Technical Screening Coding Exercise

Goal:

This is an exercise to simulate our daily work: a ticket is assigned to you, and you are expected to complete the task in a timely fashion with a high-quality solution. Through this coding exercise, you can partially demonstrate the technical skills required in the job description.

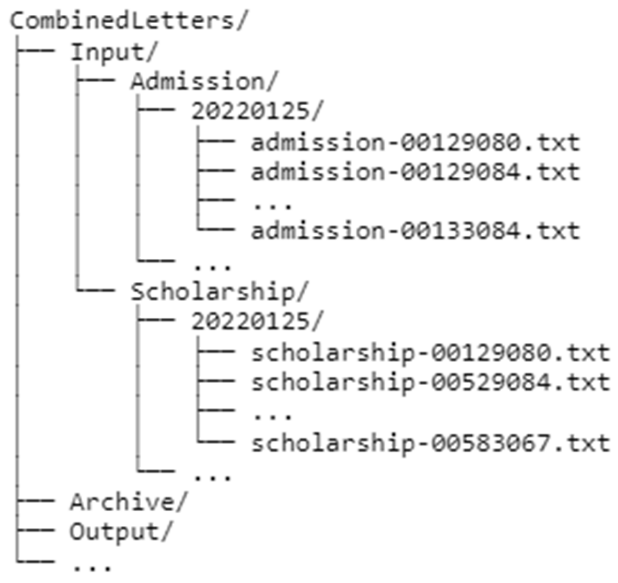
Expectation:

1. Before coding, please estimate how many hours you may need for this task.
2. Track your time during the implementation and report the actual hours you spent.
3. Create a Git repository and commit your code there.
4. Create a `README.md` file and document how to test your app.
5. Complete the task independently. Note that you are working on this coding exercise in the same time frame as other applicants, please don't share this file through internet.
6. It is OK if you are not able to complete all features, but please submit your results by the due date. Also, please review the "Task Submission Checklist" before responding to us.

Task Description:

Our department serves many other departments within our university. One of them is the Printing and Mailing Services department. They print and mail thousands of letters every day except weekends. To help with the printing process and billing process, we have built dozens of Console applications to preprocess the PDF letters before printing them.

Now assume you are going to work on a Console app to process admission letters and scholarship letters which will be sent to students. The letters come into the input folders daily from the Admissions Office and the Financial Aid Office, respectively. There could be about three thousand admission letters and about two hundred scholarship letters per day, or there could be no letters at all on certain days. The folder structure follows the diagram below.



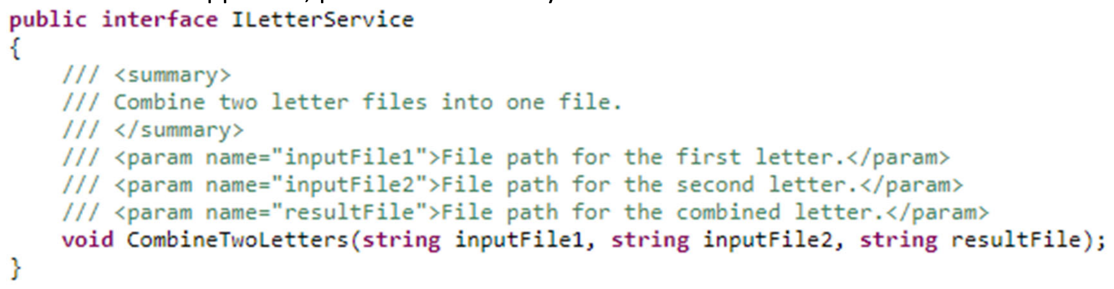
```
CombinedLetters/
├── Input/
│   ├── Admission/
│   │   ├── 20220125/
│   │   │   ├── admission-00129080.txt
│   │   │   ├── admission-00129084.txt
│   │   │   ├── ...
│   │   │   └── admission-00133084.txt
│   │   └── ...
│   └── Scholarship/
│       ├── 20220125/
│       │   ├── scholarship-00129080.txt
│       │   ├── scholarship-00529084.txt
│       │   ├── ...
│       │   └── scholarship-00583067.txt
│       └── ...
├── Archive/
├── Output/
└── ...
```

The root folder is **CombinedLetters**, which has a few fixed (cannot be deleted or renamed) subfolders: **Input**, **Archive**, **Output**. And you can create new folders if needed. In the **Input** folder, there are two fixed folders: **Admission** and **Scholarship**. These two folders contain admission letters and scholarship letters, respectively. Under the two folders, you can find dated folders which are named in the format of "**yyyyMMdd**". Each dated folder contains all letters on the day indicated in the folder name. Note that if no letters were produced on a certain day, then there won't be a dated folder for that date. For simplicity, we assume letters are in ***.txt** format and are named in the patterns of "**admission-########.txt**" or "**scholarship-########.txt**". The eight-digit strings represent students' University IDs, which uniquely identify each student in our university.

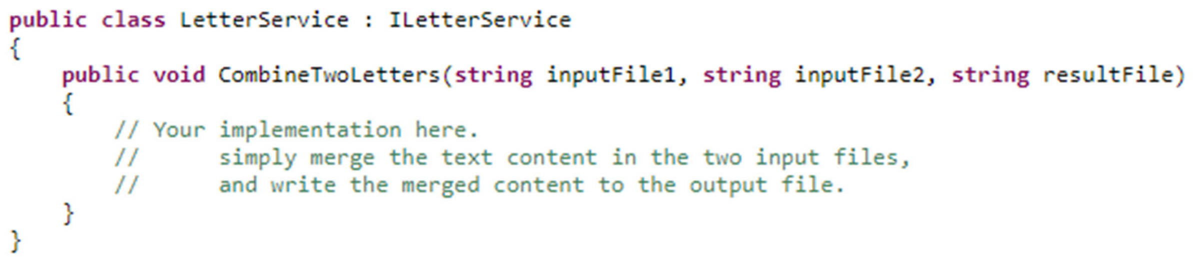
The goal of this Console app is to save mailing fees. For example, if a student has both an admission letter and a scholarship letter, then we can combine the two letters, and mail them into one envelop thus saving postage.

For this coding exercise, you are recommended to use the .NET framework in C#. The Console app has at least the following features:

1. Archive the files from the **Input** folder to the **Archive** folder.
2. Find out students who have both admission letters and scholarship letters on the day of processing. Then combine letters for each student and save all combined letters to a place in the **Output** folder. You are recommended to use the following interface and its implementation class. If you decide to take a different approach, please let us know your reasons.

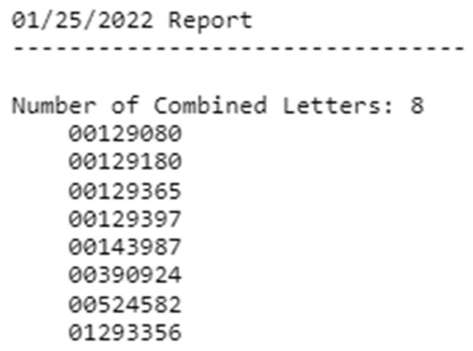


```
public interface ILetterService
{
    /// <summary>
    /// Combine two letter files into one file.
    /// </summary>
    /// <param name="inputFile1">File path for the first letter.</param>
    /// <param name="inputFile2">File path for the second letter.</param>
    /// <param name="resultFile">File path for the combined letter.</param>
    void CombineTwoLetters(string inputFile1, string inputFile2, string resultFile);
}
```



```
public class LetterService : ILetterService
{
    public void CombineTwoLetters(string inputFile1, string inputFile2, string resultFile)
    {
        // Your implementation here.
        //      simply merge the text content in the two input files,
        //      and write the merged content to the output file.
    }
}
```

3. Generate a text report in the following format, which contains the processing date, the total number of combined letters and their corresponding University IDs. The text report file should be placed in the same folder as the combined letters.



```
01/25/2022 Report
--------------------------------

Number of Combined Letters: 8
    00129080
    00129180
    00129365
    00129397
    00143987
    00390924
    00524582
    01293356
```

When you test your application, you can use any string (for example, the filename) as the content for each text file.

After you have finished implementing and testing the Console app, it will be scheduled to run once per day at 10 AM from Monday to Friday. You don't need to worry about the scheduling part. But it might be helpful that you could think about the following questions during implementation and testing.

1. What will happen if a person accidentally runs the Console app before or after the scheduled time?
2. What will happen if the Console app wasn't run previous day, and it runs today?

In the real world, you will have a set of documents before implementation. However, for this coding exercise, you don't need to consider peripheral requirements because you are not going to implement a full-fledged application. Please only focus on basic functionalities.

Task Submission Checklist:

1. The link to your Git repository.
   a. This repository must be visible to us, and it's easier if you set it public. We will review the code and the commit history.
   b. The repository must have a `README.md` file which documents how to test your app.
2. The numbers of the estimated hours and the actual hours. You can optionally write a short paragraph with details and explanations.
3. Comments (Optional). You can describe the following:
   a. The assumptions you made during implementation.
   b. The problems you came across, and how you solved them.
   c. The highlights in your code or coding practice that you want us to notice.
   d. Anything else.