CPSC 3200 Object-Oriented Development

Programming Assignment #4: Due Thursday May 19, 2022 before MIDNIGHT

P4 exercises your understanding of operator overloading and memory management in C++. P4 reuses P2 directly with the following changes:

1) Modify the driver, P4.cpp, to use smart pointers and the STL
   a. Your move semantics from P2 will thus be tested again.
   b. If needed, correct P2 move constructor and/or move assignment operator
2) Augment the type definition of **gridFlea**, as noted below
3) Expand the interfaces of both **gridFlea** and **inFest** to include the overloading of comparison and addition (see below) and all other appropriate operators.
   a. ***Focus on client expectations for use of type.***
   b. *Remember type definition is concerned with consistency.*

Use ProgrammingByContract for documentation. ***DO NOT hard code***

Part I: Class design

1) class (type) definitions from P2 are the same EXCEPT now any **gridFlea** may jump outside grid boundaries, *at most once*, **and** the jump cannot be more than z squares away from the boundary.
2) Overload operators in both type definitions to meaningfully support comparison
3) Support addition for both types, via one or more operators
   a. you decide meaning and extent**
   b. At least one type should support ++
   c. Determine the ripple effect(s) of supporting addition
      i. mixed-mode addition?
      ii. short-cut assignment?
      iii. pre & post increment.?
   d. Make reasonable design decisions so that your classes streamline the manipulation of **gridFlea** and **inFest** objects for clients
   e. Communicate assumptions and use via ProgrammingByContract
4) Define and implement any additional overloaded operators needed for consistency

**Consider, for example, if it is reasonable to add:
a number to either type?
a ***gridFlea*** to a ***gridFlea***?
a ***inFest*** to a ***inFest***?
a ***gridFlea*** to a ***inFest***? etc.

Clearly, many, many details are missing.

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.
Unit testing is not required or expected.

You should have *a collection* of distinct objects, initialized appropriately, i.e.
random distribution of objects with arbitrary, initial, reasonable values
meaningful values for non-arbitrary initial values, etc.
To test move semantics, you must use some type -- *vectors, lists, etc., from the STL*

Additionally:

1) Do NOT use raw pointers
2) Demonstrate your understanding of smart pointers by using both `unique_ptr` and `shared_ptr` instead of raw pointers to reference heap-allocated *`inFest`* objects
3) Use a STL container
   Add and remove heap-allocated *`inFest`* objects to this container
4) Demonstrate copying of *`inFest`* objects via call by value

Upload your files to BOTH Canvas and cs1
cs1 uses same submission process – substitute ‘p4’ for ‘p2’
/home/fac/dingle/submit/22sq3200/p4_runme

Do NOT upload zip files