Programming Assignment 2 (PA2)
Functions, Header Files, File I/O, Arrays, Random Numbers, Recursion, Pointers

Out: September 22nd, 2017, Friday -- DUE: October 30th, 2017, Monday, 11:59pm
EC327 Introduction to Software Engineering – Fall 2017

Total: 120 points

- *You may use any development environment you wish, as long as it is ANSI C++ compatible. Please make sure your code compiles and runs properly under the Linux/Unix environment on the PHO 305/307 (or eng-grid) machines before submitting.*
- *PAs may be submitted up to a week late at the cost of a **30% fixed penalty** (e.g., submitting a day late and a week late is equivalent). It is in your best interest to complete as many PA questions as possible before the deadline. If you have missing questions in your original submission, you may complete and submit the missing solutions during the following week. Any submissions after the deadline will be subject to the 30% penalty. No credit will be given to solutions submitted after the 1-week late submission period following the deadline.*
- *Follow the assignment submission guidelines in this document or you will lose points.*

Submission Format (Please Read)

- Use the exact file names specified in each problem for your solutions.
- Complete submissions should have **9 files.** Put all of your files in a single folder named: *<your username>_PA2* (e.g., dougd_PA2), zip it, and submit it as a single file (e.g., dougd_PA2.zip).
- Submit to Blackboard
- Please do **NOT** submit *.exe and *.o or any other files that are not required by the problem.
- **Comment your code (good practice!)**

Coding Style

As you become an experienced programmer, you will start to realize the importance of good programming style. There are many coding "guidelines" out there and it is important that you become to adopt your own. Naming conventions will help you recognize variable names, functions, CONSTANTS, Classes etc. Similarly, there are many ways to elegantly format your code so that it is easy to read. All of these issues do not affect compilation and hence are easy to overlook at this stage in your development as a programmer. However, you ability (or inability) to create clean, readable code could be the difference in your career when you leave college.

Here are two links you should explore for more on good coding conventions:

http://www.c-xx.com/ccc/ccc.php .

http://geosoft.no/development/cppstyle.html#General Recomendations

Q1. Functions, Header Files, File I/O (60 pts)

READ THE WHOLE QUESTION BEFORE YOU START. IT IS LONG AND HAS MANY PARTS.
You need to write a C++ program that will interactively accept commands from the user. Each command is a single character followed by one, two, or three parameters and specifies a function that needs to be performed. Your program must do the following:

- Ask the user for the next command code – "`Please enter command code:`"
- Read the command code
- If the command is illegal, print an appropriate error message – "`Invalid command code`"
- else if the command is quit, terminate the program
- else ask the user for the parameters – "`Please enter command parameters:`"
- Read the arguments
- Execute command

Your program must continue this loop until the "quit" command is given.

Commands that you need to include:

F or f	Compute the **F**actorial value	Given an integer number
B or b	Compute the fI**B**onacci number	Given an integer index
R or r	Compute square **R**oots	The sqrt <math> function
D or d	Compute o**D**d numbers	List odd numbers between "first" and "last" number provided
L or l	Compute natural **L**og numbers	The log <math> function
N or n	Compute **N**yanCat value	(See function description)
I or i	Read from **I**nput file	
O or o	Write to **O**utput file	
Q or q	**Q**uit program	

You need to define a global program constant named "ENTRIES" which specifies the maximum calculations performed by your program. Initialize this to 10 at the beginning of your program; NOT INSIDE ANY FUNCTION. **(Think about why it cannot be in a function)**

```
extern const int ENTRIES = 10
```

This sets up a variable that is available to be used across multiple files. Look up how to do this properly.

The parameters for F/f and B/b are single integers. There are no parameters for q/Q. The parameters for O/o and I/i are names of the files you want to write to or read from. The parameters for D/d are the first and last values as integers.

The parameters for R/r, L/l, N/n are the double values "first," "last," and "delta." These values will be inputted by the user as three doubles. When a function is called, the corresponding function must be called for all arguments values from first to last, in steps of delta. For instance, for command 'R':

```
Square root of (first)
Square root of (first + delta)
Square root of (first + delta*2)
```

```
Square root of (first + delta*3)
…
Etc
```

If delta is ≤ 0 or first > last, then no computation takes place. **Print "No computation needed."** If these special cases are not triggered, then compute the values until you have completed the full range of values or you have computed more than **ENTRIES**, whichever comes first. Your program must check for legal command codes, legal delta values, and make sure first ≤ last. However, you DO NOT need to check that a given argument range is legal for the function being called. For instance, you do not need to check that sqrt is being called with positive argument. You are allowed to assume that the user will only enter one keystroke when asked for the command. You should always include the first value in your computation, then proceed in delta increments until you hit the last value or exceed the ENTRIES value. **If a delta increment exceeds the "last" value, just use the last value for your final computation.**

All the functions must be declared in a *.h file and defined in a *.cpp file. They will be defined as follows:

```
Functions.h
Functions.cpp
```

Functions.h will hold all the function prototypes while **Functions.cpp** will hold the implementations. In addition to these two files, you will have a **Q1.cpp** where you will run the main function and call the functions in your library.

Functions.h must include the following prototypes, **EXACTLY** as given here.

```
void initialize();
```

This function must print the program output header. The output header should contain the name and number of the course, the semester and year, the assignment name, and the value of ENTRIES. The values of ENTRIES must be obtained from Q1.cpp. This function is called from main().

(Header Sample)

```
EC327: Introduction to Software Engineering
Fall 2017
Programming Assignment 2
Value of Entries is: 10
```

```
bool checkCode(char);
```

This function should return true if the command code in parameter "code" is a legal code character; otherwise, the function should return false. You have to use this function to check your user input. This function is called from main().

```
void writeDataToFile(const char *);
```

This function will write the output of the program to a file specified by user input. This only works for input provided AFTER this command has been provided. The output being written to a file should also be displayed to the console as well. Writing to a file ends when the quit command is provided.

`void readDataFromFile(const char *);`

The function will read commands from a file specified by user input. This function is called from main(). The commands in this file should be formatted each on two lines consistent with how the user would enter them from the command line.

e.g.

```
f
10
R
10 20 .2
```

`double findNyanCatValue(double);`

If this function takes in "myNum" as an argument this function with return `(2 * myNum) + (myNum * (6)^myNum)`

`int factorial(int);`

This function returns the factorial value of the given integer argument.

`int fibonacci(int);`

This function returns the Fibonacci number at the index given by the provided argument.

`double findSqrtValue(double);`

This function returns the square root of the argument.

`double naturalLog(double);`

This function returns the natural log of the provided argument.

`int findNextOddValue(int);`

This function returns an odd number **closest and higher** to the provided number.

Final requirements

- All functions (other than main) must be prototyped (declared) in **Functions.h**
- All functions (other than main) must be implemented (defined) in**Functions.cpp**
- Main function must be implemented in **Q1.cpp**
- Comment your code!!!
- All input and output (other than file I/O) must be done with `cin` and `cout`, respectively

SAMPLE OUTPUT WILL BE PROVIDED TO BLACKBOARD. Please follow this formatting!!

Q2. File I/O, Arrays, Random Numbers, and Functions (30 pts)

1. Write a function named `WriteRandomData()` that generates N random integers from 0 to M (inclusive) and writes them to a text file named `data.txt`, one number per line. You should use the following function prototype:

```
void WriteRandomData(int N, int M, const char *filename);
```

2. Write a second function called `ReadData()` that reads the data from the specified text file, and also finds the size (number of data elements). You can use the following function prototype:

```
void ReadData(const char *filename, int &size, int myArray[]);
```

3. Write a function called `DoubleAndReverse()` that doubles the size of an integer array and fills the second half with a copy of the first half in reverse order. You can use the following function prototype:

```
int * DoubleAndReverse (int *list, int size);
```

   Example: Given array: [1 2 3 4 5]
   The resulting array obtained by `DoubleAndReverse` function: [1 2 3 4 5 5 4 3 2 1]

   The function should return a pointer to an integer array that has a size equal to 2*size. The values in the new array at indices from 0 to size-1 should have the same values as in the original input array in the same order. The rest of the values in the new array should be the reverse order of the original array.

4. Write functions `getMedian()` and `getLargest()` to compute the median and largest number in an array. These functions should accept an array and size as an argument and return a double and an integer respectively. The function prototypes are left to you to figure out.

5. Write a `main()` function that creates the `data.txt` file, assumes the `data.txt` file is given in the same folder as the code files, reads that file, calculates the size of an array, calls the `DoubleAndReverse` function, prints that array, and calculate the median and largest values. For example, your final output screen should look like this (for some random `data.txt` file):

```
Writing file: data.txt
Reading file: data.txt
Array size is: 5
Doubled and Reversed is [1 2 3 4 5 5 4 3 2 1]
Median is: 3.5
Largest is : 5
```

We will test your code on our `numbers.txt` file. Assume the file can be **any** size between (and including) 0 and 1000.

Submission requirements: You should submit **five separate files:** `Q2.cpp, FileFunctions.h, FileFunctions.cpp, Statistics.h, Statistics.cpp.`

- `Main() and DoubleAndReverse()`, should be saved in `Q2.cpp`.

- `WriteRandomData ()` and `ReadData()` should be stored in a single `FileFunctions.cpp` file with the header file named `FileFunctions.h`.
- `getMedian()` and `getLargest()` should be stored in a single `Statistics.cpp` file with the header file named `Statistics.h`.

Important: As in Q1, you can use `<cmath>` library in your main program but you **must not** use `<cmath>` library (or any other math library) in `Statistics.cpp` file.

Q3. Recursion (30 pts)

Write a recursive function (`void PrintRhombus(int n);`) that takes an integer as an input, and prints the number rhombus shown below. Write a test program that prompts the user to enter an integer between 1 and 9, and then calls the function. Include a check statement in your main program or in the PrintRhombus function to print an error message if the user enters a number outside [1,9].

Turn in one file called Q3.cpp that contains all of your code for this problem.

Text in < > in the examples demonstrates the user inputs entered via the keyboard.

Sample run:
Enter a number [1-9]: <5> <enter>

```
        1
      1 2 1
    1 2 3 2 1
  1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
  1 2 3 4 3 2 1
    1 2 3 2 1
      1 2 1
        1
```

**Spaces on the right and left sides of the rhombus and the exact number of spaces between two columns are not significant as long as the numbers are aligned as shown above.*

Submission format (same as the one on the first page)

- Use the exact file names specified in each problem for your solutions.
- Complete submission should have **nine files.** Put all of your files in a single folder named: `<your username>_PA2` (e.g., `dougd_PA2`), zip it, and submit it as a single file (e.g., `dougd_PA2.zip`).
- For revisions and late assignments please follow the submission guidelines on Blackboard.
- Please do **NOT** submit *.exe and *.o or any other files that are not required by the problem.
- **Comment your code (good practice)!**

Complete submission should have **nine files**: `Q1.cpp`, `Functions.cpp`, `Functions.h`, `Q2.cpp`, `FileFunctions.h`, `FileFunctions.cpp`, `Statistics.h`, `Statistics.cpp` `Q3.cpp`