CISP 400 - Lab Assignment #6
Due Thursday, October 11, 2018 at 11:59 PM

This assignment is a programming exercise intended to allow you to work with many of the object interaction programming techniques described in the lecture. These include:

- Friends of Classes of objects
- Overloaded functions
- Overloaded operators
- Copy constructors
- Type conversion operators
- Converter constructors
- Constant functions (combined with overloaded operators)

SPECIFICATIONS:

You are writing software to keep track of the number of hours a worker has been on the job.

The part of the program you are currently writing is a class that tracks the hours of the worker. Let's call this class WorkerHours. It stores the number of hours worked so far as an int.

The class is going to need to be very versatile. It will need to do the following:

1. Add two objects using the + operator, with the result being that the hours worked are added together.
2. Add one object to another using the += operator
3. A default constructor that sets hours to 0.
4. A convert constructor that accepts an int parameter.
5. Use casting to convert to a double. The double will be the total number of hours recalculated to the number of days worked. To do this calculation, divide the number of hours by 24.
6. The class will need an external function that prints out the data. This will need to be a friend to the WorkerHours class, and access the private members directly in order to do this. It should show the number of hours worked, the number of days worked total, and the number of copies made.
7. There will be two overloaded functions, called setData(). These will take an int parameter and a double parameter. setData(int) is intended to set HOURS, should take the value and set the appropriate member variable. setData(double) should be calculated into hours, and the hours should then be stored as if setData(int) had been used. You can throw away the decimal point in this.
8. A copy constructor. The copy constructor will behave in a way described elsewhere in the lab prompt.

Additionally, there will be one function called showInternalData. It will produce much of the output. It takes a label of type string, and then outputs the label, and on the next line, the contents of the WorkerHours object. **<u>The showInternalData function MUST be a friend of the WorkerHours class.</u>** It should not use member functions, but instead, access the member variables directly.

Programs that are written to test code that is not a complete program are sometimes called "drivers," which should NOT be confused with hardware drivers. I am providing a driver for you to use to test your code with. You can download this file separately as a .cpp file.

```
#include <iostream>

using namespace std;

const int JANE_HOURS = 30, JIM_HOURS = 20, SETTER_HOURS = 40;
const double SETTER_DAYS = 3.0, HOURS_PER_DAY = 24.0;

// This is the prototype of the showInternalData function.
// It must access INTERNAL STRUCTURES in the worker object.
// Do NOT use member functions to get the data for that object.

void showInternalData(string label, const WorkerHours &worker);

int main()
{
     // Conversion constructor
     WorkerHours jane = JANE_HOURS, jim = JIM_HOURS;
     // Copy constructor
     WorkerHours janeCopy = jane;
     // + operator
     WorkerHours combined = jane + jim;
     // Default constructor
     WorkerHours testSetters;

     // Variables set aside for calculations
     double daysWorked;
     int hoursWorked;

     // You can use static_cast here, but it shouldn't be required.
     // static_cast<double>(combined);

     // Type conversion operator - int
     daysWorked = combined;
     cout << "TEST DAYS WORKED : " << daysWorked << endl;

     // You can use static_cast here, but it shouldn't be required.
     // static_cast<int>(combined);

     // Type conversion operator - double
     hoursWorked = combined;
     cout << "TEST HOURS WORKED: " << hoursWorked << endl;

     // Now we start using the internal function
     showInternalData("Jane", jane);
     showInternalData("JaneCopy", janeCopy);
```

```
    showInternalData("Jim", jim);
    showInternalData("Combined", combined);

    // += operators

    jane += janeCopy;
    showInternalData("Jane + JaneCopy", jane);

    // Now we test the overloaded setters
    testSetters.setData(SETTER_HOURS);
    showInternalData("Testing int setter", testSetters);
    testSetters.setData(SETTER_DAYS);
    showInternalData("Testing double setter", testSetters);

    // We're done
    return 0;
}
```

COPY CONSTRUCTOR:

We want to keep track of how many copies have actually been made of a specific object. As such, we will have a member variable called "copies," which will be an int. ANY constructor EXCEPT the copy constructor should set this variable to 0. The COPY CONSTRUCTOR should take this value from the object that you are copying, and add 1 to it. So, for example, if you create an object using the convert constructor, 'copies' will be 0. Pass it by value to a function, then it will have a value of one. The copy constructor will also copy the hours worked (since it would be pretty useless otherwise).

TIPS:

- I have provided a lot of examples. Use them.
- Look up the prototype for the += operator overload. It will take a little while. Also remember the difference in terms of returning a reference (&) and returning a copy (no &) when returning an object.
- Be careful not to overthink the problem. Most of the things asked here take a few lines of code each.
- It may be helpful to put together the class first, step by step.
- Read the Copy Constructor instructions very carefully if you are confused.
- Remember, we are dealing with TWO types of numerical data. Hours are expressed as integers (int). Days are expressed as floating point (double). The class member variables store it as an int. You will need to multiply or divide to change between the two. This should be handled internally by the class. It SHOULD NOT be stored; I want to see your program do the actual calculation dynamically when needed.

USER INPUT:

You do not need to process user input for this assignment.

DELIVERABLES:

Submit the .cpp file to Canvas by the due date. Multiple files are not necessary for this assignment.