```
#include      <atomic>
#include      <array>

enum {LOCKFREE_CACHELINE_BYTES = 64 };

template<typename T,  std::size_t Size>  class alignas(LOCKFREE_CACHELINE_BYTES)
SPSC_LockfreeQueue{
 // first hot members
  std::atomic<std::size_t>  m_writeIndex{0};
  enum {PADDING_SIZE = LOCKFREE_CACHELINE_BYTES - sizeof(std::atomic<std::size_t>)};
  std::array<char, PADDING_SIZE> m_pad1{};
  std::atomic<std::size_t>  m_readIndex{0};
  std::array<T, PADDING_SIZE> m_pad2{};
  std::array<T, Size> m_buffer{};

  static std::size_t nextIndex( std::size_t index){
      return (index +1 )% Size;
  }
  static bool empty (std::size_t writeIndex, std::size_t readIndex){
      return (writeIndex == readIndex);
  }

  static bool full (std::size_t writeIndex, std::size_t readIndex){
      return (nextIndex(writeIndex) == readIndex);
  }

  bool push(const T &t){
     std::size_t writeIndex =  m_writeIndex.load(std::memory_order_relaxed);
     std::size_t readIndex=  m_readIndex.load(std::memory_order_acquire);
     if( full(writeIndex, readIndex ) ){
         return false;
     }
     m_buffer[writeIndex] = t;
     m_writeIndex.store(nextIndex(writeIndex), std::memory_order_release);
     return true;
  }

   bool pop(T &t){
     std::size_t readIndex=    m_readIndex. load(std::memory_order_relaxed);
     std::size_t writeIndex =  m_writeIndex.load(std::memory_order_acquire);
     if( empty(writeIndex, readIndex ) ){
         return false;
     }
     t = m_buffer[readIndex];
     m_writeIndex.store(nextIndex(readIndex), std::memory_order_release);
  }

};
```