

```

1  #include <cassert>
2  #include "Sequence.h"
3  #include <iostream>
4  using namespace std;
5
6  namespace CS3358_FA2019
7  {
8      // CONSTRUCTORS and DESTRUCTOR
9      sequence::sequence(size_type initial_capacity) : used(0), current_index(0),
10     capacity(initial_capacity)
11     {
12         // Verifying pre-condition: initial_capacity > 0
13         if (initial_capacity < 1)
14         {
15             capacity = 1;
16         }
17
18         // Creating new empty dynamic array of size 'capacity'
19         data = new value_type[capacity];
20     }
21
22     sequence::sequence(const sequence& source) : used(source.used),
23     current_index(source.current_index), capacity(source.capacity)
24     {
25         // Creating new empty dynamic array of size 'capacity'
26         data = new value_type[capacity];
27
28         // Copying over all elements from 'source'
29         for (size_type i = 0; i < used; i++)
30         {
31             data[i] = source.data[i];
32         }
33     }
34
35     sequence::~~sequence()
36     {
37         // Deallocating dynamic variables
38         delete [] data;
39         data = NULL;
40     }
41
42     // MODIFICATION MEMBER FUNCTIONS
43     void sequence::resize(size_type new_capacity)
44     {
45         // Checking Pre-condition
46         if (used != 0 && new_capacity < used)
47         {
48             capacity = used;
49         }
50         else if (new_capacity < 1)
51         {
52             capacity = 1;
53         }
54         else
55         {
56             capacity = new_capacity;
57         }
58
59         // Creating temp dynamic array with new capacity value
60         value_type * temp_data = new value_type[capacity];
61
62         // Copying contents from 'data' to new resized array
63         for (size_type i = 0; i < used; i++)
64         {
65             temp_data[i] = data[i];
66         }
67
68         // Deallocating old dynamic variable 'data' and assigning 'data' to new
69         // resized dynamic array

```

```

70     delete [] data;
71     data = temp_data;
72 }
73
74 void sequence::start()
75 {
76     // Assigning current item to the first item on sequence array
77     current_index = 0;
78 }
79
80 void sequence::advance()
81 {
82     // Validating pre-condition
83     assert(is_item());
84
85     current_index = current_index + 1;
86 }
87
88 void sequence::insert(const value_type& entry)
89 {
90     // If sequence at capacity then resize
91     if (used == capacity)
92     {
93         resize(size_type ((capacity * 1.5) + 1));
94     }
95
96     // Inserting new entry at current_index and shifting elements to the right
97     if (is_item())
98     {
99         for (size_type i = used; i > current_index; --i)
100         {
101             data[i] = data[i - 1];
102         }
103     }
104     else
105     {
106         current_index = 0;
107         for (size_type i = used; i > current_index; --i)
108         {
109             data[i] = data[i - 1];
110         }
111     }
112
113     data[current_index] = entry;
114     ++used;
115 }
116
117 void sequence::attach(const value_type& entry)
118 {
119     // If sequence is not empty
120     if (current_index != used)
121     {
122         // If sequence at capacity then resize
123         if (used == capacity)
124         {
125             resize(size_type ((capacity * 1.5) + 1));
126         }
127
128         // Inserting new entry after current_index and shifting elements to the right
129         current_index = current_index + 1;
130         for (size_type i = used; i > current_index; --i)
131         {
132             data[i] = data[i - 1];
133         }
134         data[current_index] = entry;
135     }
136     else
137     {
138         data[current_index] = entry;

```

```

139     }
140
141     ++used;
142
143 }
144
145 void sequence::remove_current()
146 {
147     // Validating pre-condition
148     assert(is_item());
149
150     // Removing current item and shifting everything to the left
151     for (size_type i = current_index; i < used - 1; i++)
152     {
153         data[i] = data[i + 1];
154     }
155
156     // Reducing used count by one
157     --used;
158 }
159
160 sequence& sequence::operator=(const sequence& source)
161 {
162     if (!(this == &source))
163     {
164         // Allocating space in temp_data to hold elements in source
165         value_type * temp_data = new value_type[source.capacity];
166
167         // Copying over source array elements into temp_data array
168         for (size_type i = 0; i < source.used; i++)
169         {
170             temp_data[i] = source.data[i];
171         }
172
173         // Deallocating dynamic array currently pointed at by data
174         delete [] data;
175
176         // Assigning data to temp_data array
177         data = temp_data;
178
179         // Reflecting source properties onto this
180         used = source.used;
181         current_index = source.current_index;
182         capacity = source.capacity;
183     }
184     return *this;
185 }
186
187 // CONSTANT MEMBER FUNCTIONS
188 sequence::size_type sequence::size() const
189 {
190     // Returning the number of distinct elements which is stored in
191     // the variable used
192     return used;
193 }
194
195 bool sequence::is_item() const
196 {
197     // Returning true if current_index != used
198     return (current_index != used);
199 }
200
201 sequence::value_type sequence::current() const
202 {
203     // Validating pre-condition
204     assert(is_item());
205     return data[current_index];
206 }
207 }

```