**Lesson Plan CSD3354 Section 2 for March 2 2020 MON**

## Lab Learning Outcomes:

In this lab we will learn to read and write text files and to work with C# Data Structures such as Arrays. (Once you learn one of the basic data structures, you will easily and quickly see how to use the other ones).

The purpose of the Lab is to learn some of the things we need to know to complete the assignment Hugi and Astrilde, and to set the stage to do the Monopoly Game assignment will involve interacting with a Database.

## Preamble:

You will be given a text file of real estate data.

You will convert this text file into a Data Structure.

You will write code to analyze your Data Structure and answer questions like:

*What is the AVERAGE PRICE of a DETACHED house with 3 bedrooms.*

*How many Semi Detached Houses were sold by community? Output the Average Price.*

*Output a REPORT on all properties that have ‘pool’ in the description.*

**By doing this Lab, you have reproduced the IBM product CICS [Customer Information Control System], developed in the 1960s and still a multi-billion dollar a year Income Generator for IBM.**

## Reading and writing files

In this part of the lab, we will look into reading and writing simple files with C#.

The File class, from the Syste.IO namespace comes with pretty much everything we could possibly want, making it very easy to do simple reading and writing of a file.

In our first example, we will construct an extremely minimalistic text editor. In fact, it is so simple that we can only read one file and then write new content to it, and only a single line of text at a time. But it shows just how easy it is to use the File class:

|  |
| --- |
| using System;  using System.IO;  namespace FileHandlingArticleApp  {  class Program  {  static void Main(string[] args)  {  if(File.Exists("test.txt"))  {  string content = File.ReadAllText("test.txt");  Console.WriteLine("Current content of file:");  Console.WriteLine(content);  }  Console.WriteLine("Please enter new content for the file:");  string newContent = Console.ReadLine();  File.WriteAllText("test.txt", newContent);  }  }  } |

We use the File class in three places:

1. We use it to check if our file exists, we use the ReadAllText() method to read the content of the file,
2. we use the WriteAllText() method to write new content to the file.

We are not using absolute paths, but just a simple filename. This will place the file in the same directory as the executable file, which is fine for now.

The example is easy to understand: We check for the file, if it exists, we read its content and output it to the console. Then we prompt the user for new content, and once we have it, we write it to the file.

That will overwrite the previous content, but for now, that's just fine. We could however use the AppendAllText method instead. Try changing the WriteAllText line to this instead:

File.AppendAllText("test.txt", newContent);

If you run it, you will see that the new text is added to the existing text instead of overwriting it. As simple as that. But we still get only one line of text per execution of our application. Let's be a bit creative and change that. Replace the last lines in our example with this:

Console.WriteLine("Please enter new content for the file - type exit and press enter to finish editing:");

string newContent = Console.ReadLine();

while(newContent != "exit")

{

File.AppendAllText("test.txt", newContent + Environment.NewLine);

newContent = Console.ReadLine();

}

As you can see, we instruct the user to enter the word exit when they wish to stop editing the file, and until they do just that, we append the user input to the file and then prompt for a new line. We also append a newline character, the Environment.NewLine, to make it look like actual lines of text.

However, instead of writing to the file each time, a more elegant solution is:

Console.WriteLine("Please enter new content for the file - type exit and press enter to finish editing:");

using(StreamWriter sw = new StreamWriter("test.txt"))

{

string newContent = Console.ReadLine();

while(newContent != "exit")

{

sw.Write(newContent + Environment.NewLine);

newContent = Console.ReadLine();

}

}

The usage of Streams is a out of the scope of this Lab. In this example we only open the file once, and then write the changes to it once we're satisfied. In this case, we're taking advantage of the using() statement of C#, which ensures that the file reference is closed once it goes out of scope, which is when it's block of { } is done. If you don't use the using() statement, you will have to manually call the Close() method on the StreamWriter instance.

**Lab Algorithm: Importing a text file contents into an Array:**

Step 1: Make an OBJECT for each line of Ajax.csv

## Step 2: Make an Array Data Object. [**Refer to section Arrays in C# in this Lab**]

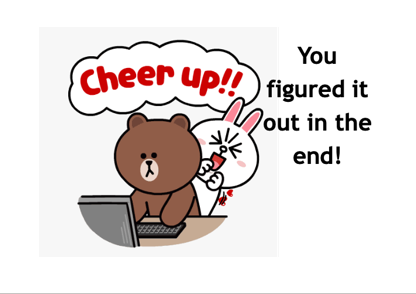
Step 3: Write the ALGORITHM for your Application

Write code to access the file Ajax.csv.

Walk over the file. Create an OBJECT for each line of the file. Store the object in your Array.

When you are done, you with have a Complex Data Object (CDO) that will have 1 record (=object) per Line of the Input Text File.

You can now walk over your Array and ACCESS data records from each object to answer questions like the one in the pre-amble to this Lab.

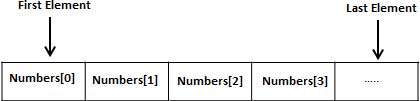


## Arrays in C#

An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



## **Declaring Arrays**

To declare an array in C#, you can use the following syntax −

datatype[] arrayName;

where,

* *datatype* is used to specify the type of elements in the array.
* *[ ]* specifies the rank of the array. The rank specifies the size of the array.
* *arrayName* specifies the name of the array.

For example,

double[] balance;

## **Initializing an Array**

Declaring an array does not initialize the array in the memory. When the array variable is initialized, you can assign values to the array.

Array is a reference type, so you need to use the **new** keyword to create an instance of the array. For example,

double[] balance = new double[10];

## **Assigning Values to an Array**

You can assign values to individual array elements, by using the index number, like −

double[] balance = new double[10];

balance[0] = 4500.0;

You can assign values to the array at the time of declaration, as shown −

double[] balance = { 2340.0, 4523.69, 3421.0};

You can also create and initialize an array, as shown −

int [] marks = new int[5] { 99, 98, 92, 97, 95};

You may also omit the size of the array, as shown −

int [] marks = new int[] { 99, 98, 92, 97, 95};

You can copy an array variable into another target array variable. In such case, both the target and source point to the same memory location −

int [] marks = new int[] { 99, 98, 92, 97, 95};

int[] score = marks;

When you create an array, C# compiler implicitly initializes each array element to a default value depending on the array type. For example, for an int array all elements are initialized to 0.

## **Accessing Array Elements**

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example,

double salary = balance[9];

The following example, demonstrates the above-mentioned concepts declaration, assignment, and accessing arrays −

[Live Demo](http://tpcg.io/iFdIey)

using System;

namespace ArrayApplication {

class MyArray {

static void Main(string[] args) {

int [] n = new int[10]; /\* n is an array of 10 integers \*/

int i,j;

/\* initialize elements of array n \*/

for ( i = 0; i < 10; i++ ) {

n[ i ] = i + 100;

}

/\* output each array element's value \*/

for (j = 0; j < 10; j++ ) {

Console.WriteLine("Element[{0}] = {1}", j, n[j]);

}

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result −

Element[0] = 100

Element[1] = 101

Element[2] = 102

Element[3] = 103

Element[4] = 104

Element[5] = 105

Element[6] = 106

Element[7] = 107

Element[8] = 108

Element[9] = 109

## **Using the *foreach* Loop**

In the previous example, we used a for loop for accessing each array element. You can also use a **foreach** statement to iterate through an array.

[Live Demo](http://tpcg.io/uvQmGo)

using System;

namespace ArrayApplication {

class MyArray {

static void Main(string[] args) {

int [] n = new int[10]; /\* n is an array of 10 integers \*/

/\* initialize elements of array n \*/

for ( int i = 0; i < 10; i++ ) {

n[i] = i + 100;

}

/\* output each array element's value \*/

foreach (int j in n ) {

int i = j-100;

Console.WriteLine("Element[{0}] = {1}", i, j);

}

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result −

Element[0] = 100

Element[1] = 101

Element[2] = 102

Element[3] = 103

Element[4] = 104

Element[5] = 105

Element[6] = 106

Element[7] = 107

Element[8] = 108

Element[9] = 109

## **C# Arrays**

There are following few important concepts related to array which should be clear to a C# programmer −

|  |  |
| --- | --- |
| **Sr.No.** | **Concept & Description** |
| 1 | [Multi-dimensional arrays](https://www.tutorialspoint.com/csharp/csharp_multi_dimensional_arrays.htm)  C# supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array. |
| 2 | [Jagged arrays](https://www.tutorialspoint.com/csharp/csharp_jagged_arrays.htm)  C# supports multidimensional arrays, which are arrays of arrays. |
| 3 | [Passing arrays to functions](https://www.tutorialspoint.com/csharp/csharp_passing_arrays_to_functions.htm)  You can pass to the function a pointer to an array by specifying the array's name without an index. |
| 4 | [Param arrays](https://www.tutorialspoint.com/csharp/csharp_param_arrays.htm)  This is used for passing unknown number of parameters to a function. |
| 5 | [The Array Class](https://www.tutorialspoint.com/csharp/csharp_array_class.htm)  Defined in System namespace, it is the base class to all arrays, and provides various properties and methods for working with arrays. |

## How to read a text file one line at a time

This example reads the contents of a text file, one line at a time, into a string using the ReadLine method of the StreamReader class. Each text line is stored into the string line and displayed on the screen.

int counter = 0;

string line;

// Read the file and display it line by line.

System.IO.StreamReader file =

new System.IO.StreamReader(@"c:\test.txt");

while((line = file.ReadLine()) != null)

{

System.Console.WriteLine(line);

counter++;

}

file.Close();

System.Console.WriteLine("There were {0} lines.", counter);

// Suspend the screen.

System.Console.ReadLine();