CS 131 - Homework 6

Deadline:

5:00 pm on Friday, October 22

How to submit:

When you are done with the following problems:

- make sure that your current working directory on nrs-labs is the one where your C++ function files for this homework are;
  - for example, you might need:

    `cd 131hw6`

    ...to change to you directory `131hw6`, and
  - then look at what files are there using `ls`
- then use `~st10/131submit` to submit your `.cpp` and `.h` files for homework number **6**
- make sure that `~st10/131submit` shows that it submitted your `.cpp` and `.h` files for all of your C++ functions for this homework

Purpose:

Practice writing simple C++ functions

Important notes:

- Each student should work **individually** on this homework.
- You are still expected to follow the Design Recipe for all functions that you write.
  - Remember, you will receive **significant** credit for the signature, purpose, header, and examples portions of your functions.
  - (but remember to use **C++ types** in signatures for C++ functions)
  - (and, use == or < for your C++ example expressions -- for example,

    ```
    my_funct(3) == 27
    abs(my_dbl(4.7) - 100.43) < 0.01
    ```

    ...and note that these example tests are **expressions** rather than C++ statements, so do NOT end them with a semicolon!)
  - Typically you'll get at least half-credit for a correct signature, purpose, header, and examples/, even if your function body is not correct
  - (and, you'll **lose at least half-credit** if you omit these or do them poorly, even if your function body is correct).
- Be especially careful to include at least one specific example/check-expect for each "kind"/category of data, and (when appropriate) for boundaries between data. You can lose credit for not doing so.
- IF you missed the Week 8 Lab (Friday, October 15), then you need to do the steps in

APPENDIX 1 in the "C++ tools" handout on the public course web page, under "References"; see the UNIX reference handout in the same location, too.

- Also: note that the C++ `cmath` library, included by the course C++ tools by default, includes such goodies as an absolute value function (`abs`), `sqrt`, `pow`, and more.

The Problems:

Problem 0

Create, protect, and change to a directory `131hw6` -- type the following from your home directory on nrs-labs:

```
[you1@nrs-labs ~]$ mkdir 131hw6
[you1@nrs-labs ~]$ chmod 700 131hw6
[you1@nrs-labs ~]$ cd 131hw6
```

As we noted in lab, the `chmod` above (with `700`) protects your new directory, allowing ONLY YOU to read, write, or execute it. THIS IS WHAT YOU SHOULD USE FOR HOMEWORK DIRECTORIES or any directory you do NOT want others to be able to read.

(If you log out and come back later, remember to `cd 131hw6` each time to return to this directory!)

Problem 1

Recall the function from Homework 2 that computes the volume of a rectangular tank (Problem 2). Use `funct_play2` to develop a C++ version of this function named `tank_volume`. (`tank_volume` expects a rectangular tank's length, height, and width, and produces the volume of that tank)

Submit your resulting `tank_volume.cpp`, `tank_volume.h`, and `tank_volume_ck_expect.cpp` files.

Problem 2

Recall the function from Homework 2 that determines the total number of inches in a given number of feet and inches (Problem 6). Use `funct_play2` to develop a C++ version of this function named `total_inches`. (Appropriate use of a named constant is required!) (`total_inches` expects a number of feet and a number of inches, and produces the total number of inches)

(Reminder: a C++ constant is defined using:

```
const <type> <IDENTIFIER> = <expression>;
```

...for example:

```
const int WIDTH = 300;
```

And remember: it is a COURSE STYLE STANDARD that named constants should be written in all-uppercase.)

Submit your resulting `total_inches.cpp`, `total_inches.h`, and `total_inches_ck_expect.cpp` files.

Problem 3

Use `funct_play2` to develop a C++ function `in_range` that expects an integer quantity and produces whether that quantity is in the interval [50, 100).

Submit your resulting `in_range.cpp`, `in_range.h`, and `in_range_ck_expect.cpp` files.

Problem 4

Use `funct_play2` to develop a C++ function `short_enough` that expects a string and a desired length limit, and produces whether the number of characters in that string is less than or equal to that desired length limit.

Submit your resulting `short_enough.cpp`, `short_enough.h`, and `short_enough_ck_expect.cpp` files.

Problem 5

Use `funct_play2` to develop a C++ function `gross_total` that expects a quantity of an item and the cost per item, and produces the cost for that many of that item. For this function, your solution *must* be written so that the quantities *MUST* be integers, but the cost per item *CAN* be fractional.

Submit your resulting `gross_total.cpp`, `gross_total.h`, and `gross_total_ck_expect.cpp` files.

Problem 6

Use `funct_play2` to develop a C++ function `tax_owed` that expects an amount and a tax rate, and produces the tax owed for that amount based on that tax rate.

Submit your resulting `tax_owed.cpp`, `tax_owed.h`, and `tax_owed_ck_expect.cpp` files.

Problem 7

We of course like to use functions in other functions, regardless of the computer language!

To do so in C++, we need some pieces that `funct_play2` will build for you *IF* you list those functions that a new function needs when it asks, at the beginning,

```
Are there any already-created C++ functions (in the current
working directory) which you would like to be able to use within
your new function?

(type y if so, n if not)
```

If you type `y`, it will prompt you to enter each of those function(s) names.

You need to enter those names again when you use `funct_compile` for this function.

If you want to use `expr_play` to execute your new function, then when you see:

```
Are there any already-created C++ functions (in the current
working directory) which you would like to be able to use within
C++ expressions?

(type y if so, n if not)
```

...you will need to enter that function's name, **AND** the names of **all** functions your function uses, when prompted. Then you can write expressions trying out all of those functions!

Why am I noting all of this? Because now your should use `funct_play2` to develop a C++ function `final_total` that expects a quantity of an item and a cost per item, and produces the total cost for that many of that item including tax based on a **set tax rate of 7.25%**. Your function must appropriately call `gross_total` and `tax_owed`, and appropriate use of named constants is required.

Submit your resulting `final_total.cpp`, `final_total.h`, and `final_total_ck_expect.cpp` files.