```cpp
// FILE: Sequence.cpp
// CLASS IMPLEMENTED: sequence (see sequence.h for documentation)
// INVARIANT for the sequence ADT:
//   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 dynamic array, pointed to by
//      the member variable data. 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.
//   3. The size of the dynamic array is in the member variable
//      capacity.
//   4. The index of the current item is in the member variable
//      current_index. If there is no valid current item, then
//      current_index 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"
#include <iostream>
using namespace std;

namespace CS3358_FA2019
{
   // CONSTRUCTORS and DESTRUCTOR
   sequence::sequence(size_type initial_capacity) : used(0), current_index(0),
   capacity(initial_capacity)
   {
      // Verifying pre-condition: initial_capacity > 0
      if (initial_capacity < 1)
      {
         capacity = 1;
      }

      // Creating new empty dynamic array of size 'capacity'
      data = new value_type[capacity];
   }

   sequence::sequence(const sequence& source) : used(source.used),
   current_index(source.current_index), capacity(source.capacity)
   {
      // Creating new empty dynamic array of size 'capacity'
      data = new value_type[capacity];

      // Copying over all elements from 'source'
      for (size_type i = 0; i < used; i++)
```

```
70          {
71              data[i] = source.data[i];
72          }
73      }
74
75      sequence::~sequence()
76      {
77          // Deallocating dynamic variables
78          delete [] data;
79          data = NULL;
80      }
81
82      // MODIFICATION MEMBER FUNCTIONS
83      void sequence::resize(size_type new_capacity)
84      {
85          // Checking Pre-condition
86          if (used != 0 && new_capacity < used)
87          {
88              capacity = used;
89          }
90          else if (new_capacity < 1)
91          {
92              capacity = 1;
93          }
94          else
95          {
96              capacity = new_capacity;
97          }
98
99          // Creating temp dynamic array with new capacity value
100         value_type * temp_data = new value_type[capacity];
101
102         // Copying contents from 'data' to new resized array
103         for (size_type i = 0; i < used; i++)
104         {
105             temp_data[i] = data[i];
106         }
107
108         // Deallocating old dynamic variable 'data' and assigning 'data' to new
109         // resized dynamic array
110         delete [] data;
111         data = temp_data;
112     }
113
114     void sequence::start()
115     {
116         // Assigning current item to the first item on sequence array
117         current_index = 0;
118     }
119
120     void sequence::advance()
121     {
122         // Validating pre-condition
123         assert(is_item());
124
125         current_index = current_index + 1;
126     }
127
128     void sequence::insert(const value_type& entry)
129     {
130         // If sequence at capacity then resize
131         if (used == capacity)
132         {
133             resize(size_type ((capacity * 1.5) + 1));
134         }
135
136         // Inserting new entry at current_index and shifting elements to the right
137         if (is_item())
138         {
```

```
139             for (size_type i = used; i > current_index; --i)
140             {
141                data[i] = data[i - 1];
142             }
143          }
144          else
145          {
146             current_index = 0;
147             for (size_type i = used; i > current_index; --i)
148             {
149                data[i] = data[i - 1];
150             }
151          }
152
153          data[current_index] = entry;
154          ++used;
155       }
156
157       void sequence::attach(const value_type& entry)
158       {
159          // If sequence is not empty
160          if ( current_index != used)
161          {
162             // If sequence at capacity then resize
163             if (used == capacity)
164             {
165                resize(size_type ((capacity * 1.5) + 1));
166             }
167
168             // Inserting new entry after current_index and shifting elements to the right
169             current_index = current_index + 1;
170             for (size_type i = used; i > current_index; --i)
171             {
172                data[i] = data[i - 1];
173             }
174             data[current_index] = entry;
175          }
176          else
177          {
178             data[current_index] = entry;
179          }
180
181          ++used;
182
183       }
184
185       void sequence::remove_current()
186       {
187          // Validating pre-condition
188          assert(is_item());
189
190          // Removing current item and shifting everything to the left
191          for (size_type i = current_index; i < used - 1; i++)
192          {
193             data[i] = data[i + 1];
194          }
195
196          // Reducing used count by one
197          --used;
198       }
199
200       sequence& sequence::operator=(const sequence& source)
201       {
202          if (!(this == &source))
203          {
204             // Allocating space in temp_data to hold elements in source
205             value_type * temp_data = new value_type[source.capacity];
206
207             // Copying over source array elements into temp_data array
```

```
208             for (size_type i = 0; i < source.used; i++)
209             {
210                temp_data[i] = source.data[i];
211             }
212
213             // Deallocating dynamic array currently pointed at by data
214             delete [] data;
215
216             // Assigning data to temp_data array
217             data = temp_data;
218
219             // Reflecting source properties onto this
220             used = source.used;
221             current_index = source.current_index;
222             capacity = source.capacity;
223          }
224          return *this;
225       }
226
227       // CONSTANT MEMBER FUNCTIONS
228       sequence::size_type sequence::size() const
229       {
230          // Returning the number of distinct elements which is stored in
231          // the varable used
232          return used;
233       }
234
235       bool sequence::is_item() const
236       {
237          // Returning true if current_index != used
238          return (current_index != used);
239       }
240
241       sequence::value_type sequence::current() const
242       {
243          // Validating pre-condition
244          assert(is_item());
245          return data[current_index];
246       }
247    }
248
```