# Reading Text File in C#

## Introduction

Reading text files is a fundamental task in programming that allows us to access and extract valuable information from stored data. Whether it's processing user inputs, analyzing log files, or working with configuration files, the ability to read text files is crucial in many programming tasks.

In this lesson, we will explore different approaches for reading text files in C#. We will cover various techniques, from reading files character by character to reading them line by line. These techniques will equip you with the necessary skills to extract data efficiently and perform further operations on it.

By the end of this lesson, you will have a solid understanding of how to open, read, and close text files in C#. You will also learn about error handling and best practices to ensure smooth execution of your file reading operations.

So, let's dive into the world of text file reading in C# and discover the power and versatility it brings to your programming tasks. Get ready to unlock the potential of reading text files and enhance your programming skills!

## Objectives

The objectives of this lesson on reading text files in C# are designed to help you gain a comprehensive understanding of this essential programming task. Throughout this lesson, you will have the opportunity to learn various techniques and best practices for efficiently reading text files in C#. By focusing on four key objectives - understand, learn, practice, and apply - you will develop the necessary skills to confidently work with text files in your C# projects.

1. Understand the importance of reading text files in various programming tasks:
   - Recognize the significance of text file reading in extracting and manipulating data.
   - Understand the scenarios where reading text files is essential in programming.
2. Learn different approaches for reading text files in C#:
   - Explore techniques for reading files character by character.
   - Discover methods for reading files line by line.
   - Learn about reading text files using formatted data extraction.
3. Practice implementing file reading techniques in C#:
   - Gain hands-on experience by working through coding examples and exercises.
   - Apply the concepts learned to read text files and extract meaningful information.
   - Develop a solid understanding of the syntax and usage of file reading functions in C#.
4. Apply best practices and error handling techniques when reading text files:
   - Learn about resource cleanup and memory management while working with files.
   - Understand the importance of error handling to ensure smooth execution of file reading operations.
   - Apply best practices for efficient and effective text file reading in C#.

By focusing on these objectives, you will not only enhance your understanding of text file reading in C#, but also gain practical experience and the ability to apply these techniques to real-world programming scenarios.

**Source code example**

```csharp
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string filePath = @"C:\file\file_to_read.txt";

        try
        {
            string[] lines = File.ReadAllLines(filePath);
            int lineCount = lines.Length;
            int wordCount = 0;
            int charCount = 0;

            foreach (string line in lines)
            {
                charCount += line.Length;

                // Splitting line into words using space as delimiter
                string[] words = line.Split(' ');
                wordCount += words.Length;
            }

            Console.WriteLine("Contents of the file:");
            Console.WriteLine(string.Join(Environment.NewLine, lines));
            Console.WriteLine();

            Console.WriteLine($"Number of lines: {lineCount}");
            Console.WriteLine($"Number of words: {wordCount}");
            Console.WriteLine($"Number of characters: {charCount}");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine("File not found. Please make sure the file path is correct.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
        Console.ReadKey();
    }
}
```

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string filePath = @"C:\file\file_to_read.txt";

        try
        {
            string[] lines = File.ReadAllLines(filePath);
            int lineCount = lines.Length;
            int wordCount = 0;
            int charCount = 0;

            foreach (string line in lines)
            {
                charCount += line.Length;

                // Splitting line into words using space as delimiter
                string[] words = line.Split(' ');
                wordCount += words.Length;
            }

            Console.WriteLine("Contents of the file:");
            Console.WriteLine(string.Join(Environment.NewLine, lines));
            Console.WriteLine();

            Console.WriteLine($"Number of lines: {lineCount}");
            Console.WriteLine($"Number of words: {wordCount}");
            Console.WriteLine($"Number of characters: {charCount}");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine("File not found. Please make sure the file path is correct.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
        Console.ReadKey();
    }
}
```

**Explanation**

The source code is a collection of instructions written by programmers using a human-readable programming language. It serves as the foundation for creating computer programs. In simpler terms, it's like a recipe that tells the computer what to do.

In the provided source code, we have a program that reads the contents of a text file and displays them on the console. It also includes additional features to count the number of words, lines, and characters in the file.

To start, the program needs the file path, which is the location of the text file on your computer. You can specify the file path by updating the filePath variable in the code.

The program uses a try-catch block to handle any potential errors that may occur during the execution. If the file is not found or any other error occurs, it will display an appropriate error message.

Once the file is successfully read, the program stores each line of the file in an array called lines. It then counts the number of lines by getting the length of the lines array.

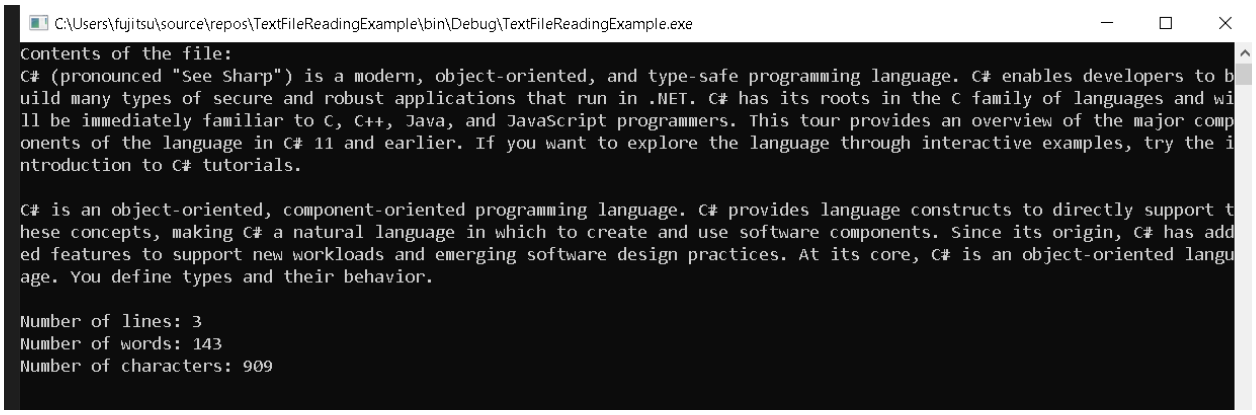
To count the number of words and characters, the program iterates over each line. It adds the length of each line to the charCount variable to get the total number of characters. It also splits each line into words using spaces as delimiters and counts the number of words.

After reading the file and counting the words, lines, and characters, the program displays the contents of the file on the console using Console.WriteLine(). It also shows the counts for lines, words, and characters.

To run the program, you can copy the provided source code into a new C# project in your preferred development environment. Make sure to update the filePath variable with the correct path to your text file. Then, compile and run the program to see the results on the console.

Remember, this program is just one example of how to read and analyze text files in C#. You can modify and expand upon it to suit your specific needs and explore further possibilities with text file manipulation in C#.

**Output**



```
C:\Users\fujitsu\source\repos\TextFileReadingExample\bin\Debug\TextFileReadingExample.exe
Contents of the file:
C# (pronounced "See Sharp") is a modern, object-oriented, and type-safe programming language. C# enables developers to b
uild many types of secure and robust applications that run in .NET. C# has its roots in the C family of languages and wi
ll be immediately familiar to C, C++, Java, and JavaScript programmers. This tour provides an overview of the major comp
onents of the language in C# 11 and earlier. If you want to explore the language through interactive examples, try the i
ntroduction to C# tutorials.

C# is an object-oriented, component-oriented programming language. C# provides language constructs to directly support t
hese concepts, making C# a natural language in which to create and use software components. Since its origin, C# has add
ed features to support new workloads and emerging software design practices. At its core, C# is an object-oriented langu
age. You define types and their behavior.

Number of lines: 3
Number of words: 143
Number of characters: 909
```

**Summary**

The lesson on "Reading Text Files in C#" explores the fundamental process of reading and analyzing the contents of a text file using C# programming. It begins by emphasizing the significance of this skill in various programming tasks and introduces different approaches for reading text files. The tutorial provides a working example that reads a text file, displays its contents, and calculates statistics such as the number of lines, words, and characters. The use of try-catch blocks for error handling is also demonstrated, ensuring a robust and error-tolerant program. Learners are exposed to the practical application of file manipulation and gain insights into the importance of this skill in data-driven applications. Overall, the lesson equips beginners with the foundational knowledge needed to read and process text files in C#.

**Exercises and Assessment**

1. Enhanced Display: Modify the program to display line numbers alongside each line of the file.
2. Custom File Path: Allow the user to input the file path during runtime. Update the program to handle this user-specified file path.
3. File Extension Check: Implement a check to ensure that the provided file is a text file (.txt) before attempting to read it. Display an appropriate message if the file has an incorrect extension.
4. Word Frequency Counter: Extend the program to count the frequency of each unique word in the file and display the results.
5. User Input Interaction: Allow users to interactively choose what statistics they want to display (lines, words, characters) and implement the corresponding functionalities.

**Lab Exam:**

In a lab exam scenario, present students with a set of requirements and constraints. For instance:

Task: Create a C# program that reads a user-specified text file, displays its contents with line numbers, and provides options for calculating either the number of lines, words, or characters. The program should gracefully handle errors, and additional features are encouraged.

Constraints:

- Use a modular approach with well-defined functions.
- Include error handling mechanisms.
- Implement user-friendly prompts and messages.

This lab exam assesses students' ability to apply the concepts learned in the lesson to solve a real-world problem and encourages creativity in extending the program's functionality.

**Quiz**

1. Which method reads the entire file as a single string?
   a) ReadLine
   b) ReadToEnd
   c) ReadAllLines
   d) ReadAllText

2. What error occurs if you try to read a non-existent file?
   a) NullReferenceException
   b) FileNotFoundException
   c) ArgumentException
   d) IndexOutOfRangeException

3. How can you count the number of words in a line?
   a) Split the line by spaces and count the array elements.
   b) Use the Length property of the line string.
   c) Iterate through each character and check if it's a space.
   d) None of the above.

4. How can you write data to a text file?
   a) Using File.WriteAllLines
   b) Using File.WriteAllText
   c) Using a StreamWriter object
   d) All of the above

5. Which is NOT a good practice for working with text files?
   a) Closing the file stream after reading.
   b) Using a try...catch block to handle errors.
   c) Reading the entire file into memory at once for large files.
   d) Splitting lines by tabs or commas instead of spaces.

d) ReadAllText
b) FileNotFoundException
a) Split the line by spaces and count the array elements.
d) All of the above
c) Reading the entire file into memory at once for large files.

**Meta Description**

"Master the art of reading text files in C#, from understanding techniques to practical application. Enhance your programming skills today!"