CIS 130 - Homework #5

Due:

Thursday, March 25th, 11:59 pm

Purpose:

Practice writing simple C++ functions, including a little practice with writing `if` statements.

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 these C++ functions on nrs-labs using the `funct_play2/funct_compile/expr_play` tools)
- 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 130hw05`)
- 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.)
- IF you missed the Week 7 lab (on Wednesday, March 3), 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.
- Remember to follow the class style guidelines (more have been added for C++ -- see the in-class examples and postings).

Homework problems:

Problem 0

Create, protect, and change to a directory `130hw05` -- type the following from your home directory on nrs-labs:

```
[you1@nrs-labs ~]$ mkdir 130hw05
[you1@nrs-labs ~]$ chmod 700 130hw05
[you1@nrs-labs ~]$ cd 130hw05
```

The `chmod` command 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 130hw05` each time to return to this directory!)

Problem 1

Recall the function from Homework 2, Problem 2 that computes the volume of a rectangular tank. Use `funct_play2` to develop a C++ version of this function named `tank_volume` . (`tank_volume` expects a rectangular tank's length, width, and height, 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, Problem 3 that produces the average gas consumption (in miles-per-gallon) used for a trip. Use `funct_play2` to develop a C++ version of this function named `avg_mpg` . (`avg_mpg` expects the number of miles traveled and the number of gallons of gas used, and produces the average gas consumption in miles-per-gallon for that trip.)

Submit your resulting `avg_mpg.cpp`, `avg_mpg.h`, and `avg_mpg_ck_expect.cpp` files.

Problem 3

Recall the function from Homework 3, Problem 1 that determines a semester grade. Use `funct_play2` to develop a C++ version of this function named `semester_grade` . (`semester_grade` expects a homework average, a quiz average, and a final exam score, and produces a semester grade based on the following weightings: 45% from the homework average, 20% from the quiz average, and 35% from the final exam score.)

Note that appropriate use of named constants is required; remember, a C++ named constant is defined using:

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

...for example:

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

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

Submit your resulting `semester_grade.cpp`, `semester_grade.h`, and `semester_grade_ck_expect.cpp` files.

Problem 4

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 total cost for that many of that item. For this function, your solution *must* be written so that the quantity *must* be an integer, 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 5

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 6

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.

PLEASE NOTE: You need to enter those names **again** *each time* you use `funct_play2` or `funct_compile` for such a 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)
```

...then, after answering `y`, 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 you 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 9.25%. (Yes, `final_total` needs that set tax rate to be a named constant -- it is not a named constant in Problem 5, but it is a named constant here in Problem 6.) Your function must appropriately *call*

`gross_total` and `tax_owed`.

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

Problem 7

Recall the function from Homework 4, Problem 3 that determines the appropriate discount for a given frequent shopper level. Use `funct_play2` to develop a C++ version of this function named `get_disc` .

(As a reminder: A store gives discounts to frequent shoppers based on their past level of purchases; they are either "silver" level, "gold" level, or "platinum" level. Silver level frequent shoppers receive a 10% discount, gold level frequent shoppers receive a 25% discount, and platinum level frequent shoppers receive a 33% discount. All other shoppers receive no discount.

Design a function `get_disc` that, given a string representing the level of frequent shopper, produces the appropriate discount for that level written as a decimal fraction. (It should return a discount of 0 if the symbol is not one of those noted above.)

Be especially careful to use named constants as appropriate, and to provide sufficient examples.)

Submit your resulting `get_disc.cpp`, `get_disc.h`, and `get_disc_ck_expect.cpp` files.

Problem 8

Recall the function from Homework 4, Problem 4, that expects a string representing the level of frequent shopper and the total of a purchase before discount, and produces the appropriate discounted total for that purchase. Use `funct_play2` to to develop a C++ version of this function named `disc_total` .

For full credit, your solution must appropriately use your function from Problem 7.

Submit your resulting `disc_total.cpp`, `disc_total.h`, and `disc_total_ck_expect.cpp` files.