**ECE 1111: Engineering Computation I**

**Laboratory No. 8: Pointers and Arrays**

**Goals:** This lab is designed to integrate the concepts of pointers, arrays and dynamic memory allocation. Dynamic memory allocation is a critical concept since we typically want programs to operate successfully independent of the size of the input data, and yet we don’t want to waste space allocating memory that goes unused.

In this lab, we also introduce the use of Microsoft Visual Studio Code. Tutorials on this can be found here:

*https://isip.piconepress.com/courses/temple/ece\_1111/resources/tutorials/tips\_vscode/*

**Deliverables:**

1. *Source Code:* three directories (p01, p02 and p03) containing the files that contain your solutions to the problems below. Each directory should contain: (1) a Makefile, (2) a header, (3) a main program (e.g., p01.cc), and (4) an implementation file (e.g., p01\_00.cc). Everything should compile with make.
2. *Check Off:* Demonstrate that your program runs successfully on a series of example files. Answer questions about dynamic memory allocation.

**Deposit your work in:**

**/data/courses/ece\_1111/current/labs/lab\_08/<lastname\_firstname>**

**Description:**

Using what we have learned previously, write a program that opens a text file and reads the data into memory. Your program should work for any size file.

The specific tasks are:

1. Create a test file called *example.txt* that contains some arbitrary text. For example, convert this Word document to a text file and use that as your test file.
2. Adapt your previously written program that opens a text file and reads it line by line. Read each line from the file into a fixed character string array with dimension MAX\_LINE\_LENGTH = 9999 (allocate an additional character for the null character). This represents the largest line length you can process (including the null termination character).
3. For the first version of your program:
	1. Compute the length of the string – call this a variable named *len*.
	2. Allocate a string of length (*len + 1*) using the new operator. Call this buffer “*tmp\_string*”.
	3. Copy the string you read from the file into *tmp\_string*, and then print the value of *tmp\_string* in this format:

<line\_num>: <value>

For example:

27: this is a test

28: this is the next line in the file

...

* 1. Delete the memory you have allocated.
	2. After you have read the last line in the file and printed it, close the file.
1. For the second version:
	1. Open the input text file, count the numbers of lines and store this in a variable *N*, and then close the input text file.
	2. Create an array of character strings: *char\* strs[N];*
	3. Open the text file a second time.
	4. Read each line and store it in *strs*. Be sure to allocate memory for each string.
	5. Close the input file.
	6. Using a second loop, print the data contained in *strs* following the format above.
	7. Deallocate the memory you have allocated for *strs*. (Hint: loop over each pointer and deallocate the memory associated with each pointer.)
	8. Exit.
2. For the third version, repeat (4), but do this using function calls. Your program should consist of four functions:
3. A function that opens a file, counts the number of lines, closes the file and returns the number of lines as a return value. If an error occurred, return a value of -1.
4. A function that opens a file, reads the data into memory, closes the file, and returns a pointer to the array of strings containing the data.
5. A function that prints the strings to the terminal. The function should have two arguments – the array of character strings and a FILE\* pointer. The main program should call this function using stdout.
6. A function that cleans up memory.
7. Demonstrate that you can load this code into MS Visual Studio Code and debug it. Demonstrate that you can step through the code, set breakpoints, examine variable values, etc.

**Summary:**

The second program you have written is a good example of a program that does dynamic memory allocation. Such programming techniques are crucial to writing clean, robust code. The second program could have been written simply by allocating a large matrix that would hold the largest file you intend to process. For example, you could assume no file had more than 1M lines, and each line could be no more than 1M characters long. What would be wrong with writing the program this way and using this program on a time-sharing operating system in which you are sharing resources with other users?

If you fail to deallocate the memory you have allocated, your program will have what is considered a memory leak. Memory leaks and memory management problems are a common source of problems. For example, it is quite common that your cell phone will hang if you let an application run continuously indefinitely.

Source level debugging of code is a very important skill for a programmer. You have to get in the habit of using a debugger to fix problems with your code.