CIS 130 - Homework #7

Due:

Thursday, April 22nd, 11:59 pm

Purpose:

Mostly practice writing C++ functions and programs involving repetition.

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 130hw07`)
- 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 (more have been added for C++ -- see the in-class examples and postings).

Homework problems:

Problem 0

Create, protect, and change to a directory `130hw07` -- type the following from your home directory on nrs-labs:

```
[you1@nrs-labs ~]$ mkdir 130hw07
```

```
[you1@nrs-labs ~]$ chmod 700 130hw07
[you1@nrs-labs ~]$ cd 130hw07
```

(If you log out and come back later, remember to `cd 130hw07` each time to return to this directory!)

Problem 1

Using `funct_play2`, develop a C++ function `line_of_X` that expects a desired number of X's, and doesn't return anything, but has the side-effect of printing to the screen that many X's, followed by a newline character.

That is, the call:

```
line_of_X(3);
```

...would return nothing, but would have the side-effect of causing the following to be printed to the screen:

```
XXX
```

(remember that it SHOULD output a newline at the end of those 3 X's.)

Because `line_of_X` is a `void` function, to test it, use `nano` (or your favorite text editor) to write a small testing `main` function in a file named `line_of_X_test.cpp`, using the `main` function template (`main_template.txt`) available on the public course web page. (Helpful hints: make sure you have a `#include` line for `line_of_X.h` at the beginning of this `main` function, and remember you can compile this using `g++`, with or without help from `compile_helper`.)

Here are the requirements for `line_of_X_test.cpp`:

- it must call `line_of_X` at least 3 times, each time with a different argument value
- precede each of these calls with a `cout` saying how many X's ought to be seen on the next line -- then anyone looking at this program's results can reasonably tell if the testing calls worked.

Submit your files `line_of_X.h`, `line_of_X.cpp`, and `line_of_X_test.cpp`.

Problem 2

Using `funct_play2`, develop a C++ function `box_of_X` that expects a desired number of rows and a desired number of X's per row, and doesn't return anything, but has the side-effect of printing to the screen that many rows of X's, each with that many X's per row, ending up with printing a newline character. This function must appropriately call `line_of_X`.

(After all -- each time you want the side-effect of a row of X's printed to the screen, you should know after Problem 1 how to get that...)

That is, the call:

```
box_of_X(3, 5);
```

...would return nothing, but would have the side-effect of causing the following to be printed to the screen:

```
XXXXX
XXXXX
```

```
XXXXX
```
(remember that it SHOULD output a newline at the end of the "box".)

Again, because `box_of_X` is a `void` function, to test it, use `nano` (or your favorite text editor) to write a small testing `main` function in a file named `box_of_X_test.cpp`, using the `main` function template (`main_template.txt`). (When you compile this program, what *three* `.cpp` files do you need to be sure to list in the `g++` command?)

Here are the requirements for `box_of_X_test.cpp`:

- it must call `box_of_X` at least 2 times, each time with different argument values
- precede each of these calls with a `cout` saying how many rows of how many X's ought to be seen starting on the next line -- then anyone looking at this program's results can reasonably tell if the testing calls worked.

Submit your files `box_of_X.h`, `box_of_X.cpp`, and `box_of_X_test.cpp`.

Problem 3

Using `funct_play2`, develop a C++ function `triangle` that expects the number of rows desired for a triangle, and doesn't return anything, but has the side-effect of printing to the screen a "triangle" of X's that many rows tall, with one X in the first row, two X's in the second row, and so on, ending up with printing a newline character. This function must appropriately call `line_of_X`.

That is, the call:

```
triangle(5);
```

...would return nothing, but would have the side-effect of causing the following to be printed to the screen:

```
X
XX
XXX
XXXX
XXXXX
```

(remember that it SHOULD output a newline at the end of the "triangle".)

Again, because `triangle` is a `void` function, to test it, use `nano` (or your favorite text editor) to write a small testing `main` function in a file named `triangle_test.cpp`, using the `main` function template (`main_template.txt`). (When you compile this program, what *three* `.cpp` files do you need to be sure to list in the `g++` command?)

Here are the requirements for `triangle_test.cpp`:

- it must call `triangle` at least 2 times, each time with a different argument value
- precede each of these calls with a `cout` saying how how tall a triangle ought to be seen starting on the next line -- then anyone looking at this program's results can reasonably tell if the testing calls worked.

Submit your files `triangle.h`, `triangle.cpp`, and `triangle_test.cpp`.

Problem 4

Consider your `triangle` program from Problem 3. A client has decided that she would like a program that interactively asks how many such triangles (of different heights) are desired, and then it should ask for precisely that many triangle heights, each time then printing to the screen (with the help of the `triangle` function) a triangle of that height.

Use `nano` (or your favorite text editor) to write a `main` function in a file `many_tris.cpp` that does this. You are required to call `triangle` appropriately in your solution.

We don't have a good way to write testing functions for `main` functions, but of course you can and are expected to test-run such a program on an appropriate variety of test cases until you are convinced that it is working properly. (In this case, it is also wise to test it on a case in which the user says, when asked, that he/she wants 0 triangles. What should happen in that case?) Test-run your `many_tris` program until you are satisfied that it works properly, and submit your file `many_tris.cpp`.

Problem 5

Recall the function from Homework 6, Problem 1, `workout`, that expects 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.

This function could be used within a sentinel-controlled loop to let a user know the amount of exercise time needed for different quantities of pizza.

Use `nano` (or your favorite text editor) to write a `main` function in a file `advice.cpp` that provides an interactive "interface" for the function `workout`. (Remember, you can obtain a copy of an example version of `workout.h` and `workout.cpp` from the Homework 6 "Selected solutions" section on the course Moodle site, if you need or would like to.) You will need to have copies of `workout.h` and `workout.cpp` in your `130hw07` directory.

That is, your `main` function will prompt the user for different amounts of daily pizza consumption in slices, and then use `workout` to determine the amount of exercise time needed to work off each such pizza quantity, printing a clear message to the screen including that exercise amount, until the user indicates that he/she would like to stop.

Make sure that this `main` function uses a properly structured sentinel-controlled loop in asking the user to enter for the next daily pizza consumption value, using `workout` appropriately to then compute and then display the exercise amount needed for working off that level of pizza consumption. What would be an appropriate sentinel value for this situation? Decide, and include what the user needs to type in to quit as part of your interactive prompt to the user.

Test-run your `advice` program on an appropriate variety of test cases until you are convinced that it is working properly. (In this case, it is also wise to test it on a case in which the user immediately enters the sentinel value, the first time he/she is asked. What should happen in that case?) Then, submit your file `advice.cpp`.