```
// FILE: sequence.h
////////////////////////////////////////////////////////////////////
// NOTE: Two specialized versions of sequence (seqDouble for a sequence
//       of real numbers and seqChar for a sequence of characters) are
//       specified in this header file. For both versions, the same
//       documentation applies; simply replace sequence in the
//       documentation with seqDouble or seqChar as appropriate.
////////////////////////////////////////////////////////////////////
// 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 CONSTANTS 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 already 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
```

```
67  //     Pre:  (none)
68  //     Post: The return value is the number of items in the sequence.
69  //   bool is_item() const
70  //     Pre:  (none)
71  //     Post: A true return value indicates that there is a valid
72  //           "current" item that may be retrieved by activating the current
73  //           member function (listed below). A false return value indicates
74  //           that there is no valid current item.
75  //   value_type current() const
76  //     Pre:  is_item() returns true.
77  //     Post: The item returned is the current item in the sequence.
78  // VALUE SEMANTICS for the sequence class:
79  //    Assignments and the copy constructor may be used with sequence
80  //    objects.
81
82  #ifndef SEQUENCE_H
83  #define SEQUENCE_H
84
85  #include <cstdlib>  // provides size_t
86
87  namespace CS3358_FA2021_A04
88  {
89     template <class Item>
90     class sequence
91     {
92     public:
93        typedef size_t size_type;
94        static const size_type CAPACITY = 10;
95        // CONSTRUCTOR
96        sequence();
97        // MODIFICATION MEMBER FUNCTIONS
98        void start();
99        void end();
100       void advance();
101       void move_back();
102       void add(const Item& entry);
103       void remove_current();
104       // CONSTANT MEMBER FUNCTIONS
105       size_type size() const;
106       bool is_item() const;
107       Item current() const;
108
109    private:
110       Item data[CAPACITY];
111       size_type used;
112       size_type current_index;
113    };
114 }
115
116 #include "sequence.template"
117 #endif
118
119
120
121 //=============================================================//
122
123
124
125 // FILE: sequence.cpp
126 // CLASS IMPLEMENTED: sequence (see sequence.h for documentation).
127 // INVARIANT for the sequence class:
128 //   1. The number of items in the sequence is in the member variable
129 //      used;
130 //   2. The actual items of the sequence are stored in a partially
131 //      filled array. The array is a compile-time array whose size
132 //      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
{
   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 Item& entry)
```

```
199     {
200        assert( size() < CAPACITY );
201
202        size_type i;
203
204        if ( ! is_item() )
205        {
206           if (used > 0)
207              for (i = used; i >= 1; --i)
208                 data[i] = data[i - 1];
209           data[0] = entry;
210           current_index = 0;
211        }
212        else
213        {
214           ++current_index;
215           for (i = used; i > current_index; --i)
216              data[i] = data[i - 1];
217           data[current_index] = entry;
218        }
219        ++used;
220     }
221
222     template <class Item>
223     void sequence<Item>::remove_current()
224     {
225        assert( is_item() );
226
227        size_type i;
228
229        for (i = current_index + 1; i < used; ++i)
230           data[i - 1] = data[i];
231        --used;
232     }
233
234     template <class Item>
235     typename sequence<Item>::size_type sequence<Item>::size() const
236        { return used; }
237
238     template <class Item>
239     bool sequence<Item>::is_item() const { return (current_index < used); }
240
241     template <class Item>
242     Item sequence<Item>::current() const
243     {
244        assert( is_item() );
245
246        return data[current_index];
247     }
248
249  }
250
251
252
253  //========================================================================//
254
255
256  // FILE: sequenceTest.cpp
257  // An interactive test program for the sequence class
258
259  #include <cctype>      // provides toupper
260  #include <iostream>    // provides cout and cin
261  #include <cstdlib>     // provides EXIT_SUCCESS
262  #include "sequence.h"
263
264  using namespace std;
```

```
265 using namespace CS3358_FA2021_A04;
266
267 // PROTOTYPES for functions used by this test program:
268
269 void print_menu();
270 // Pre:  (none)
271 // Post: A menu of choices for this program is written to cout.
272 char get_user_command();
273 // Pre:  (none)
274 // Post: The user is prompted to enter a one character command.
275 //       The next character is read (skipping blanks and newline
276 //       characters), and this character is returned.
277 template <class Item>
278 void show_list(Item src);
279 // Pre: (none)
280 // Post: The items of src are printed to cout (one per line).
281 int get_object_num();
282 // Pre:  (none)
283 // Post: The user is prompted to enter either 1 or 2. The
284 //       prompt is repeated until a valid integer can be read
285 //       and the integer's value is either 1 or 2. The valid
286 //       integer read is returned. The input buffer is cleared
287 //       of any extra input until and including the first
288 //       newline character.
289 double get_number();
290 // Pre:  (none)
291 // Post: The user is prompted to enter a real number. The prompt
292 //       is repeated until a valid real number can be read. The
293 //       valid real number read is returned. The input buffer is
294 //       cleared of any extra input until and including the
295 //       first newline character.
296 char get_character();
297 // Pre:  (none)
298 // Post: The user is prompted to enter a non-whitespace character.
299 //       The prompt is repeated until a non-whitespace character
300 //       can be read. The non-whitespace character read is returned.
301 //       The input buffer is cleared of any extra input until and
302 //       including the first newline character.
303
304 int main(int argc, char *argv[])
305 {
306    sequence<double> s1;     // A sequence of double for testing
307    sequence<char> s2;       // A sequence of char for testing
308    int objectNum;    // A number to indicate selection of s1 or s2
309    double numHold;   // Holder for a real number
310    char charHold;    // Holder for a character
311    char choice;      // A command character entered by the user
312
313    cout << "An empty sequence of real numbers (s1) and\n"
314         << "an empty sequence of characters (s2) have been created."
315         << endl;
316
317    do
318    {
319       if (argc == 1)
320          print_menu();
321       choice = toupper( get_user_command() );
322       switch (choice)
323       {
324          case '!':
325             objectNum = get_object_num();
326             if (objectNum == 1)
327             {
328                s1.start();
329                cout << "s1 started" << endl;
330             }
```

```
            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 '?':
```

```
397             objectNum = get_object_num();
398             if (objectNum == 1)
399             {
400                if ( s1.is_item() )
401                   cout << "s1 has a current item." << endl;
402                else
403                   cout << "s1 has no current item." << endl;
404             }
405             else
406             {
407                if ( s2.is_item() )
408                   cout << "s2 has a current item." << endl;
409                else
410                   cout << "s2 has no current item." << endl;
411             }
412             break;
413          case 'C':
414             objectNum = get_object_num();
415             if (objectNum == 1)
416             {
417                if ( s1.is_item() )
418                   cout << "Current item in s1 is: "
419                        << s1.current() << endl;
420                else
421                   cout << "s1 has no current item." << endl;
422             }
423             else
424             {
425                if ( s2.is_item() )
426                   cout << "Current item in s2 is: "
427                        << s2.current() << endl;
428                else
429                   cout << "s2 has no current item." << endl;
430             }
431             break;
432          case 'P':
433             objectNum = get_object_num();
434             if (objectNum == 1)
435             {
436                if (s1.size() > 0)
437                {
438                   cout << "s1: ";
439                   show_list(s1);
440                   cout << endl;
441                }
442                else
443                   cout << "s1 is empty." << endl;
444             }
445             else
446             {
447                if (s2.size() > 0)
448                {
449                   cout << "s2: ";
450                   show_list(s2);
451                   cout << endl;
452                }
453                else
454                   cout << "s2 is empty." << endl;
455             }
456             break;
457          case 'S':
458             objectNum = get_object_num();
459             if (objectNum == 1)
460                cout << "Size of s1 is: " << s1.size() << endl;
461             else
462                cout << "Size of s2 is: " << s2.size() << endl;
```

```
463             break;
464          case 'A':
465             objectNum = get_object_num();
466             if (objectNum == 1)
467             {
468                numHold = get_number();
469                s1.add(numHold);
470                cout << numHold << " added to s1." << endl;
471             }
472             else
473             {
474                charHold = get_character();
475                s2.add(charHold);
476                cout << charHold << " added to s2." << endl;
477             }
478             break;
479          case 'R':
480             objectNum = get_object_num();
481             if (objectNum == 1)
482             {
483                if ( s1.is_item() )
484                {
485                   numHold = s1.current();
486                   s1.remove_current();
487                   cout << numHold << " removed from s1." << endl;
488                }
489                else
490                   cout << "s1 has no current item." << endl;
491             }
492             else
493             {
494                if ( s2.is_item() )
495                {
496                   charHold = s2.current();
497                   s2.remove_current();
498                   cout << charHold << " removed from s2." << endl;
499                }
500                else
501                   cout << "s2 has no current item." << endl;
502             }
503             break;
504          case 'Q':
505             cout << "Quit option selected...bye" << endl;
506             break;
507          default:
508             cout << choice << " is invalid...try again" << endl;
509       }
510    }
511    while (choice != 'Q');
512
513    cin.ignore(999, '\n');
514    cout << "Press Enter or Return when ready...";
515    cin.get();
516    return EXIT_SUCCESS;
517 }
518
519 void print_menu()
520 {
521    cout << endl;
522    cout << "The following choices are available:\n";
523    cout << "  !  Activate the start() function\n";
524    cout << "  &  Activate the end() function\n";
525    cout << "  +  Activate the advance() function\n";
526    cout << "  -  Activate the move_back() function\n";
527    cout << "  ?  Print the result from the is_item() function\n";
528    cout << "  C  Print the result from the current() function\n";
```

```
529     cout << "  P  Print a copy of the entire sequence\n";
530     cout << "  S  Print the result from the size() function\n";
531     cout << "  A  Add a new item with the add(...) function\n";
532     cout << "  R  Activate the remove_current() function\n";
533     cout << "  Q  Quit this test program" << endl;
534  }
535
536  char get_user_command()
537  {
538     char command;
539
540     cout << "Enter choice: ";
541     cin >> command;
542
543     cout << "You entered ";
544     cout << command << endl;
545     return command;
546  }
547
548  template <class Item>
549  void show_list(Item src)
550  {
551     for ( src.start(); src.is_item(); src.advance() )
552        cout << src.current() << "  ";
553  }
554
555  int get_object_num()
556  {
557     int result;
558
559     cout << "Enter object # (1 = s1, 2 = s2) ";
560     cin  >> result;
561     while ( ! cin.good() )
562     {
563        cerr << "Invalid integer input..." << endl;
564        cin.clear();
565        cin.ignore(999, '\n');
566        cout << "Re-enter object # (1 = s1, 2 = s2) ";
567        cin  >> result;
568     }
569     // cin.ignore(999, '\n');
570
571     while (result != 1 && result != 2)
572     {
573        cin.ignore(999, '\n');
574        cerr << "Invalid object # (must be 1 or 2)..." << endl;
575        cout << "Re-enter object # (1 = s1, 2 = s2) ";
576        cin  >> result;
577        while ( ! cin.good() )
578        {
579           cerr << "Invalid integer input..." << endl;
580           cin.clear();
581           cin.ignore(999, '\n');
582           cout << "Re-enter object # (1 = s1, 2 = s2) ";
583           cin  >> result;
584        }
585        // cin.ignore(999, '\n');
586     }
587
588     cout << "You entered ";
589     cout << result << endl;
590     return result;
591  }
592
593  double get_number()
594  {
```

```
595     double result;
596
597     cout << "Enter a real number: ";
598     cin  >> result;
599     while ( ! cin.good() )
600     {
601        cerr << "Invalid real number input..." << endl;
602        cin.clear();
603        cin.ignore(999, '\n');
604        cout << "Re-enter a real number ";
605        cin  >> result;
606     }
607     // cin.ignore(999, '\n');
608
609     cout << "You entered ";
610     cout << result << endl;
611     return result;
612  }
613
614  char get_character()
615  {
616     char result;
617
618     cout << "Enter a non-whitespace character: ";
619     cin  >> result;
620     while ( ! cin )
621     {
622        cerr << "Invalid non-whitespace character input..." << endl;
623        cin.ignore(999, '\n');
624        cout << "Re-enter a non-whitespace character: ";
625        cin  >> result;
626     }
627     // cin.ignore(999, '\n');
628
629     cout << "You entered ";
630     cout << result << endl;
631     return result;
632  }
633
634
635
636
```