Concurrent Queue Implementation Solutions

std::queue

- Briefly explain why, as it stands, std::queue is not suitable for use as a concurrent queue
  - Needs to be protected against data races
  - "Popping" an element requires two operations (one to get its value, one to remove it) and another thread could interfere between these operations
  - Should be more robust when trying to pop from an empty container

Concurrent Queue Implementation

- Implement a concurrent queue which uses locks
- Make sure your implementation is thread-safe and exception-safe
- In the event that a client tries to pop() from an empty queue, your implementation throws an exception
- Write a multi-threaded program to exercise your implementation in which the exception is caught in the thread which calls pop()
- Modify your program so that the exception is caught in the main thread

Condition Variable

- Modify your implementation as follows:
- In the event that a client tries to pop() from an empty queue, your implementation waits until there is some data on the queue
- Write a multi-threaded program to exercise your implementation

Conclusion

- Suggest how your implementation could be improved (you are not required to write any code for this exercise)
  - The locking is "coarse-grained", meaning that the entire queue is locked every time a thread accesses it. In effect, accessing the queue temporarily converts the program into a single-threaded application
  - "Fine-grained" locking would allow threads to perform different operations concurrently, but would make the code more complex and might not improve the performance
  - Locking is very slow, making the queue inefficient
  - Lock-free programming would improve concurrency and efficiency, but is more difficult to implement correctly