```
// FILE: sequence.h
////////////////////////////////////////////////////////////////////
// NOTE: Two separate versions of sequence (one for a sequence of real
//       numbers and another for a sequence characters are specified,
//       in two separate namespaces in this header file. For both
//       versions, the same documentation applies.
////////////////////////////////////////////////////////////////////
// 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_FA2021_A04_sequenceOfAll
{
   template<class Item>
   class sequence
   {
   public:
      // TYPEDEFS and MEMBER SP2020
      typedef Item 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 Item& 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;
   };
}
#include "sequence.template"
#endif
```

```
// FILE: sequence.cpp
// CLASS IMPLEMENTED: sequence (see sequence.h for documentation).
// INVARIANT for the sequence class:
// 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_FA2021_A04_sequenceOfAll
{
   template<class Item>
   sequence<Item>::sequence() : used(0), current_index(0) { }

   template<class Item>
   void sequence<Item>::start() { current_index = 0; }

   template<class Item>
   void sequence<Item>::end()
   { current_index = (used > 0) ? used - 1 : 0; }
```

```
template<class Item>
void sequence<Item>::advance()
{
   assert( is_item() );
   ++current_index;
}

template<class Item>
void sequence<Item>::move_back()
{
   assert( is_item() );
   if (current_index == 0)
      current_index = used;
   else
      --current_index;
}

template<class Item>
void sequence<Item>::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;
}

template<class Item>
void sequence<Item>::remove_current()
{
   assert( is_item() );

   size_type i;

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

```
   template<class Item>
   typename sequence<Item>::size_type sequence<Item>::size() const{ return
used; }

   template<class Item>
   bool sequence<Item>::is_item() const { return (current_index < used); }

   template<class Item>
   typename sequence<Item>::value_type sequence<Item>::current() const
   {
      assert( is_item() );

      return data[current_index];
   }
}
```

```
// FILE: sequenceTest.cpp
// An interactive test program for the sequence class

#include <cctype>      // provides toupper
#include <iostream>    // provides cout and cin
#include <cstdlib>     // provides EXIT_SUCCESS
#include "sequence.h"
using namespace CS3358_FA2021_A04_sequenceOfAll;
using namespace std;

typedef sequence<char> seqChar;
typedef sequence<double> seqDouble;
// PROTOTYPES for functions used by this test program:

void print_menu();
// Pre:  (none)
// Post: A menu of choices for this program is written to cout.
char get_user_command();
// Pre:  (none)
// Post: The user is prompted to enter a one character command.
//       The next character is read (skipping blanks and newline
//       characters), and this character is returned.
template<typename T>
void show_list(T);
// Pre: (none)
// Post: The items of src are printed to cout (one per line).
int get_object_num();
// Pre:  (none)
// Post: The user is prompted to enter either 1 or 2. The
//       prompt is repeated until a valid integer can be read
//       and the integer's value is either 1 or 2. The valid
//       integer read is returned. The input buffer is cleared
//       of any extra input until and including the first
//       newline character.
double get_number();
// Pre:  (none)
// Post: The user is prompted to enter a real number. The prompt
//       is repeated until a valid real number can be read. The
//       valid real number read is returned. The input buffer is
//       cleared of any extra input until and including the
//       first newline character.
char get_character();
// Pre:  (none)
// Post: The user is prompted to enter a non-whitespace character.
//       The prompt is repeated until a non-whitespace character
//       can be read. The non-whitespace character read is returned.
//       The input buffer is cleared of any extra input until and
//       including the first newline character.

int main(int argc, char *argv[])
{
   seqDouble s1;  // A sequence of double for testing
   seqChar s2; // A sequence of char for testing
   int objectNum;    // A number to indicate selection of s1 or s2
   double numHold;   // Holder for a real number
```

```
char charHold;    // Holder for a character
char choice;      // A command character entered by the user

cout << "An empty sequence of real numbers (s1) and\n"
     << "an empty sequence of characters (s2) have been created."
     << endl;

do
{
   if (argc == 1)
      print_menu();
   choice = toupper( get_user_command() );
   switch (choice)
   {
      case '!':
         objectNum = get_object_num();
         if (objectNum == 1)
         {
            s1.start();
            cout << "s1 started" << endl;
         }
         else
         {
            s2.start();
            cout << "s2 started" << endl;
         }
         break;
      case '&':
         objectNum = get_object_num();
         if (objectNum == 1)
         {
            s1.end();
            cout << "s1 ended" << endl;
         }
         else
         {
            s2.end();
            cout << "s2 ended" << endl;
         }
         break;
      case '+':
         objectNum = get_object_num();
         if (objectNum == 1)
         {
            if ( ! s1.is_item() )
               cout << "Can't advance s1." << endl;
            else
            {
               s1.advance();
               cout << "Advanced s1 one item."<< endl;
            }
         }
         else
         {
            if ( ! s2.is_item() )
```

```
            cout << "Can't advance s2." << endl;
         else
         {
            s2.advance();
            cout << "Advanced s2 one item."<< endl;
         }
      }
      break;
   case '-':
      objectNum = get_object_num();
      if (objectNum == 1)
      {
         if ( ! s1.is_item() )
            cout << "Can't move back s1." << endl;
         else
         {
            s1.move_back();
            cout << "Moved s1 back one item."<< endl;
         }
      }
      else
      {
         if ( ! s2.is_item() )
            cout << "Can't move back s2." << endl;
         else
         {
            s2.move_back();
            cout << "Moved s2 back one item."<< endl;
         }
      }
      break;
   case '?':
      objectNum = get_object_num();
      if (objectNum == 1)
      {
         if ( s1.is_item() )
            cout << "s1 has a current item." << endl;
         else
            cout << "s1 has no current item." << endl;
      }
      else
      {
         if ( s2.is_item() )
            cout << "s2 has a current item." << endl;
         else
            cout << "s2 has no current item." << endl;
      }
      break;
   case 'C':
      objectNum = get_object_num();
      if (objectNum == 1)
      {
         if ( s1.is_item() )
            cout << "Current item in s1 is: "
                 << s1.current() << endl;
```

```
      else
         cout << "s1 has no current item." << endl;
   }
   else
   {
      if ( s2.is_item() )
         cout << "Current item in s2 is: "
              << s2.current() << endl;
      else
         cout << "s2 has no current item." << endl;
   }
   break;
case 'P':
   objectNum = get_object_num();
   if (objectNum == 1)
   {
      if (s1.size() > 0)
      {
         cout << "s1: ";
         show_list(s1);
         cout << endl;
      }
      else
         cout << "s1 is empty." << endl;
   }
   else
   {
      if (s2.size() > 0)
      {
         cout << "s2: ";
         show_list(s2);
         cout << endl;
      }
      else
         cout << "s2 is empty." << endl;
   }
   break;
case 'S':
   objectNum = get_object_num();
   if (objectNum == 1)
      cout << "Size of s1 is: " << s1.size() << endl;
   else
      cout << "Size of s2 is: " << s2.size() << endl;
   break;
case 'A':
   objectNum = get_object_num();
   if (objectNum == 1)
   {
      numHold = get_number();
      s1.add(numHold);
      cout << numHold << " added to s1." << endl;
   }
   else
   {
      charHold = get_character();
```

```
                s2.add(charHold);
                cout << charHold << " added to s2." << endl;
             }
             break;
          case 'R':
             objectNum = get_object_num();
             if (objectNum == 1)
             {
                if ( s1.is_item() )
                {
                   numHold = s1.current();
                   s1.remove_current();
                   cout << numHold << " removed from s1." << endl;
                }
                else
                   cout << "s1 has no current item." << endl;
             }
             else
             {
                if ( s2.is_item() )
                {
                   charHold = s2.current();
                   s2.remove_current();
                   cout << charHold << " removed from s2." << endl;
                }
                else
                   cout << "s2 has no current item." << endl;
             }
             break;
          case 'Q':
             cout << "Quit option selected...bye" << endl;
             break;
          default:
             cout << choice << " is invalid...try again" << endl;
       }
    }
    while (choice != 'Q');

    cin.ignore(999, '\n');
    cout << "Press Enter or Return when ready...";
    cin.get();
    return EXIT_SUCCESS;
}

void print_menu()
{
    cout << endl;
    cout << "The following choices are available:\n";
    cout << "  !  Activate the start() function\n";
    cout << "  &  Activate the end() function\n";
    cout << "  +  Activate the advance() function\n";
    cout << "  -  Activate the move_back() function\n";
    cout << "  ?  Print the result from the is_item() function\n";
    cout << "  C  Print the result from the current() function\n";
    cout << "  P  Print a copy of the entire sequence\n";
```

```
   cout << "  S  Print the result from the size() function\n";
   cout << "  A  Add a new item with the add(...) function\n";
   cout << "  R  Activate the remove_current() function\n";
   cout << "  Q  Quit this test program" << endl;
}

char get_user_command()
{
   char command;

   cout << "Enter choice: ";
   cin >> command;

   cout << "You entered ";
   cout << command << endl;
   return command;
}

template<typename T>
void show_list(T src)
{
   for ( src.start(); src.is_item(); src.advance() )
      cout << src.current() << "  ";
}

int get_object_num()
{
   int result;

   cout << "Enter object # (1 = s1, 2 = s2) ";
   cin  >> result;
   while ( ! cin.good() )
   {
      cerr << "Invalid integer input..." << endl;
      cin.clear();
      cin.ignore(999, '\n');
      cout << "Re-enter object # (1 = s1, 2 = s2) ";
      cin  >> result;
   }
   // cin.ignore(999, '\n');

   while (result != 1 && result != 2)
   {
      cin.ignore(999, '\n');
      cerr << "Invalid object # (must be 1 or 2)..." << endl;
      cout << "Re-enter object # (1 = s1, 2 = s2) ";
      cin  >> result;
      while ( ! cin.good() )
      {
         cerr << "Invalid integer input..." << endl;
         cin.clear();
         cin.ignore(999, '\n');
         cout << "Re-enter object # (1 = s1, 2 = s2) ";
         cin  >> result;
      }
```

```
      // cin.ignore(999, '\n');
   }

   cout << "You entered ";
   cout << result << endl;
   return result;
}

double get_number()
{
   double result;

   cout << "Enter a real number: ";
   cin  >> result;
   while ( ! cin.good() )
   {
      cerr << "Invalid real number input..." << endl;
      cin.clear();
      cin.ignore(999, '\n');
      cout << "Re-enter a real number ";
      cin  >> result;
   }
   // cin.ignore(999, '\n');

   cout << "You entered ";
   cout << result << endl;
   return result;
}

char get_character()
{
   char result;

   cout << "Enter a non-whitespace character: ";
   cin  >> result;
   while ( ! cin )
   {
      cerr << "Invalid non-whitespace character input..." << endl;
      cin.ignore(999, '\n');
      cout << "Re-enter a non-whitespace character: ";
      cin  >> result;
   }
   // cin.ignore(999, '\n');

   cout << "You entered ";
   cout << result << endl;
   return result;
}
```