Multi-threaded Element Queue component (v5)

Component Requirements

Develop a class from scratch to queue a finite number of primitive types (e.g., `int`). This class supports multi-threading as follows:

- Queue users shall input a maximum capacity during creation.
- Queue users shall be able to get the number of enqueued elements at any moment.
- As any queue, `push()` method adds elements to the Queue, while `pop()` method retrieves elements on a FIFO approach. Pushing and popping shall be synchronized.
- Both push and pop shall have an optional parameter for timeout. If no timeout is given, the pushing/popping shall block indefinitely in case the Queue is full/empty. Otherwise, an exception is thrown after the timeout.
- To support different primitive types, a template class shall be implemented.
- Dynamic memory allocation shall be used to store queue elements (no `std` library for storing elements).
- Performance issues are optionally addressed.

Project requirements

- Develop unit tests for the Queue class with the support of a framework (e.g., *gtest*).
- A CMake configuration shall be used to ease the build process (including tests).
- Consider your class will be used as a utility library by others, so your code shall be documented, preferably using *Doxygen* style.
- Share your code at your *GitHub* or *Bitbucket* account and send us the link.

Example Implementation

As an example, implement the following in `main()` - Reading thread pops elements while Writing thread pushes elements.

```
New Queue<int>(2)
Push(1)     // Queue: 1
             Pop() -> 1
Push(2)     // Queue: 2
Push(3)     // Queue: 2,3
Push(4)     // blocks
             Pop() -> 2 // is released
             // Queue: 3,4
             Pop() -> 3
             Pop() -> 4
             Pop() // blocks
Push(5)     // Queue: 5
             Pop() -> 5 // is released
```