,y\_min);

draw(x\_min,y\_min,x\_min,y\_max);

draw(x\_min,y\_max,x\_max,y\_max);

clip();

}

if(flag==4 || flag==0)

{

x\_max=ui->xmax->toPlainText().toInt();

y\_max=ui->ymax->toPlainText().toInt();

x\_min=ui->xmin->toPlainText().toInt();

y\_min=ui->ymin->toPlainText().toInt();

draw(x\_max,y\_max,x\_max,y\_min);

draw(x\_min,y\_min,x\_max,y\_min);

draw(x\_min,y\_min,x\_min,y\_max);

draw(x\_min,y\_max,x\_max,y\_max);

}

}

void lineclipping::mousePressEvent(QMouseEvent \*)

{

if(i<n)

{

x[i]=QCursor::pos().x();

y[i]=QCursor::pos().y();

i++;

}

}

void lineclipping :: draw(int a, int b ,int c,int d)

{

QPainter painter(this);

i=flag=0;

painter.drawLine(a,b,c,d);

//painter.setPen(QPen((Qt::blue),1));

//painter.drawLine(0,480,1280,480);

//painter.drawLine(640,0,640,960);

}

void lineclipping::clip ()

{

/\*x\_max=ui->xmax->toPlainText().toInt();

y\_max=ui->ymax->toPlainText().toInt();

x\_min=ui->xmin->toPlainText().toInt();

y\_min=ui->ymin->toPlainText().toInt();\*/

cohenSutherlandClip(x[0],y[0],x[1],y[1]);

}

// Function to compute region code for a point(x, y)

int lineclipping ::computeCode(double x, double y)

{

// initialized as being inside

int code = INSIDE;

if (x < x\_min) // to the left of rectangle

code |= LEFT;

else if (x > x\_max) // to the right of rectangle

code |= RIGHT;

if (y < y\_min) // below the rectangle

code |= BOTTOM;

else if (y > y\_max) // above the rectangle

code |= TOP;

return code;

}

// Implementing Cohen-Sutherland algorithm

// Clipping a line from P1 = (x2, y2) to P2 = (x2, y2)

void lineclipping ::cohenSutherlandClip(double x1, double y1,

double x2, double y2)

{

// Compute region codes for P1, P2

int code1 = computeCode(x1, y1);

int code2 = computeCode(x2, y2);

// Initialize line as outside the rectangular window

bool accept = false;

while (true)

{

if ((code1 == 0) && (code2 == 0))

{

// If both endpoints lie within rectangle

accept = true;

break;

}

else if (code1 & code2)

{

// If both endpoints are outside rectangle,

// in same region

break;

}

else

{

// Some segment of line lies within the

// rectangle

int code\_out;

double x, y;

// At least one endpoint is outside the

// rectangle, pick it.

if (code1 != 0)

code\_out = code1;

else

code\_out = code2;

// Find intersection point;

// using formulas y = y1 + slope \* (x - x1),

// x = x1 + (1 / slope) \* (y - y1)

if (code\_out & TOP)

{

// point is above the clip rectangle

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

y = y\_max;

}

else if (code\_out & BOTTOM)

{

// point is below the rectangle

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

y = y\_min;

}

else if (code\_out & RIGHT)

{

// point is to the right of rectangle

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

x = x\_max;

}

else if (code\_out & LEFT)

{

// point is to the left of rectangle

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

x = x\_min;

}

if (code\_out == code1)

{

x1 = x;

y1 = y;

code1 = computeCode(x1, y1);

}

else

{

x2 = x;

y2 = y;

code2 = computeCode(x2, y2);

}

accept=true;

}

}

if (accept)

{

draw(x1,y1,x2,y2);

}

}

void lineclipping::on\_select\_clicked()

{

n=2;

flag=0;

update();

}

void lineclipping::on\_draw\_clicked()

{

flag=2;

update();

}

void lineclipping::on\_clip\_clicked()

{

flag=3;

update();

}

void lineclipping::on\_ShowWindow\_clicked()

{

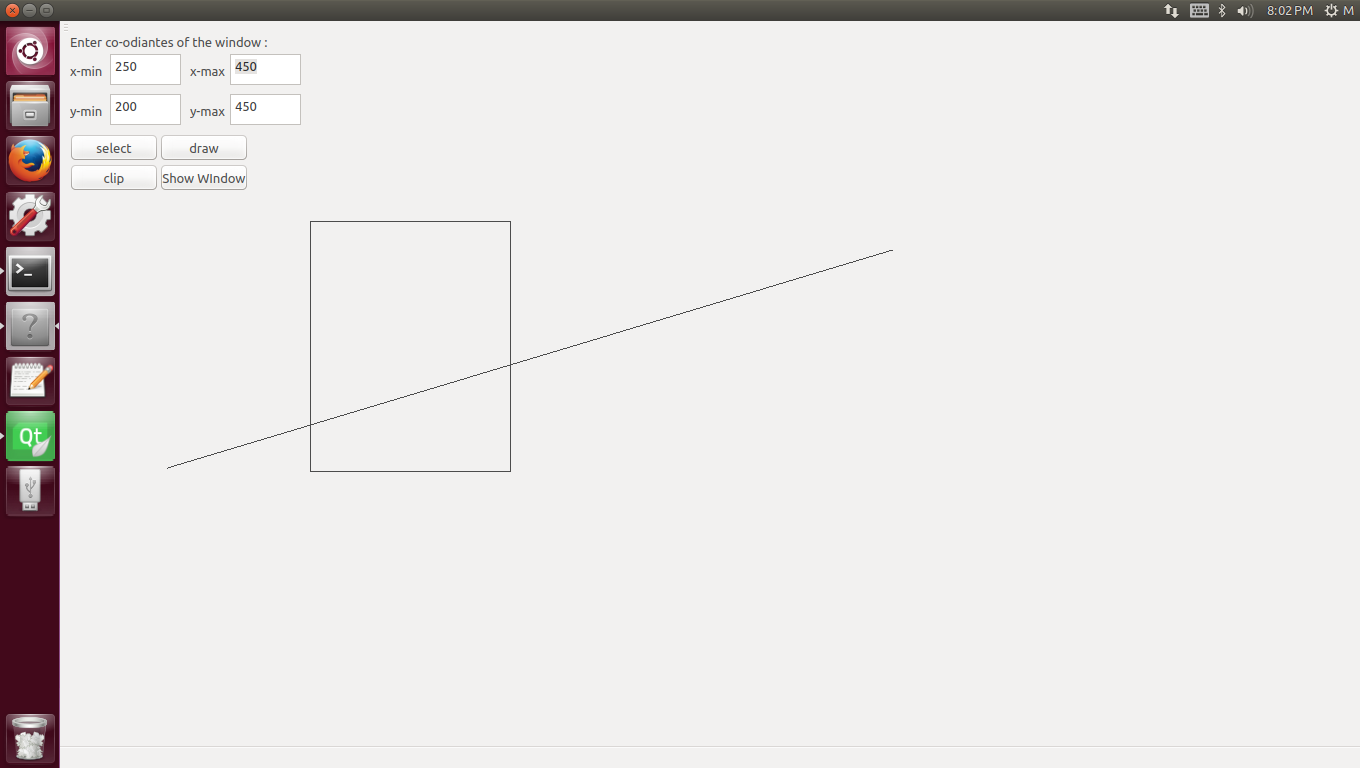
flag=4;

update();

}

**OUTPUT:**

Line Before Clipping :



Line After being Clipped : 