**CPSC 5011 Object-Oriented Concepts**

***This assignment exercises your understanding of operator overloading in C++***

*For an acceptable P4 submission:*

1. **use C++ --** the g++ compiler (C++17) on cs1
   a. programs developed in Visual Studio often do NOT compile on cs1
   b. HIGHLY recommended to use C++ tools -- g++, CLion, Xcode etc.
   c. Submissions that do not compile on cs1 will NOT receive credit
2. upload all .h and .cpp files to cs1 AND to Canvas (do not use .hpp)
   ***Do not upload zip files!!***
   (jumpPrime.h, jumpPrime.cpp, duelingJP.h, duelingJP.cpp, p4.cpp)
   use the submission script to upload and compile:
   /home/fac/sreeder/submit/cpsc5011/p4_runme
3. Overload all appropriate operators. ***Focus on the expectations of the client.***
4. Fulfill requirements as specified in steps 3-7 from P1

**Part I: Class design**

Overload all appropriate operators for the classes from P2: **jumpPrime** and **duelingJP**
The class (type) definitions are the same EXCEPT now may encapsulate a 3-digit number as well as those of 4 or more digits.

A primary design goal is to streamline manipulation of **jumpPrime** and **duelingJP** objects.

**Comparison** should be widely supported for all types.

**Addition MUST** be supported consistently for both types: you decide the meaning and extent.

Consider, for example, if it is reasonable to: add a ***duelingJP*** to a ***duelingJP***? a ***jumpPrime*** to a ***duelingJP***? a ***jumpPrime*** to a ***jumpPrime***? etc.

Determine the ripple effect(s) of supporting addition, including mixed-mode, short-cut assignment and pre & post increment. Make reasonable design decisions so that your classes satisfy the stated goals, communicate assumptions and use, and yield clear and maintainable software

Clearly, many, many details are missing. **You must use operator overloading.**

**Use ProgrammingByContract** to specify:
pre and post conditions; interface, implementation and class invariants.
Intent of operator overloading should be well-documented

**Part II: Driver**

Design a ***functionally decomposed*** driver to demonstrate program requirements.

Clearly specify the intent and structure of your driver

You should have arrays of distinct objects, initialized appropriately, i.e.

random distribution of objects with arbitrary, initial, reasonable values

meaningful values for non-arbitrary initial values, etc.

do NOT use vectors, lists, etc.