```
#include <cassert>
#include "Sequence.h"
#include <iostream>
using namespace std;

namespace CS3358_SP2022
{
   // CONSTRUCTORS and DESTRUCTOR
   sequence::sequence(size_type initial_capacity) {
      data = new value_type [initial_capacity];
      capacity = initial_capacity;
      used = 0;
      current_index = 0;
   }

   sequence::sequence(const sequence& source) {
      if (this != &source) {
         capacity = source.capacity;
         used = source.used;
         data = new value_type [capacity];
         for (size_type i = 0; i < used; i++) data[i] = source.data[i];
         if (source.is_item()) current_index = source.current_index;
         else current_index = used;
      }
   }

   sequence::~sequence() {
      delete [] data;
   }

   // MODIFICATION MEMBER FUNCTIONS
```

```
  void sequence::resize(size_type new_capacity) {
    if (new_capacity < used) new_capacity = used;
    if (new_capacity < DEFAULT_CAPACITY) new_capacity =
DEFAULT_CAPACITY;
    capacity = new_capacity;
    value_type* temp = new value_type[capacity];
    for (size_type i = 0; i < used; i++) temp[i] = data[i];
    delete [] data;
    data = temp;
  }

  void sequence::start() {
    if (size() > 0) current_index = 0;
  }

  void sequence::advance() {
    assert(is_item());
    current_index += 1;
  }

  void sequence::insert(const value_type& entry) {
    if (size() == capacity) resize(capacity * 1.5);
    if(!is_item()) 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 (size() == capacity) resize(capacity * 1.5);
    if(!is_item()) current_index = used-1;
    current_index++;
    for (size_type i = used; i > current_index; i--) data[i] =
data[i-1];
    data[current_index] = entry;
    used++;
  }

  void sequence::remove_current() {
    assert(is_item());
    for (size_type i = current_index; i < used; i++) data[i] =
data[i+1];
    used--;
  }

  sequence& sequence::operator=(const sequence& source) {
    if (this != &source) {
      value_type* temp = new value_type [source.capacity];
      for (size_type i = 0; i < source.used; i++) temp[i] =
source.data[i];
      delete [] data;
      data = temp;
      used = source.used;
      capacity = source.capacity;
      if (source.is_item()) current_index = source.current_index;
      else current_index = used;
    }
  }

  // CONSTANT MEMBER FUNCTIONS
  sequence::size_type sequence::size() const {
        return used;
    }

    bool sequence::is_item() const {
        return (current_index >= 0 && current_index < used);
    }

    sequence::value_type sequence::current() const {
        assert (is_item());
        return data[current_index];
    }
}
```