```
// FILE: sequence.cpp
// 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_SP2022_A04_sequenceOfNum {
  sequence::sequence() : used(0), current_index(0) { }

  void sequence::start() { current_index = 0; }

  void sequence::end() { current_index = (used > 0) ? used - 1 : 0; }

  void sequence::advance() {
    assert( is_item() );
    current_index++;
  }

  void sequence::move_back() {
    assert( is_item() );
    if (current_index == 0)
      current_index = used;
    else
      --current_index;
  }

  void sequence::add(const value_type& entry)
  {
    assert( size() < CAPACITY );

    size_type i;

    if ( ! is_item() )
    {
      if (used > 0)
        for (i = used; i >= 1; --i)
          data[i] = data[i - 1];
      data[0] = entry;
      current_index = 0;
    }
    else
    {
      ++current_index;
      for (i = used; i > current_index; --i)
        data[i] = data[i - 1];
      data[current_index] = entry;
    }
    ++used;
  }

  void sequence::remove_current()
  {
    assert( is_item() );
```

```
    size_type i;

    for (i = current_index + 1; i < used; ++i)
      data[i - 1] = data[i];
    --used;
  }

  sequence::size_type sequence::size() const { return used; }

  bool sequence::is_item() const { return (current_index < used); }

  sequence::value_type sequence::current() const
  {
    assert( is_item() );

    return data[current_index];
  }
}

namespace CS3358_SP2022_A04_sequenceOfChar
{
  sequence::sequence() : used(0), current_index(0) { }

  void sequence::start() { current_index = 0; }

  void sequence::end() { current_index = (used > 0) ? used - 1 : 0; }

  void sequence::advance()
  {
    assert( is_item() );
    ++current_index;
  }

  void sequence::move_back() {
    assert( is_item() );
    if (current_index == 0)
      current_index = used;
    else
      current_index--;
  }

  void sequence::remove_current() {
    assert( is_item() );
    for (size_type i = current_index + 1; i < used; i++)
      data[i - 1] = data[i];
    used--;
  }

  sequence::size_type sequence::size() const { return used; }

  bool sequence::is_item() const { return (current_index < used); }
}

template class sequence<double>;
template class sequence<char>;
```

```
// FILE: sequence.template
//IMPLEMENTS: The functions of the sequence template class which
//          rely on inputs/outputs of different value types
//NOTE: Since sequence is a template class, this file is included in sequence.h

#include <cstdlib>  // provides size_t
#include <cassert>  // Provides assert

namespace CS3358_SP2022_A04_template {
    template <class T>
    void sequence<T>::add(const T& entry) {
      assert( size() < CAPACITY );
      if ( ! is_item() ) {
        if (used > 0)
          for (size_type i = used; i >= 1; i--)
            data[i] = data[i - 1];
        data[0] = entry;
        current_index = 0;
      }
      else
      {
        current_index++;
        for (size_type i = used; i > current_index; i--)
          data[i] = data[i - 1];
        data[current_index] = entry;
      }
      used++;
   }

    T sequence<T>::current() const {
```

```
    assert( is_item() );
    return data[current_index];
  }
}
```

```
// FILE: sequence.h
/////////////////////////////////////////////////////////////////////
// CLASS PROVIDED: sequence (a container class for a list of items,
//                where each list may have a designated item called
//                the current item)
//
// TYPEDEFS and MEMBER functions for the sequence class:
//   typedef ____ value_type
//     sequence::value_type is the data type of the items in the sequence.
//     It may be any of the C++ built-in types (int, char, etc.), or a
//     class with a default constructor, an assignment operator, and a
//     copy constructor.
//   typedef ____ size_type
//     sequence::size_type is the data type of any variable that keeps
//     track of how many items are in a sequence.
//   static const size_type CAPACITY = _____
//     sequence::CAPACITY is the maximum number of items that a
//     sequence can hold.
//
// CONSTRUCTOR for the sequence class:
//   sequence()
//     Pre:  (none)
//     Post: The sequence has been initialized as an empty sequence.
//
// MODIFICATION MEMBER FUNCTIONS for the sequence class:
//   void start()
//     Pre:  (none)
//     Post: The first item on the sequence becomes the current item
//           (but if the sequence is empty, then there is no current item).
//   void end()
//    Pre:  (none)
//    Post: The last item on the sequence becomes the current item
//         (but if the sequence is empty, then there is no current item).
//  void advance()
//    Pre:  is_item() returns true.
//    Post: If the current item was the last item in the sequence, then
//         there is no longer any current item. Otherwise, the new current
//         item is the item immediately after the original current item.
//  void move_back()
//    Pre:  is_item() returns true.
//    Post: If the current item was the first item in the sequence, then
//         there is no longer any current item. Otherwise, the new current
//         item is the item immediately before the original current item.
//  void add(const value_type& entry)
//    Pre:  size() < CAPACITY.
//    Post: A new copy of entry has been inserted in the sequence after
//         the current item. If there was no current item, then the new
//         entry has been inserted as new first item of the sequence. In
//         either case, the newly added item is now the current item of
//         the sequence.
//  void remove_current()
//    Pre:  is_item() returns true.
//    Post: The current item has been removed from the sequence, and
//         the item after this (if there is one) is now the new current
//         item. If the current item was already the last item in the
//         sequence, then there is no longer any current item.
//
// CONSTANT MEMBER FUNCTIONS for the sequence class:
//  size_type size() const
//    Pre:  (none)
//     Post: The return value is the number of items in the sequence.
//   bool is_item() const
//     Pre:  (none)
//     Post: A true return value indicates that there is a valid
//          "current" item that may be retrieved by activating the current
//          member function (listed below). A false return value indicates
//          that there is no valid current item.
//   value_type current() const
//     Pre:  is_item() returns true.
//     Post: The item returned is the current item in the sequence.
// VALUE SEMANTICS for the sequence class:
//    Assignments and the copy constructor may be used with sequence
//    objects.

#ifndef SEQUENCE_H
#define SEQUENCE_H

#include <cstdlib>  // provides size_t

namespace CS3358_SP2022_A04 {
  template <class Item>
  class sequence {
  public:
    // TYPEDEFS and MEMBER SP2020
    typedef T value_type;
    typedef size_t size_type;
    static const size_type CAPACITY = 10;
    // CONSTRUCTOR
    sequence();
    // MODIFICATION MEMBER FUNCTIONS
```

```
      void start();
      void end();
      void advance();
      void move_back();
      void add(const value_type& entry);
      void remove_current();
      // CONSTANT MEMBER FUNCTIONS
      size_type size() const;
      bool is_item() const;
      value_type current() const;

    private:
      value_type data[CAPACITY];
      size_type used;
      size_type current_index;
    };
    template <class Item>
}

#include "sequence.template"
#endif
```