МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ «БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Лабораторная работа №4 по дисциплине:

«Компьютерная графика»

«Алгоритмы вычерчивания окружностей»

Вариант №15

Выполнил: ст. гр. АСОИ-181

Самусев Д. А.

Проверил: Шилов А. В.

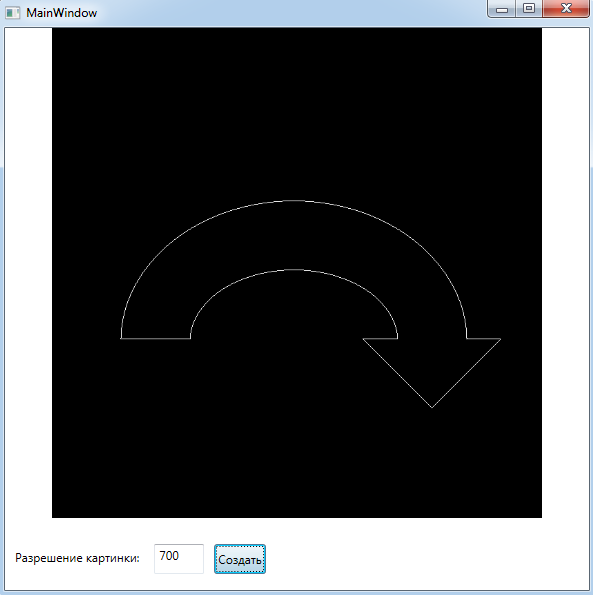
**Задание:** Составить алгоритм и программу для отображения объекта №2 методом Брезенхема.

Листинг кода программы:

namespace CG\_lab4\_Samusev  
{  
 class Line  
 {  
 public double X1 { get; set; }  
 public double Y1 { get; set; }  
  
 public double X2 { get; set; }  
 public double Y2 { get; set; }  
  
 public Line()  
 {  
 }  
  
 public Line(double x1, double y1, double x2, double y2)  
 {  
 X1 = x1;  
 Y1 = y1;  
 X2 = x2;  
 Y2 = y2;  
 }  
 }  
}

using System;  
using System.Linq;  
using System.Windows;  
using System.Windows.Media;  
using System.Windows.Media.Imaging;  
using System.Windows.Shapes;  
  
namespace CG\_lab4\_Samusev  
{  
 */// <summary>  
 /// Interaction logic for MainWindow.xaml  
 /// </summary>* public partial class MainWindow : Window  
 {  
 private int \_imageSideLength;  
  
 private delegate void DrawPixelInQuarters(int x, int y, int xOffset, int yOffset);  
 private WriteableBitmap bitmap;  
 delegate bool ConditionCheker();  
  
 public MainWindow()  
 {  
 InitializeComponent();  
 }  
  
 private void CreateFigureButton\_Click(object sender, RoutedEventArgs e)  
 {  
 try  
 {  
 \_imageSideLength = int.Parse(ResolutionTextBox.Text);  
 }  
 catch  
 {  
 \_imageSideLength = 210;  
 }  
  
 bitmap = new WriteableBitmap(\_imageSideLength + 10, \_imageSideLength + 10, 96, 96, PixelFormats.Bgr32, null);  
 ImageBox.Source = bitmap;  
  
 DrawEllipse(\_imageSideLength / 2, \_imageSideLength / 14 \* 9, \_imageSideLength / 14 \* 3, \_imageSideLength / 7, Quarter.**First**, Quarter.**Second**);  
 DrawEllipse(\_imageSideLength / 2, \_imageSideLength / 14 \* 9, \_imageSideLength / 14 \* 5, \_imageSideLength / 7 \* 2, Quarter.**First**, Quarter.**Second**);  
  
 DrawLine(new Line  
 {  
 X1 = \_imageSideLength / 7,  
 Y1 = \_imageSideLength / 14 \* 9,  
 X2 = \_imageSideLength / 7 \* 2,  
 Y2 = \_imageSideLength / 14 \* 9  
 });  
  
 DrawLine(new Line  
 {  
 X1 = \_imageSideLength / 14 \* 13,  
 Y1 = \_imageSideLength / 14 \* 9,  
 X2 = \_imageSideLength / 7 \* 6,  
 Y2 = \_imageSideLength / 14 \* 9  
 });  
  
 DrawLine(new Line  
 {  
 X1 = \_imageSideLength / 7 \* 5,  
 Y1 = \_imageSideLength / 14 \* 9,  
 X2 = \_imageSideLength / 14 \* 9,  
 Y2 = \_imageSideLength / 14 \* 9  
 });  
  
 DrawLine(new Line  
 {  
 X1 = \_imageSideLength / 14 \* 13,  
 Y1 = \_imageSideLength / 14 \* 9,  
 X2 = \_imageSideLength / 14 \* 11,  
 Y2 = \_imageSideLength / 14 \* 11  
 });  
  
 DrawLine(new Line  
 {  
 X1 = \_imageSideLength / 14 \* 9,  
 Y1 = \_imageSideLength / 14 \* 9,  
 X2 = \_imageSideLength / 14 \* 11,  
 Y2 = \_imageSideLength / 14 \* 11  
 });  
 }  
  
 private void DrawLine(Line line)  
 {  
  
 double err = -(1.0 / 2.0);  
 double delta = (double)Math.Abs(line.Y1 - line.Y2) / (double)Math.Abs(line.X1 - line.X2);  
  
 int y;  
 double y2;  
 int x;  
 double x2;  
  
 if ((int)line.X1 < line.X2)  
 {  
 x = (int)line.X1;  
 x2 = line.X2;  
 y = (int)line.Y1;  
 y2 = line.Y2;  
 }  
 else  
 {  
 x2 = line.X1;  
 x = (int)line.X2;  
 y2 = line.Y1;  
 y = (int)line.Y2;  
 }  
  
 ConditionCheker isLineEnded;  
  
 int xCrement = 1;  
 int yCrement;  
  
 if (x == x2)  
 {  
 xCrement = 0;  
 }  
  
 if (y > y2)  
 {  
 isLineEnded = () => y <= y2 && x >= x2;  
 yCrement = -1;  
 }  
 else  
 {  
 isLineEnded = () => y >= y2 && x >= x2;  
 yCrement = 1;  
 }  
  
 while (!isLineEnded())  
 {  
 try  
 {  
 FillPixel(x, y);  
 }  
 catch (Exception ex)  
 {  
 MessageBox.Show(ex.Message);  
 break;  
 }  
  
 err += delta;  
 if (err > 0)  
 {  
 y += yCrement;  
 err--;  
 }  
  
 x += xCrement;  
 }  
 }  
  
 private void DrawEllipse(int xOffset, int yOffset, int a, int b, params Quarter[] quarters)  
 {  
 DrawPixelInQuarters drawPixel = null;  
  
 foreach (var quarter in quarters)  
 {  
 switch (quarter)  
 {  
 case Quarter.**First**:  
 drawPixel += FillPixelInFirstQuarter;  
 break;  
 case Quarter.**Second**:  
 drawPixel += FillPixelInSecondQuarter;  
 break;  
 case Quarter.**Third**:  
 drawPixel += FillPixelInThirdQuarter;  
 break;  
 case Quarter.**Fourth**:  
 drawPixel += FillPixelInFourthQuarter;  
 break;  
 default:  
 drawPixel = FillPixelInFirstQuarter;  
 break;  
 }  
 }  
  
 int x = 0;  
 int y = b;  
 double delta = 4 \* Math.Pow(b \* (x + 1), 2) + Math.Pow(a \* (2 \* y - 1), 2) - 4 \* Math.Pow(a \* b, 2);  
  
 while (Math.Pow(a, 2) \* (2 \* y - 1) > 2 \* Math.Pow(b, 2) \* (x + 1))  
 {  
 drawPixel(x, y, xOffset, yOffset);  
 x++;  
  
 if (delta < 0)  
 {  
 delta += 4 \* Math.Pow(b, 2) \* (2 \* x + 3);  
 }  
 else  
 {  
 delta = delta - 8 \* Math.Pow(a, 2) \* (y - 1) + 4 \* Math.Pow(b, 2) \* (2 \* x + 3);  
 y--;  
 }  
 }  
  
 delta = Math.Pow(b \* (2 \* x + 1), 2) + 4 \* Math.Pow(a \* (y + 1), 2) - 4 \* Math.Pow(a \* b, 2);  
  
 while (y + 1 != 0)  
 {  
 drawPixel(x, y, xOffset, yOffset);  
 y--;  
  
 if (delta < 0)  
 {  
 delta += 4 \* Math.Pow(a, 2) \* (2 \* y + 3);  
 }  
 else  
 {  
 delta = delta - 8 \* Math.Pow(b, 2) \* (x + 1) + 4 \* Math.Pow(a, 2) \* (2 \* y + 3);  
 x++;  
 }  
 }  
 }  
  
 private void FillPixelInFirstQuarter(int x, int y, int xOffset, int yOffset)  
 {  
 FillPixel(xOffset + x, yOffset - y);  
 }  
  
 private void FillPixelInSecondQuarter(int x, int y, int xOffset, int yOffset)  
 {  
 FillPixel(xOffset - x, yOffset - y);  
 }  
  
 private void FillPixelInThirdQuarter(int x, int y, int xOffset, int yOffset)  
 {  
 FillPixel(xOffset - x, yOffset + y);  
 }  
  
 private void FillPixelInFourthQuarter(int x, int y, int xOffset, int yOffset)  
 {  
 FillPixel(xOffset + x, yOffset + y);  
 }  
  
 private void FillPixel(int x, int y)  
 {  
 byte red = 255;  
 byte green = 255;  
 byte blue = 255;  
 byte[] colorData = { blue, green, red, 255 };  
  
 var rect = new Int32Rect(x, y, 1, 1);  
 bitmap.WritePixels(rect, colorData, 4, 0);  
 }  
 }  
}

Результат работы программы:



Блок-схема алгоритма:

