```
// FILE: sequence.template
// TEMPLATE CLASS IMPLEMENTED: sequence (see sequence.h for documentation).
// INVARIANT for the sequence class:
//   1. The number of items in the sequence is in the member variable
//      used;
//   2. The actual items of the sequence are stored in a partially
//      filled array. The array is a compile-time array whose size
//      is fixed at CAPACITY; the  member variable data references
//      the array.
//   3. For an empty sequence, we do not care what is stored in any
//      of data; for a non-empty sequence the items in the sequence
//      are stored in data[0] through data[used-1], and we don't care
//      what's in the rest of data.
//   4. The index of the current item is in the member variable
//      current_index. If there is no valid current item, then
//      current item will be set to the same number as used.
//      NOTE: Setting current_index to be the same as used to
//            indicate "no current item exists" is a good choice
//            for at least the following reasons:
//            (a) For a non-empty sequence, used is non-zero and
//                a current_index equal to used indexes an element
//                that is (just) outside the valid range. This
//                gives us a simple and useful way to indicate
//                whether the sequence has a current item or not:
//                a current_index in the valid range indicates
//                that there's a current item, and a current_index
//                outside the valid range indicates otherwise.
//            (b) The rule remains applicable for an empty sequence,
//                where used is zero: there can't be any current
//                item in an empty sequence, so we set current_index
//                to zero (= used), which is (sort of just) outside
//                the valid range (no index is valid in this case).
//            (c) It simplifies the logic for implementing the
//                advance function: when the precondition is met
//                (sequence has a current item), simply incrementing
//                the current_index takes care of fulfilling the
//                postcondition for the function for both of the two
//                possible scenarios (current item is and is not the
//                last item in the sequence).

#include <cassert>
#include "sequence.h"

namespace CS3358_FA2019_A04_sequence
{
   template <class T>
   sequence<T>::sequence() : used(0), current_index(0)
   {
      //Initialization list used
   }

   template <class T>
   void sequence<T>::start()
   {
      // Setting current_index to first index (0)
      current_index = 0;
   }

   template <class T>
   void sequence<T>::end()
   {
      // If used greater than 0 set current_index
      // to used-1 else set it to 0
      current_index = (used > 0) ? used - 1 : 0;
   }

   template <class T>
   void sequence<T>::advance()
   {
```

```
70          // Checking precondition that current indexes
71          // holds an item, if so move to next index
72          assert( is_item() );
73          ++current_index;
74       }
75
76       template <class T>
77       void sequence<T>::move_back()
78       {
79          // Checking precondition that current indexe
80          // holds an item, if current index is the first
81          // then move to the end, else move back an index
82          assert( is_item() );
83          if (current_index == 0)
84             current_index = used;
85          else
86             --current_index;
87       }
88
89       template <class T>
90       void sequence<T>::add(const T& entry)
91       {
92          // Checking precondition that the sequence is not
93          // at capacity
94          assert( size() < CAPACITY );
95
96          size_type i;
97
98          if ( ! is_item() )
99          {
100            if (used > 0)
101               for (i = used; i >= 1; --i)
102                  data[i] = data[i - 1];
103            data[0] = entry;
104            current_index = 0;
105         }
106         else
107         {
108            ++current_index;
109            for (i = used; i > current_index; --i)
110               data[i] = data[i - 1];
111            data[current_index] = entry;
112         }
113         ++used;
114      }
115
116      template <class T>
117      void sequence<T>::remove_current()
118      {
119         // Checking precondition that current index
120         // holds and item, if so remove
121         assert( is_item() );
122
123         size_type i;
124
125         for (i = current_index + 1; i < used; ++i)
126            data[i - 1] = data[i];
127         --used;
128      }
129
130      template <class T>
131      typename sequence<T>::size_type sequence<T>::size() const
132      {
133         // used holds the size or number of elements used
134         return used;
135      }
136
137      template <class T>
138      bool sequence<T>::is_item() const
```

```
139      {
140         return (current_index < used);
141      }
142
143      template <class T>
144      T sequence<T>::current() const
145      {
146         // Checking the precondition that the
147         // current index holds an item, if so
148         // return that item.
149         assert( is_item() );
150
151         return data[current_index];
152      }
153   }
154
```