

EECE.2160: ECE Application Programming

Spring 2018

Programming Assignment #9: File I/O

Due **Wednesday, 5/2/18**, 11:59:59 PM

1. Introduction

In this assignment, you will work with files to handle input and output. Your program can be used to determine the minimum, maximum, and average values for voltage, current, and power across a group of resistors. The input data will be stored in a series of binary files; your program will be responsible for outputting the results to a text file.

2. Deliverables

This assignment uses multiple files, which you must write from scratch:

- ***prog9_files.c***: Source file containing your main function.
- ***prog9_functions.h***: Header file containing function prototypes.
- ***prog9_functions.c***: Source file containing other user-defined functions

Submit all three files by uploading these files to your Dropbox folder. Ensure your file names match the names specified above. Failure to meet this specification will reduce your grade, as described in the program grading guidelines.

3. Specifications

Input: Your program should prompt the user to enter the name of two binary files. Each file contains a series of double-precision values. One represents a series of resistance values; the other contains the corresponding voltage drop across those resistors. Note:

- If the user enters the name of a file that cannot be opened, print an error message and repeat the prompt for the name.
- You may assume that each input file holds the same number of values, and that number will be no more than 20.
- After reading each file name, the program should read the contents of the file into an array that stores these values. Look at the output test files on the course web page (*r1v1_out.txt*, *r2v2_out.txt*, *r3v3_out.txt*) to see a list of the values in each input file.
 - If an error occurs when reading a file (not when opening the file), print an error message and exit the program.
 - Note that reaching the end of the file without reading the maximum amount of data is not an error. If you attempt to read 10 values and only get 5, for example, the remainder of the program should work strictly with those 5 values.

Your program should also prompt for the name of a file to store your output, which is described below.

Output: After reading the input files, your program should print the following information:

- A table of the resistance/voltage pairs read from the input files.
 - Each quantity should be shown using 2 decimal places.
 - You may assume the maximum resistance is 99,999.99 ohms.
 - You may assume the minimum and maximum voltages are -99.99 and 99.99 volts, respectively.
- A separate table showing the minimum, maximum, and average values for three quantities: the voltage drops, current flow, and power consumption across all resistors. Note that:
 - Voltage should be printed using 2 decimal places; current and power should be printed using 4.
 - You should calculate the current and power for each pair of R/V values and store each number in an appropriate array. For example, if the resistance, voltage, and current values are stored in arrays `res[]`, `voltage[]`, and `current[]`, then `current[0]` is based on `res[0]` and `voltage[0]`.
 - To calculate current, use the equation $I \text{ (current)} = V / R$
 - To calculate power, use the equation $P = V * I$
 - After filling the current and power arrays, you can find the minimum, maximum, and average value in each array before printing.
 - Common errors to avoid that I've seen in previous semesters:
 - The average value must be calculated based on all elements of each array—you can't simply take $(\text{max} + \text{min}) / 2$.
 - The maximum current and power values may not be based on the maximum voltage values.

4. Hints

Functions: You may wish to write functions to handle the following repeated operations in this program:

- Opening a file: Handles the three different files to be opened
 - Example: `FILE *openFile(char *desc, char *mode)` would open a file with description `desc` and access type `mode`, returning a valid `FILE` pointer. Function will prompt user for file name until valid name is entered. Prompt will include string `desc` so different files have different prompts. Sample function call: `FILE *fp = openFile("input", "r");`
- Calculating min/max/average: Arguments: array and number of elements in array; determines the minimum, maximum, and average of all values in the array.

Binary files: Remember that the `fread()` function returns the number of values correctly read from a binary input file. If you read less than the maximum possible number of values, you can use the `ferror()` and `feof()` functions to determine the cause—either an error occurred, or you reached the end of the file.

Also note that, for input files to be read correctly, they should be stored in the same directory as your source file or executable.

Mac users: Xcode requires that input files be placed in the same directory as the executable—putting them with the source file does not work. The easiest way to find your executable directory is through the executable image in the “Products” folder shown on the left side of your Xcode window (see Figure 1 below). Right click on the executable name and choose “Show in Finder” from the resulting menu. Place your input files in the folder that appears, which contains your executable file.

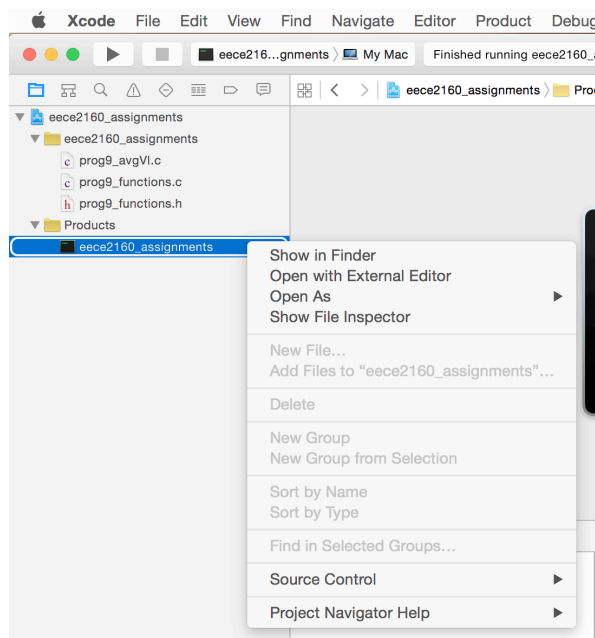


Figure 1: Xcode screenshot demonstrating how to find the location of your executable file

Field width: Specifying the field width of an output allows you to generate multiple lines of output that line up with one another, as in the tables you have to print for this assignment. Field width is a minimum number of characters to be printed, with leading spaces added to fill the field if the value to be printed is shorter than the field width. Say `int x = 123` and `int y = 1`, and the program contains these lines:

```
printf("x:%4d\n", x);  
printf("y:%4d\n", y);
```

The output would be as shown below, with 1 space before 123 and 3 spaces before 1 so each value is printed in a “field” of 4 characters:

```
x: 123  
y:   1
```

Field width and precision: Field width and precision can be combined when printing floating-point values. Remember, since field width specifies a minimum number of characters to be printed (not digits), you must account for all characters in a number, including a decimal point and a minus sign (if necessary).

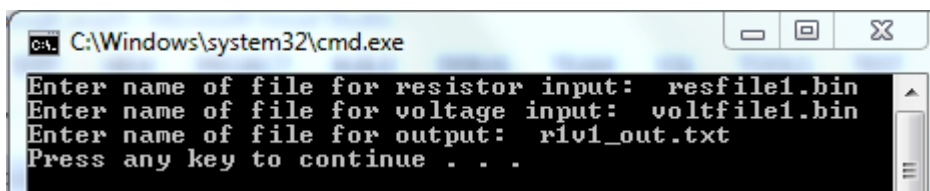
The following print statement prints the variable `double d1` with a field width of 6 and a precision of 1: `printf("d1:%6.1lf\n", d1);`

So, for example:

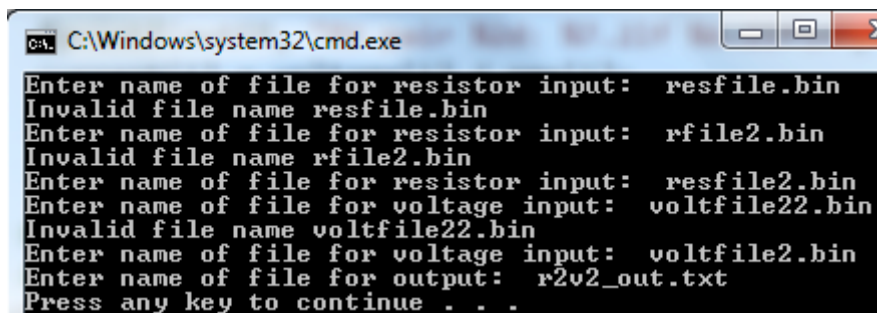
- If `d1 = 1.23`, output is: `d1: 1.2` (3 spaces before 1.2)
- If `d1 = 123.4`, output is: `d1: 123.4` (1 space before 123.4)
- If `d1 = 1234.5`, output is: `d1:1234.5` (no spaces, since 1234.5 uses 6 characters)

5. Test Cases

The screenshots below show the console input and output from two different program runs—one in which all files open successfully, and another in which the user provides invalid input file names before choosing a correct one:



```
C:\Windows\system32\cmd.exe
Enter name of file for resistor input: resfile1.bin
Enter name of file for voltage input: voltfile1.bin
Enter name of file for output: r1v1_out.txt
Press any key to continue . . .
```



```
C:\Windows\system32\cmd.exe
Enter name of file for resistor input: resfile.bin
Invalid file name resfile.bin
Enter name of file for resistor input: rfile2.bin
Invalid file name rfile2.bin
Enter name of file for resistor input: resfile2.bin
Enter name of file for voltage input: voltfile22.bin
Invalid file name voltfile22.bin
Enter name of file for voltage input: voltfile2.bin
Enter name of file for output: r2v2_out.txt
Press any key to continue . . .
```

The more useful test cases can be found on the web page, in the form of input and output files. The “Program 9 test files” link will take you to a web page where you can download three separate pairs of resistance/voltage input files. You can also view the output files produced by using each of these input sets. Your output files should match these test cases exactly.