# Exercise: Streams and Files

Problems for exercises and homework for the ["CSharp Advanced" course @ Software University](https://softuni.bg/courses/csharp-advanced).

## Odd Lines

Write a program that reads a **text** file and prints on the console its **odd** **lines**. Line numbers start from 0. Use **StreamReader**.

|  |  |
| --- | --- |
| **text.txt** | **Output** |
| -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 this some kind of joke?! Is it? |

## Line Numbers

Write a program that **reads** a **text** **file** and inserts **line** **numbers** in front of **each** of its **lines**. The result should be **written** to **another** text file. Use **StreamReader** in combination with **StreamWriter**.

|  |  |
| --- | --- |
| **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.  Line 2: -Is this some kind of joke?! Is it?  Line 3: -Quick, hide here…It is safer. |

## Word Count

Write a program that reads a **list** of **words** from the file **words.txt** and finds **how** **many** **times** each of the words is **contained** in another file **text.txt**. Matching should be **case-insensitive**.

Write the results in file **results.txt**. **Sort** the words by **frequency** in **descending** order. Use **StreamReader** in combination with **StreamWriter**.

|  |  |  |
| --- | --- | --- |
| **words.txt** | **text.txt** | **result.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 |

## Copy Binary File

Write a program that copies the contents of a binary file (e.g. image, video, etc.) to another using **FileStream**. You are **not allowed** to use the **File** class or similar helper classes.

## Slicing File

Write a program that takes **any** **file** and **slices** it to **n** parts. Write the following methods:

* **Slice(string sourceFile, string destinationDirectory, int parts)** - **slices** the given source file into **n** parts and **saves** them in **destinationDirectory**.

|  |  |
| --- | --- |
| **Source File** | **Destination Directory** |
| parts = 5C:\Users\Jamal\AppData\Local\Microsoft\Windows\INetCache\Content.Word\solid-logger.png | C:\Users\Jamal\AppData\Local\Microsoft\Windows\INetCache\Content.Word\sliced-png.png |

* **Assemble(List<string> files, string destinationDirectory)** - **combines** all files into one, in the **order** they are **passed**, and **saves** the result in **destinationDirectory**.

|  |  |
| --- | --- |
| **Source Files** | **Destination Directory** |
| C:\Users\Jamal\AppData\Local\Microsoft\Windows\INetCache\Content.Word\sliced-png.png | C:\Users\Jamal\AppData\Local\Microsoft\Windows\INetCache\Content.Word\assembled.png |

Use **FileStreams**. You are **not allowed** to use the **File** class or similar helper classes.

## Zipping Sliced Files

**Modify** your previous program to also **compress** the bytes while slicing parts and **decompress** them when assembling them back to the **original** file. Use **GzipStream**.

**Tip**: When getting files from directory, make sure you only get files with **.gz** extension (there might be hidden files).

|  |  |  |
| --- | --- | --- |
| **Source File** | **Compressed & Sliced** | **Decompressed & Assembled** |
| parts = 5C:\Users\Jamal\AppData\Local\Microsoft\Windows\INetCache\Content.Word\solid-logger.png |  | C:\Users\Jamal\AppData\Local\Microsoft\Windows\INetCache\Content.Word\assembled.png |

## Directory Traversal

Traverse a given **directory** for **all** **files** with the given **extension**. Search through the **first** **level** of the **directory** **only** and write information about each **found** file in **report.txt**.

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.

|  |  |  |
| --- | --- | --- |
| **Input** | **Directory View** | **report.txt** |
| . |  | .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 |

## \* Full Directory Traversal

Modify your previous program to **recursively traverse** the **sub-directories** of the starting directory as well.

## \*\* HTTP Server

Create a simple HTTP server that should be able to **receive requests** and **return appropriate responses** accordingly to the **request path**. **Read** more on the internet to see what HTTP request and response should look like. Create a web site of 3 pages:

* **1st** page should be accessible at localhost:{port}/ - (that is the root directory). That page holds just **welcome message** and **link** to the second page
* **2nd** page should be accessible at localhost:{port}/info – that page shows the **current time** and the **count of logical processors** on the machine
* **3rd** page should be an **error page** if the user tries to access any other page return as response the error page.

You are provided with some HTML files of the needed pages to make it easier for you. You are free to modify them and even use your own HTML files.

### Examples





