CS 131 - Homework 7

Deadline:

5:00 pm on Friday, October 29

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 131hw7`

    ...to change to you directory `131hw7`, and
  - then look at what files are there using `ls`
- then use `~st10/131submit` to submit your `.cpp` and `.h` files for homework number **7**
- make sure that `~st10/131submit` shows that it submitted your `.cpp` and `.h` files for all of your C++ functions and classes for this homework

Purpose:

To practice using C++ `if` statements, creating simple C++ classes, creating testing functions for those classes, and creating functions expecting instances of those classes.

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.

- Remember 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 `131hw7` -- type the following from your home directory on nrs-labs:

```
[you1@nrs-labs ~]$ mkdir 131hw7
[you1@nrs-labs ~]$ chmod 700 131hw7
[you1@nrs-labs ~]$ cd 131hw7
```

(If you log out and come back later, remember to `cd 131hw7` each time to return to *this* directory!)

Problem 1

Develop a C++ version `get_discount` of the Racket function you developed for Homework 3's Problem 2, except with the following slightly-altered specification:

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 C++ function `get_discount` 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_discount.cpp`, `get_discount.h`, and `get_discount_ck_expect.cpp` files.

Problem 2

Develop a C++ version `get_disct_total` of the Racket function you developed for Homework 3's Problem 3, except with the following slightly-altered specification:

Design a function `get_disct_total` that, given a string representing the level of frequent shopper and the total of a purchase before discount, produces the appropriate discounted total for that purchase. For full credit, your solution must appropriately use your function from Problem 1.

Submit your resulting `get_disct_total.cpp`, `get_disct_total.h`, and `get_disct_total_ck_expect.cpp` files.

Problem 3

Develop a C++ version `workout` of the Racket function you developed for Homework 3's Problem 4, except with the following slightly-altered specification:

This problem considers an important consequence of increased pizza consumption – the need for additional exercise.

Develop a function `workout` that determines the number of hours of exercise required to counter the excess fat from eating pizza. `workout` expects a number that represents daily pizza consumption, in slices, and produces a number, in hours, that represents the amount of exercise time that you need.

For a daily intake of :	You need to work out for :
0 slices	1/2 hour
1 to 3 slices	1 hour
>3 slices	1 hour +1/2 hour per slice above 3

Submit your resulting `workout.cpp`, `workout.h`, and `workout_ck_expect.cpp` files.

Problem 4

Develop a C++ class `rhino`. Rhinos, for our purposes, have a weight, a color, and an irritability index (which is in the interval [1, 10] and is always an integer).

4 part a

Write a C++ class **data definition** for `rhino` using the same style as used in C++ class `boa`, as well as a **template for a function** that expects a C++ `rhino` class instance `a_rhino`.

You will pasting comments containing these into both of your files for Problem 4 part b.

4 part b

Create `rhino.h` , containing class `rhino`'s class definition, and `rhino.cpp`, containing class `rhino`'s class implementation. Each of these files should include comments containing your `rhino` data definition form Problem 4 part a.

(Note that you will probably type these in by-hand using `nano`, although if you prefer you can paste in the `boa` or class template version first and then edit that for `rhino`.)

4 part c

Using `funct_play2`, design and implement a test function `rhino_test` for testing class `rhino`'s constructor and selector methods. When `funct_play2` asks at the beginning about whether this new function uses other things, make sure you answer `y`, it uses class `rhino`!

Submit your files `rhino.h`, `rhino.cpp`, `rhino_test.h`, `rhino_test.cpp`, and `rhino_test_ck_expect.cpp`.

Problem 5

Using `funct_play2`, develop a C++ function `is_safe_to_feed` which expects a `rhino`, and produces whether it is safe to feed that rhino right now or not. How do you know? If his/her irritability index is greater than or equal to 5, then it is NOT safe to feed that rhino right now...

Again, when `funct_play2` asks at the beginning if this new function uses other things, remember to answer `y`, it uses class `rhino`.

Submit your files `is_safe_to_feed.h`, `is_safe_to_feed.cpp`, and `is_safe_to_feed_ck_expect.cpp`.

Problem 6

Consider a small taxi company. In this scenario, we are interesting in the following information about each company taxi: how many people it can hold (an integer), how many suitcases it can hold (an integer), how many gallons its gas tank can hold (a double), and whether or not it can accommodate a child car safety seat (a bool).

6 part a

Write a C++ class **data definition** for `taxi` using the same style as used in C++ class `boa`, as well as a **template for a function** that expects a C++ `taxi` class instance `a_taxi`.

You will pasting comments containing these into both of your files for Problem 6 part b.

6 part b

Create `taxi.h` , containing class `taxi`'s class definition, and `taxi.cpp`, containing class `taxi`'s class implementation. Each of these files should include comments containing your `taxi` data definition form Problem 6 part a.

(Note that you will probably type these in by-hand using `nano`, although if you prefer you can paste in the `boa` or class template version first and then edit that for `taxi`.)

6 part c

Using `funct_play2`, design and implement a test function `taxi_test` for testing class `taxi`'s constructor and selector methods.

Submit your files `taxi.h`, `taxi.cpp`, `taxi_test.h`, `taxi_test.cpp`, and `taxi_test_ck_expect.cpp`.

Problem 7

Using `funct_play2`, develop a C++ function `fillup_cost` which expects a `taxi` and a price per gallon of gasoline, and produces how much it would cost to fill up that taxi for that gasoline price.

(Since the examples for `fillup_cost` are likely to involve comparisons of `double` values, remember the testing technique of seeing if the absolute value of the difference between the actual call result and the expected value is "small" enough, say less than .001.)

Submit your files `fillup_cost.h`, `fillup_cost.cpp`, and `fillup_cost_ck_expect.cpp`.

Problem 8

Using `funct_play2`, develop a C++ function `is_big_enough` which expects a `taxi`, a desired number of passengers, and a desired number of bags, and produces whether that many passengers and bags can fit in that taxi.

Submit your files `is_big_enough.h`, `is_big_enough.cpp`, and `is_big_enough_ck_expect.cpp`.