Exercise C++

Note: it is not required to provide fully compilable code. It can also be pseudo code or a description of the solution. Please do not hesitate to ask for clarifications when necessary.

Provide classes to hold n-dimensional keys and their associated values (let's call them data storage classes). A data storage can be configured (at construction) with a number of dimensions and the common size of the dimensions (e.g. 3 dimensions of size 10). The keys are implicitly defined by the size of the dimension, i.e. size 10 means keys range from 0 to 9 in each dimension. For each key a random integer value should be associated (e.g. key [0,0,0] has value 14, key [0,0,1] has value 88, key [9,9,9] has value 5). The association of values to keys should be done in the storage construction.

Name the classes ReadableStorageA and ReadableStorageB. They should have different implementations regarding their "read()" function, but also share common functionality. Make sure to apply OOP concepts to avoid code duplication.

common:
- initialization in constructor (random value generation)
- both provide a "info()" function that outputs the total number of stored values to the console (e.g. "storage has 1000 values")
- both provide a "read()" function (but behaviour differs, see below)

specialities:
- ReadableStorageA:
  function "read()" writes the next key in sorted order and its associated value (e.g. first call returns "key: [0,0,0] value: 14", second call returns "key: [0,0,1] value: 88", last call returns "key: [9,9,9] value: 5") to the console
  calling read() after the last key has been written produces an error message "end of storage reached".
- ReadableStorageB:
  function "read()" writes a random key of the storage and its associated value to the console, but never writes the same key/value more than once.
  calling read() after all keys have been written exactly once produces an error message "end of storage reached". *Note: it suffices to describe (not code) the approach for ReadableStorageB.*

Be prepared to discuss how the storage classes can be made thread safe, i.e. multiple threads should be able to read the same storage object such that each thread reads a (random) subset of the storage when iteratively calling "read()".
What are pros or cons when considering mutex vs. atomic based implementations.

Thank you!