Following are the basic expected functionalities of `ConcurrentQueue` :

- `Push(x)` : add an element x in the queue.
- `Pop()` : returns the front element of the queue.
- `peek()` : returns true if the queue is not empty.
- `busyWaitforPush()` : wait for a pop operation so that there is space for new elements.
- All functions of the queue should be thread-safe as well.

Segmentation fault:

Let's say at some point in time queue is filled as shown below :

Address	00001	00002	00003	00004	00005	00006	00007	00008	00009
Task Id	6	7	8		1	2	3	4	5

↑ ↑
mWritePtr mReadPtr

Let's say the producer thread pushes a task with `id` = 9, our queue becomes:

Address	00001	00002	00003	00004	00005	00006	00007	00008	00009
Task Id	6	7	8	9	1	2	3	4	5

↑↑
mWritePtr
mReadPtr

Now a pop is issued by the consumer thread following steps are performed:

1. Check if the queue is not empty.
2. If not empty then pop the element and get the reference of the element from the queue. I.e. reference of element at mReadPtr is returned.
3. The function object is called
4. And finally, the functor is deleted.

step 1 and 2 are locked but 3 and 4 are not let's say just after step 2 (i.e. reference of object at memory 00005 is returned) producer resumes and since the queue is not full producer is ready to push and pushes task 10 in the queue now consumer wants to call and delete task 1

but instead, it leads to the function call and deletion of task 10 since object at location 00005 is changed during the push.

Let's say task 10 is deleted but the functor pointer is still in the queue so during future pop we will again pop the pointer to task 10 which is already deleted which leads to double free and segmentation fault. The main cause of segmentation fault is concurrent deletion and creation of heap objects using the `delete` and `new` keywords.
For example, if the producer creates an object and just after that object is deleted by the consumer and again producer tries to allocate memory. However, if the memory allocate to the newly created object is the same as the memory that was just deleted so newly created task points to invalid memory. Now if again a consumer tries to access this invalid memory this will lead to a segmentation fault.