LAB ASSIGNMENT 1

For Lab Assignment 1, you are required to download the zip file posted from the course lab web page and solve all of the problems given. Your assignment is due at 11:59:59pm on Monday, September 28th. (Note; that this assignment has been updated from its initial posting and you need to re-download both the folder and the description to be able to complete all of the problems listed herein). Not all of the problems are of equal difficulty; hence they will be weighted differently when they are marked. The different problems are divided into separate subfolders (prob1 through prob6) and their "solution code" needs to be put in the same folder as the "problem" code. Your completion of this Lab Assignment will be evaluated in the lab ***and*** electronically. **To ensure that your code can be properly evaluated electronically, please make sure you follow the *exact* instructions on the web page for code submission. Also, be sure that you do not change the name of the file folder or its directory structure.**

Finally: For each problem (excluding Problem 1) be sure to try *numerous* test cases to verify your code (not just the ones we give you here). Sharing your test cases with each other via Piazza is a good way to have a larger pool of test conditions to try.

Problem 1: Use the sizeof() function to determine the size of the following predefined types as well as the size of the custom data types described in CustomTypes.cpp found inside the prob1 folder.

1) short, 2) int, 3) long, 4) float, 5) double, 6) uint8_t, 7) uint16_t, 8) char, 9) void, 10) short*,

11) int*, 12) long*, 13) float*, 14) double*, 15) uint8_t*, 16) uint16_t*, 17) char*, 18) void*

I've numbered each of these types (the custom types are numbered in their comments). From your main function, use the following format to print out the results to the monitor:

- short = ?? bytes [Command line 1]
- int = ?? bytes [Command line 2]
- long = ?? bytes [Command line 3]
- float = ?? bytes [Command line 4]
- …

Problem 2: The code inside the prob2 folder implements a 'quick sort' algorithm, but there are ***multiple*** bugs in it. Use commenting to localize each bug by commenting out different parts of the code one by one and to try to debug it. You will also need to write code to get the input values to be sorted from a file called "input.txt" that is located in the folder prob2. NOTE: Do NOT hard code the variable used to indicate the number of values to be sorted (i.e. how many are stored in the file) as this may change. Instead, you may create a "constant" variable named "max_length" and set it equal to 200 (as this will be the maximum number of inputs we give you) for error checking.

Problem 3: Write member functions for the Class Complex (found inside the prob4 folder) to define (overload) the basic arithmetic operations

- '+','-','*','/','!=','==','>','<'
- print_magnitude, and calculate_magnitude

- *Be sure you test them well* ***(don't forget the corner cases)***.

Problem 4: This question has three Parts. The code is found inside folder prob4.

- **PART 1.** For the class ETriangle (Equilateral Triangle), add the calculation to initialize the height member variable for the constructor and mutator functions. Add a Member function to the called get_surface_area()
- **PART 2:** Create a new derived class called ETriangularPrism that includes an additional member variable, "length." Write a constructor to initialize the TriangularPrism's private member variable, "length", as well as its inherited member variables, "side_length" and "height." Write the member function get_volume() and write a member function to overload get_area().
  PART 3: Create a new derived class called "SquarePyramid" and write member functions to overload get_volume() and get_surface_area() . Write a constructor to initialize the SquarePyramid's private member variable "elevation" (from the base of the pyramid to the apex), which can be used to initialize the inherited private member variables, "side_length" and "height". Write a SquarePyramid constructor that takes the triangle's side length as input and uses it to initialize all the object's member variables.

You should use the version of main provided here to test and verify these classes and for ***in-lab marking only***. Your electronic submission will require you to use a different "main" function. You should ***rewrite*** the main function to ask the user to do the following:

>Enter side length of equilateral triangle:
>Enter length of triangle prism:

Our auto-checker will then expect the following output. The output should be written in 8 lines. Only print the Numbers and NOTHING ELSE:

- First line :- Contains the height of the ETriangle object
- Second Line:- Contains the perimeter of the ETriangle object
- Third Line :- Contains the "surface" area (i.e. Area) of the ETriangle object
- Fourth Line :- Contains the surface area of the ETriangularPrism object
- Fifth Line :- Contains the volume of the ETriangularPrism object
- Sixth Line :- Contains the surface area of the SquarePyramid object
- Seventh Line :- Contains the volume of the SquarePyramid object
- Eighth Line:- Contains the elevation of the SquarePyramid object

Problem 5: Use the code given inside of the prob5 folder to create a class that initializes an array and allows the user to search for values AND REMOVE them from the array. To complete this code you need to:

1) In the main function, read the numbers from the file "input.txt" and insert the elements into the array. The terminating "character" for the data in the array is -1.
2) Write a member function that returns the number of array members that have valid data.
3) Write a search function - the function should look for a number the user gives and if it exists - DELETE it from the array. NOTE: This will require adjusting the positions of the remaining values in the array accordingly so that data is still stored contiguously and terminated with a -1.

4) To obtain the values you need to search for, you need to read them from file "search.txt". (as found in this folder).

Hints:

- The search function would benefit from helper functions to make your code more readable (they should be private member functions).
- A "value" may appear more than once in the array in *ANY* position.
- A value may not appear in your array at all.
- The input.txt and search.txt files are only **samples** of the versions we will use to test your code. Be sure to create your own for more thorough testing (and consider sharing them with your fellow class mates to have a bigger search pool as mentioned previously).

Challenge Task: Problem 6 is a challenge task. Challenge tasks are tasks that you should only perform if you have extra time, are keen, and maybe want to show off a little. It is only worth 10% of the assignment grade, but it will require you to do a bit more work (e.g. reading ahead in the textbook, googling to find some additional information, etc.). It will also (generally) rely on your successfully completing other parts of the assignment. If you don't demo the challenge task, the maximum score you can get on this lab assignment is 90% (which is still an A+).

Problem 6: Using the CoordinatePairs class code provided in the directory prob6, read in values from "input.txt" to initialize the array and sort the array of objects using the quick sort algorithm learned in part 2. I've provided a pseudocode template for your main function to show how you should read in and write out the data.

I've also provided all of the member functions for the class you ***require*** to adapt the quick sort algorithm to use these objects. However, you may choose to add your own public or private member functions if you think it will make your code cleaner and more elegant. Note: This is an implementation choice that will be based partially on the way you think, but also on how comfortable you are with C/C++. The only requirement is that if you change the class (i.e. add member functions and change the class), the class must still be an Abstract Data Type (ADT) after you are done. Therefore, you ***CANNOT*** remove/change the declarations of any public member functions (do you understand why?).

You will be sorting the objects in the array based on increasing Manhattan Distance between the coordinate pairs. The Manhattan Distance between two coordinate pairs, (a,b) and (c,d), is defined as the sum of the absolute distances between the points in both planes. In other words, the Manhattan Distance = |a-c| + |b-d|.

Hints:

- Use the I/O code from prob2/prob5 to get yourself started.
- When adapting your sorting algorithm, your trying to swap the type int for CoordinatePairs, but pay attention to the comparison points as you are comparing the Manhattan Distance.