**Assignment Report**

**By**

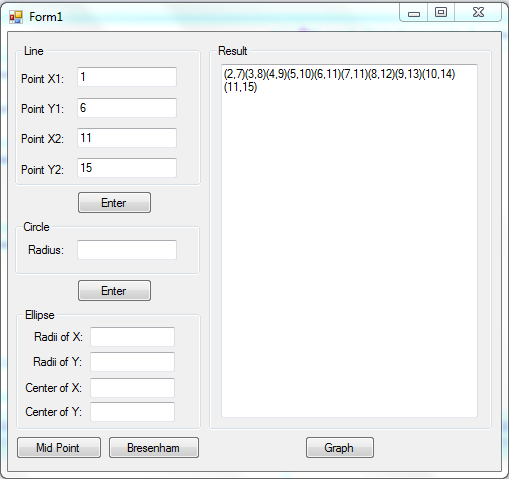
**Miss Kanokwan Teachasriphaitoon**

**ID: 55070701601**

**Assignment Report**

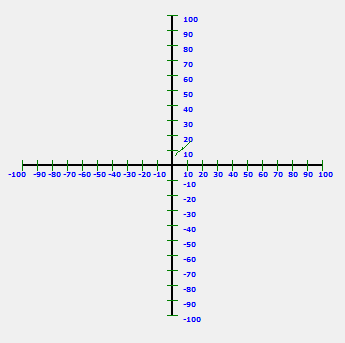
1. (10 Points) Scan Conversion (Line Segment) using original Bresenham’s line algorithm.

1.1 (2.5 Points) Scan conversion of a line segment, where two points are (1, 6) and (11, 15).

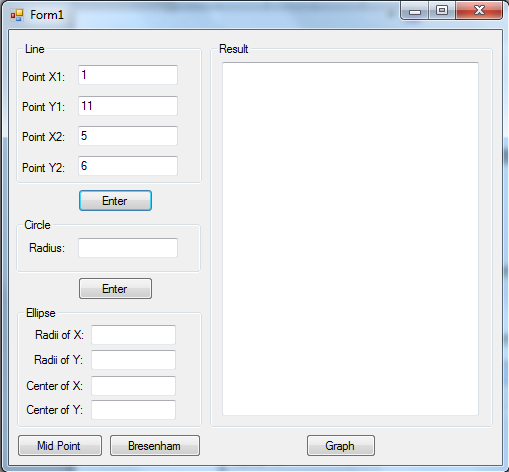


Result: (1,6)(2,7)(3,8)(4,9)(5,10)(6,11)(7,11)(8,12)(9,13)(10,14)(11,15)

Graph

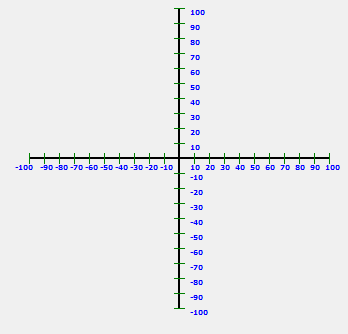


1.2 (2.5 Points) Scan conversion of a line segment, where two points are (1, 11) and (5, 6).

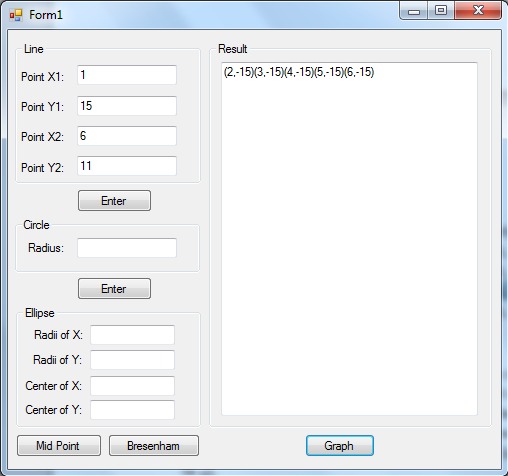


Result: (1, 11)

Graph

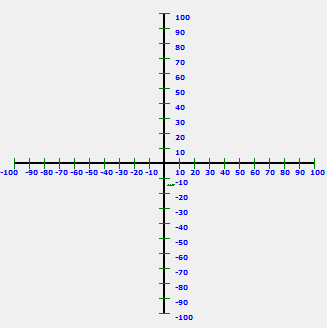


1.3 (2.5 Points) Scan conversion of a line segment, where two points are (1, 15) and (6, 11).

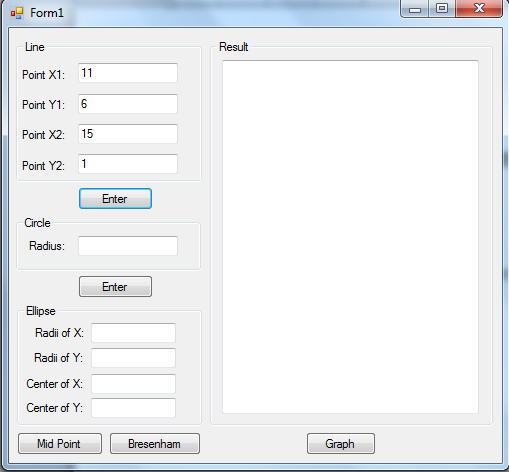


Result: (1,-15)(2,-15)(3,-15)(4,-15)(5,-15)(6,-15)

Graph

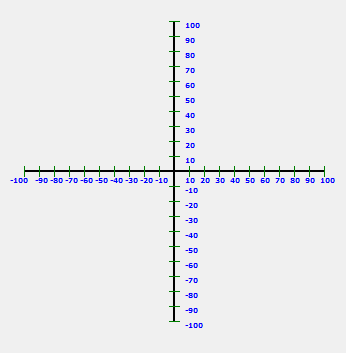


1.4 (2.5 Points) Scan conversion of a line segment, where two points are (11, 6) and (15, 1).



Result: (11, 6)

Graph



Source Code for Line Segment

Get Input:

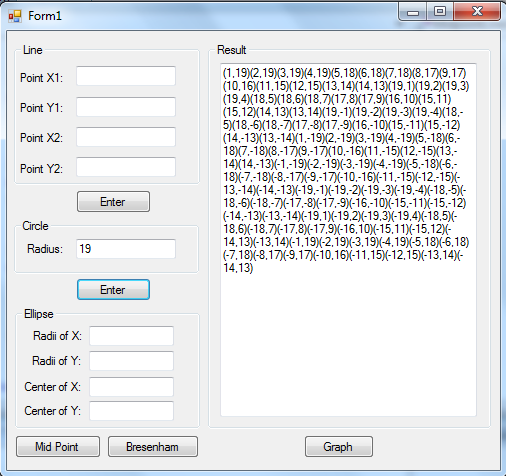
|  |
| --- |
| private void bt\_result\_Click(object sender, EventArgs e)  {  lb\_result.Text = "";  if (tb\_x1.Text == "" || tb\_x2.Text == "" || tb\_y1.Text == "" || tb\_y2.Text == "")  {  MessageBox.Show("Input is not complete!!!");  }  else  {  PointsList = new List<Points>();  double m = 0;  int x1 = Convert.ToInt32(tb\_x1.Text);  int y1 = Convert.ToInt32(tb\_y1.Text);  int x2 = Convert.ToInt32(tb\_x2.Text);  int y2 = Convert.ToInt32(tb\_y2.Text);  m = (Convert.ToDouble(y2 - y1)) / (Convert.ToDouble(x2 - x1));  if (m >= 0 && m <= 1 && x1 < x2)  {  PointsList = MidpointLine(Math.Abs(x1), Math.Abs(x2), Math.Abs(y1), Math.Abs(y2), 1);//case1  }  else if (m > 1 && y1 < y2)  {  PointsList = MidpointLine(Math.Abs(y1), Math.Abs(y2), Math.Abs(x1), Math.Abs(x2), 2);//case2  }  else if (m >= -1 && m < 0 && x1 < x2)  {  PointsList = MidpointLine(Math.Abs(x1), Math.Abs(x2), Math.Abs(y1), Math.Abs(y2), 3);//case3  }  else if (m < -1 && y2 < y1)  {  PointsList = MidpointLine(Math.Abs(y1), Math.Abs(y2), Math.Abs(x1), Math.Abs(x2), 4);//case4  }  else if (m > 1 && y2 < y1)  {  PointsList = MidpointLine(Math.Abs(y1), Math.Abs(y2), Math.Abs(x1), Math.Abs(x2), 5);//case5  }  else if (m > 0 && m <= 1 && x2 < x1)  {  PointsList = MidpointLine(Math.Abs(x1), Math.Abs(x2), Math.Abs(y1), Math.Abs(y2), 6);//case6  }  else if (m <= 0 && m >= -1 && x2 < x1)  {  PointsList = MidpointLine(Math.Abs(x1), Math.Abs(x2), Math.Abs(y1), Math.Abs(y2), 7);//case7  }  else if (m < -1 && y1 < y2)  {  PointsList = MidpointLine(Math.Abs(y1), Math.Abs(y2), Math.Abs(x1), Math.Abs(x2), 8);//case8  }  result = "";  foreach (var data in PointsList)  {  result = result + "(" + data.Pointx.ToString() + "," + data.Pointy.ToString() + ")";  }  lb\_result.Text = result.ToString();  }  } |

Line Segment function:

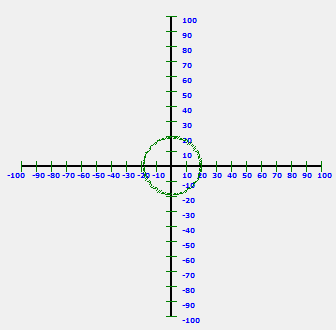
|  |
| --- |
| private List<Points> MidpointLine(int x1, int x2, int y1, int y2, int case\_qou)  {  List<Points> PointList = new List<Points>();  Points p = new Points();  int x = 0, y = 0;  int dx, dy;  dx = x2 - x1;  dy = y2 - y1;  int d = 0, dD = 0, dU = 0;  d = 2 \* dy - dx;  dD = 2 \* dy;  dU = 2 \* (dy - dx);  x = x1;  y = y1;  while (x < x2)  {  if (d < 0)  {  d = d + dD;  x = x + 1;  }  else  {  d = d + dU;  x = x + 1;  y = y + 1;  }  if (case\_qou == 1)  {  p = new Points() { Pointx = x, Pointy = y };  PointList.Add(p);  }  else if (case\_qou == 2)  {  p = new Points() { Pointx = y, Pointy = x };  PointList.Add(p);  }  else if (case\_qou == 3)  {  p = new Points() { Pointx = x, Pointy = -y };  PointList.Add(p);  }  else if (case\_qou == 4)  {  p = new Points() { Pointx = y, Pointy = -x };  PointList.Add(p);  }  else if (case\_qou == 5)  {  p = new Points() { Pointx = -y, Pointy = -x };  PointList.Add(p);  }  else if (case\_qou == 6)  {  p = new Points() { Pointx = -x, Pointy = -y };  PointList.Add(p);  }  else if (case\_qou == 7)  {  p = new Points() { Pointx = -x, Pointy = y };  PointList.Add(p);  }  else  {  p = new Points() { Pointx = -y, Pointy = x };  PointList.Add(p);  }  }  return PointList;  } |

2. (10 Points) Scan Conversion (Circle)

2.1 (4 Points) Scan conversion of a circle with the radius 19 pixels using original midpoint circle algorithm. Show only the second octant of the circle.



Graph



Source code for Circle

|  |
| --- |
| private void button1\_Click(object sender, EventArgs e)  {  lb\_result.Text = "";  if (tb\_radian.Text == "")  {  MessageBox.Show("Input is not complete!!!");  }  else  {  PointsList = new List<Points>();  List<Points> TmpPoints = new List<Points>();  Points p = new Points();  int h = 0, dU = 0, dD = 0;  int r = 0;  Console.Write("Enter r: ");  r = Convert.ToInt32(tb\_radian.Text);  h = 1 - r;  int x = 0, y = 0;  y = r;  while (y > x)  {  if (h < 0)  {  dU = 2 \* x + 3;  h = h + dU;  x = x + 1;  }  else  {  dD = 2 \* (x - y) + 5;  h = h + dD;  x = x + 1;  y = y - 1;  }  p = new Points() { Pointx = x, Pointy = y };  TmpPoints.Add(p);  }  //case 1: (y,x)  foreach (var datas in TmpPoints)  {  p = new Points() { Pointx = datas.Pointx, Pointy = datas.Pointy };  PointsList.Add(p);  }  //case 2: (x,y)  foreach (var datas in TmpPoints)  {  p = new Points() { Pointx = datas.Pointy, Pointy = datas.Pointx };  PointsList.Add(p);  }  //case 3: (x,-y)  foreach (var datas in TmpPoints)  {  p = new Points() { Pointx = datas.Pointy, Pointy = -(datas.Pointx) };  PointsList.Add(p);  }  //case 4: (y,-x)  foreach (var datas in TmpPoints)  {  p = new Points() { Pointx = datas.Pointx, Pointy = -(datas.Pointy) };  PointsList.Add(p);  }  //case 5: (-y,-x)  foreach (var datas in TmpPoints)  {  p = new Points() { Pointx = -(datas.Pointx), Pointy = -(datas.Pointy) };  PointsList.Add(p);  }  //case 6: (-x,-y)  foreach (var datas in TmpPoints)  {  p = new Points() { Pointx = -(datas.Pointy), Pointy = -(datas.Pointx) };  PointsList.Add(p);  }  //case 7: (-x, y)  foreach (var datas in TmpPoints)  {  p = new Points() { Pointx = -(datas.Pointy), Pointy = (datas.Pointx) };  PointsList.Add(p);  }  //case 8: (-y, x)  foreach (var datas in TmpPoints)  {  p = new Points() { Pointx = -(datas.Pointx), Pointy = (datas.Pointy) };  PointsList.Add(p);  }  result = "";  foreach (var data in PointsList)  {  result = result + "(" + data.Pointx.ToString() + "," + data.Pointy.ToString() + ")";  }  lb\_result.Text = result.ToString();  }  } |

5. (10 Points) Line Clipping.

Let the window coordinates be (100, 100) and (150, 160) for the lower-left and upper-right corners, respectively. Fill in the blank for the following line clipping processes:

1. Compute 4-Bit codes for each point represented line segments.

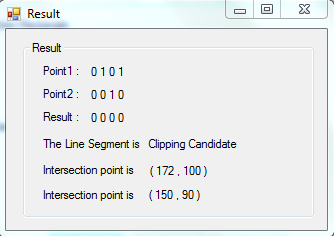
2. Use Cohen-Sutherland algorithm for the classification of these line segments into: Visible, Invisible or Clipping candidate.

3. Find the visible parts of the clipped line segments. In cases of clipping candidate, the student has to calculate and answer the intersection points.

Line1:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Point 1 | Point 2 | 4-Bit Code point 1 | 4-Bit Code point 2 | Categories | Visible Point 1 | Visible Point 2 |
| (50, 50) | (175, 101) | 0101 | 0010 | Clipping Candidate | (172, 100) | (150, 90) |

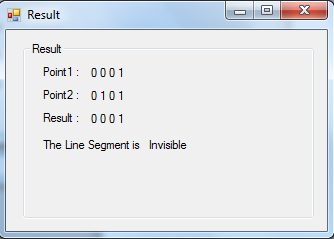
Program Result:



Line2:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Point 1 | Point 2 | 4-Bit Code point 1 | 4-Bit Code point 2 | Categories | Visible Point 1 | Visible Point 2 |
| (90, 90) | (155, 80) | 0001 | 0101 | Invisible | - | - |

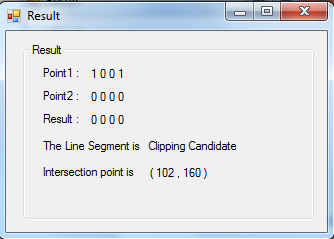
Program Result:



Line3:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Point 1 | Point 2 | 4-Bit Code point 1 | 4-Bit Code point 2 | Categories | Visible Point 1 | Visible Point 2 |
| (95, 110) | (165, 155) | 0101 | 0000 | Clipping Candidate | (102,160) | - |

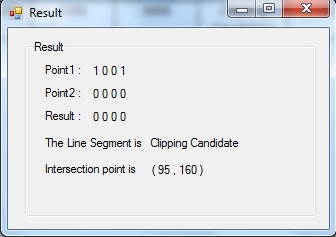
Program Result:



Line4:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Point 1 | Point 2 | 4-Bit Code point 1 | 4-Bit Code point 2 | Categories | Visible Point 1 | Visible Point 2 |
| (95, 100) | (165, 135) | 1001 | 0000 | Clipping Candidate | (95,160) | - |

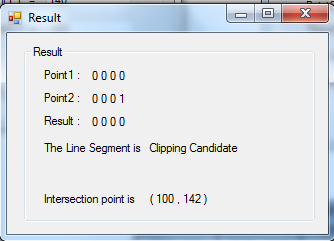
Program Result:



Line5:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Point 1 | Point 2 | 4-Bit Code point 1 | 4-Bit Code point 2 | Categories | Visible Point 1 | Visible Point 2 |
| (110, 50) | (140, 153) | 0000 | 0001 | Clipping Candidate | - | (100,142) |

Program Result:

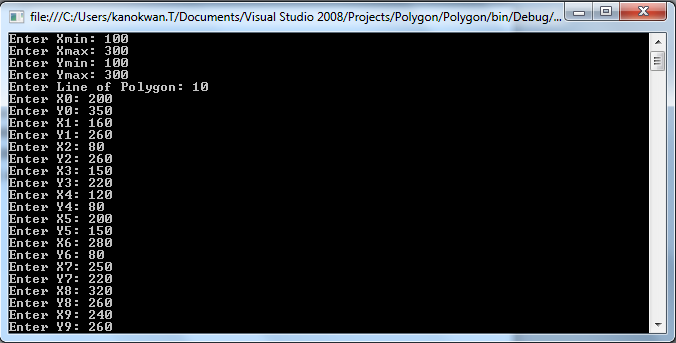


Source code for Line Clipping:

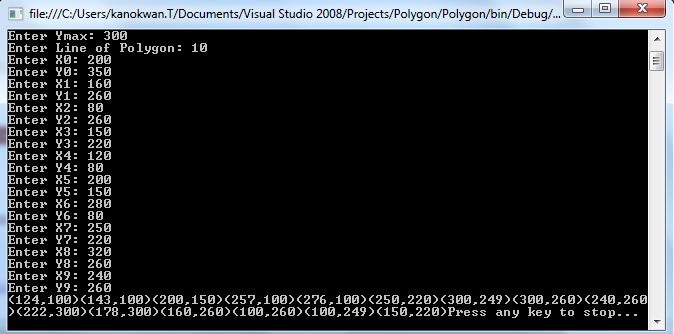
|  |
| --- |
| public Result(int Xmin, int Xmax, int Ymin, int Ymax, int PointX1\_Line, int PointY1\_Line, int PointX2\_Line, int PointY2\_Line)  {  InitializeComponent();  label3.Visible = false;  label4.Visible = false;  lb\_intersec1.Visible = false;  lb\_intersec2.Visible = false;  bool TopFlagStartPoint = false;  bool DownFlagStartPoint = false;  bool RightFlagStartPoint = false;  bool LeftFlagStartPoint = false;  bool TopFlagEndPoint = false;  bool DownFlagEndPoint = false;  bool RightFlagEndPoint = false;  bool LeftFlagEndPoint = false;  String Result = null;  float m = 0;  float P = 0;  float x1 = 0;  float x2 = 0;  float y1 = 0;  float y2 = 0;  TopFlagStartPoint = CalTopFlag(Ymax, PointY1\_Line);  DownFlagStartPoint = CalDownFlag(Ymin, PointY1\_Line);  RightFlagStartPoint = CalRightFlag(Xmax, PointX1\_Line);  LeftFlagStartPoint = CalLeftFlag(Xmin, PointX1\_Line);  TopFlagEndPoint = CalTopFlag(Ymax, PointY2\_Line);  DownFlagEndPoint = CalDownFlag(Ymin, PointY2\_Line);  RightFlagEndPoint = CalRightFlag(Xmax, PointX2\_Line);  LeftFlagEndPoint = CalLeftFlag(Xmin, PointX2\_Line);  if (TopFlagStartPoint == true || DownFlagStartPoint == true || RightFlagStartPoint == true || LeftFlagStartPoint == true ||  TopFlagEndPoint == true || DownFlagEndPoint == true || RightFlagEndPoint == true || LeftFlagEndPoint == true)  {  if (TopFlagStartPoint & TopFlagEndPoint == true || DownFlagStartPoint & DownFlagEndPoint == true ||  RightFlagStartPoint & RightFlagEndPoint == true || LeftFlagStartPoint & LeftFlagEndPoint == true)  {  Result = "Invisible";  }  else  {  Result = "Clipping Candidate";  //point 1  if (TopFlagStartPoint)  {  x1 = PointX1\_Line + (PointX2\_Line - PointX1\_Line) \* (Ymax - PointY1\_Line) / (PointY2\_Line - PointY1\_Line);  y1 = Ymax;  label3.Visible = true;  lb\_intersec1.Visible = true;  lb\_intersec1.Text = "( " + x1 + " , " + y1 + " )";  }  else if (DownFlagStartPoint)  {  x1 = PointX1\_Line + (PointX2\_Line - PointX1\_Line) \* (Ymin - PointY1\_Line) / (PointY2\_Line - PointY1\_Line);  y1 = Ymin;  label3.Visible = true;  lb\_intersec1.Visible = true;  lb\_intersec1.Text = "( " + x1 + " , " + y1 + " )";  }  else if (RightFlagStartPoint)  {  y1 = PointY1\_Line + (PointY2\_Line - PointY1\_Line) \* (Xmax - PointX1\_Line) / (PointX2\_Line - PointX1\_Line);  x1 = Xmax;  label3.Visible = true;  lb\_intersec1.Visible = true;  lb\_intersec1.Text = "( " + x1 + " , " + y1 + " )";  }  else if (LeftFlagStartPoint)  {  y1 = PointY1\_Line + (PointY2\_Line - PointY1\_Line) \* (Xmin - PointX1\_Line) / (PointX2\_Line - PointX1\_Line);  x1 = Xmin;  label3.Visible = true;  lb\_intersec1.Visible = true;  lb\_intersec1.Text = "( " + x1 + " , " + y1 + " )";  }  //point 2  if (TopFlagEndPoint)  {  x2 = PointX1\_Line + (PointX2\_Line - PointX1\_Line) \* (Ymax - PointY1\_Line) / (PointY2\_Line - PointY1\_Line);  y2 = Ymax;  label4.Visible = true;  lb\_intersec2.Visible = true;  lb\_intersec2.Text = "( " + x2 + " , " + y2 + " )";  }  else if (DownFlagEndPoint)  {  x2 = PointX1\_Line + (PointX2\_Line - PointX1\_Line) \* (Ymin - PointY1\_Line) / (PointY2\_Line - PointY1\_Line);  y2 = Ymin;  label4.Visible = true;  lb\_intersec2.Visible = true;  lb\_intersec2.Text = "( " + x2 + " , " + y2 + " )";  }  else if (RightFlagEndPoint)  {  y2 = PointY1\_Line + (PointY2\_Line - PointY1\_Line) \* (Xmax - PointX1\_Line) / (PointX2\_Line - PointX1\_Line);  x2 = Xmax;  label4.Visible = true;  lb\_intersec2.Visible = true;  lb\_intersec2.Text = "( " + x2 + " , " + y2 + " )";  }  else if (LeftFlagEndPoint)  {  y2 = PointY1\_Line + (PointY2\_Line - PointY1\_Line) \* (Xmin - PointX1\_Line) / (PointX2\_Line - PointX1\_Line);  x2 = Xmin;  label4.Visible = true;  lb\_intersec2.Visible = true;  lb\_intersec2.Text = "( " + x2 + " , " + y2 + " )";  }  }  }  else  {  Result = "Visible";  }  lb\_Line\_Result.Text = Result;  lb\_Point1\_Result.Text = Convert.ToInt32(TopFlagStartPoint).ToString() + " " + Convert.ToInt32(DownFlagStartPoint).ToString() + " " + Convert.ToInt32(RightFlagStartPoint).ToString() + " " + Convert.ToInt32(LeftFlagStartPoint).ToString();  lb\_Point2\_Result.Text = Convert.ToInt32(TopFlagEndPoint).ToString() + " " + Convert.ToInt32(DownFlagEndPoint).ToString() + " " + Convert.ToInt32(RightFlagEndPoint).ToString() + " " + Convert.ToInt32(LeftFlagEndPoint).ToString();  int a = Convert.ToInt32(GetResult(TopFlagStartPoint,TopFlagEndPoint));  int b = Convert.ToInt32(GetResult(DownFlagStartPoint,DownFlagEndPoint));  int c = Convert.ToInt32(GetResult(RightFlagStartPoint,RightFlagEndPoint));  int d = Convert.ToInt32(GetResult(LeftFlagStartPoint, LeftFlagEndPoint));  lb\_Result.Text = a +" "+ b +" "+ c +" "+ d;  }  private bool CalTopFlag(int Ymax, int Y\_Line)  {  if (Y\_Line > Ymax)  {  return true;  }  return false;  }  private bool CalDownFlag(int Ymin, int Y\_Line)  {  if (Y\_Line < Ymin)  {  return true;  }  return false;  }  private bool CalRightFlag(int Xmax, int X\_line)  {  if (X\_line > Xmax)  {  return true;  }  return false;  }  private bool CalLeftFlag(int Xmin, int X\_line)  {  if (X\_line < Xmin)  {  return true;  }  return false;  }  private bool GetResult(bool point1, bool point2)  {  bool result = point1 & point2;  return result;  }    } |

6. (10 Points) Polygon Clipping Let a window have a lower left corner at the point X(100, 100) and a upper-right corner at the point Y(300, 300). Perform the polygon clipping of a polygon with a sequence of vertices ABCDEFGHIJ, where A(200, 350), B(160, 260), C(80, 260), D(150, 220), E(120, 80), F(200, 150), G(280, 80), H(250, 220), I(320, 260) and J(240, 260).

Initial Input:



Result:



Source code for Polygon clipping

|  |
| --- |
| static void Main(string[] args)  {  int Ymin, Ymax, Xmin, Xmax, NumOfPolygon;  Console.Write("Enter Xmin: ");  Xmin = Convert.ToInt32(Console.ReadLine());  Console.Write("Enter Xmax: ");  Xmax = Convert.ToInt32(Console.ReadLine());  Console.Write("Enter Ymin: ");  Ymin = Convert.ToInt32(Console.ReadLine());  Console.Write("Enter Ymax: ");  Ymax = Convert.ToInt32(Console.ReadLine());    Console.Write("Enter Line of Polygon: ");  NumOfPolygon = Convert.ToInt32(Console.ReadLine());  List<Point> pointList = new List<Point>();  List<Point> TmppointList = new List<Point>();  Point p = new Point();  int X = 0;  int Y = 0;  //get input  for (int i = 0; i < NumOfPolygon; i++)  {  Console.Write("Enter X" + i + ": ");  X = Convert.ToInt32(Console.ReadLine());  Console.Write("Enter Y" + i + ": ");  Y = Convert.ToInt32(Console.ReadLine());  p = new Point() {Pointx = X, Pointy = Y};  pointList.Add(p);  }  #region  //check down  TmppointList = pointList;  pointList = new List<Point>();  result res;  int count = 0;  foreach(var data in TmppointList)  {  if (count+1 < TmppointList.Count())  {  res = Down(Ymin, TmppointList[count].Pointx,  TmppointList[count + 1].Pointx,  TmppointList[count].Pointy,  TmppointList[count+1].Pointy);  if (res.x1.HasValue && res.y1.HasValue)  {  p = new Point() { Pointx = res.x1.Value, Pointy = res.y1.Value };  pointList.Add(p);  }  if (res.x2.HasValue && res.y2.HasValue)  {  p = new Point() { Pointx = res.x2.Value, Pointy = res.y2.Value };  pointList.Add(p);  }    }  else  {  res = Down(Ymin, TmppointList[count].Pointx,  TmppointList[0].Pointx,  TmppointList[count].Pointy,  TmppointList[0].Pointy);  if (res.x1.HasValue && res.y1.HasValue)  {  p = new Point() { Pointx = res.x1.Value, Pointy = res.y1.Value };  pointList.Add(p);  }  if (res.x2.HasValue && res.y2.HasValue)  {  p = new Point() { Pointx = res.x2.Value, Pointy = res.y2.Value };  pointList.Add(p);  }  }  count++;  }  #endregion  #region  //check right  TmppointList = pointList;  pointList = new List<Point>();  count = 0;  foreach (var data in TmppointList)  {  if (count + 1 < TmppointList.Count())  {  res = Right(Xmax, TmppointList[count].Pointx,  TmppointList[count + 1].Pointx,  TmppointList[count].Pointy,  TmppointList[count + 1].Pointy);  if (res.x1.HasValue && res.y1.HasValue)  {  p = new Point() { Pointx = res.x1.Value, Pointy = res.y1.Value };  pointList.Add(p);  }  if (res.x2.HasValue && res.y2.HasValue)  {  p = new Point() { Pointx = res.x2.Value, Pointy = res.y2.Value };  pointList.Add(p);  }  }  else  {  res = Right(Xmax, TmppointList[count].Pointx,  TmppointList[0].Pointx,  TmppointList[count].Pointy,  TmppointList[0].Pointy);  if (res.x1.HasValue && res.y1.HasValue)  {  p = new Point() { Pointx = res.x1.Value, Pointy = res.y1.Value };  pointList.Add(p);  }  if (res.x2.HasValue && res.y2.HasValue)  {  p = new Point() { Pointx = res.x2.Value, Pointy = res.y2.Value };  pointList.Add(p);  }  }  count++;  }  #endregion  #region  //check up  TmppointList = pointList;  pointList = new List<Point>();  count = 0;  foreach (var data in TmppointList)  {  if (count + 1 < TmppointList.Count())  {  res = Up(Ymax, TmppointList[count].Pointx,  TmppointList[count + 1].Pointx,  TmppointList[count].Pointy,  TmppointList[count + 1].Pointy);  if (res.x1.HasValue && res.y1.HasValue)  {  p = new Point() { Pointx = res.x1.Value, Pointy = res.y1.Value };  pointList.Add(p);  }  if (res.x2.HasValue && res.y2.HasValue)  {  p = new Point() { Pointx = res.x2.Value, Pointy = res.y2.Value };  pointList.Add(p);  }  }  else  {  res = Up(Ymax, TmppointList[count].Pointx,  TmppointList[0].Pointx,  TmppointList[count].Pointy,  TmppointList[0].Pointy);  if (res.x1.HasValue && res.y1.HasValue)  {  p = new Point() { Pointx = res.x1.Value, Pointy = res.y1.Value };  pointList.Add(p);  }  if (res.x2.HasValue && res.y2.HasValue)  {  p = new Point() { Pointx = res.x2.Value, Pointy = res.y2.Value };  pointList.Add(p);  }  }  count++;  }  #endregion  #region  //check left  TmppointList = pointList;  pointList = new List<Point>();  count = 0;  foreach (var data in TmppointList)  {  if (count + 1 < TmppointList.Count())  {  res = Left(Xmin, TmppointList[count].Pointx,  TmppointList[count + 1].Pointx,  TmppointList[count].Pointy,  TmppointList[count + 1].Pointy);  if (res.x1.HasValue && res.y1.HasValue)  {  p = new Point() { Pointx = res.x1.Value, Pointy = res.y1.Value };  pointList.Add(p);  }  if (res.x2.HasValue && res.y2.HasValue)  {  p = new Point() { Pointx = res.x2.Value, Pointy = res.y2.Value };  pointList.Add(p);  }  }  else  {  res = Left(Xmin, TmppointList[count].Pointx,  TmppointList[0].Pointx,  TmppointList[count].Pointy,  TmppointList[0].Pointy);  if (res.x1.HasValue && res.y1.HasValue)  {  p = new Point() { Pointx = res.x1.Value, Pointy = res.y1.Value };  pointList.Add(p);  }  if (res.x2.HasValue && res.y2.HasValue)  {  p = new Point() { Pointx = res.x2.Value, Pointy = res.y2.Value };  pointList.Add(p);  }  }  count++;  }  #endregion  var results = pointList.Where(pp => pp.Pointx != null).GroupBy(pp => new { pp.Pointx, pp.Pointy }).Select(grp => grp.First())  .ToList<Point>();  count = 0;  foreach (var data in results)  {  Console.Write("(" + results [count].Pointx + "," + results[count].Pointy + ")");  count++;  }  Console.WriteLine("Press any key to stop...");  Console.ReadKey();  }  public static result Down(int? Ymin, int? X1, int? X2, int? Y1, int? Y2)  {  if (Y1 >= Ymin && Y2 >= Ymin)  {  return new result(X2, Y2, null, null);  }  else if (Y1 < Ymin && Y2 < Ymin)  {  return new result(null, null, null, null);  }  else if (Y1 < Ymin && Y2 >= Ymin)  {  //y=mx+b  float m = 0, b = 0;  int x = 0;  m = ((float)Y2 - (float)Y1) / ((float)X2 - (float)X1);  b = (float)Y2 - (m \* (float)X2);  x = Convert.ToInt32(((float)Ymin - b) / m);  return new result(x, Ymin, X2, Y2);  }  else if (Y1 >= Ymin && Y2 < Ymin)  {  float m = 0, b = 0;  int x = 0;  m = ((float)Y2 - (float)Y1) / ((float)X2 - (float)X1);  b = (float)Y2 - (m \* (float)X2);  x = Convert.ToInt32(((float)Ymin - b) / m);  return new result(x, Ymin, null, null);  }  return new result(null, null, null, null);  }  public static result Right(int? Xmax, int? X1, int? X2, int? Y1, int? Y2)  {  if(X1 <= Xmax && X2 <= Xmax)  {  return new result(X2, Y2, null, null);  }  else if (X1 > Xmax && X2 > Xmax)  {  return new result(null, null, null, null);  }else if(X1 > Xmax && X2 <= Xmax)  {  //y=mx+b  float m = 0, b = 0;  int y = 0;  m = ((float)Y2 - (float)Y1) / ((float)X2 - (float)X1);  b = (float)Y2 - (m \* (float)X2);  y = Convert.ToInt32((m \* (float)Xmax) + b);  return new result(Xmax, y, X2, Y2);  }else if(X1 <= Xmax && X2 > Xmax)  {  //y=mx+b  float m = 0, b = 0;  int y = 0;  m = ((float)Y2 - (float)Y1) / ((float)X2 - (float)X1);  b = (float)Y2 - (m \* (float)X2);  y = Convert.ToInt32((m \* (float)Xmax) + b);  return new result(Xmax, y, null, null);  }  return new result(null, null, null, null);  }  public static result Up(int? Ymax, int? X1, int? X2, int? Y1, int? Y2)  {  if(Y1 <= Ymax && Y2 <= Ymax)  {  return new result(X2, Y2, null, null);  }  else if (Y1 > Ymax && Y2 > Ymax)  {  return new result(null, null, null, null);  }  else if (Y1 > Ymax && Y2 <= Ymax)  {  //y=mx+b  float m = 0, b = 0;  int x = 0;  m = ((float)Y2 - (float)Y1) / ((float)X2 - (float)X1);  b = (float)Y2 - (m \* (float)X2);  x = Convert.ToInt32(((float)Ymax - b) / m);  return new result(x, Ymax, X2, Y2);  }  else if (Y1 <= Ymax && Y2 > Ymax)  {  //y=mx+b  float m = 0, b = 0;  int x = 0;  m = ((float)Y2 - (float)Y1) / ((float)X2 - (float)X1);  b = (float)Y2 - (m \* (float)X2);  x = Convert.ToInt32(((float)Ymax - b) / m);  return new result(x, Ymax, null, null);  }  return new result(null, null, null, null);  }  public static result Left(int? Xmin, int? X1, int? X2, int? Y1, int? Y2)  {  if (X1 >= Xmin && X2 >= Xmin)  {  return new result(X2, Y2, null, null);  }  else if (X1 < Xmin && X2 < Xmin)  {  return new result(null, null, null, null);  }  else if (X1 < Xmin && X2 >= Xmin)  {  //y=mx+b  float m = 0, b = 0;  int y = 0;  m = ((float)Y2 - (float)Y1) / ((float)X2 - (float)X1);  b = (float)Y2 - (m \* (float)X2);  y = Convert.ToInt32((m \* (float)Xmin) + b);  return new result(Xmin, y, X2, Y2);  }  else if (X1 >= Xmin && X2 < Xmin)  {  //y=mx+b  float m = 0, b = 0;  int y = 0;  m = ((float)Y2 - (float)Y1) / ((float)X2 - (float)X1);  b = (float)Y2 - (m \* (float)X2);  y = Convert.ToInt32((m \* (float)Xmin) + b);  return new result(Xmin, y, null, null);  }  return new result(null, null, null, null);  } |

All Source code Link: https://www.dropbox.com/sh/bgz4xaelgoi5hc0/AAD76AnB1hqVAAzvWIZ7TiZ0a?dl=0