CIS 130 - Intro to Programming - Fall 2005
Homework Assignment #6

HW #6 PART 1 is due by **12:00 NOON** on **Wednesday, October 19, 2005;**
HW #6 PART 2 is due by **12:00 NOON** on **Friday, October 21, 2005**

Week "8" Lab Exercise - there is NO Week "8" Lab Exercise

* Note: when I tried running expressions such as:

```
is_legal_PAT_score(199) == false
```

...using expr_play, it worked just fine. However, funct_play0, funct_play1, and funct_play2 were using some obsolete code in their running-expressions sections. Since these have now been fixed, please re-copy them and make sure that you are running the **10-12-05** version (the opening message includes the version date):

(make sure you are in your home directory on cs-server...)

```
cp ~st10/bin/funct_play0 bin
cp ~st10/bin/funct_play1 bin
cp ~st10/bin/funct_play2 bin
chmod 700 bin/funct_play*
```

Now, expressions such as that above should work in the **funct_play*** family of tools, too (assuming you have answered the opening questions appropriately, of course).

<u>HOMEWORK #6</u>

You are to work **individually** on this assignment.

PART 1: (30% of the HW #6 grade) Create a file **130hw6_part1.txt** on cs-server. Inside this file, type the following:

* your Contract for the function you are to write for **problem #1** below;
* your Examples part of the design recipe for any one of the functions that you are to write for **problem #2** below;
* your Contract for **overtime_owed** in **problem #4** below;
* your Contract for **workOut** in **problem #5** below.

(You may decide to write auxiliary functions in the course of some of these problems; this time, those contracts are not required to be turned in as part of Part 1. Only the above pieces are required for Part 1.)

You must submit this file using ~st10/130submit (typed at the cs-server prompt, from the directory where your 130hw6_part1.txt file resides!!) by **12:00 noon** on **Wednesday, October 19th** to receive **any** credit for Part 1 of HW #6.

PART 2: (70% of the HW #6 grade) Finish the problems below. For all except problem #3, you must enter your functions using an appropriate choice of the **funct_play*** tools (remembering that, except as noted, all design recipe elements ARE expected and required), and test them using either **funct_play*** or **expr_play**.

When you are ready, you must submit:

* problem #3's **130hw06_num3.txt** file;

* all of your **.cpp** and **.h** files for:
    * problem #1's function;
    * problem #2's three functions;
    * problem #4's **overtime_owed** function (and any **new** auxiliary functions, besides worked_overtime, that you use, if any)
    * problem #5's **workOut** function (and any auxiliary function that you use, if any)

...using **~st10/130submit**. These must be submitted by **12:00 noon** on **Friday, October 21st** to receive **any** credit for Part 2 of HW #6.

REMINDERS:

* You are now REQUIRED to use the **program design recipe** for all functions, from here on out, and your functions need to each include the program design recipe elements (including contract, purpose, and specific examples) unless explicitly stated otherwise.

* When possible, you should write your Examples using == notation, as discussed in lecture.

* You are expected to use **bool** rather than int for Boolean situations, and you are expected to write **true** and **false** instead of 1 and 0 for logical literal values (again, as discussed in lecture).

* You are expected to test all Examples using funct_play* or expr_play; you may of course run as many additional testing calls as you wish.

1. Trying boolean functions, part 1 (adapted from Exercise 4.2.3, pp. 31-32. You may want to look at this, as it provides a useful example and some good "set-up".)

Consider the following equation (written in mathematical notation, *not* in C++ syntax):

$4 \cdot n^2 + 6 \cdot n + 2 = 462$

Write a boolean function that will return true if a value **n** satisfies this equation, and will return false if it does not. Because of the nature of this particular function, named constants are **not** required, and I'll be expecting generic names for the function and for its parameter. Also, you do not have to have an Examples section for this function; however, you should test it on the values **10**, **12**, and **14** using either one of the funct_play* tools or expr_play.

(You DO need to follow the rest of the design recipe...)

2. Trying boolean functions, part 2 (adapted from Exercise 4.2.2, p. 30): use **funct_play1** or **funct_play2** to translate the following intervals on the real number line into C++ functions that each expect a floating-point number as its parameter and returns true if that number is in the interval shown and returns false if that number is outside the interval.

Because of the nature of these particular functions, named constants are not required, and I'll be expecting generic names for the functions and for each one's parameter. You **do** need an appropriate Examples section for each, however, with a sufficient selection of examples as discussed in lecture. And, they should be expressed using ==.

(Note that, in a "real" problem involving an interval, the "boundaries" would likely have meanings such that named constants would be reasonable and useful.)

(a) the interval (3,7]:

```
            (---------------]
|---|---|---|---|---|---|---|---|---|---|---|---
0                   5                   10
```

(b) the union of (1,3) and (9,11):

```
    (-------)                       (-------)
|---|---|---|---|---|---|---|---|---|---|---|---
0                   5                   10
```

(c) the range of numbers outside of [1,3].

```
<----)       (------------------------------------>
 |---|---|---|---|---|---|---|---|---|---|---|---
 0                   5                   10
```

3. Create a file **130hw06_num3.txt**. In it, type answers to the following:

(a) (adapted from Exercise 4.2.2, p. 31)
Consider the following number line template:

```
|---|---|---|---|---|---|---|---|---|---|---|---
0                   5                   10
```

And, consider how the intervals were depicted on such number lines in problem #2, above, and in the reading packet.

Type out intervals depicted on number lines that correspond to the following functions:

a1.
```
bool inInterval1 (double num)
{
    return (-3 < num) && (num < 0);
}
```

a2.
```
bool inInterval2 (double num)
{
    return (num < 1) || (num > 2);
}
```

a3.
```
bool inInterval3 (double num)
{
    return !((1 <= num) && (num <= 5));
}
```

(b) Type out answers for Exercise **4.3.1**, pp. 35-36. (Be sure to answer both parts.)

4. Consider the function **worked_overtime** from HW #5.

Use this function in another function, **overtime_owed**. This function expects, as its parameters, the number of hours worked, and the overtime rate of pay. If the number of hours worked exceeds 40, then **overtime_owed** should return how much overtime wages are due in this case --- overtime wages are computed by multiplying JUST the hours over forty by the overtime rate of pay. But, if no overtime was worked, then the function should return 0 (no overtime is owed in this case).

For example, **overtime_owed(43, 12.50)** == **37.5**, because there are 3 overtime hours at 12.50 overtime per hour. But, **overtime_owed(37, 15.00) == 0**, because no overtime wages are owed for this case.

Carefully note the following requirements:

* you must appropriately call your **worked_overtime** function from HW #5.

* you must use an **if-statement** appropriately in your solution.

* make sure that your named constant for 40 hours a week being the "boundary" for overtime is included in your **worked_overtime.h** file. (If it wasn't there before --- put it there now!)

Then, you will also be able to use it in **overtime_owed** without re-declaring it. And, you should do so.

5. (Adapted from Keith Cooper's section of Rice University's COMP 210, Spring 2002) Conditionals (and Pizza Economics)

In class, we have developed a function intended to deal with some of the economic aspects of eating pizza. Our earlier function, however, ignores an important consequence of increased pizza consumption —the need for additional exercise.

Develop a function **workOut** that computes the number of hours of exercise required to counter the excess fat from eating pizza. **workOut** expects as its parameter a number that represents daily pizza consumption, in slices, and returns 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