

```
1 // FILE: Sequence.cpp
2 // CLASS IMPLEMENTED: sequence (see sequence.h for documentation)
3 // INVARIANT for the sequence ADT:
4 //   1. The number of items in the sequence is in the member variable
5 //      used;
6 //   2. The actual items of the sequence are stored in a partially
7 //      filled array. The array is a dynamic array, pointed to by
8 //      the member variable data. For an empty sequence, we do not
9 //      care what is stored in any of data; for a non-empty sequence
10 //      the items in the sequence are stored in data[0] through
11 //      data[used-1], and we don't care what's in the rest of data.
12 //   3. The size of the dynamic array is in the member variable
13 //      capacity.
14 //   4. The index of the current item is in the member variable
15 //      current_index. If there is no valid current item, then
16 //      current_index will be set to the same number as used.
17 //   NOTE: Setting current_index to be the same as used to
18 //         indicate "no current item exists" is a good choice
19 //         for at least the following reasons:
20 //         (a) For a non-empty sequence, used is non-zero and
21 //             a current_index equal to used indexes an element
22 //             that is (just) outside the valid range. This
23 //             gives us a simple and useful way to indicate
24 //             whether the sequence has a current item or not:
25 //             a current_index in the valid range indicates
26 //             that there's a current item, and a current_index
27 //             outside the valid range indicates otherwise.
28 //         (b) The rule remains applicable for an empty sequence,
29 //             where used is zero: there can't be any current
30 //             item in an empty sequence, so we set current_index
31 //             to zero (= used), which is (sort of just) outside
32 //             the valid range (no index is valid in this case).
33 //         (c) It simplifies the logic for implementing the
34 //             advance function: when the precondition is met
35 //             (sequence has a current item), simply incrementing
36 //             the current_index takes care of fulfilling the
37 //             postcondition for the function for both of the two
38 //             possible scenarios (current item is and is not the
39 //             last item in the sequence).
40
41 #include <cassert>
42 #include "Sequence.h"
43 #include <iostream>
44 using namespace std;
45
46 namespace CS3358_SP2023
47 {
48     // CONSTRUCTORS and DESTRUCTOR
49     sequence::sequence(size_type initial_capacity)
```

```
50     :used(0), capacity(initial_capacity),current_index(0)
51     {
52         if (initial_capacity <= 0)
53         {
54             capacity = 1;
55         }
56
57         data = new value_type[capacity];
58     }
59
60     sequence::sequence(const sequence& source)
61         :used(source.used), capacity(source.capacity), current_index
           (source.current_index)
62     {
63         data = new value_type[capacity];
64
65         //initialize all elem of data[]
66         for (int i = 0; i < used; i++)
67         {
68             data[i] = source.data[i];
69         }
70     }
71
72     sequence::~~sequence()
73     {
74         delete[] data;
75     }
76
77     // MODIFICATION MEMBER FUNCTIONS
78     void sequence::resize(size_type new_capacity)
79     {
80         //set new_capacity to the minimum if needed
81         if (new_capacity < used)
82         {
83             new_capacity = used;
84         }
85
86         //prevent array of capacity 0
87         if (new_capacity == 0)
88         {
89             new_capacity = DEFAULT_CAPACITY;
90         }
91
92
93         value_type* newData = new value_type[new_capacity];
94
95         for (int i = 0; i < used; i++)
96         {
97             newData[i] = data[i];
```

```
98     }
99
100     //deallocate data and replace it with newData
101     delete[] data;
102     data = newData;
103
104     capacity = new_capacity;
105 }
106
107 void sequence::start()
108 {
109     if (used == 0)
110     {
111         current_index = used;
112     }
113     else
114     {
115         current_index = 0;
116     }
117 }
118
119 void sequence::advance()
120 {
121     if (this->is_item())
122     {
123         current_index++;
124     }
125 }
126
127 void sequence::insert(const value_type& entry)
128 {
129     if (this->is_item())
130     {
131         data[current_index + 1] = data[current_index];
132         data[current_index] = entry;
133         current_index--;
134         used++;
135     }
136     else
137     {
138         //if no current_index entry is set to the front
139         data[0] = entry;
140         current_index = 0;
141         used++;
142     }
143 }
144
145 void sequence::attach(const value_type& entry)
146 {
```

```
147     if (this->is_item())
148     {
149         data[current_index + 1] = entry;
150         current_index++;
151         used++;
152     }
153     else
154     {
155         //if no current_index entry is set to the end
156         data[current_index] = entry;
157         used++;
158     }
159 }
160
161 void sequence::remove_current()
162 {
163     if (this->is_item())
164     {
165         //if current is the last item
166         if (current_index + 1 == used)
167         {
168             used--;
169         }
170         else
171         {
172             for (int i = current_index; i < used - 1; i++)
173             {
174                 data[i] = data[i + 1];
175             }
176             used--;
177         }
178     }
179 }
180
181 sequence& sequence::operator=(const sequence& source)
182 {
183     //self assignment check
184     if (this != &source)
185     {
186         used = source.used;
187         capacity = source.capacity;
188         current_index = source.current_index;
189
190         //Deallocate data array
191         delete[] data;
192
193         //Initialize new data array
194         data = new value_type[capacity];
195     }
```

```
196         //repopulate this->data[] with rhs.data[]
197         for (int i = 0; i < source.used; i++)
198         {
199             data[i] = source.data[i];
200         }
201     }
202
203     return *this;
204 }
205
206 // CONSTANT MEMBER FUNCTIONS
207 sequence::size_type sequence::size() const
208 {
209     return used;
210 }
211
212 bool sequence::is_item() const
213 {
214     if (current_index == used)
215     {
216         return false;
217     }
218
219     return true;
220 }
221
222 sequence::value_type sequence::current() const
223 {
224     if (this->is_item())
225     {
226         return data[current_index];
227     }
228 }
229 }
230
231
```