EE2510 - Lab 1: Basic C++ Programming

2 weeks total

Goals:

1. Become familiar with functional programming in C/C++
2. Dynamic memory utilization
3. Cin/Cout
4. File management

Assignment Description:

Overview:

Create a multi-function calculator – with 3 distinct modes.

Mode 1 – Arithmetic calculator mode

Add, subtract, multiply, divide, or exponentiate (no math.h allowed)

Mode 2 – Data analysis mode

Calculate the average, biggest, or smallest of a set of user entered numbers

Mode 3 – Resistor analysis mode

Calculate the series or parallel combination of a set of user entered values

The user will provide the number of entries and the values

Interface:

The user will be asked which mode to operate in

The user will then be asked which operation in that mode to perform

The user will be prompted for appropriate inputs (must use cin >>)

The program will display the results (must use cout <<)

The program will repeat

Structural requirements:

The top-level program file (includes main) should be used for control only

The functions for each mode must be developed in a separate .cpp / .h file structure

Attached are prototypes for the required functions for this project

Additional functions may be needed or desired (user IO)

NO global variables No math.h

The "array" must be stored in the heap

Grading:

Functionality	Structure
Comments – readability	Documentation
Cleanliness (beauty) of the code	On-time

Deliverables:

Program flow diagram(s)

All code

Eclipse "project explorer" capture showing all files in the project

Screen capture/printouts of input/output for each mode/selection

Hardcopy – no need to put into a PowerPoint or pdf, just print/label/staple

Due: 5:00 pm 1 day after week 2 lab – in the box outside my office

The following prototypes borrow heavily from work done by Dr. Carl

Prototypes for Arithmetic calculator mode

```
//////////////////////////////////////////////////
// Name:        basic_math
// Purpose:     Does basic mathematical calculations
// Return:      0 if successful
//              not-0 if error
// Inputs       int - operation to perform:
//                  1 - addition
//                  2 - subtraction
//                  3 - multiplication
//                  4 - division
//                  5 - exponentiate
//              int - first operand
//              int - second operand
// Outputs      float* - pointer to memory where
//                       result is stored
//////////////////////////////////////////////////////////
int basic_math(int operation, int val1, int val2, float* result);
```

Prototypes for Data Analysis mode

```
//////////////////////////////////////////////////////
// Name:        average
// Purpose:     Averages the values in an array.
// Return:      0 if successful
//              not-0 if error
// Inputs       int[] - array of data
//              int - length of array
// Output       float* - pointer to memory where
//                       result is stored
//////////////////////////////////////////////////////
int average(int array[], int len, float* ave);


//////////////////////////////////////////////////
// Name:        largest
// Purpose:     Finds the largest value in an array.
// Return:      0 if successful
//              not-0 if error
// Inputs       int[] - array of data
//              int - length of array
// Outputs      int* - pointer to memory where largest
//                     value in array is stored
//
//////////////////////////////////////////////////////
int largest(int array[], int len, int* largest);

//////////////////////////////////////////////////
// Name:        smallest
// Purpose:     Finds the smallest value in an array.
// Return:      0 if successful
//              not-0 if error
```

```
// Inputs       int[] - array of data
//              int - length of array
// Outputs      int* - pointer to memory where smallest
//                     value in array is stored
//
//////////////////////////////////////////////////////
int smallest(int array[], int len, int* smallest);
```

Prototypes for Resistor Analysis mode

```
//////////////////////////////////////////////////////
// Name:        series_parallel
// Purpose:     Finds the equivalent resistance of a series or parallel
//              sequence of resistors.
// Return:      0 if successful
//              not-0 if error
// Inputs       int[] - array of resistor values
//              int - length of array
//              int - resistor configuration
//                      1 = series
//                      2 = parallel
// Output       float* - pointer to memory where equivalent
//                       resistance is stored
//////////////////////////////////////////////////////////
int series_parallel(int array[], int len, int config, float* result);
```