16.216: ECE Application Programming

Summer 2012

Programming Assignment #7: Arrays and Functions

Due **Friday, 8/10/12**, 11:59:59 PM

1. Introduction

This assignment gives you experience working with arrays and functions. You will write a program that can be used to determine the minimum, maximum, and average values for voltage, current, and power across a group of resistors, using functions to read in the appropriate data and calculate the necessary data.

2. Deliverables

Submit your source file directly to Dr. Geiger (Michael_Geiger@uml.edu) as an e-mail attachment. Ensure your source file name is ***prog7_arrays.c***. You should submit only the .c file. Failure to meet this specification will reduce your grade, as described in the program grading guidelines.

3. Specifications

Input: Your program should repeatedly prompt the user to enter resistance and voltage values for each resistor, concluding when one of two conditions is met:

- The user enters Ctrl-Z (and then Enter) to indicate the end of the list
  - See Section 4 for hints on detecting this condition
- The user enters a total of 20 resistors (the maximum allowed number).

For example (user input is underlined):

```
Enter R and V for resistor 1: 270 5.5
Enter R and V for resistor 2: 12000 -33.1
Enter R and V for resistor 3: 4700 8.3
Enter R and V for resistor 4: 2200 4.9
Enter R and V for resistor 5: 6800 -1.5
Enter R and V for resistor 6: 330 .45
Enter R and V for resistor 7: 10000 5.5
Enter R and V for resistor 8: ^Z
```

Your program must also validate input—all resistor values must be greater than zero. Below is a partial run showing how a negative resistor value is handled:

```
Enter R and V for resistor 1: -10 2.3
ERROR: R must be greater than 0
Enter R and V for resistor 1: 10 2.3
Enter R and V for resistor 2:
```

Output: After reading all input, your program should print a table showing the minimum, maximum, and average values for three quantities: the voltage drops, current flow, and power consumption across all resistors. Recall that:

- I = V / R
- P = V * I

Your output for the first input sequence of 7 resistors above would be:

```
         Voltage   Current      Power
MIN        -33.1   -0.0028     0.0003
MAX          8.3    0.0204     0.1120
AVG         -1.4    0.0033     0.0333
```

Error checking: Your program should print an error and immediately exit under any of the following conditions:

- Any of the inputs are incorrectly formatted and therefore cannot be read correctly using `scanf()`.

Functions: In addition to the `main()` function, you are required to provide two other functions that are called by `main()`:

`int ReadRes(int n, double *r, double *v);`

Function arguments	
`int n`	Number of next resistor to be entered
`double *r`	Address of main program variable storing next resistor value
`double *v`	Address of main program variable storing resistor voltage
Return values	
1	Function has read valid data and stored those data in the variables at addresses indicated by `r` and `v`
0	End of file (Ctrl-Z) was entered, so no valid data were stored in variables at address indicated by `r` and `v`.
-1	Input data incorrectly formatted—could not be read by `scanf()`

This function is used to read in pairs of resistances and voltages. It will prompt for each resistor by number, starting with resistor 1, and then accept the resistance and voltage for a *single resistor*. Your main program must therefore call this function repeatedly to read all resistor values.

If the entered resistance is not positive, `ReadRes()` will display an error and try again (see the second example input on page 1). If the resistance is valid, the function returns to main with `r` and `v` pointing to new values.

If the user enters Ctrl-Z, `ReadRes()` must detect the end of file and signal to the main program that there are no more data, using the function return value, as described above. Note that the function should not return if an invalid resistance value is entered.

Functions (cont.):

```
void MinMaxAvg(double x[], int n, double *min, double *max,
               double *avg);
```

Function arguments	
`double x[]`	Array of doubles to be analyzed
`int n`	Number of elements in array
`double *min`	Address of main program variable storing array minimum
`double *max`	Address of main program variable storing array maximum
`double *avg`	Address of main program variable storing array average

This function will be used to calculate the appropriate quantities for each input array—minimum, maximum, and average. Because multiple values are required to be "returned" to the main program, addresses of those variables are passed.

4. Formulating a solution

Detecting end of input: As noted, your program should stop reading input when the user enters Ctrl-Z. That sequence indicates the end of the input stream. When `scanf()` reaches the end of the input stream before reading all values, it will return a special value, `EOF` ("end of file"). You can therefore test for this value in your `ReadRes()` function; note that the example code below only covers one possible return value for this function:

```
int scanfRetVal;
scanfRetVal = scanf(<appropriate scanf arguments>);
if (scanfRetVal == EOF)
   return 0;              // Indicates no valid data read
```

Pointers and scanf(): Until now, we have used `scanf()` to read data into simple variables and have therefore needed the address operator (`&`) to pass the variable addresses to `scanf()`.

If you want to read a value into a variable, and you already have a pointer holding the variable's address, you can simply use that pointer as an argument to `scanf()`. For example, the following code uses a pointer `p` that points to an integer `x`. That pointer is passed to `scanf()`, allowing an input value to be read into `x`:

```
int x;
int *p = &x;
scanf("%d", p);   // If user enters 3, x == 3 at end of
                  //   this function call
                  // This code is equivalent to:
                  //   scanf("%d", &x);
```

Partially filled arrays: In this program, I suggest using arrays to store all voltage, current, and power values. Each of these arrays can then be passed to the `MinMaxAvg()` function to calculate the appropriate values.

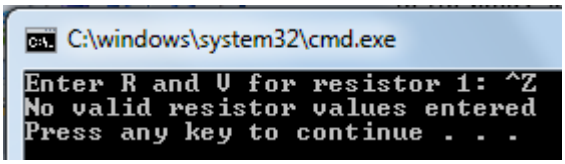
In most cases, these arrays will not be completely filled. Each array should be declared with the maximum possible size, which I recommend declaring as a symbolic constant. A separate variable should hold the number of values actually stored in the array; that variable can be passed to `MinMaxAvg()` as the number of array elements.

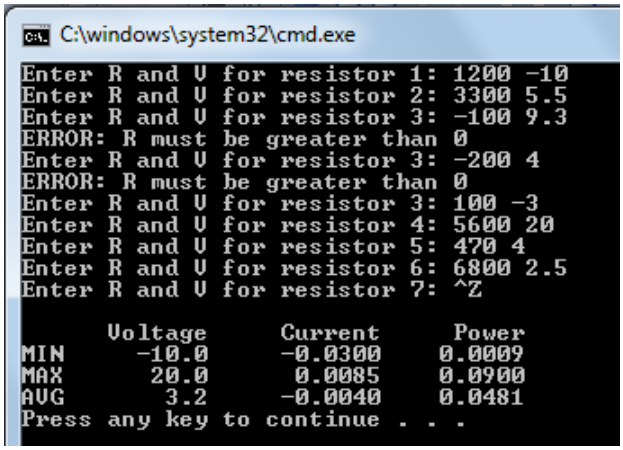
The following code provides a basic example of the approach I suggest for using partially filled arrays:

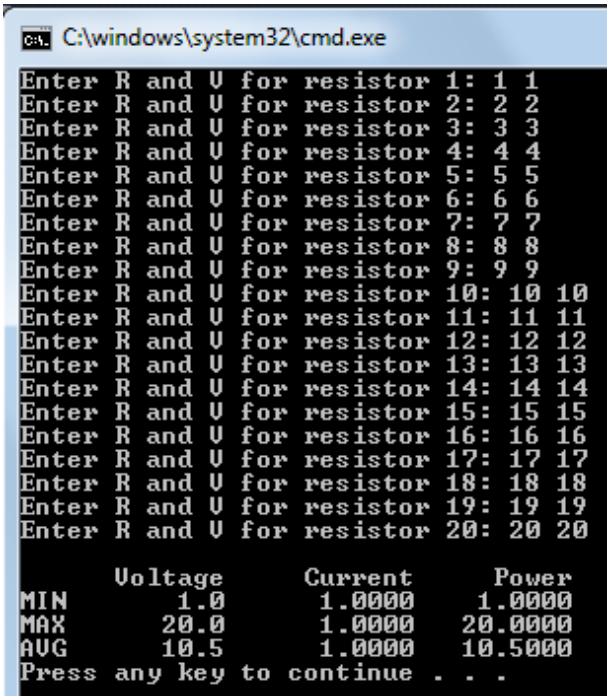
```
int n;                          // # elements in array
double current[MAX_RESISTORS];  // Current values
double maxC, minC, avgC;        // Current min/max/avg
<code to read input values and change array and n>

// Calculate min, max, and average values for current
MinMaxAvg(current, n, &minC, &maxC, &avgC);
```

5. Test Cases

Your output should match these test cases exactly for the given input values. I will use these test cases in grading of your lab, but will also generate additional cases that will not be publicly available. Note that these test cases may not cover all possible program outcomes. You should create your own tests to help debug your code and ensure proper operation for all possible inputs.







Remember, to get your program to terminate with a message saying, “Press any key to continue …”, use the **Start Without Debugging** command (press Ctrl + F5) to run your code.