```cpp
#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++)
      {
         data[i] = source.data[i];
      }
   }

   sequence::~sequence()
   {
      // Deallocating dynamic variables
      delete [] data;
      data = NULL;
   }

   // MODIFICATION MEMBER FUNCTIONS
   void sequence::resize(size_type new_capacity)
   {
      // Checking Pre-condition
      if (used != 0 && new_capacity < used)
      {
         capacity = used;
      }
      else if (new_capacity < 1)
      {
         capacity = 1;
      }
      else
      {
         capacity = new_capacity;
      }

      // Creating temp dynamic array with new capacity value
      value_type * temp_data = new value_type[capacity];

      // Copying contents from 'data' to new resized array
      for (size_type i = 0; i < used; i++)
      {
         temp_data[i] = data[i];
      }

      // Deallocating old dynamic variable 'data' and assigning 'data' to new
      // resized dynamic array
```

```cpp
        delete [] data;
        data = temp_data;
    }

    void sequence::start()
    {
        // Assigning current item to the first item on sequence array
        current_index = 0;
    }

    void sequence::advance()
    {
        // Validating pre-condition
        assert(is_item());

        current_index = current_index + 1;
    }

    void sequence::insert(const value_type& entry)
    {
        // If sequence at capacity then resize
        if (used == capacity)
        {
            resize(size_type ((capacity * 1.5) + 1));
        }

        // Inserting new entry at current_index and shifting elements to the right
        if (is_item())
        {
            for (size_type i = used; i > current_index; --i)
            {
                data[i] = data[i - 1];
            }
        }
        else
        {
            current_index = 0;
            for (size_type i = used; i > current_index; --i)
            {
                data[i] = data[i - 1];
            }
        }

        data[current_index] = entry;
        ++used;
    }

    void sequence::attach(const value_type& entry)
    {
        // If sequence is not empty
        if ( current_index != used)
        {
            // If sequence at capacity then resize
            if (used == capacity)
            {
                resize(size_type ((capacity * 1.5) + 1));
            }

            // Inserting new entry after current_index and shifting elements to the right
            current_index = current_index + 1;
            for (size_type i = used; i > current_index; --i)
            {
                data[i] = data[i - 1];
            }
            data[current_index] = entry;
        }
        else
        {
            data[current_index] = entry;
```

```
      }

      ++used;

   }

   void sequence::remove_current()
   {
      // Validating pre-condition
      assert(is_item());

      // Removing current item and shifting everything to the left
      for (size_type i = current_index; i < used - 1; i++)
      {
         data[i] = data[i + 1];
      }

      // Reducing used count by one
      --used;
   }

   sequence& sequence::operator=(const sequence& source)
   {
      if (!(this == &source))
      {
         // Allocating space in temp_data to hold elements in source
         value_type * temp_data = new value_type[source.capacity];

         // Copying over source array elements into temp_data array
         for (size_type i = 0; i < source.used; i++)
         {
            temp_data[i] = source.data[i];
         }

         // Deallocating dynamic array currently pointed at by data
         delete [] data;

         // Assigning data to temp_data array
         data = temp_data;

         // Reflecting source properties onto this
         used = source.used;
         current_index = source.current_index;
         capacity = source.capacity;
      }
      return *this;
   }

   // CONSTANT MEMBER FUNCTIONS
   sequence::size_type sequence::size() const
   {
      // Returning the number of distinct elements which is stored in
      // the varable used
      return used;
   }

   bool sequence::is_item() const
   {
      // Returning true if current_index != used
      return (current_index != used);
   }

   sequence::value_type sequence::current() const
   {
      // Validating pre-condition
      assert(is_item());
      return data[current_index];
   }
}
```