CIS 130 - Homework #9

Due:

Thursday, May 6th, 11:59 pm

Purpose:

Mostly practice writing C++ functions and programs involving pass-by-reference, repetition, and arrays

How to submit:

When you are done with the following problems:

- (Assuming that you are still ssh'd and logged into to **nrs-labs**, since you will be writing and testing some of these C++ functions on nrs-labs using the `funct_play2/funct_compile/expr_play` tools, and writing and testing others using `nano` and `g++`)
- Make sure your current working directory on **nrs-labs** is the one where your homework files are: do the command:

  ```
  ls
  ```

  ...and make sure you see the names of your homework files listed.
  - If you are not in the proper directory, use `cd` *`directory_name`* to go to the proper directory. (For example, `cd 130hw09`)
- use `~st10/130submit` to submit all of your `.cpp` and `.h` files in the current directory.
  - Remember, I don't mind if extra `.cpp` and `.h` files get submitted as well.
  - Make sure that `~st10/130submit` shows that it submitted all of your homework files.
- (ASK ME if this is not clear, or if you have any problems with submission!)

Important notes:

- Each student should work individually on this assignment.
- Remember, you are still expected to follow the Design Recipe for all functions that you write. (Remember to use C++ type names in C++ function contracts, and to write C++ specific examples/tests as discussed in class.)
- Remember to follow the class style guidelines.
- **IF** you choose to use Dev-C++ for any of these problems, remember that:
  - you need to make a .h file for each non-`main` function (make sure it follows the template given on the public course web page for .h files)
  - you need `system("PAUSE");` before the `return EXIT_SUCCESS;` in each `main` function
  - you need to include your contract, purpose, examples, name, and date last modified in an opening comment block in each .cpp file

Homework problems:

Problem 0

Create, protect, and change to a directory `130hw09` -- type the following from your home directory on nrs-labs:

```
[you1@nrs-labs ~]$ mkdir 130hw09

[you1@nrs-labs ~]$ chmod 700 130hw09

[you1@nrs-labs ~]$ cd 130hw09
```

(If you log out and come back later, remember to `cd 130hw09` each time to return to this directory!)

Problem 1

Write a function `accelerate` that expects a single pass-by-reference parameter, representing a speed, and it increases the corresponding argument's value by 10%.

(Note that this, then, is an input/output parameter, as it is used both for providing information to the function and for giving information back to the caller.)

That is, if you did:

```
double curr_speed = 48;
accelerate(curr_speed);
```

...then after these two statements, the following expression should be true:

```
curr_speed == 52.8
```

(although you might need: `abs(curr_speed - 52.8) < .001`)

What should you remember to use to represent 10% in your program?

To test your function, either modify `accelerate_ck_expect.cpp` or write a `main` function in `accelerate_test.cpp` that performs at least the above actions and then outputs the result of the expression shown (being sure to use:

```
    cout << boolalpha;
```

...before trying to print out the results of the expression comparing `curr_speed` to its expected value. That way, you'll see `true` or `false` printed for the value of this `bool` expression.)

Submit your files `accelerate.h`, `accelerate.cpp`, and either `accelerate_ck_expect.cpp` or `accelerate_test.cpp`

Problem 2

(This problem does *not* involve pass-by-reference.) There are times when you just want to see what is in an array.

Write a function `show_values` that expects an array of doubles, its size, and a desired title, and it doesn't return anything, but simply has the side-effect of printing to the screen that title, followed by a blank line, followed by the array's values.

That is,

```
const int NUM_TEMPS = 6;
```

```
double temps[NUM_TEMPS] = {3, 1, 6.6, 2, 8, 4};
show_values(temps, NUM_TEMPS, "This Week's Temperatures");
```

...would cause the following to be printed to the screen:

```
This Week's Temperatures:

3
1
6.6
2
8
4
```

To test your function, either modify `show_values_ck_expect.cpp` or write a `main` function in `show_values_test.cpp` that performs at least the above actions and then outputs the result of the expression shown.

Submit your files `show_values.h`, `show_values.cpp`, and either `show_values_ck_expect.cpp` or `show_values_test.cpp`

Problem 3

Now, put these pieces from Problems 1 and 2 together.

Write a `main` function in a file `combo.cpp` that:

- declares and initializes an example array of doubles of your choice.
- calls `show_values` appropriately with the title `"BEFORE"` to show the initial version of the array
- calls `accelerate` for each item in the array
- calls `show_values` appropriately with the title `"AFTER"` to show what is now in the array.

Submit your file `combo.cpp`.

Problem 4

Consider: we discussed a function `swap` that simply swaps the values of its two pass-by-reference parameters.

There are occasionally times where we just want to two variables' values to be in order -- say, for our purposes here, increasing/ascending order.

Write a function `arrange` that expects two pass-by-reference `double` parameters, and if the first argument is greater than the second, it rearranges their values -- swaps their values -- so that the corresponding first argument now contains the smaller value, and the corresponding second argument now contains the larger value. But if the first isn't greater than the second, the corresponding argument remain unchanged.

(Thus, after calling this, you can be assured that the first argument's value will be <= the second argument's value...)

(Note that both of these happen to be input/output parameters -- both are used as input to the function, and both may be used to give a result back to the caller.)

That is, if you did:

```
double alpha = 100.1;
double beta = 50.6;
double gamma = 200.3;
double theta = 200.3;

arrange(alpha, beta);
```

then afterwards the following expressions should be true:

```
alpha == 50.6
beta == 100.1
```

for:

```
arrange(beta, gamma);
```

then afterwards the following expressions should be true:

```
beta == 100.1
gamma = 200.3
```

(yes, in this case the arguments are unchanged -- they are already in the desired order)

for:

```
arrange(gamma, theta);
```

then afterwards the following expressions should be true:

```
gamma = 200.3
theta = 200.3
```

(yes, in this case the arguments are unchanged -- they are already in the desired order)

To test your function, either modify `arrange_ck_expect.cpp` or write a `main` function in `arrange_test.cpp` that performs at least the above actions and then outputs the result of the expression shown.

Submit your files `arrange.h`, `arrange.cpp`, and either `arrange_ck_expect.cpp` or `arrange_test.cpp`

Problem 5

Write a function `max_and_count` that expects 2 input parameters, an array of doubles and its size, and 2 output parameters, that it should set to the maximum value in the array, and how many times that maximum value appears, respectively. It should not return anything.

To test your function, either modify `max_and_count_ck_expect.cpp` or write a `main` function in `max_and_count_test.cpp` that calls this for at least two different example arrays, at least one of which contains more than one instance of its maximum value. It should print to the screen the result of comparing the actual resulting values in the calls' 3rd and 4th arguments to the expected values for those arguments (using `boolalpha` to make sure that `true/false` are shown instead of `1/0` when the results of these `bool` expressions are printed to the screen).

Submit your files `max_and_count.h`, `max_and_count.cpp`, and either `max_and_count_ck_expect.cpp` or `max_and_count_test.cpp`doubl