CIS 130 - Intro to Programming - Spring 2007
Homework Assignment #8 - **INDIVIDUAL** assignment

Homework #8 DUE**: BEGINNING** of class, Wednesday, April 18, 2007

Purpose: get practice with simple C++ functions involving if statements, local variables, while statements, and simple interactive input/output.

Note that use of the design recipe is still required for all functions, including C++ functions! But, use **C++ types** in **contracts** for C++ functions, and use == for examples for non-void C++ functions.

1. Remember **work_out** from HW #3? Write a C++ version of **work_out** that takes the number of slices of pizza eaten daily as its parameter, and returns the number of hours of exercise required to work off that level of pizza-slice consumption, based on the following:

intake	work-out required
0 slices	0.5 hr
1-3 slices	1 hr
>3 slices	1 hr + (0.5 hr per slice > 3)

2. Remember **pieces_produced** from HW #4? Write a C++ version of **pieces_produced** that takes the number of workers on duty during a given hour of the day, along with that hour of the day (expressed in 24-hour format), as its two parameters, and returns the number of pieces that can be produced by that many workers during that hour, based on the following productivity:

between the hours of 6 am and 10 am --- that is, from 6-7, 7-8, 8-9, and 9-10 --- they could produce 30 pieces/hour/worker;

between 10 am and 2 pm --- from 10-11, 11-12, 12-1, and 1-2 --- they could produce 40 pieces/hour/worker;

and between 2 pm and 6 pm --- from 2-3, 3-4, 4-5, and 5-6 --- they could produce 35 pieces/hour/worker.

At all other hours, 0 pieces are produced per hour per worker --- the company is closed then.

3. Before we throw interactive input and output into the mix, I'd like you to try your hand at a C++ function with a "plainer" while-loop.

Important note: **expr_play** (and **funct_play2**'s testing) sometimes do NOT do well with **void** functions (functions that do not return anything). I hope to talk about why next week. BUT, in the meantime, we are going to avoid writing functions that don't return SOMETHING. Please keep this in mind, and don't forget that **return** statement... However, if a function prints to the screen AND returns something, these tools actually still work (as long as they print what they want BEFORE they try to return... 8-))

Write a C++ function **multiples** that takes two parameters, the integer for which multiples are desired, and how many multiples are desired. It then prints to the screen that many multiples of that value, one per line including the multiplication done as shown below, starting with that value times 0, and when complete it **returns** the the final multiple.

That is, if you called **multiples(6, 8)**, to the screen should be printed:

```
6 * 0 = 0
6 * 1 = 6
```

```
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
```

...and then it would **return** 42**. (Note** that, run using **expr_play**, this would actually show up as:

```
multiples(6, 8)
would have the value:
6 * 0 = 0
6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
42
```

...which is *somewhat* reminiscent of how prints and function expression returns are jumbled in the **python** interpreter, too...)

4. Remember **average_grades** from HW #5? Write a C++ version of **average_grades** that takes the number of grades to be averaged as its parameter, and then it tries to interactively ask for that many numeric grades, then computing and returning the average of those grades.

5. But, how about an averaging function for those who don't want to say in advance how many they want to enter? Write a C++ function **average_grades2** that takes NO arguments, but asks for grades to be entered until a grade that is less than 0 is entered (signalling that the user is done) --- it then **returns** the average of the grades entered (NOT including the negative grade signalling completion --- that should NOT be considered a "real" grade). For full credit, you are expected to use the "classical" loop style discussed for this kind of situation.

6. Write a C++ function **num_above** that takes two parameters, a number of values desired and a **double** limit value. It then tries to interactively ask for that many values, comparing each to the parameter limit, and **counts** how many of the entered values **exceed** that limit. When done, it **returns** the number of values entered that exceeded that limit.

For example, if the user called **num_above(4, 50.0)** and typed in 75, 2.7, 49, and 51.1, then **num_above(4, 50.0) == 2** (two of the four values entered were greater than the argument limit of 50.0).

This has *some* similarities to **count_As** from HW #5...

7. Remember **dollar_to_euro** from HW #7? Write a C++ function exchange that takes a single parameter, the number of dollar amounts to be entered, and then interactively asks the user to enter that many dollar amounts, each time **calling** dollar_to_euro to determine the amount of Euros for that dollar amount, and printing ithat dollar amount's Euro equivalent to the screen, but also summing up these Euros equivalents and at the end returning the total in Euros for all of the dollar amounts entered.

For example, if someone calls **exchange(4)** and enters **10**, **20**, **1**, and **4**, then it would print to the screen (interleaved with the prompts, and yours might vary based on a more-current exchange rate):

```
in Euros, that is: 7.86985
in Euros, that is: 15.3793
in Euros, that is: .786985
in Euros, that is: 3.07586
```

...and then it would **return** 26.913775.

(Remember, when you test this – you need to enter both **dollar_to_euro**'s and **exchange**'s names into ~st10/expr_play, to test exchange!)

When you are happy with these functions, you can either submit the .cpp and .h files for each, OR you can use the following quickie-tool to build a file containing all of them (a tar file) and submit that one file instead (IF you have named your functions PRECISELY as given above...):

...if you are interesting in the quickie tool, then type the following at the cs-server prompt:
~st10/het_hw08

...give the name of a directory you want built, and when done, if all 14 files are listed on-screen as being in your new file, then you can submit the file whose name it tells you at the end.

(Note: you STILL use ~st10/130submit to submit this homework! But it is your choice if you submit the 14 .cpp and .h files in the usual way, OR use ~st10/get_hw08 and submit the single file it builds containing (hopefully) your 14 .cpp and .h files.)