SWO3	Übung zu Softwareentwicklung mit klassischen Sprachen und Bibliotheken 3	WS 2018/19, Angabe 9

☐ Gruppe 1 (J. Heinzelreiter)

☐ Gruppe 2 (M. Hava) Name: _____ Aufwand [h]: _____

☐ Gruppe 3 (P. Kulczycki) Übungsleiter/Tutor: _____ Punkte: _____

Beispiel	Lösungsidee (max. 100%)	Implement. (max. 100%)	Testen (max. 100%)
1 (100 P)			

Beispiel 1: swo3::deque (`src/deque/`)

Implementieren Sie einen ADT `swo3::deque` (*double-ended queue*, siehe https://en.wikipedia.org/wiki/Double-ended_queue) gemäß dem im Folgenden definierten Interface. Eine swo3::deque speichert ihre Elemente in einem Ringpuffer (siehe https://en.wikipedia.org/wiki/Circular_buffer). Testen Sie ausführlich unter Zuhilfenahme von generischen Algorithmen und *range-based for loops* (siehe https://en.cppreference.com/w/cpp/language/range-for). Für eine genaue Spezifikation der einzelnen Komponenten der `swo3::deque` (Typen, Methoden etc.) verweisen wir auf https://en.cppreference.com/w/cpp/container/deque und https://en.cppreference.com/w/cpp/named_req/RandomAccessIterator.

```
namespace swo3 {

/**
 * see https://en.cppreference.com/w/cpp/container/deque and
 *     https://en.cppreference.com/w/cpp/named_req/RandomAccessIterator
 */
template <typename T> class deque final {
   using value_type = ...
   using reference  = ...
   using size_type  = ...

   class iterator final {   // implements RandomAccessIterator
      ...
   };

   deque ();
   explicit deque (size_type count);
   deque (size_type count, T const & value);

   deque (deque const & other);
   deque (deque && other);
   deque (std::initializer_list <T> init);

  ~deque ();

   deque & operator = (deque const & other);
   deque & operator = (deque && other) noexcept;
   deque & operator = (std::initializer_list <T> init);

   reference operator [] (size_type pos);

   reference at    (size_type pos);
   reference back  ();
   reference front ();
```

```
   iterator begin () noexcept;
   iterator end   () noexcept;

   bool      empty () const noexcept;
   size_type size  () const noexcept;

   void clear () noexcept;

   void push_back (T const & value);
   void push_back (T && value);
   void pop_back  ();

   void push_front (T const & value);
   void push_front (T && value);
   void pop_front  ();

   void resize (size_type count);
   void swap   (deque & other) noexcept;

   ...
};

template <typename T> bool operator == (deque const & lhs, deque const & rhs);
template <typename T> bool operator != (deque const & lhs, deque const & rhs);
template <typename T> bool operator <  (deque const & lhs, deque const & rhs);
template <typename T> bool operator <= (deque const & lhs, deque const & rhs);
template <typename T> bool operator >  (deque const & lhs, deque const & rhs);
template <typename T> bool operator >= (deque const & lhs, deque const & rhs);

}   // namespace swo3
```