CMPT 470/816 HOME EXERCISE 2

CODE CONTRACTS

Within this exercise, you will be asked to provide *preconditions* and *postconditions* for the answer to take home exercise 1. For those not familiar with them from earlier classes, we provide some introduction to these topics here.

Preconditions represent things on which the function is depending – conditions counted on for it to do its job, and should be guaranteed by the caller. These are almost always expressed such that they involve the parameters (e.g., saying that a given parameter is non-null, or another parameter must be greater than 0, or that one parameter is greater or equal to another. Preconditions also can involve the state of global variables or constructs at the time that the function is called. They can also involve aspects of history (e.g., the fact that this function should only be called once).

Postconditions indicate properties that are guaranteed when the function finishes (e.g., returns a value, or – for "procedures" such as are associated with functions "returning" void in C -- simply returns). Please note that postconditions desired go beyond simply stating broad criteria for the results of a function (such as that the return value is >= 0, or that a specified global array is modified, although those are welcome), but also include characterizing how those postconditions relate to the inputs provided – i.e., ***what*** is it that the function is *doing*? (i.e., what is the purpose of the function)? Clearly, such information is of key significance for a user trying to determine whether the function can meet their needs – especially if the code is not available (e.g., if your code were part of a library, and that user were using that library). Much of this behaviour of the function can be provided in the comments for the function as a whole (such as in the examples given in class), but if so can be referred to by the postcondition. It can further be sometimes be useful to emphasize history-related properties associated with the function if the function is *idempotent* – that is, if the return value or return state of the function is the same regardless as to how many times it is called on the same arguments.

Building on the last take home exercise, please use comments to provide code contracts for each of the functions in the refactored code in the example solution using provided to you via moodle for home exercise 1.

Due in class January 17, 2017