|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CS 1400 Lab 24**  **Reading From a File**  **Version 1.0**  **Objectives:**  The objective of this lab is to give you practice reading data from a file. After completing this lab, you should be able to design solutions to problems that read data from a file.  **Study Material**   |  | | --- | | **File I/O**  **Introduction**  File I/O refers to the reading a file as input and writing to a file as output of data from/to a file that is stored on the hard disk of your computer. C# treats a file as a sequential stream of bytes. C# contains a number of stream classes that are used for I/O operations. In this lab we will introduce two of these classes, the ***StreamReader*** class for reading text data from a file and the ***StreamWriter*** class for writing text data to a file.  The ***StreamReader*** class and the ***StreamWriter*** class are instance classes, that contain methods that work identically to the way they do for the Console class for the Keyboard and Monitor. For example, the ***StreamReader*** class uses the ***ReadLine***( ) method to read a line of data from a file, in exactly the same way that the ***ReadLine***( ) method reads data from the Console.  **Declaring and Opening a Stream**  **Declaring a Stream Object:** You instantiate a stream object, just as you do any other instance class. For example, to create a ***StreamReader*** object you would write  StreamReader **inFile** **=** **new** StreamReader("Data.txt");    In this case a ***StreamReader*** object is created and can be referenced by the variable ***inFile***.  **Opening a Stream:** Before you can read from a file, you must open the file. This can be done when the object is declared, by writing the following statement       StreamReader **inFile** **=** **new** StreamReader("Data.txt");  The parameter ***Data.txt*** is the path to the file. If no path is given, then the file is read from the same directory as the program that you are currently executing. If the path that you use as a parameter contains a backslash, you must use two backslashes, for example  **StreamReader** theTextFile = new StreamReader("***F:\\Data.txt***");  **Reading from a File**  Once the file has been opened, you can now read from the file, just as you would read from the Console. All of the same rules apply. For example,      StreamReader **inFile** **=** **new** StreamReader("Data.txt");  **int** **number** **=** 0;  **double** **size** **=** 0.0;  **number** **=** (**int.TryParse**(**inFile.ReadLine**(), **out** **number**)) **?** **number** : **-**1;  **size** **=** (**double.TryParse**(**inFile.ReadLine**(), **out** **size**)) **?** **size** : **-**1.0;  **inFile.Close**(); |  |  | | --- | | **File Paths**  **The Default File Path**  When creating a ***StreamReader*** or ***StreamWriter*** object, if you pass a simple file name as the parameter, then your program will look in the directory of the currently executing program’s .exe file. This is the directory where your executable is being executed from.  ***Specifying a Full Path***  If you want your program to read or write a file that is someplace other than in the current directory, then you have to specify the fully qualified path to the file. C# provides several ways of writing a path. Suppose that you want to read from the file ***scores.txt*** located on an attached ***USB*** drive ***g:*** the path would be ***g:\\TestData\\****.* You could write the code to create the ***StreamReader*** object and specify the path in any of the following formats:  StreamReader dataFile = new StreamReader(***"F:\\TestData\\scores.txt"***); StreamReader dataFile = new StreamReader(***"F:/TestData/scores.txt"***); StreamReader dataFile = new StreamReader***(@"F:\TestData\scores.txt"***);  **Reading From the My Documents folder**  If you are using Windows Vista or Windows 7 or later versions, Microsoft has made it very difficult, to store a file or create a file that is in the ***root C:\*** drive. For security reasons, they want you to store all of your data in your ***My Documents*** folder or a ***USB drive***. This creates special problems when reading and writing files, because the path to your file can be quite lengthy. Fortunately the .Net library gives us a way to get this path from the computer. The following program shows you how to let the user just enter in the name of a file in his or her My Documents folder, and get the path from the computer.  **static** **bool** **ReadFile**(**int**[] **grades**)  {  **string** **filename** **=** **""**;  **string** **fullpath** **=** **""**;  **string** **data** **=** **""**;  StreamReader **inFile** **=** **null**;  **string** **path** **=** **""**;  Console**.Clear**();  **path** **=** Environment**.GetFolderPath**(Environment**.**SpecialFolder**.Personal**);  Console**.WriteLine**(**"------------------ Reading File Example -----------------------\n"**);  Console**.Write**(**"Enter the name of the file to open (Enter to Exit): "**);  **if**(((**filename=**Console**.ReadLine**())**==""**) **?** **true** : **false**)  **return** **false**;  **else**  **filename** **=** **"\\"** **+** **filename**;  **fullpath** **=** **path** **+** **filename**;  **if** (**!**File**.Exists**(**fullpath**))  {  Console**.WriteLine**(**"File "** **+** **filename** **+** **" does NOT exist\nPlease try again!"**);  **return** **true**;  }  **else**  {  **inFile** **=** **new** StreamReader(**fullpath**);  **int** **count** **=** 0;  **do**  {  **data** **=** **inFile.ReadLine**();  **if** (**data** **==** **null** **||** **data==""**)  {  **grades**[**count**] **=** **-**1;  **break**;  }  **if**(**int.TryParse**(**data**, **out** **grades**[**count++**]))  **continue**;  **else**  **throw** **new** Exception(**string.Format**(((**data==""**)**?** **"empty"** : **"Value read \"{0}\" was invalid!"**),**data**));  } **while** (**true**);  **inFile.Close**();  **inFile** **=** **null**;  Console**.WriteLine**(**"File read properly"**);  **DisplayData**(**grades**);  **return** **true**;  }  }**//End ReadFile()** | | CS1400: Reading to End of FileTesting for End of File (eof) It is not very often that you know exactly how much data will be in a file, when you are writing a program. Programs have to be written so that they can read data from files of all different sizes. So, what do you do if you want to read data from a file, but you don't know how much data is in the file? In C#, the solution is easy. When you try to read past the end of a file, the **ReadLine( )** method returns a ***null*** reference for the string. We can test this condition by writing  **if** (**data** **==** **null**)  The following example reads integer data from a file and prints it until it reaches the end of the file.  **inFile** **=** **new** StreamReader(**fullpath**);  **int** **count** **=** 0;  **do**  {  **data** **=** **inFile.ReadLine**();  **if** (**data** **==** **null** **||** **data==""**)  {  **grades**[**count**] **=** **-**1;  **break**;  }  **if**(**int.TryParse**(**data**, **out** **grades**[**count++**]))  **continue**;  **else**  **throw** **new** Exception(**string.Format**(((**data==""**)**?** **"empty"** : **"Value read \"{0}\" was invalid!"**),**data**));  **inFile.Close**();  **inFile** **=** **null**;  The general algorithm for reading and processing data from a file then looks like this:   * Open the File * Verify that it exists * Write a loop   + Using ReadLIne( ), reads in a line of data and stores it in a string.   + Checks to see if the string read is ***null (eof) or “” (empty string)***. If it is not, it does whatever processing you need to do on the data in the loop.   + If it is ***null or “”***, break out of the loop | |   **Programming Exercise**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **CS 1400 Lab #24**  **Reading Data From a File**  **Introduction**  In this lab, you will create a program that reads data from a file. It will read data until it reaches the end of the file.  **The Problem**  In this lab, exam scores for a class of students is stored in a file. You are to write a program that successfully opens the file, reads in the exam scores, until the end-of-file (***eof***) is read and outputs the scores on the Console.  **Deriving the Solution**  First of all, copy the data file provided for this assignment into your "***Documents***" folder. Name the file **"Grades.txt"**. Use this file to test your program.  Now do the following:   * In your program create an array of 50 int’s to hold the data that comes from the file. * Your program must get the path to the user's Documents folder as described in the reading material on File Paths. The name of the file will be ***"Grades.txt"***. * Create a ***StreamReader*** object, using this path. This will open the file. * Write a loop that reads data from the file, until it discovers the end-of-file or an empty string. * As each integer value is read in, display it, and store it in the array. * Using the concepts taught earlier about partially filled arrays, write a method that takes the array as a parameter and calculates and returns the average value of the integers stored in the array. * Output the average.   **File(s) to Submit:**  Place your complete project folder in a zip file and name the zip file Lab\_24\_your-initials\_V1.0.zip. For example, I would name my file Lab\_24\_DAF\_V1.0.zip. Submit this assignment as Lab #24 on Canvas.   |  |  |  | | --- | --- | --- | |  | **Grading Guidelines** |  | | # | Program | C(correct)  X(incorrect) | | 1 | Meets & works to specifications | 6 points | | 2 | Error Free, elegant & efficient | 4 points | | 3 | Pseudo-Code | -3 points | | 4 | Style Guidelines | -2 points | | 5 | Header File(s) & Formatting | -2 points | | 6 | Source Files(s) & Formatting | -1 points | | 7 | Project Prolog | -1 points | | 8 | Function Prologs | -1 points | | 9 | Zip Filename | -1 points | | 10 | Lab & Project Names | -1 points | | 11 | Zip File is invalid or will not unzip | Lab = 0 pts | |  | Total Points | 10 | 0-9 | |   **Files**  The following is the data file for the programming exercise:  Grades.txt |

88

90

78

65

50

83

75

23

60

94

\_