TASK

C# File I/O, which stands for Input/Output, is a crucial aspect of C# programming that enables the interaction between computer programs and external files. In essence, it allows programs to read data from files, write data to files, and manipulate file-related operations. This capability is fundamental in a wide range of software applications, from simple text editors to complex database systems.

File I/O serves several key purposes:

1. Data Storage: File I/O enables the permanent storage and retrieval of data. Programs can save information to files for later use, ensuring data persistence even after the program terminates.

2. Data Import/Export: It facilitates the exchange of data between applications. Data can be imported from external sources, manipulated within the program, and then exported back to a file in a different format.
3. Configuration Management: Many programs use configuration files to store settings and preferences. File I/O allows applications to read and update these configurations, providing customization options to users.

4. Logging and Debugging: Developers use file I/O to log program activities, errors, and debugging information. Log files are invaluable for diagnosing issues and monitoring software performance.

5. Database Interaction: Database systems often use files to store structured data. File I/O is an integral part of reading and writing data to and from databases.

6. Text File Processing: Text files are commonly used for storing structured or unstructured data. File I/O enables the parsing, manipulation, and extraction of information from text files.

7. File Management: Programs can create, delete, copy, move, and rename files and directories on the file system using File I/O, allowing for effective file management.

Step 1: Create a New Console Application Project

1.1. Open your integrated development environment (IDE), such as Visual Studio.
1.2. Create a new project by selecting "File" > "New" > "Project" (or equivalent) from the IDE's menu.
1.3. Choose the "Console Application" template for C# and provide a suitable project name, like "FileIOOperations."

Step 2: Implement File Creation and Writing

2.1. Within the newly created project, locate the main program file, typically named "Program.cs.".

2.2. Create a new text file named "sample.txt" in the project directory using the following code:

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        // Specify the file path
        string filePath = "sample.txt";

        // Create the file if it doesn't exist, or overwrite it if
it does
        using (StreamWriter sw = new StreamWriter(filePath))
        {
            // Write the text to the file
            sw.WriteLine("C# File I/O allows you to read and write
data to files. It's a crucial part of many applications.");
        }
    }
}
```

2.3. Save the changes.

Step 3: Implement File Reading and Display

3.1. Modify the program to read the contents of "sample.txt" and display them on the console:

```
// Read and display the contents of the file
string fileContents = File.ReadAllText(filePath);
Console.WriteLine("Contents of sample.txt:");
Console.WriteLine(fileContents);
```

Step 4: Create "numbers.txt" and Write Numbers

4.1. Create a new text file named "numbers.txt" in the project directory:

```
// Specify the file path for numbers.txt
string numbersFilePath = "numbers.txt";

// Create or overwrite the file
using (StreamWriter sw = new StreamWriter(numbersFilePath))
{
    // Write numbers from 1 to 10, one per line
    for (int i = 1; i <= 10; i++)
    {
        sw.WriteLine(i);
    }
}
```

4.2. Save the changes.

Step 5: Calculate and Display Sum of Numbers

5.1. Implement a function to calculate and display the sum of the numbers in "numbers.txt":

```
// Read the numbers from numbers.txt
string[] numberStrings = File.ReadAllLines(numbersFilePath);

int sum = 0;

foreach (string numStr in numberStrings)
{
    if (int.TryParse(numStr, out int number))
    {
        sum += number;
    }
}

Console.WriteLine($"Sum of numbers in numbers.txt: {sum}");
```

Step 6: Create "grades.txt" and Write Grades

6.1. Create a new text file named "grades.txt" in the project directory.

6.2. Implement a loop to prompt the user to enter five grades (as integers) and write each grade to "grades.txt" on a new line:

```
// Specify the file path for grades.txt
string gradesFilePath = "grades.txt";

// Prompt the user for grades and write them to grades.txt
using (StreamWriter sw = new StreamWriter(gradesFilePath))
{
    for (int i = 1; i <= 5; i++)
    {
        Console.Write($"Enter grade {i}: ");
        int grade = int.Parse(Console.ReadLine());
        sw.WriteLine(grade);
    }
}
```

Step 7: Calculate and Display Average Grade

7.1. Implement a function to read the grades from "grades.txt," calculate the average grade, and display it:

```
// Read the grades from grades.txt
string[] gradeStrings = File.ReadAllLines(gradesFilePath);
```

```
int total = 0;

foreach (string gradeStr in gradeStrings)
{
    if (int.TryParse(gradeStr, out int grade))
    {
        total += grade;
    }
}

double average = (double)total / gradeStrings.Length;

Console.WriteLine($"Average grade: {average:F2}");
```

Step 8: Implement Error Handling

8.1. Surround file operations with try-catch blocks to handle exceptions gracefully:

```
try
{
    // File operations here
}
catch (FileNotFoundException ex)
{
    Console.WriteLine($"File not found: {ex.Message}");
}
catch (IOException ex)
{
    Console.WriteLine($"An error occurred while accessing the file:
{ex.Message}");
}
catch (Exception ex)
{
    Console.WriteLine($"An unexpected error occurred:
{ex.Message}");
}
```

Step 9: Test Your Program

9.1. Build and run your program to ensure it performs all file operations correctly.

9.2. Test various scenarios, including valid and invalid inputs, to verify that error handling works as expected.

Step 10: Add Comments and Instructions

10.1. Add comments to your code to explain each section's purpose and functionality.

10.2. Within the console application, provide clear instructions to the user, explaining what actions they need to take and what information they will receive.

Step 11: Submit Your Project

11.1. Once you have completed the task and thoroughly tested your program, prepare a submission package that includes all code files and any necessary documentation.

11.2. Submit your project according to the guidelines provided by your course instructor or organization.

This technical guide provides a step-by-step approach to completing the File I/O task in C#. Ensure you follow each step carefully to achieve the desired functionality and error handling in your program.