LAB: STREAMS, FILES AND DIRECTORIES

You can check your solutions in Judge

NOTE: For these problems follow the instructions for the required methods and classes. For each problem **submit zipped folder** of your project **without** the **"bin"** and **"obj"** folders in it.

FILE OPERATIONS

1. ODD LINES

Write a program that reads a text file (e. g. **`input.txt`**) and writes every **odd** line in another file. Line numbers **start from 0**.

Note: use the following structure:

```
namespace OddLines
{
    public class OddLines
    {
        static void Main()
        {
            string inputFilePath = @"..\..\..\Files\input.txt";
            string outputFilePath = @"..\..\..\Files\output.txt";

            ExtractOddLines(inputFilePath, outputFilePath);
        }

        public static void ExtractOddLines(string inputFilePath, string outputFilePath)
        {
            // TODO: write your code here…
        }
    }
}
```

EXAMPLES

input.txt	output.txt
Two households, both alike in dignity, In fair Verona, where we lay our scene, From ancient grudge break to new mutiny, Where civil blood makes civil hands unclean. From forth the fatal loins of these two foes A pair of star-cross'd lovers take their life; Whose misadventured piteous overthrows Do with their death bury their parents' strife.	In fair Verona, where we lay our scene, Where civil blood makes civil hands unclean. A pair of star-cross'd lovers take their life; Do with their death bury their parents' strife

2. LINE NUMBERS

Write a program that **reads a text file** (e. g. **input.txt**) and **inserts line numbers** in front of each of its lines. The result should be **written to another text file** (e. g. **output.txt**). Use **StreamReader** and **StreamWriter**.

NOTE: use the following structure:

```
namespace LineNumbers
{
    public class LineNumbers
    {
        static void Main()
        {
            string inputPath = @"..\..\..\Files\input.txt";
            string outputPath = @"..\..\..\Files\output.txt";

            RewriteFileWithLineNumbers(inputPath, outputPath);
        }

        public static void RewriteFileWithLineNumbers(string inputFilePath, string
outputFilePath)
        {
            // TODO: write your code here…
        }
    }
}
```

EXAMPLES

input.txt	output.txt

`Two households, both alike in dignity,`
`In fair Verona, where we lay our scene,`
`From ancient grudge break to new mutiny,`
`Where civil blood makes civil hands unclean.`
`From forth the fatal loins of these two foes`
`A pair of star-cross'd lovers take their life;`
`Whose misadventured piteous overthrows`
`Do with their death bury their parents' strife.`
	`1. Two households, both alike in dignity,`
	`2. In fair Verona, where we lay our scene,`
	`3. From ancient grudge break to new mutiny,`
	`4. Where civil blood makes civil hands unclean.`
	`5. From forth the fatal loins of these two foes`
	`6. A pair of star-cross'd lovers take their life;`
	`7. Whose misadventured piteous overthrows`
	`8. Do with their death bury their parents' strife.`

3. WORD COUNT

Write a program that **reads a list of words** from a given file (e. g. **`words.txt`**) and finds how many times each of the words occurs in another file (e. g. **`text.txt`**). Matching should be **case-insensitive**. The **result** should be written to an output text file (e. g. **`output.txt`**). Sort the words by frequency in descending order.

NOTE: use the following structure:

```
namespace WordCount
{
    public class WordCount
    {
        static void Main()
        {
            string wordPath = @"..\..\..\Files\words.txt";
            string textPath = @"..\..\..\Files\text.txt";
            string outputPath = @"..\..\..\Files\output.txt";

            CalculateWordCounts(wordPath, textPath, outputPath);
        }

        public static void CalculateWordCounts(string wordsFilePath, string textFilePath,
string outputFilePath)
        {
            // TODO: write your code here…
        }
    }
}
```

EXAMPLES

words.txt	text.txt	output.txt

quick is fault	-I was quick to judge him, but it wasn't his fault. -Is this some kind of joke?! Is it? -Quick, hide here…It is safer.	is - 3 quick - 2 fault - 1

4. MERGE FILES

Write a program that reads the contents of **two input text files** (e. g. **`input1.txt`** and **`input2.txt`**) and **merges them line by line** together into a third text file (e. g. **`output.txt`**). The merging is done as follows:

- Line 1 from input1.txt
- Line 1 from input2.txt
- Line 2 from input1.txt
- Line 2 from input2.txt
- …

If some of the files have more lines than the other, append at the end of the output the lines, which cannot be matched with the other file.

NOTE: use the following structure:

```
namespace MergeFiles
{
    public class MergeFiles
    {
        static void Main()
        {
            var firstInputFilePath = @"..\..\..\Files\input1.txt";
            var secondInputFilePath = @"..\..\..\Files\input2.txt";
            var outputFilePath = @"..\..\..\Files\output.txt";

            MergeTextFiles(firstInputFilePath, secondInputFilePath, outputFilePath);
        }

        public static void MergeTextFiles(string firstInputFilePath, string
secondInputFilePath, string outputFilePath)
        {
            // TODO: write your code here…
        }
    }
}
```

EXAMPLES

input1.txt	input2.txt	output.txt
1 3 5	2 4 6 7	1 2 3 4 5

		6 7

Directory Operations

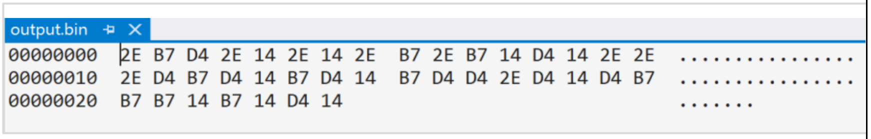
5. Extract Special Bytes

You are given a binary file (e. g. **`example.png`**) and a text file (e. g. **`bytes.txt`**), holding a list of bytes in the range [0…255]. Write a program to extract occurrences of all given bytes from the input file to an output binary file (e. g. **`output.bin`**).

NOTE: use the following structure:

```
namespace ExtractSpecialBytes
{
    public class ExtractSpecialBytes
    {
        static void Main()
        {
            string binaryFilePath = @"..\..\..\Files\example.png";
            string bytesFilePath = @"..\..\..\Files\bytes.txt";
            string outputPath = @"..\..\..\Files\output.bin";

            ExtractBytesFromBinaryFile(binaryFilePath, bytesFilePath, outputPath);
        }

        public static void ExtractBytesFromBinaryFile(string binaryFilePath, string
bytesFilePath, string outputPath)
        {
            // TODO: write your code here…
        }
    }
}
```

Examples

example.png	bytes.txt	output.bin
	20 46 183 212	

output.bin

```
00000000  2E B7 D4 2E 14 2E 14 2E  B7 2E B7 14 D4 14 2E 2E  ................
00000010  2E D4 B7 D4 14 B7 D4 14  B7 D4 D4 2E D4 14 D4 B7  ................
00000020  B7 B7 14 B7 14 D4 14                               .......
```

6. Split / Merge Binary Files

You are given an input binary file (e. g. **`example.png`**). Write a program to **split it into two equal-sized files** (e. g. **`part-1.bin`** and **`part-2.bin`**). When the input file size is an odd number, the first part should be 1 byte bigger than the second.

After splitting the input file, **join the obtained files** into a new file (e. g. **`example-joined.png`**). The obtained result file should be the same as the initial input file.

NOTE: use the following structure:

```
namespace SplitMergeBinaryFile
{
    public class SplitMergeBinaryFile
    {
        static void Main()
        {
            string sourceFilePath = @"..\..\..\Files\example.png";
            string joinedFilePath = @"..\..\..\Files\example-joined.png";
            string partOnePath = @"..\..\..\Files\part-1.bin";
            string partTwoPath = @"..\..\..\Files\part-2.bin";

            SplitBinaryFile(sourceFilePath, partOnePath, partTwoPath);
            MergeBinaryFiles(partOnePath, partTwoPath, joinedFilePath);
        }

        public static void SplitBinaryFile(string sourceFilePath, string partOneFilePath, 
string partTwoFilePath)
        {
            // TODO: write your code here…
        }

        public static void MergeBinaryFiles(string partOneFilePath, string partTwoFilePath, 
string joinedFilePath)
        {
            // TODO: write your code here…
        }
    }
}
```

DIRECTORY OPERATIONS

7. FOLDER SIZE

You are given a folder in the file system (e. g. **`TestFolder`**). Calculate the size of all files in the folder and its **subfolders.** The result should be written to another text (e. g. **`output.txt`**) file in **kilobytes**.

NOTE: use the following structure:

```
namespace FolderSize
{
    public class FolderSize
    {
        static void Main()
        {
            string folderPath = @"..\..\..\Files\TestFolder";
            string outputPath = @"..\..\..\Files\output.txt";

            GetFolderSize(folderPath, outputPath);
        }

        public static void GetFolderSize(string folderPath, string outputFilePath)
        {
            // TODO: write your code here…
        }
    }
}
```

EXAMPLES

output.txt
`0.0869140625 KB`

Exercise: Streams, Files and Directories

- You can check your solutions in Judge
- Ask your questions here https://www.slido.com/ by entering the code **#csharp-advanced**

NOTE: For these problems follow the instructions for the required methods and classes. For each problem **submit zipped folder** of your project **without** the **"bin"** and **"obj"** folders in it.

1. Even Lines

Write a program that reads a **text** file (e. g. **text.txt)** and prints on the console its **even lines**. Line numbers start from 0. Use **StreamReader**. Before you print the result, replace **{'-', ', ', '. ', '! ', '? '}** with **'@'** and **reverse the order of the words**.

Note: use the following structure:

```
namespace EvenLines
{
    using System;

    public class EvenLines
    {
        static void Main()
        {
            string inputFilePath = @"..\..\..\text.txt";

            Console.WriteLine(ProcessLines(inputFilePath));
        }

        public static string ProcessLines(string inputFilePath)
        {
        }
    }
}
```

Examples

Input file: text.txt	**Output (at the console)**
-I was quick to judge him, but it wasn't his fault. -Is this some kind of joke?! Is it? -Quick, hide here. It is safer.	fault@ his wasn't it but him@ judge to quick was @I safer@ is It here@ hide @Quick@

2. Line Numbers

Write a program that **reads** a **text file** (e. g. **text.txt**) and inserts **line numbers** in front of **each** of its **lines and count all the letters and punctuation marks**. The result should be **written** to **another** text file (e. g. **output.txt**). Use the static class **File** to read and write all the lines of the input and output files.

Note: use the following structure:

```
public class LineNumbers
{
    public static void ProcessLines(string inputFilePath, string outputFilePath)
    {
    }
}
```

Examples

text.txt	output.txt
-I was quick to judge him, but it wasn't his fault. -Is this some kind of joke?! Is it? -Quick, hide here. It is safer.	Line 1: -I was quick to judge him, but it wasn't his fault. (37)(4) Line 2: -Is this some kind of joke?! Is it? (24)(4) Line 3: -Quick, hide here. It is safer. (22)(4)

3. Copy Binary File

Write a program that copies the contents of a binary file (e. g. **copyMe.png**) to another binary file (e. g. **copyMe-copy.png**) using **FileStream**. You are **not allowed** to use the **File** class or similar helper classes.

Note: use the following structure:

```
namespace CopyBinaryFile
{
    using System;

    public class CopyBinaryFile
    {
        static void Main()
        {
            string inputFilePath = @"..\..\..\copyMe.png";
            string outputFilePath = @"..\..\..\copyMe-copy.png";

            CopyFile(inputFilePath, outputFilePath);
        }

        public static void CopyFile(string inputFilePath, string outputFilePath)
        {
        }
    }
}
```

4. Directory Traversal

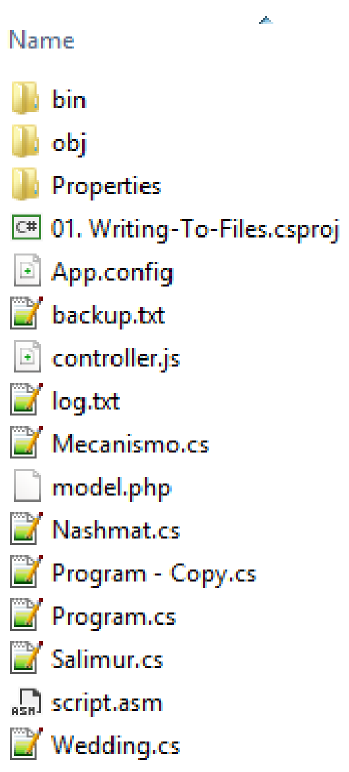
Write a program that traverses a given **directory** for **all files** with the given **extension**. Search through the **first level** of the **directory only.** Write information about each **found** file in a text file named **report.txt** and it should be saved on the **Desktop**. The files should be **grouped** by their **extension**. **Extensions** should be **ordered** by the **count** of their files **descending**, then by **name alphabetically**. **Files** under an extension should be **ordered** by their **size**. **report.txt** should be saved on the **Desktop**. Ensure the desktop path is always valid, regardless of the user.

Note: use the following structure:

```
namespace DirectoryTraversal
{
    using System;

    public class DirectoryTraversal
    {
        static void Main()
        {
            string path = Console.ReadLine();
            string reportFileName = @"\report.txt";

            string reportContent = TraverseDirectory(path);
            Console.WriteLine(reportContent);

            WriteReportToDesktop(reportContent, reportFileName);
        }

        public static string TraverseDirectory(string inputFolderPath)
        {
        }

        public static void WriteReportToDesktop(string textContent, string reportFileName)
        {
        }
    }
}
```

Examples

Input	Directory View	report.txt

.	 Name bin obj Properties 01. Writing-To-Files.csproj App.config backup.txt controller.js log.txt Mecanismo.cs model.php Nashmat.cs Program - Copy.cs Program.cs Salimur.cs script.asm Wedding.cs	.cs --Mecanismo.cs - 0.994kb --Program.cs - 1.108kb --Nashmat.cs - 3.967kb --Wedding.cs - 23.787kb --Program - Copy.cs - 35.679kb --Salimur.cs - 588.657kb .txt --backup.txt - 0.028kb --log.txt - 6.72kb .asm --script.asm - 0.028kb .config --App.config - 0.187kb .csproj --01. Writing-To-Files.csproj - 2.57kb .js --controller.js - 1635.143kb .php --model.php - 0kb

5. Copy Directory

Write a method, which **copies a directory with files** (**without its subdirectories**) to another directory. The input folder and the output folder should be given as parameters from the console. If the output folder already exists, first delete it (together with all its content).

Note: use the following structure:

```
namespace CopyDirectory
{
    using System;

    public class CopyDirectory
    {
        static void Main()
        {
            string inputPath = @$"{Console.ReadLine()}";
            string outputPath = @$"{Console.ReadLine()}";

            CopyAllFiles(inputPath, outputPath);
        }

        public static void CopyAllFiles(string inputPath, string outputPath)
        {
        }
    }
}
```

6. *Zip and Extract

Write a program that **creates** a **ZIP** file (archive), holding a given **input file,** and **extracts** the ZIP-ed file from the archive into in separate **output file**.

- Use the **copyMe.png** file from your resources as input and **zip** it into a ZIP file of your choice, e. g. **archive.zip**.
- **Extract** the file from the archive into a new file of your choice, e. g. **extracted.png**.

If your code works correctly, the input and output files should be the same.

Note: use the following structure:

```
namespace ZipAndExtract
{
    using System;
    using System.IO;

    public class ZipAndExtract
    {
        static void Main()
        {
        }
        public static void ZipFileToArchive(string inputFilePath, string zipArchiveFilePath)
        {
        }

        public static void ExtractFileFromArchive(string zipArchiveFilePath, string fileName, string outputFilePath)
        {
        }
    }
}
```

Hints

- Use the **ZipFile** class.
- The **entry** in the ZIP file should hold the **file name only** without its path.

EXERCISE: STREAMS, FILES AND DIRECTORIES

- You can check your solutions in Judge
- Ask your questions here https://www.slido.com/ by entering the code **`#csharp-advanced`**

NOTE: For these problems follow the instructions for the required methods and classes. For each problem **submit zipped folder** of your project **without** the **"`bin`"** and **"`obj`"** folders in it.

7. EVEN LINES

Write a program that reads a **text** file (e. g. **`text.txt`**) and prints on the console its **even lines**. Line numbers start from 0. Use **`StreamReader`**. Before you print the result, replace **{'-',',','.','!','?'}** with **'@'** and **reverse the order of the words**.

Note: use the following structure:

```
namespace EvenLines
{
    using System;

    public class EvenLines
    {
        static void Main()
        {
            string inputFilePath = @"..\..\..\text.txt";

            Console.WriteLine(ProcessLines(inputFilePath));
        }

        public static string ProcessLines(string inputFilePath)
        {
        }
    }
}
```

EXAMPLES

Input file: text.txt	Output (at the console)

`-I was quick to judge him, but it wasn't his fault.`
`-Is this some kind of joke?! Is it?`
`-Quick, hide here. It is safer.`
	`fault@ his wasn't it but him@ judge to quick was @I`
	`safer@ is It here@ hide @Quick@`

8. LINE NUMBERS

Write a program that **reads** a **text file** (e. g. **`text.txt`**) and inserts **line numbers** in front of **each** of its **lines and count all the letters and punctuation marks**. The result should be **written** to **another** text file (e. g. **`output.txt`**). Use the static class **`File`** to read and write all the lines of the input and output files.

Note: use the following structure:

```
public class LineNumbers
{
    public static void ProcessLines(string inputFilePath, string outputFilePath)
    {
    }
}
```

EXAMPLES

text.txt	output.txt

`-I was quick to judge him, but it wasn't his fault.`
`-Is this some kind of joke?! Is it?`
	`Line 1: -I was quick to judge him, but it wasn't his fault. (37)(4)`

`-Quick, hide here. It is safer.`	`Line 2: -Is this some kind of joke?! Is it? (24)(4)`
	`Line 3: -Quick, hide here. It is safer. (22)(4)`

9. Copy Binary File

Write a program that copies the contents of a binary file (e. g. **`copyMe.png`**) to another binary file (e. g. **`copyMe-copy.png`**) using **`FileStream`**. You are **not allowed** to use the **`File`** class or similar helper classes.

Note: use the following structure:

```
namespace CopyBinaryFile
{
    using System;

    public class CopyBinaryFile
    {
        static void Main()
        {
            string inputFilePath = @"..\..\..\copyMe.png";
            string outputFilePath = @"..\..\..\copyMe-copy.png";

            CopyFile(inputFilePath, outputFilePath);
        }

        public static void CopyFile(string inputFilePath, string outputFilePath)
        {
        }
    }
}
```

10. Directory Traversal

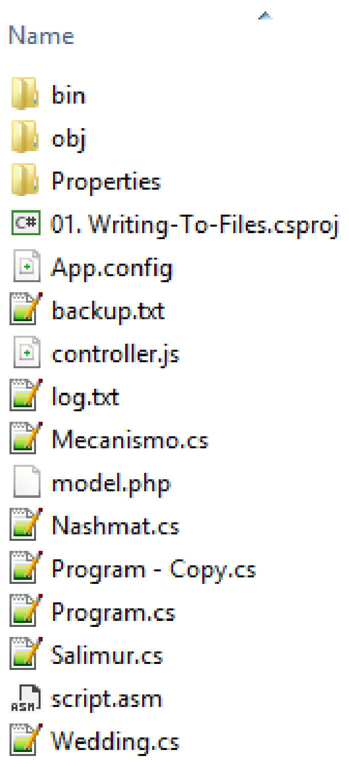
Write a program that traverses a given **directory** for **all files** with the given **extension**. Search through the **first level** of the **directory only.** Write information about each **found** file in a text file named **`report.txt`** and it should be saved on the **`Desktop`**. The files should be **grouped** by their **extension**. **Extensions** should be **ordered** by the **count** of their files **descending**, then by **name alphabetically**. **Files** under an extension should be **ordered** by their **size**. **`report.txt`** should be saved on the **`Desktop`**. Ensure the desktop path is always valid, regardless of the user.

Note: use the following structure:

```
namespace DirectoryTraversal
{
    using System;

    public class DirectoryTraversal
    {
        static void Main()
        {
            string path = Console.ReadLine();
            string reportFileName = @"\report.txt";

            string reportContent = TraverseDirectory(path);
            Console.WriteLine(reportContent);

            WriteReportToDesktop(reportContent, reportFileName);
        }

        public static string TraverseDirectory(string inputFolderPath)
        {
        }

        public static void WriteReportToDesktop(string textContent, string reportFileName)
        {
        }
    }
}
```

EXAMPLES

Input	Directory View	report.txt

.	 Name
	bin
	obj
	Properties
	01. Writing-To-Files.csproj
	App.config
	backup.txt
	controller.js
	log.txt
	Mecanismo.cs
	model.php
	Nashmat.cs
	Program - Copy.cs
	Program.cs
	Salimur.cs
	script.asm
	Wedding.cs
		`.cs`
		`--Mecanismo.cs - 0.994kb`
		`--Program.cs - 1.108kb`
		`--Nashmat.cs - 3.967kb`
		`--Wedding.cs - 23.787kb`
		`--Program - Copy.cs - 35.679kb`
		`--Salimur.cs - 588.657kb`
		`.txt`
		`--backup.txt - 0.028kb`
		`--log.txt - 6.72kb`
		`.asm`
		`--script.asm - 0.028kb`
		`.config`
		`--App.config - 0.187kb`
		`.csproj`
		`--01. Writing-To-Files.csproj - 2.57kb`
		`.js`
		`--controller.js - 1635.143kb`
		`.php`
		`--model.php - 0kb`

11. COPY DIRECTORY

Write a method, which **copies a directory with files** (**without its subdirectories**) to another directory. The input folder and the output folder should be given as parameters from the console. If the output folder already exists, first delete it (together with all its content).

Note: use the following structure:

```
namespace CopyDirectory
{
    using System;

    public class CopyDirectory
    {
        static void Main()
        {
            string inputPath = @$"{Console.ReadLine()}";
            string outputPath = @$"{Console.ReadLine()}";

            CopyAllFiles(inputPath, outputPath);
        }

        public static void CopyAllFiles(string inputPath, string outputPath)
        {
        }
    }
}
```

12. *ZIP AND EXTRACT

Write a program that **creates** a **ZIP** file (archive), holding a given **input file,** and **extracts** the ZIP-ed file from the archive into in separate **output file**.

- Use the **`copyMe.png`** file from your resources as input and **zip** it into a ZIP file of your choice, e. g. **`archive.zip`**.
- **Extract** the file from the archive into a new file of your choice, e. g. **`extracted.png`**.

If your code works correctly, the input and output files should be the same.

Note: use the following structure:

```
namespace ZipAndExtract
{
    using System;
    using System.IO;

    public class ZipAndExtract
    {
        static void Main()
        {
        }

        public static void ZipFileToArchive(string inputFilePath, string
zipArchiveFilePath)
        {
        }

        public static void ExtractFileFromArchive(string zipArchiveFilePath, string
fileName, string outputFilePath)
        {
        }
    }
}
```

HINTS

- Use the **`ZipFile`** class.
- The **entry** in the ZIP file should hold the **file name only** without its path.