



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики  
Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота № 4**  
з дисципліни “Основи програмування”  
тема “Бібліотеки і обробка зображень”

Виконав  
студент I курсу

групи КП-01

Пецеля Артем Володимирович  
(*прізвище, ім'я, по батькові*)

Перевірів  
“ \_\_\_\_ ” “ \_\_\_\_ ” 20\_\_ р.  
викладач

Гадиняк Руслан Анатолійович  
(*прізвище, ім'я, по батькові*)

варіант №12

Київ 2021

## Мета роботи

Реалізувати різні алгоритми редагування зображень.  
Розбити проект програми на декілька проектів у одному рішенні з використанням бібліотек класів.

## Постановка завдання

Створити консольну програму, що дозволяє виконувати редагування зображень.

### Аргументи командного рядка

Приклад аргументів:

``dotnet run {module} ./file.jpg ./out.jpg`` - аргументи обов'язкові і зберігають такий порядок, тільки цих аргументів недостатньо, після них задавати команду редагування і її параметри

- ``{module}`` - ``pixel`` або ``fast``, визначає яким саме модулем редагування змінити зображення.
- ``./file.jpg`` - перший аргумент після ``{module}`` - приклад шляху вхідного зображення
- ``./out.jpg`` - другий аргумент після ``{module}`` - приклад шляху вихідного зображення

Команди:

- Команда отримання частини зображення за координатами:  
``crop {width}x{height}+{left}+{top}`` - всі аргументи обов'язкові і зберігають такий порядок.

Приклад: ``dotnet run pixel ./file.jpg ./out.jpg crop 100x100+30+90``

- Команди методів за варіантом із таблиць 1-4 (див. Додаток А).

## Вимоги до структури коду

Розбити програму на модулі:

- **модуль обробки аргументів командного рядка** - модуль аналізує задані користувачем аргументи командного рядка і використовує інші модулі.
- **модулі редагування зображень** - містять функції, що на основі вхідного зображення створюють змінене зображення.
  - реалізація за допомогою стандартних функцій, матриць кольору, матриць трансформації або будь-якої графічної бібліотеки
  - реалізація за допомогою піксельних змін

Створити бібліотеки:

- **ProgbaseLab.ImageEditor.Common** - містить контракт модулів редагування
- **ProgbaseLab.ImageEditor.Pixel** - містить модуль редагування зображень пікселями
- **ProgbaseLab.ImageEditor.Fast** - містить модуль редагування зображень стандартними функціями або з використанням інших графічних бібліотек

Підключити і використати бібліотеки у проекті консольної програми.

# Текст коду програми

## Program.cs

```
using System;
using System.Drawing;
using ProgbaseLab.ImageEditor.Pixel;
using ProgbaseLab.ImageEditor.Fast;
using ProgbaseLab.ImageEditor.Common;
using System.Diagnostics;

namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                ArgumentProcessor.Run(args);
            }
            catch (Exception ex)
            {
                Console.Error.WriteLine($"Error: {ex.Message}");
            }
        }
    }
    static class ArgumentProcessor
    {
        public static void Run(string[] args)
        {
            ProgramArguments programArguments = ParseArguments(args);

            switch (programArguments.operation)
            {
                case "crop":
                {
                    ProcessCrop(programArguments);
                    break;
                }
                case "rotate180":
                {
                    ProcessRotate180(programArguments);
                    break;
                }
                case "removeRed":
                {
                    ProcessRemoveRed(programArguments);
                    break;
                }
                case "grayscale":
                {
                    ProcessGrayscale(programArguments);
                    break;
                }
                case "changeBrightness":
                {
                    ProcessChangeBrighness(programArguments);
                    break;
                }
            }
        }
        private static void ProcessCrop(ProgramArguments args)
        {
            if (args.otherArguments.Length != 1)
            {
                throw new ArgumentException("Invalid number of parameters in crop command");
            }
        }
    }
}
```

```

    }

    IImageEditor editor = ChooseEditor(args.module);

    Rectangle rectangle = ParseRectangle(args.otherArguments[0]);

    Stopwatch stopwatch = new Stopwatch();

    Bitmap bmp = new Bitmap(args.inputFile);

    stopwatch.Start();
    Bitmap result = editor.Crop(bmp, rectangle.left, rectangle.top, rectangle.width, rectangle.height);
    stopwatch.Stop();

    Console.WriteLine($"Operation {args.operation} done in {stopwatch.ElapsedMilliseconds} ms");
    result.Save(args.outputFile);
}
struct Rectangle
{
    public int width;
    public int height;
    public int left;
    public int top;
}
private static Rectangle ParseRectangle(string source)
{
    string[] widthAndOther = source.Split('x');
    if (widthAndOther.Length != 2)
    {
        throw new ArgumentException("Invalid rectangle format. Should be: \"{width}x{height}+{left}+{top}\"");
    }
    if (!int.TryParse(widthAndOther[0], out int width))
    {
        throw new ArgumentException("Width should be integer");
    }
    string[] heightAndPosition = widthAndOther[1].Split("+");
    if (heightAndPosition.Length != 3)
    {
        throw new ArgumentException("Invalid rectangle format. Should be: \"{width}x{height}+{left}+{top}\"");
    }
    if (!int.TryParse(heightAndPosition[0], out int height))
    {
        throw new ArgumentException("Height should be integer");
    }
    if (!int.TryParse(heightAndPosition[1], out int left))
    {
        throw new ArgumentException("Left coordinate should be integer");
    }
    if (!int.TryParse(heightAndPosition[2], out int top))
    {
        throw new ArgumentException("Top coordinate should be integer");
    }
    return new Rectangle()
    {
        width = width,
        height = height,
        left = left,
        top = top
    };
}
private static void ProcessRotate180(ProgramArguments args)
{
    if (args.otherArguments.Length != 0)
    {
        throw new ArgumentException($"Operation should not have any arguments. Got: {args.otherArguments.Length}");
    }

    IImageEditor editor = ChooseEditor(args.module);

    Stopwatch stopwatch = new Stopwatch();

```

```

        Bitmap bmp = new Bitmap(args.inputFile);

        stopwatch.Start();
        Bitmap result = editor.Rotate180(bmp);
        stopwatch.Stop();

        Console.WriteLine($"Operation {args.operation} done in {stopwatch.ElapsedMilliseconds} ms");
        result.Save(args.outputFile);
    }
    private static void ProcessRemoveRed(ProgramArguments args)
    {
        if (args.otherArguments.Length != 0)
        {
            throw new ArgumentException($"Operation should not have any arguments. Got: {args.otherArguments.Length}");
        }

        IImageEditor editor = ChooseEditor(args.module);

        Stopwatch stopwatch = new Stopwatch();

        Bitmap bmp = new Bitmap(args.inputFile);

        stopwatch.Start();
        Bitmap result = editor.RemoveRed(bmp);
        stopwatch.Stop();

        Console.WriteLine($"Operation {args.operation} done in {stopwatch.ElapsedMilliseconds} ms");
        result.Save(args.outputFile);
    }
    private static void ProcessGrayscale(ProgramArguments args)
    {
        if (args.otherArguments.Length != 0)
        {
            throw new ArgumentException($"Operation should not have any arguments. Got: {args.otherArguments.Length}");
        }

        IImageEditor editor = ChooseEditor(args.module);

        Stopwatch stopwatch = new Stopwatch();

        Bitmap bmp = new Bitmap(args.inputFile);

        stopwatch.Start();
        Bitmap result = editor.Grayscale(bmp);
        stopwatch.Stop();

        Console.WriteLine($"Operation {args.operation} done in {stopwatch.ElapsedMilliseconds} ms");
        result.Save(args.outputFile);
    }
    private static void ProcessChangeBrighness(ProgramArguments args)
    {
        if (args.otherArguments.Length != 1)
        {
            throw new ArgumentException($"Operation should have 1 argument. Got: {args.otherArguments.Length}");
        }
        int brightnessValue = ParseBrightness(args.otherArguments[0]);

        IImageEditor editor = ChooseEditor(args.module);

        Stopwatch stopwatch = new Stopwatch();

        Bitmap bmp = new Bitmap(args.inputFile);

        stopwatch.Start();
        Bitmap result = editor.ChangeBrightness(bmp, brightnessValue);
        stopwatch.Stop();

        Console.WriteLine($"Operation {args.operation} done in {stopwatch.ElapsedMilliseconds} ms");
        result.Save(args.outputFile);
    }
    private static int ParseBrightness(string source)

```

```

{
    if (!int.TryParse(source, out int brightnessValue))
    {
        throw new ArgumentException("Brightness value must be integer");
    }
    if (brightnessValue < -100 || brightnessValue > 100)
    {
        throw new ArgumentException($"Brightness value should be in range [-100; 100]. Got {brightnessValue}");
    }
    return brightnessValue;
}
struct ProgramArguments
{
    public string module;
    public string inputFile;
    public string outputFile;
    public string operation;
    public string[] otherArguments;
}
private static ProgramArguments ParseArguments(string[] args)
{
    ValidateArgumentLength(args.Length);

    ValidateModule(args[0]);
    string module = args[0];

    ValidateInputFile(args[1]);
    string inputFile = args[1];

    string outputFile = args[2];

    ValidateOperation(args[3]);
    string operation = args[3];

    string[] otherArguments = new string[args.Length - 4];
    for (int i = 0; i < otherArguments.Length; i++)
    {
        otherArguments[i] = args[i + 4];
    }
    ProgramArguments programArguments = new ProgramArguments
    {
        module = module,
        inputFile = inputFile,
        outputFile = outputFile,
        operation = operation,
        otherArguments = otherArguments
    };
    return programArguments;
}
private static void ValidateArgumentLength(int length)
{
    if (length < 4)
    {
        throw new ArgumentException($"Number of arguments must be more than 3. Number of entered arguments: {length}");
    }
}
private static void ValidateModule(string module)
{
    string[] modules = new string[] { "pixel", "fast" };
    for (int i = 0; i < modules.Length; i++)
    {
        if (modules[i] == module)
        {
            return;
        }
    }
    throw new ArgumentException($"There is no such module: {module}");
}
private static void ValidateInputFile(string inputFile)
{
    if (!System.IO.File.Exists(inputFile))

```

```

        {
            throw new ArgumentException($"Such file does not exist: {inputFile}");
        }
    }
    private static void ValidateOperation(string operation)
    {
        string[] operations = new string[] { "crop", "rotate180", "removeRed", "grayscale", "changeBrightness" };
        for (int i = 0; i < operations.Length; i++)
        {
            if (operations[i] == operation)
            {
                return;
            }
        }
        throw new ArgumentException($"There is no such operation: {operation}");
    }
    private static IImageEditor ChooseEditor(string module)
    {
        return module == "pixel" ? new PixelImageEditor() : new FastImageEditor();
    }
}

```

## IImageEditor.cs

```

using System.Drawing;

namespace ProgbaseLab.ImageEditor.Common
{
    public interface IImageEditor
    {
        Bitmap Crop(Bitmap bmp, int left, int top, int width, int height);
        Bitmap Rotate180(Bitmap src);
        Bitmap RemoveRed(Bitmap bmp);
        Bitmap Grayscale(Bitmap bmp);
        Bitmap ChangeBrightness(Bitmap bmp, int brightnessValue);
    }
}

```

## FastImageEditor.cs

```

using System;
using System.Drawing;
using System.Drawing.Imaging;
using ProgbaseLab.ImageEditor.Common;
using OpenCvSharp;
using OpenCvSharp.Extensions;

namespace ProgbaseLab.ImageEditor.Fast
{
    public class FastImageEditor : IImageEditor
    {
        public FastImageEditor()
        {
        }

        public Bitmap ChangeBrightness(Bitmap bmp, int brightnessValue)
        {
            Bitmap newBitmap = new Bitmap(bmp.Width, bmp.Height);
            Graphics g = Graphics.FromImage(newBitmap);

            ColorMatrix colorMatrix = new ColorMatrix(CreateBrightnessMatrix(brightnessValue));

            ImageAttributes attributes = new ImageAttributes();

            attributes.SetColorMatrix(colorMatrix);
        }
    }
}

```



```

        g.DrawImage(bmp, new Rectangle(0, 0, bmp.Width, bmp.Height), 0, 0, bmp.Width, bmp.Height, GraphicsUnit.Pixel, attributes);

        attributes.Dispose();
        g.Dispose();
        return new Bitmap();
    }

    private float[][] CreateBrightnessMatrix(int brightness)
    {
        float scale = (float)(brightness / 100.0); // 0..1
        float darkness = 1; // 0..1
        if (scale < 0)
        {
            darkness = 1 + scale;
            scale = 0;
        }

        return new float[][]
        {
            new float[] {darkness, 0, 0, 0, 0},
            new float[] {0, darkness, 0, 0, 0},
            new float[] {0, 0, darkness, 0, 0},
            new float[] {0, 0, 0, 1, 0},
            new float[] {scale, scale, scale, 0, 1}
        };
    }

    public Bitmap Crop(Bitmap bmp, int left, int top, int width, int height)
    {
        Mat source = BitmapConverter.ToMat(bmp);
        Rect rectCrop = new Rect(left, top, width, height);

        Mat croppedImage = new Mat(source, rectCrop);
        return BitmapConverter.ToBitmap(croppedImage);
    }

    public Bitmap Grayscale(Bitmap bmp)
    {
        Mat source = BitmapConverter.ToMat(bmp);
        Mat result = new Mat();

        Cv2.CvtColor(source, result, ColorConversionCodes.RGB2GRAY);

        return BitmapConverter.ToBitmap(result);
    }

    public Bitmap RemoveRed(Bitmap bmp)
    {
        Mat source = BitmapConverter.ToMat(bmp);
        Mat[] channels = Cv2.Split(source);
        channels[2].SetTo(0);
        Mat result = new Mat();
        Cv2.Merge(channels, result);
        return BitmapConverter.ToBitmap(result);
    }

    public Bitmap Rotate180(Bitmap src)
    {
        Mat source = BitmapConverter.ToMat(src);
        Mat result = new Mat();

        Point2f center = new Point2f(source.Width / 2, source.Height / 2);
        Mat matrix = Cv2.GetRotationMatrix2D(center, 180, 1);
        Cv2.WarpAffine(source, result, matrix, source.Size());
        return BitmapConverter.ToBitmap(result);
    }
}

```

## PixelImageEditor.cs

```
using System;
using System.Drawing;
using ProgbaseLab.ImageEditor.Common;

namespace ProgbaseLab.ImageEditor.Pixel
{
    public class PixelImageEditor : IImageEditor
    {
        public PixelImageEditor()
        {
        }

        public Bitmap Crop(Bitmap bmp, int left, int top, int width, int height)
        {
            Rectangle rect = new Rectangle(left, top, width, height);
            ValidateImageSize(bmp, rect);

            Bitmap destination = new Bitmap(width, height);
            for (int x = 0; x < rect.Width; x++)
            {
                for (int y = 0; y < rect.Height; y++)
                {
                    Color color = bmp.GetPixel(x + rect.Left, y + rect.Top);
                    destination.SetPixel(x, y, color);
                }
            }
            return destination;
        }

        private void ValidateImageSize(Bitmap bmp, Rectangle rect)
        {
            if (rect.Top < 0 || rect.Bottom > bmp.Height || rect.Left < 0 || rect.Right > bmp.Width)
            {
                throw new Exception("Crop options is out of image bounds");
            }
        }

        public Bitmap Rotate180(Bitmap scr)
        {
            Bitmap targetBitmap = new Bitmap(scr.Width, scr.Height);
            for (int y = 0; y < targetBitmap.Height; y++)
            {
                for (int x = 0; x < targetBitmap.Width; x++)
                {
                    Color color = scr.GetPixel(x, y);
                    targetBitmap.SetPixel(scr.Width - x - 1, scr.Height - y - 1, color);
                }
            }
            return targetBitmap;
        }

        public Bitmap RemoveRed(Bitmap bmp)
        {
            Bitmap targetBitmap = new Bitmap(bmp.Width, bmp.Height);
            for (int y = 0; y < targetBitmap.Height; y++)
            {
                for (int x = 0; x < targetBitmap.Width; x++)
                {
                    Color color = bmp.GetPixel(x, y);
                    Color newColor = Color.FromArgb(255, 0, color.G, color.B);
                    targetBitmap.SetPixel(x, y, newColor);
                }
            }
            return targetBitmap;
        }

        public Bitmap Grayscale(Bitmap bmp)
        {
            for (int y = 0; y < bmp.Height; y++)
            {
                for (int x = 0; x < bmp.Width; x++)
                {
                    Color color = bmp.GetPixel(x, y);
                    int yLinear = (int)(0.2126 * color.R + 0.7152 * color.G + 0.0722 * color.B);
                }
            }
        }
    }
}
```

```

        Color newColor = Color.FromArgb(255, yLinear, yLinear, yLinear);
        bmp.SetPixel(x, y, newColor);
    }
}
return bmp;
}
public Bitmap ChangeBrightness(Bitmap bmp, int brightnessValue)
{
    Bitmap targetBmp = new Bitmap(bmp.Width, bmp.Height);
    float[][] matrix = CreateBrightnessMatrix(brightnessValue);
    for (int y = 0; y < bmp.Height; y++)
    {
        for (int x = 0; x < bmp.Width; x++)
        {
            Color oldColor = bmp.GetPixel(x, y);
            Color newColor = ApplyFilter(oldColor, matrix);
            targetBmp.SetPixel(x, y, newColor);
        }
    }
    return targetBmp;
}
private Color ApplyFilter(Color oldColor, float[][] filter)
{
    int red = (int)(oldColor.R * filter[0][0] + oldColor.G * filter[1][0] + oldColor.B * filter[2][0] + 255 * filter[4][0]);
    int green = (int)(oldColor.R * filter[0][1] + oldColor.G * filter[1][1] + oldColor.B * filter[2][1] + 255 * filter[4][1]);
    int blue = (int)(oldColor.R * filter[0][2] + oldColor.G * filter[1][2] + oldColor.B * filter[2][2] + 255 * filter[4][2]);

    int r = Math.Min(Math.Max(red, 0), 255);
    int g = Math.Min(Math.Max(green, 0), 255);
    int b = Math.Min(Math.Max(blue, 0), 255);

    return Color.FromArgb(255, r, g, b);
}
private float[][] CreateBrightnessMatrix(int brightness)
{
    float scale = (float)(brightness / 100.0);
    float darkness = 1;
    if (scale < 0)
    {
        darkness = 1 + scale;
        scale = 0;
    }



    return new float[][]
    {
        new float[] {darkness, 0, 0, 0, 0},
        new float[] {0, darkness, 0, 0, 0},
        new float[] {0, 0, darkness, 0, 0},
        new float[] {0, 0, 0, 1, 0},
        new float[] {scale, scale, scale, 0, 1}
    };
}
}
}

```

## Приклади результатів

### Приклад 1. Crop

Оригінал і очікуваний результат:

Оригінал	Paint.net
	

Порівняння реалізацій:



PS D:\progbase\labs\second_sem\lab4\ConsoleApp> dotnet run fast ./example.jpg ./out.jpg crop 100x100+500+300 Operation crop done in 21 ms
PS D:\progbase\labs\second_sem\lab4\ConsoleApp> dotnet run pixel ./example.jpg ./out.jpg crop 100x100+500+300 Operation crop done in 8 ms

Результати:

Fast	Pixel
	

### Приклад 2. Rotate 180

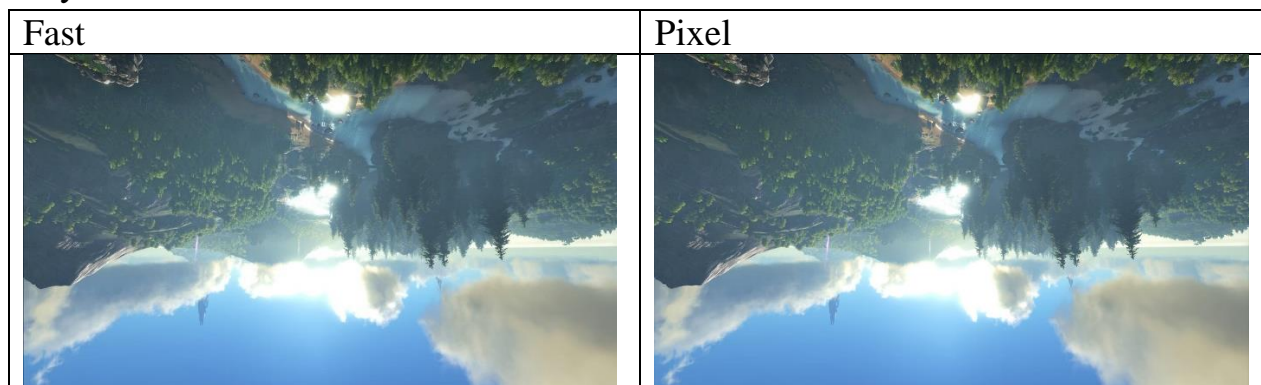
Оригінал і очікуваний результат:

Оригінал	Paint.net
	

### Порівняння реалізацій:

PS D:\progbase\labs\second_sem\lab4\ConsoleApp> dotnet run fast ./example.jpg ./out.jpg rotate180 Operation rotate180 done in 38 ms
PS D:\progbase\labs\second_sem\lab4\ConsoleApp> dotnet run pixel ./example.jpg ./out.jpg rotate180 Operation rotate180 done in 1109 ms

### Результати:



### Приклад 3. Remove red

Оригінал і очікуваний результат:





### Порівняння реалізацій:

PS D:\progbase\labs\second_sem\lab4\ConsoleApp> dotnet run fast ./example.jpg ./out.jpg removeRed Operation removeRed done in 35 ms
PS D:\progbase\labs\second_sem\lab4\ConsoleApp> dotnet run pixel ./example.jpg ./out.jpg removeRed Operation removeRed done in 981 ms





Результати:

Fast	Pixel
	

Приклад 4. Grayscale



Оригінал і очікуваний результат:

Оригінал	Paint.net
	

Порівняння реалізацій:



PS D:\progbase\labs\second_sem\lab4\ConsoleApp> dotnet run fast ./example.jpg ./out.jpg grayscale Operation grayscale done in 35 ms
PS D:\progbase\labs\second_sem\lab4\ConsoleApp> dotnet run pixel ./example.jpg ./out.jpg grayscale Operation grayscale done in 1360 ms

Результати:

Fast	Pixel
	

**Приклад 5. Brightness**

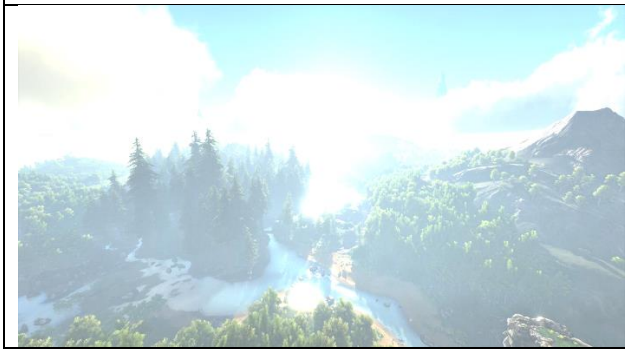

Оригінал і очікуваний результат:

Оригінал	Paint.net
	

Порівняння реалізацій:

PS D:\progbase\labs\second_sem\lab4\ConsoleApp> dotnet run fast ./example.jpg ./out.jpg changeBrightness 40 Operation changeBrightness done in 19 ms
PS D:\progbase\labs\second_sem\lab4\ConsoleApp> dotnet run pixel ./example.jpg ./out.jpg changeBrightness 40 Operation changeBrightness done in 1094 ms

Результати:

Fast	Pixel
	

## **Висновки**

Виконавши дану лабораторну роботу було реалізовано модуль редагування зображення ефектами згідно варіанту. Код було логічно розбито на декілька бібліотек.

Компіляція коду відбувалась за допомогою утиліти `dotnet`.