**ECE 1111: Engineering Computation I**

**Laboratory No. 10: File I/O**

**Goals:** Reading from and writing to files is an important part of programming. This allows your program to interact with external data. In this lab, you will learn how to read and write both formatted and unformatted data to a file. You must use fopen(), fclose, fscanf(), fread() and fwrite().

**Deliverables:**

1. *Source Code:* five programs that perform the tasks below.
2. *Check Off:* Demonstrate that your programs run successfully. Explain how your code reads and writes data. Be sure you understand tasks no. 2 and no. 5.

Deposit your work in:

**/data/courses/ece\_1111/current/labs/lab\_10/<lastname\_firstname>/p01**

**/data/courses/ece\_1111/current/labs/lab\_10/<lastname\_firstname>/p02**

**/data/courses/ece\_1111/current/labs/lab\_10/<lastname\_firstname>/p03**

**...**

**Description:**

There are five tasks in this laboratory:

1. Write a program that writes the following text to a file:

**width = 27.0**

**height = 99.0**

**length = 33.0**

We often refer to this format as name-value pairs.

1. Write a program that reads the values from this file using fopen, fclose and fscanf. Demonstrate that this works for any values of these variables (e.g., “width = 35.0” should work). How flexible is your code? What if there are extra spaces or tabs in the line? How do you ignore the “width =” part of the line?
2. Create a floating-point array of 100 values. Set the values of the array to the numbers [0,99] (the element index). Write a program that writes this data to a file as binary data. The file should be exactly 400 bytes in size. Use “od -f” to demonstrate that the file contains the correct values.
3. Write a program that reads this data from a file and prints the values to stdout using fread().
4. Write a program that loops over the file created in no. 3 and reads floating point values two numbers at a time. This program should work for any size file. Loop until there is no more data to read. Print the values you are reading to stdout. Demonstrate this works for a file of 100 floats, 200 floats, and 1,000 floats. Use your program in no. 3 to create these files. What happens if the file was 399 bytes long instead of 400 bytes long? How would your program handle the end of file condition?

**Summary:**

Binary file I/O is particularly important because most real-world data, such as signals (e.g., video streams, audio streams, biological signals), are represented as binary values. You will learn more about this is Signal and Systems and Digital Signal Processing. The buffered I/O functions (e.g., fopen) are very useful tools for dealing with such data. Whether it is reading and writing integer data, such as signal values that are delivered by a digital to analog converter, or floating-point data, such as numeric data generated during a simulation, being able to write the exact value of a variable to a file is an important capability. One interesting way we apply this is to checkpoint programs so that they can be restarted at any time from the middle of a long run.