ASSIGNMENT-5 & 6

NAME-ANUBHAV ANAND

ENROLLMENT NUMBER-2020CSB102

Subject-Assignment 5 and 6 of Computer Graphics

G-Suite Id-2020CSB102.anubhav@students.iiests.ac.in

ASSIGNMENT-5

1. Implement Sutherland-Cohen line clipping algorithm.

Ans-Code-

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Clipping extends Applet implements ActionListener,
MouseWheelListener {

  int originX, originY;
  int height, width;
  int gap = 40;
  int temp = 0;
  static final int INSIDE = 0; // 0000
  static final int LEFT = 1; // 0001
  static final int RIGHT = 2; // 0010
  static final int BOTTOM = 4; // 0100
  static final int TOP = 8; // 1000

  static final int x_max = 100;
  static final int y_max = 100;
  static final int x_min = -100;
  static final int y_min = -100;

  // static final int x_max = 25;
  // static final int y_max = 25;
  // static final int x_min = -25;
  // static final int y_min = -25;

  Button b1 = new Button(" + ");
  Button b2 = new Button(" - ");
  Button b3 = new Button(" Clip ");

  public void init() {
    setBackground(Color.black);
    b1.setBackground(Color.red);
    b2.setBackground(Color.green);
    b3.setBackground(Color.GREEN);
    add(b1);
```

```
    add(b2);
    add(b3);
    addMouseWheelListener(this);
    b1.addActionListener(this);
    b2.addActionListener(this);
    b3.addActionListener(this);
  }



  //Function for plotting points
  public void plotPoint(Graphics g, int x,int y ,Color c){
    int originX = (getX() + getWidth()) / 2;
    int originY = (getY() + getHeight()) / 2;
    g.setColor(c);
    g.fillRect(originX+(gap*x)-(gap/4), originY-(gap*y)-(gap/4),3*gap ,3*gap
);
}

  //function to make grid
  public void makeGrid(Graphics g)
  {
      if(gap<=0|| gap>getHeight())
          return ;
      int originX = (getX() + getWidth()) / 2;
      int originY = (getY() + getHeight()) / 2;
      g.setColor(Color.red);
      g.drawLine(originX, originY - getHeight() / 2, originX, originY +
getHeight() / 2);
      g.drawLine(originX - getWidth() / 2, originY, originX + getWidth() / 2,
originY);
      g.setColor(Color.black);
      for (int x = gap; x <= getWidth(); x += gap) {

          g.drawLine(originX + x, 0, originX + x, getHeight());
          g.drawLine(originX - x, 0, originX - x, getHeight());

      }
      for (int y = gap; y <= getHeight(); y += gap) {

          g.drawLine(0, originY + y, getWidth(), originY + y);
          g.drawLine(0, originY - y, getWidth(), originY - y);
      }
  }


  //Function for the buttons
  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == b1) zoom(10);
```

```
    if (e.getSource() == b2) zoom(-10);
    if (e.getSource() == b3) {
      if (temp == 0) temp = 1; else temp = 0;
      repaint();
    }
  }

  //Function for the mousewheel
  public void mouseWheelMoved(MouseWheelEvent e) {
    int z = e.getWheelRotation();
    zoom(z);
  }

  //Function for the zoom in feature
  public void zoom(int i)
  {
      if(i>0)
          gap+=gap/10+1;
      else if(i<0)
          gap-=gap/10+1;
      repaint();

  }
  //function to compute code
  static int computeCode(int x, int y) {
    // initialized as being inside
    int code = INSIDE;

    if (x < x_min) code |= LEFT; else if ( // to the left of rectangle
      x > x_max
    ) code |= RIGHT; // to the right of rectangle
    if (y < y_min) code |= BOTTOM; else if ( // below the rectangle
      y > y_max
    ) code |= TOP; // above the rectangle

    return code;
  }

  public void cohenSutherlandClip(Graphics g, int x1, int y1, int x2, int y2)
{
    // Compute region codes for P1, P2
    int code1 = computeCode(x1, y1);
    int code2 = computeCode(x2, y2);

    // Initialize line as outside the rectangular window
    boolean accept = false;

    while (true) {
```

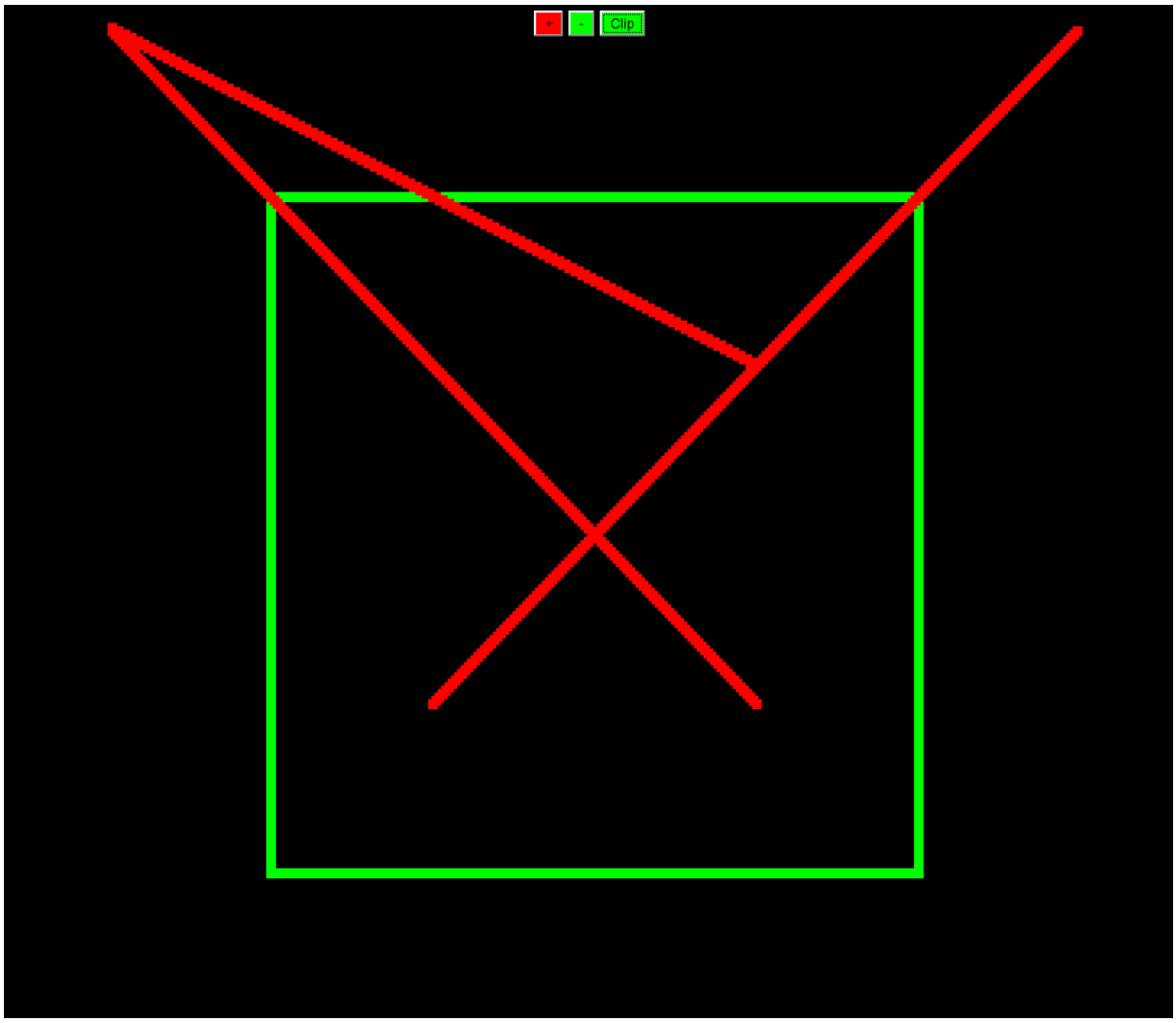
```
      if ((code1 == 0) && (code2 == 0)) {
        // If both endpoints lie within rectangle
        accept = true;
        break;
      } else if ((code1 & code2) != 0) {
        // If both endpoints are outside rectangle,
        // in same region
        break;
      } else {
        // Some segment of line lies within the
        // rectangle
        int code_out;
        int x = 0, y = 0;

        // 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) != 0) {
          // point is above the clip rectangle
          x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1);
          y = y_max;
        } else if ((code_out & BOTTOM) != 0) {
          // point is below the rectangle
          x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1);
          y = y_min;
        } else if ((code_out & RIGHT) != 0) {
          // point is to the right of rectangle
          y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1);
          x = x_max;
        } else if ((code_out & LEFT) != 0) {
          // point is to the left of rectangle
          y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1);
          x = x_min;
        }

        // Now intersection point x, y is found
        // We replace point outside rectangle
        // by intersection point
        if (code_out == code1) {
          x1 = x;
          y1 = y;
          code1 = computeCode(x1, y1);
        } else {
          x2 = x;
```

```
        y2 = y;
        code2 = computeCode(x2, y2);
      }
    }
  }
  if (accept) {
    DDALine(g, x1, y1, x2, y2,Color.RED);
  } else System.out.println("Line rejected");
}

void DDALine(Graphics g, int x0, int y0, int x1, int y1,Color c) {
  int dx = (x1 - x0);
  int dy = (y1 - y0);

  int step;
  if (Math.abs(dx) > Math.abs(dy)) {
    step = Math.abs(dx);
  } else {
    step = Math.abs(dy);
  }

  float x_incr = (float) dx / step;
  float y_incr = (float) dy / step;
  float x = (float) x0;
  float y = (float) y0;

  for (int i = 0; i < step; i++) {
    plotPoint(g, Math.round(x), Math.round(y), c);
    x += x_incr;
    y += y_incr;
  }
}
//paint function
public void paint(Graphics g) {
  g.setColor(Color.white);
  height = getHeight();
  width = getWidth();
  originX = (getX() + width) / 2;
  originY = (getY() + height) / 2;
  DDALine(g, x_min, y_min, x_max, y_min,Color.GREEN);
  DDALine(g, x_min, y_max, x_max, y_max,Color.GREEN);
  DDALine(g, x_min, y_min, x_min, y_max,Color.GREEN);
  DDALine(g, x_max, y_min, x_max, y_max,Color.GREEN);
  // makeGrid(g);
  if (temp == 0) {
 //Simple Lines

  DDALine(g, -50, -50, 150, 150,Color.red);
```

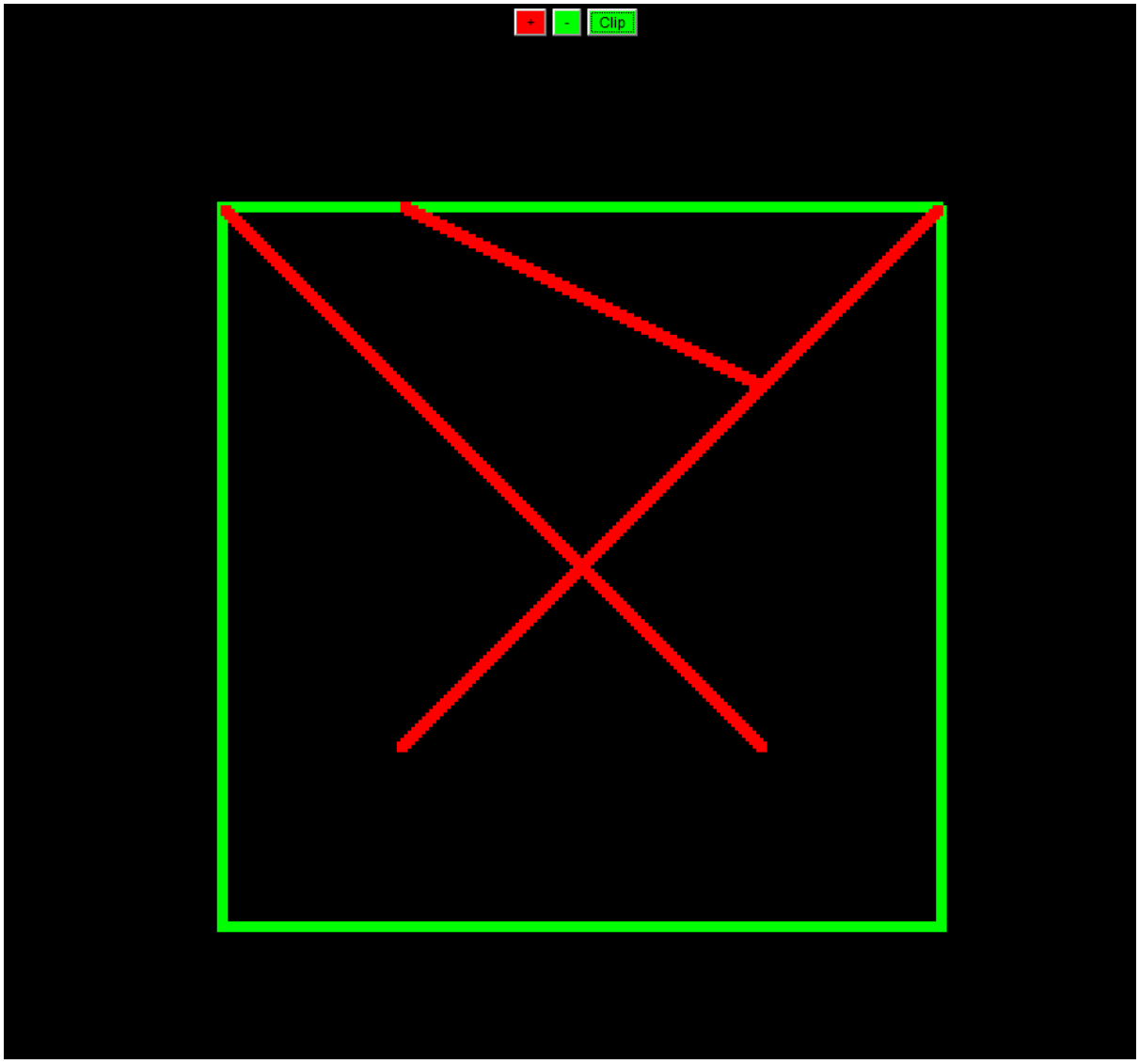
```
    DDALine(g, 50, 50, -150, 150,Color.red);
    DDALine(g, 50, -50, -150, 150,Color.red);
    } else {
        //Simple Lines

    cohenSutherlandClip(g, -50, -50, 150, 150);
    cohenSutherlandClip(g, 50, 50, -150, 150);
    cohenSutherlandClip(g, 50, -50, -150, 150);


    }
  }
}
```

For Random Lines When we Applied Clipping-

Output(Before Clipping)-



Output(After Clipping)-



For Square Code-

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Clipping extends Applet implements ActionListener,
MouseWheelListener {

  int originX, originY;
  int height, width;
  int gap = 40;
  int temp = 0;
  static final int INSIDE = 0; // 0000
  static final int LEFT = 1; // 0001
  static final int RIGHT = 2; // 0010
  static final int BOTTOM = 4; // 0100
  static final int TOP = 8; // 1000

  // static final int x_max = 100;
  // static final int y_max = 100;
  // static final int x_min = -100;
  // static final int y_min = -100;

  static final int x_max = 25;
  static final int y_max = 25;
  static final int x_min = -25;
  static final int y_min = -25;

  Button b1 = new Button(" + ");
  Button b2 = new Button(" - ");
  Button b3 = new Button(" Clip ");

  public void init() {
    setBackground(Color.black);
    b1.setBackground(Color.red);
    b2.setBackground(Color.green);
    b3.setBackground(Color.GREEN);
    add(b1);
    add(b2);
    add(b3);
    addMouseWheelListener(this);
    b1.addActionListener(this);
    b2.addActionListener(this);
    b3.addActionListener(this);
  }
```

```
  //Function for plotting points
  public void plotPoint(Graphics g, int x,int y ,Color c){
    int originX = (getX() + getWidth()) / 2;
    int originY = (getY() + getHeight()) / 2;
    g.setColor(c);
    g.fillRect(originX+(gap*x)-(gap/4), originY-(gap*y)-(gap/4),gap ,gap );
}

  //function to make grid
  public void makeGrid(Graphics g)
  {
      if(gap<=0|| gap>getHeight())
          return ;
      int originX = (getX() + getWidth()) / 2;
      int originY = (getY() + getHeight()) / 2;
      g.setColor(Color.red);
      g.drawLine(originX, originY - getHeight() / 2, originX, originY +
getHeight() / 2);
      g.drawLine(originX - getWidth() / 2, originY, originX + getWidth() / 2,
originY);
      g.setColor(Color.black);
      for (int x = gap; x <= getWidth(); x += gap) {

          g.drawLine(originX + x, 0, originX + x, getHeight());
          g.drawLine(originX - x, 0, originX - x, getHeight());

      }
      for (int y = gap; y <= getHeight(); y += gap) {

          g.drawLine(0, originY + y, getWidth(), originY + y);
          g.drawLine(0, originY - y, getWidth(), originY - y);
      }
  }


  //Function for the buttons
  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == b1) zoom(10);
    if (e.getSource() == b2) zoom(-10);
    if (e.getSource() == b3) {
      if (temp == 0) temp = 1; else temp = 0;
      repaint();
    }
  }

  //Function for the mousewheel
  public void mouseWheelMoved(MouseWheelEvent e) {
    int z = e.getWheelRotation();
```

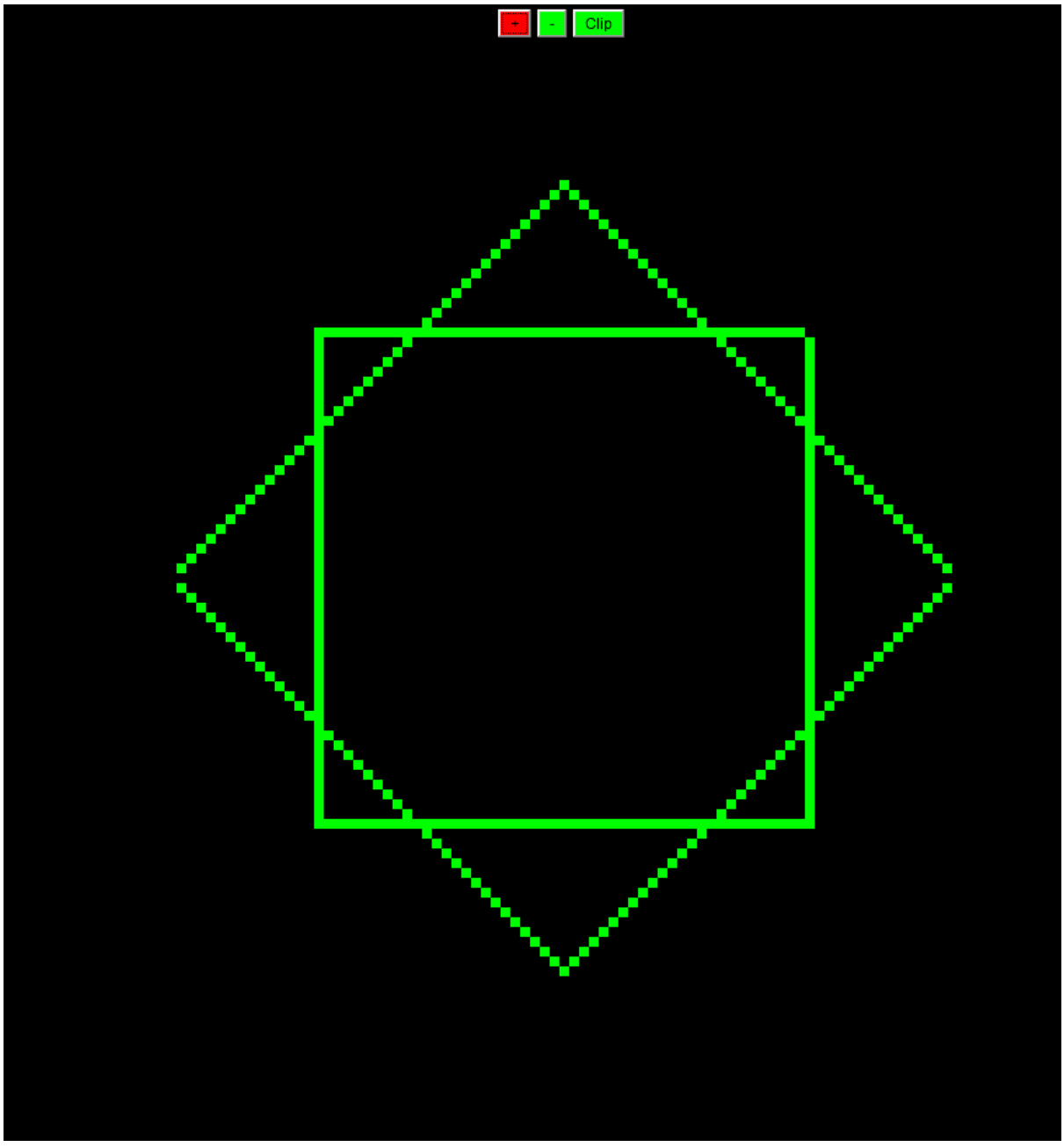
```
    zoom(z);
  }

  //Function for the zoom in feature
  public void zoom(int i)
  {
      if(i>0)
          gap+=gap/10+1;
      else if(i<0)
          gap-=gap/10+1;
      repaint();

  }
  //function to compute code
  static int computeCode(int x, int y) {
    // initialized as being inside
    int code = INSIDE;

    if (x < x_min) code |= LEFT; else if ( // to the left of rectangle
      x > x_max
    ) code |= RIGHT; // to the right of rectangle
    if (y < y_min) code |= BOTTOM; else if ( // below the rectangle
      y > y_max
    ) code |= TOP; // above the rectangle

    return code;
  }

  public void cohenSutherlandClip(Graphics g, int x1, int y1, int x2, int y2)
{
    // Compute region codes for P1, P2
    int code1 = computeCode(x1, y1);
    int code2 = computeCode(x2, y2);

    // Initialize line as outside the rectangular window
    boolean accept = false;

    while (true) {
      if ((code1 == 0) && (code2 == 0)) {
        // If both endpoints lie within rectangle
        accept = true;
        break;
      } else if ((code1 & code2) != 0) {
        // If both endpoints are outside rectangle,
        // in same region
        break;
      } else {
        // Some segment of line lies within the
```

```
        // rectangle
        int code_out;
        int x = 0, y = 0;

        // 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) != 0) {
          // point is above the clip rectangle
          x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1);
          y = y_max;
        } else if ((code_out & BOTTOM) != 0) {
          // point is below the rectangle
          x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1);
          y = y_min;
        } else if ((code_out & RIGHT) != 0) {
          // point is to the right of rectangle
          y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1);
          x = x_max;
        } else if ((code_out & LEFT) != 0) {
          // point is to the left of rectangle
          y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1);
          x = x_min;
        }

        // Now intersection point x, y is found
        // We replace point outside rectangle
        // by intersection point
        if (code_out == code1) {
          x1 = x;
          y1 = y;
          code1 = computeCode(x1, y1);
        } else {
          x2 = x;
          y2 = y;
          code2 = computeCode(x2, y2);
        }
      }
    }
    if (accept) {
      DDALine(g, x1, y1, x2, y2,Color.RED);
    } else System.out.println("Line rejected");
  }
```

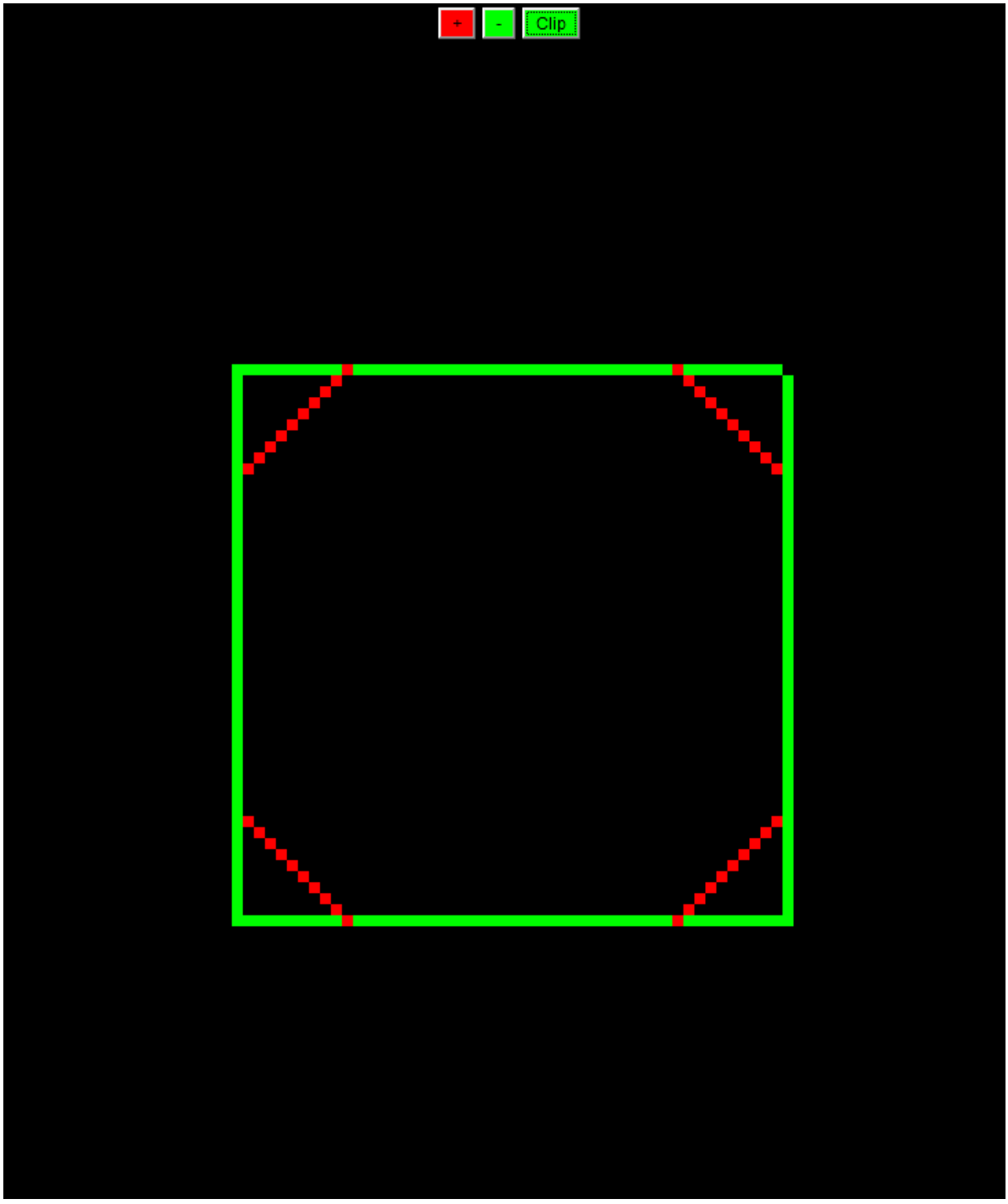
```
void DDALine(Graphics g, int x0, int y0, int x1, int y1,Color c) {
  int dx = (x1 - x0);
  int dy = (y1 - y0);

  int step;
  if (Math.abs(dx) > Math.abs(dy)) {
    step = Math.abs(dx);
  } else {
    step = Math.abs(dy);
  }

  float x_incr = (float) dx / step;
  float y_incr = (float) dy / step;
  float x = (float) x0;
  float y = (float) y0;

  for (int i = 0; i < step; i++) {
    plotPoint(g, Math.round(x), Math.round(y), c);
    x += x_incr;
    y += y_incr;
  }
}
//paint function
public void paint(Graphics g) {
  g.setColor(Color.white);
  height = getHeight();
  width = getWidth();
  originX = (getX() + width) / 2;
  originY = (getY() + height) / 2;
  DDALine(g, x_min, y_min, x_max, y_min,Color.GREEN);
  DDALine(g, x_min, y_max, x_max, y_max,Color.GREEN);
  DDALine(g, x_min, y_min, x_min, y_max,Color.GREEN);
  DDALine(g, x_max, y_min, x_max, y_max,Color.GREEN);
  // makeGrid(g);
  if (temp == 0) {
 //Square
 DDALine(g, 0, -40, -40, 0,Color.GREEN);
 DDALine(g, 0, -40, 40, 0,Color.GREEN);
 DDALine(g, 0, 40, 40, 0,Color.GREEN);
 DDALine(g, 0, 40, -40, 0,Color.GREEN);


  } else {
      //Square
      cohenSutherlandClip(g, 0, -40, -40, 0);
      cohenSutherlandClip(g, 0, -40, 40, 0);
      cohenSutherlandClip(g, 0, 40, 40, 0);
      cohenSutherlandClip(g, 0, 40, -40, 0);
```

```
    }
  }
}
```

Output-



Output(After Clipping)-



2. Hence apply the algorithm on pair of endpoints of each sides of a polygon.

Ans-Code-

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Clipping extends Applet implements ActionListener,
MouseWheelListener {

  int originX, originY;
  int height, width;
  int gap = 40;
  int temp = 0;
  static final int INSIDE = 0; // 0000
  static final int LEFT = 1; // 0001
  static final int RIGHT = 2; // 0010
  static final int BOTTOM = 4; // 0100
  static final int TOP = 8; // 1000

  static final int x_max = 100;
  static final int y_max = 100;
  static final int x_min = -100;
  static final int y_min = -100;

  // static final int x_max = 25;
  // static final int y_max = 25;
  // static final int x_min = -25;
  // static final int y_min = -25;

  Button b1 = new Button(" + ");
  Button b2 = new Button(" - ");
  Button b3 = new Button(" Clip ");

  public void init() {
    setBackground(Color.black);
    b1.setBackground(Color.red);
    b2.setBackground(Color.green);
    b3.setBackground(Color.GREEN);
    add(b1);
    add(b2);
    add(b3);
    addMouseWheelListener(this);
```

```
    b1.addActionListener(this);
    b2.addActionListener(this);
    b3.addActionListener(this);
  }



  //Function for plotting points
  public void plotPoint(Graphics g, int x,int y ,Color c){
    int originX = (getX() + getWidth()) / 2;
    int originY = (getY() + getHeight()) / 2;
    g.setColor(c);
    g.fillRect(originX+(gap*x)-(gap/4), originY-(gap*y)-(gap/4),2*gap ,2*gap
);
}

  //function to make grid
  public void makeGrid(Graphics g)
  {
      if(gap<=0|| gap>getHeight())
          return ;
      int originX = (getX() + getWidth()) / 2;
      int originY = (getY() + getHeight()) / 2;
      g.setColor(Color.red);
      g.drawLine(originX, originY - getHeight() / 2, originX, originY +
getHeight() / 2);
      g.drawLine(originX - getWidth() / 2, originY, originX + getWidth() / 2,
originY);
      g.setColor(Color.black);
      for (int x = gap; x <= getWidth(); x += gap) {

          g.drawLine(originX + x, 0, originX + x, getHeight());
          g.drawLine(originX - x, 0, originX - x, getHeight());

      }
      for (int y = gap; y <= getHeight(); y += gap) {

          g.drawLine(0, originY + y, getWidth(), originY + y);
          g.drawLine(0, originY - y, getWidth(), originY - y);
      }
  }


  //Function for the buttons
  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == b1) zoom(10);
    if (e.getSource() == b2) zoom(-10);
    if (e.getSource() == b3) {
      if (temp == 0) temp = 1; else temp = 0;
```

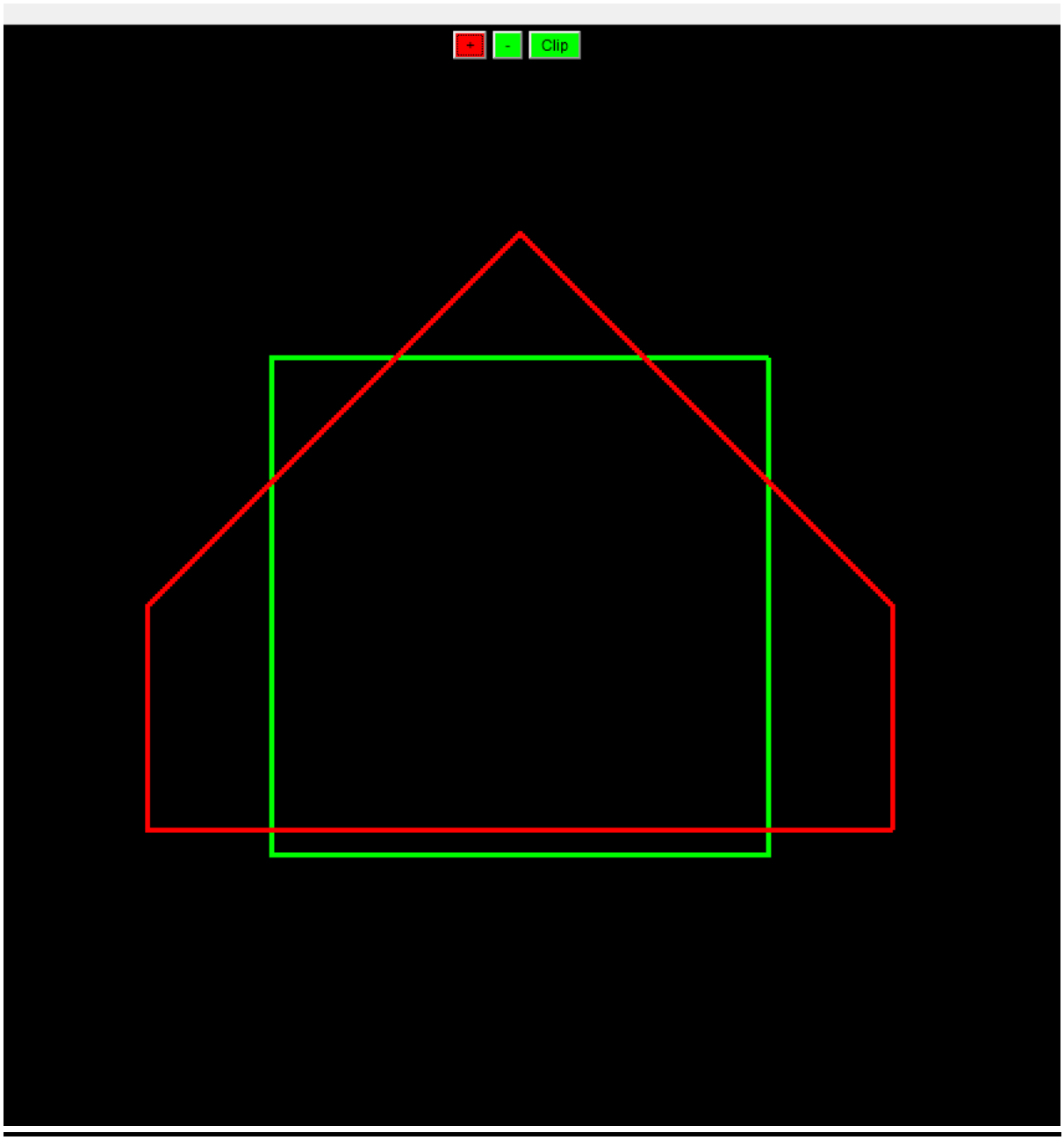
```
      repaint();
    }
  }

  //Function for the mousewheel
  public void mouseWheelMoved(MouseWheelEvent e) {
    int z = e.getWheelRotation();
    zoom(z);
  }

  //Function for the zoom in feature
  public void zoom(int i)
  {
      if(i>0)
          gap+=gap/10+1;
      else if(i<0)
          gap-=gap/10+1;
      repaint();

  }
  //function to compute code
  static int computeCode(int x, int y) {
    // initialized as being inside
    int code = INSIDE;

    if (x < x_min) code |= LEFT; else if ( // to the left of rectangle
      x > x_max
    ) code |= RIGHT; // to the right of rectangle
    if (y < y_min) code |= BOTTOM; else if ( // below the rectangle
      y > y_max
    ) code |= TOP; // above the rectangle

    return code;
  }

  public void cohenSutherlandClip(Graphics g, int x1, int y1, int x2, int y2)
{
    // Compute region codes for P1, P2
    int code1 = computeCode(x1, y1);
    int code2 = computeCode(x2, y2);

    // Initialize line as outside the rectangular window
    boolean accept = false;

    while (true) {
      if ((code1 == 0) && (code2 == 0)) {
        // If both endpoints lie within rectangle
        accept = true;
```

```
      break;
    } else if ((code1 & code2) != 0) {
      // If both endpoints are outside rectangle,
      // in same region
      break;
    } else {
      // Some segment of line lies within the
      // rectangle
      int code_out;
      int x = 0, y = 0;

      // 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) != 0) {
        // point is above the clip rectangle
        x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1);
        y = y_max;
      } else if ((code_out & BOTTOM) != 0) {
        // point is below the rectangle
        x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1);
        y = y_min;
      } else if ((code_out & RIGHT) != 0) {
        // point is to the right of rectangle
        y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1);
        x = x_max;
      } else if ((code_out & LEFT) != 0) {
        // point is to the left of rectangle
        y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1);
        x = x_min;
      }

      // Now intersection point x, y is found
      // We replace point outside rectangle
      // by intersection point
      if (code_out == code1) {
        x1 = x;
        y1 = y;
        code1 = computeCode(x1, y1);
      } else {
        x2 = x;
        y2 = y;
        code2 = computeCode(x2, y2);
      }
```

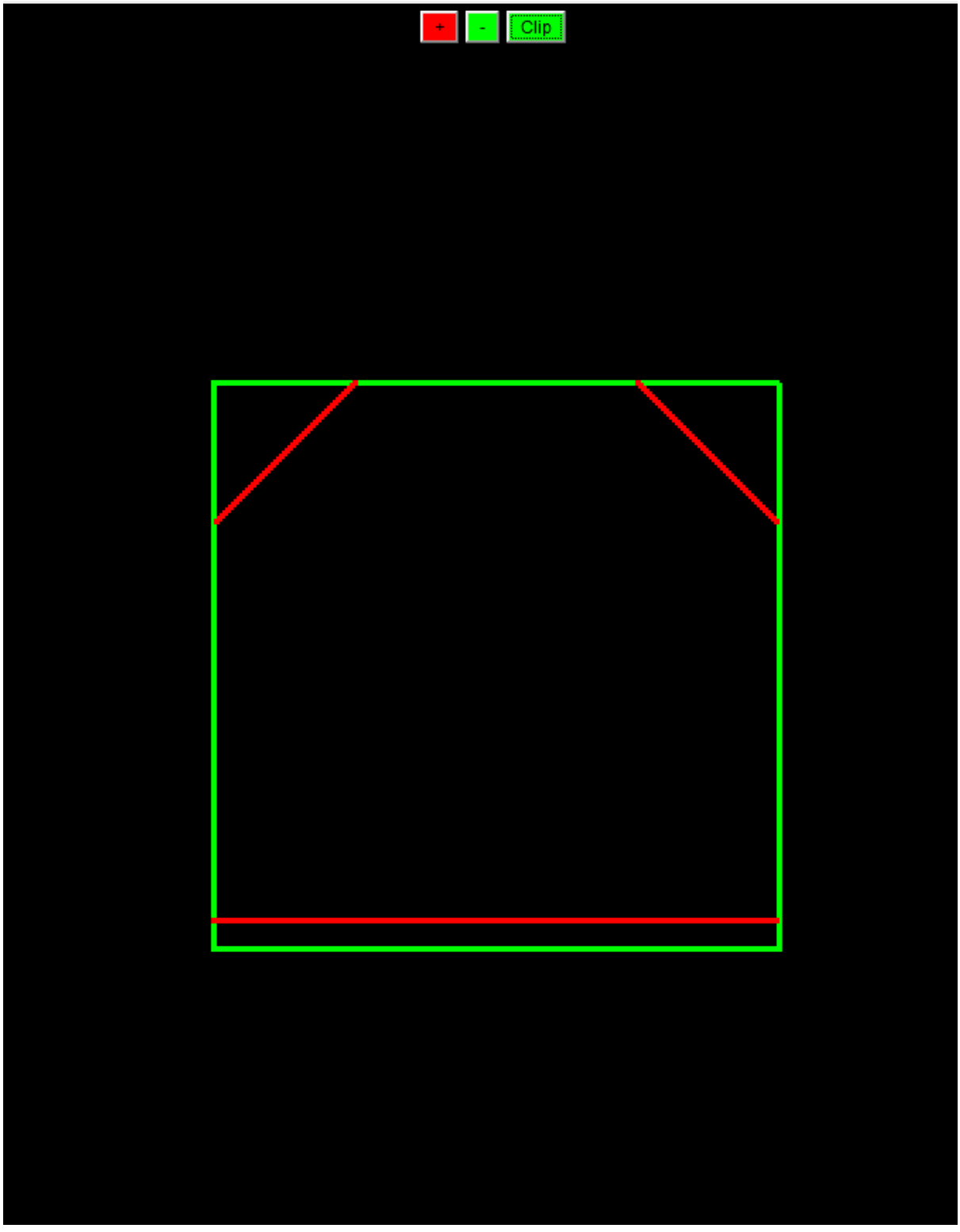
```
    }
  }
  if (accept) {
    DDALine(g, x1, y1, x2, y2,Color.RED);
  } else System.out.println("Line rejected");
}

void DDALine(Graphics g, int x0, int y0, int x1, int y1,Color c) {
  int dx = (x1 - x0);
  int dy = (y1 - y0);

  int step;
  if (Math.abs(dx) > Math.abs(dy)) {
    step = Math.abs(dx);
  } else {
    step = Math.abs(dy);
  }

  float x_incr = (float) dx / step;
  float y_incr = (float) dy / step;
  float x = (float) x0;
  float y = (float) y0;

  for (int i = 0; i < step; i++) {
    plotPoint(g, Math.round(x), Math.round(y), c);
    x += x_incr;
    y += y_incr;
  }
}
//paint function
public void paint(Graphics g) {
  g.setColor(Color.white);
  height = getHeight();
  width = getWidth();
  originX = (getX() + width) / 2;
  originY = (getY() + height) / 2;
  DDALine(g, x_min, y_min, x_max, y_min,Color.GREEN);
  DDALine(g, x_min, y_max, x_max, y_max,Color.GREEN);
  DDALine(g, x_min, y_min, x_min, y_max,Color.GREEN);
  DDALine(g, x_max, y_min, x_max, y_max,Color.GREEN);
  // makeGrid(g);
  if (temp == 0) {
 //Pentagon
 DDALine(g, 0, 150, -150, 0,Color.red);
 DDALine(g, 0, 150, 150, 0,Color.red);
 DDALine(g, -150, 0, -150, -90,Color.red);
 DDALine(g, 150, 0, 150, -90,Color.red);
 DDALine(g, -150, -90, 150, -90,Color.red);
```

```
    } else {
        //Pentagon
        cohenSutherlandClip(g, 0, 150, -150, 0);
        cohenSutherlandClip(g, 0, 150, 150, 0);
        cohenSutherlandClip(g, -150, 0, -150, -90);
        cohenSutherlandClip(g, 150, 0, 150, -90);
        cohenSutherlandClip(g, -150, -90, 150, -90);


    }
  }
}
```

Output(Pentagon Before Clipping)-



Output After Clipping(Pentagon)-



ASSIGNMENT-6

Q2. From set of control points draw a Cubic B-Spline curve.

Ans-Code-

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class B_Spline extends Applet
implements  ActionListener,MouseWheelListener{
    //It is for generating a rectangle corresponding to a particular point in
cartesian coordinate syatem
        int gap = 4;
    public void plotPoint(Graphics g,int x,int y,Color c)
    {
        g.setColor(c);
        g.fillRect(
            (getX()+getWidth())/2+(x*gap)-(gap/2),
            (getY()+getHeight())/2-(y*gap)-(gap/2),
            3*gap,3*gap
        );
    }
    public int slope(int x1,int x2,int y1,int y2)
    {
        int x=x2-x1;
        int y=y2-y1;
        int m=y/x;
        return m;
    }

    //It is for initialisation purpose
    public void init(){
        addMouseWheelListener(this);
        button1 = new Button("+");
        add(button1);
        button1.addActionListener(this);
        button2 = new Button("-");
        add(button2);
        button1.setBackground(Color.white);
        button2.setBackground(Color.white);
        button2.addActionListener(this);
```

```
        setForeground(Color.green);
        setBackground(Color.black);
    }
    //it is for implementing button function
    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource() == button1){
         gap+=gap+gap/10;
         repaint();
        }
        else if(e.getSource()==button2)
            {
             gap-=gap/10;
             repaint();
            }
    }
    //It is for mouse wheel operation
    public void mouseWheelMoved(MouseWheelEvent e)
    {
        int z=e.getWheelRotation();
        gap+=z;
        repaint();
    }

    Button button1, button2;
    // It is for creating the cartesian grids
    public void paintGrid(Graphics g,int gap,int originx,int originy)
    {
        g.setColor(Color.green);

        for(int i = gap;i<=getWidth();i+=gap)
        {
            g.drawLine(originx+i, originy-getHeight()/2, originx+i,
originy+getHeight()/2);
            g.drawLine(originx-i, originy-getHeight()/2, originx-i,
originy+getHeight()/2);
        }
        for(int i = gap;i<=getHeight();i+=gap)
        {
            g.drawLine(originx-getWidth()/2, originy+i, originx+getWidth()/2,
originy+i);
            g.drawLine(originx-getWidth()/2, originy-i, originx+getWidth()/2,
originy-i);

        }
    }

    public long ncr(int n, int r) {
```

```
        long result = 1;

        for (int i = 1; i <= r; i++) {
            result *= n - r + i;
            result /= i;
        }

        return result;
    }


    public void b_Spline(int[] x , int[] y)
    {
        double xu = 0.0 , yu = 0.0 , u = 0.0 ;

        for(u = 0.0 ; u <= 1.0 ; u += 0.0001)
        {
            xu = Math.pow(1-u,3)*x[0]+3*u*Math.pow(1-
u,2)*x[1]+3*Math.pow(u,2)*(1-u)*x[2]
                 +Math.pow(u,3)*x[3];
            yu = Math.pow(1-u,3)*y[0]+3*u*Math.pow(1-
u,2)*y[1]+3*Math.pow(u,2)*(1-u)*y[2]
                +Math.pow(u,3)*y[3];
            plotPoint(getGraphics(), (int)xu , (int)yu,Color.orange) ;
        }
    }

//It ia a normal paint function to call other functions to generate the
graphics in applet
    public void paint(Graphics g){

            g.setColor(Color.orange);
            int originx=getX()+getWidth()/2;
            int originy=getY()+getHeight()/2;
            g.drawLine(originx-getWidth()/2, originy, originx+getWidth()/2,
originy);
            g.drawLine(originx, originy-getHeight()/2, originx,
originy+getHeight()/2);
            // Point pts[] = new Point[12];
            int k=2;
            int[] x0 = {-100*k , -80*k , -60*k , -40*k};
            int[] y0 = {0 , -20*k , -30*k , 0};
            int[] x1 = {-40*k , -20*k , 0 , 20*k};
            int[] y1 = {0 , 20*k , 25*k , 0};
            int[] x2 = {20*k , 40*k , 80*k , 60*k};
            int[] y2 = {0 , -20*k , -25*k , 0};
            b_Spline(x0,y0);
            b_Spline(x1,y1);
            b_Spline(x2,y2);
```

```
            k=4;
            int[] x3 = {-100*k , -80*k , -60*k , -40*k};
            int[] y3 = {0 , -20*k , -30*k , 0};
            int[] x4 = {-40*k , -20*k , 0 , 20*k};
            int[] y4 = {0 , 20*k , 25*k , 0};
            int[] x5 = {20*k , 40*k , 80*k , 60*k};
            int[] y5 = {0 , -20*k , -25*k , 0};
            b_Spline(x3,y3);
            b_Spline(x4,y4);
            b_Spline(x5,y5);


    }
}
```

Output-