

Multi-threaded element queue in C++

1. Develop a class from scratch to queue a finite number of elements. This class will be used for multi-threaded communication as follows:

- Reading thread pops elements while writing thread pushes elements.
- If the queue is empty, reading thread will block and wait for the next element.
- If the queue is full, writing thread will block and wait for another thread to remove an item.

Extra: The reading/writing threads should optionally block for a certain period of time. If the action is successful within this time, true is returned, otherwise false.

The class interface should look like this:

```
class Queue<T> {
    Queue(int size) {...}
    void Push(T element) {...}
    T Pop() {...}
    int Count() {...} // Amount of elements stored now
    int Size() {...} // Max number of elements
}
```

As an example implement the following in *main()*:

<u>Writing Thread</u>	<u>Queue</u>	<u>Reading Thread</u>
New Queue<int>(2)		
Push(1)	1	
		Pop() -> 1
Push(2)	2	
Push(3)	2, 3	
Push(4) // blocks		
	3	Pop() -> 2
// is released	3, 4	
	4	Pop() -> 3
		Pop() -> 4
		Pop() // blocks
Push(5)	5	
		-> 5 // is released

Notes:

- An instance of `std::queue` may be used internally, but `race conditions` must be avoided.
 - Do not forget to document your code. Preferably use `Doxygen` style.
 - Performance issues are optional.
2. Develop unit tests for the Queue class with support of a framework (e.g. `cppunit`, `googletest`, `catch`).
3. Develop a `CMake` configuration file to ease the build process (inc. tests).
4. Share your code at your Github or Bitbucket account and send us the link.