// FILE: Assign03Auto.cpp

// A non-interactive test program for the sequence class using a dynamic array.

//

// DESCRIPTION:

// Each function of this program tests part of the sequence class, returning

// some number of points to indicate how much of the test was passed.

// A description and result of each test is printed to cout.

// Maximum number of points awarded by this program is determined by the

// constants POINTS[1], POINTS[2]...

#include <iostream> // provides cout.

#include <cstring> // provides memcpy.

#include <cstdlib> // provides size\_t.

#include "Sequence.h" // provides the sequence class with double items.

using namespace std;

using namespace CS3358\_FA2019;

// Descriptions and points for each of the tests:

const size\_t MANY\_TESTS = 7;

const int POINTS[MANY\_TESTS+1] =

{

21, // Total points for all tests.

4, // Test 1 points

4, // Test 2 points

4, // Test 3 points

2, // Test 4 points

2, // Test 5 points

2, // Test 6 points

3 // Test 7 points

};

const char DESCRIPTION[MANY\_TESTS+1][256] =

{

"tests for sequence class with a dynamic array",

"Testing insert, attach, and the constant member functions",

"Testing situations where the cursor goes off the sequence",

"Testing remove\_current",

"Testing the resize member function",

"Testing the copy constructor",

"Testing the assignment operator",

"Testing insert/attach when current DEFAULT\_CAPACITY exceeded"

};

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// bool test\_basic(const sequence& test, size\_t s, bool has\_cursor)

// Postcondition: A return value of true indicates:

// a. test.size() is s, and

// b. test.is\_item() is has\_cursor.

// Otherwise the return value is false.

// In either case, a description of the test result is printed to cout.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool test\_basic(const sequence& test, size\_t s, bool has\_cursor)

{

bool answer;

cout << "Testing that size() returns " << s << " ... ";

cout.flush();

answer = (test.size() == s);

cout << (answer ? "Passed." : "Failed.") << endl;

if (answer)

{

cout << "Testing that is\_item() returns ";

cout << (has\_cursor ? "true" : "false") << " ... ";

cout.flush();

answer = (test.is\_item() == has\_cursor);

cout << (answer ? "Passed." : "Failed.") << endl;

}

return answer;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// bool test\_items(sequence& test, size\_t s, size\_t i, double items[])

// The function determines if the test sequence has the correct items

// Precondition: The size of the items array is at least s.

// Postcondition: A return value of true indicates that test.current()

// is equal to items[i], and after test.advance() the result of

// test.current() is items[i+1], and so on through items[s-1].

// At this point, one more advance takes the cursor off the sequence.

// If any of this fails, the return value is false.

// NOTE: The test sequence has been changed by advancing its cursor.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool test\_items(sequence& test, size\_t s, size\_t i, double items[])

{

bool answer = true;

cout << "The cursor should be at item [" << i << "]" << " of the sequence\n";

cout << "(counting the first item as [0]). I will advance the cursor\n";

cout << "to the end of the sequence, checking that each item is correct...";

cout.flush();

while ((i < s) && test.is\_item() && (test.current() == items[i]))

{

i++;

test.advance();

}

if ((i != s) && !test.is\_item())

{ // The test.is\_item() function returns false too soon.

cout << "\n Cursor fell off the sequence too soon." << endl;

answer = false;

}

else if (i != s)

{ // The test.current() function returned a wrong value.

cout << "\n The item [" << i << "] should be " << items[i] << ",\n";

cout << " but it was " << test.current() << " instead.\n";

answer = false;

}

else if (test.is\_item())

{ // The test.is\_item() function returns true after moving off the sequence.

cout << "\n The cursor was moved off the sequence,";

cout << " but is\_item still returns true." << endl;

answer = false;

}

cout << (answer ? "Passed." : "Failed.") << endl;

return answer;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// bool correct(sequence& test, size\_t s, size\_t cursor\_spot, double items[])

// This function determines if the sequence (test) is "correct" according to

// these requirements:

// a. it has exactly s items.

// b. the items (starting at the front) are equal to

// items[0] ... items[s-1]

// c. if cursor\_spot < s, then test's cursor must be at

// the location given by cursor\_spot.

// d. if cursor\_spot >= s, then test must not have a cursor.

// NOTE: The function also moves the cursor off the sequence.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool correct(sequence& test, size\_t size, size\_t cursor\_spot, double items[])

{

bool has\_cursor = (cursor\_spot < size);

// Check the sequence's size and whether it has a cursor.

if (!test\_basic(test, size, has\_cursor))

{

cout << "Basic test of size() or is\_item() failed." << endl << endl;

return false;

}

// If there is a cursor, check the items from cursor to end of the sequence.

if (has\_cursor && !test\_items(test, size, cursor\_spot, items))

{

cout << "Test of the sequence's items failed." << endl << endl;

return false;

}

// Restart the cursor at the front of the sequence and test items again.

cout << "I'll call start() and look at the items one more time..." << endl;

test.start();

if (has\_cursor && !test\_items(test, size, 0, items))

{

cout << "Test of the sequence's items failed." << endl << endl;

return false;

}

// If the code reaches here, then all tests have been passed.

cout << "All tests passed for this sequence." << endl << endl;

return true;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// int test1()

// Performs some basic tests of insert, attach, and the constant member

// functions. Returns POINTS[1] if the tests are passed. Otherwise returns 0.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int test1()

{

sequence empty; // An empty sequence

sequence test; // A sequence to add items to

double items1[4] = { 5, 10, 20, 30 }; // These 4 items are put in a sequence

double items2[4] = { 10, 15, 20, 30 }; // These are put in another sequence

// Test that the empty sequence is really empty

cout << "Starting with an empty sequence." << endl;

if (!correct(empty, 0, 0, items1)) return 0;

// Test the attach function to add something to an empty sequence

cout << "I am now using attach to put 10 into an empty sequence." << endl;

test.attach(10);

if (!correct(test, 1, 0, items2)) return 0;

// Test the insert function to add something to an empty sequence

cout << "I am now using insert to put 10 into an empty sequence." << endl;

test = empty;

test.insert(10);

if (!correct(test, 1, 0, items2)) return 0;

// Test the insert function to add an item at the front of a sequence

cout << "I am now using attach to put 10,20,30 in an empty sequence.\n";

cout << "Then I move the cursor to the start and insert 5." << endl;

test = empty;

test.attach(10);

test.attach(20);

test.attach(30);

test.start();

test.insert(5);

if (!correct(test, 4, 0, items1)) return 0;

// Test the insert function to add an item in the middle of a sequence

cout << "I am now using attach to put 10,20,30 in an empty sequence.\n";

cout << "Then I move the cursor to the start, advance once, ";

cout << "and insert 15." << endl;

test = empty;

test.attach(10);

test.attach(20);

test.attach(30);

test.start();

test.advance();

test.insert(15);

if (!correct(test, 4, 1, items2)) return 0;

// Test the attach function to add an item in the middle of a sequence

cout << "I am now using attach to put 10,20,30 in an empty sequence.\n";

cout << "Then I move the cursor to the start and attach 15 ";

cout << "after the 10." << endl;

test = empty;

test.attach(10);

test.attach(20);

test.attach(30);

test.start();

test.attach(15);

if (!correct(test, 4, 1, items2)) return 0;

// All tests have been passed

cout << "All tests of this first function have been passed." << endl;

return POINTS[1];

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// int test2()

// Performs a test to ensure that the cursor can correctly be run off the end

// of the sequence. Also tests that attach/insert work correctly when there is

// no cursor. Returns POINTS[2] if the tests are passed. Otherwise returns 0.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int test2()

{

sequence test;

size\_t i;

// Put three items in the sequence

cout << "Using attach to put 20 and 30 in the sequence, and then calling\n";

cout << "advance, so that is\_item should return false ... ";

cout.flush();

test.attach(20);

test.attach(30);

test.advance();

if (test.is\_item())

{

cout << "failed." << endl;

return 0;

}

cout << "passed." << endl;

// Insert 10 at the front and run the cursor off the end again

cout << "Inserting 10, which should go at the sequence's front." << endl;

cout << "Then calling advance three times to run cursor off the sequence ...";

cout.flush();

test.insert(10);

test.advance(); // advance to the 20

test.advance(); // advance to the 30

test.advance(); // advance right off the sequence

if (test.is\_item())

{

cout << " failed." << endl;

return false;

}

cout << " passed." << endl;

// Attach more items until the sequence becomes full.

// Note that the first attach should attach to the end of the sequence.

cout << "Calling attach to put the numbers 40, 50, 60 ...";

cout << test.DEFAULT\_CAPACITY\*10 << " at the sequence's end." << endl;

for (i = 4; i <= test.DEFAULT\_CAPACITY; i++)

test.attach(i\*10);

// Test that the sequence is correctly filled.

cout << "Now I will test that the sequence has 10, 20, 30, ...";

cout << test.DEFAULT\_CAPACITY\*10 << "." << endl;

test.start();

for (i = 1; i <= test.DEFAULT\_CAPACITY; i++)

{

if ((!test.is\_item()) || test.current() != i\*10)

{

cout << " Test failed to find " << i\*10 << endl;

return 0;

}

test.advance();

}

if (test.is\_item())

{

cout << " There are too many items on the sequence." << endl;

return false;

}

// All tests passed

cout << "All tests of this second function have been passed." << endl;

return POINTS[2];

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// int test3()

// Performs basic tests for the remove\_current function.

// Returns POINTS[3] if the tests are passed. Returns POINTS[3] / 4 if almost

// all the tests are passed. Otherwise returns 0.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int test3()

{

// In the next declarations, I am declaring a sequence called test.

// Both before and after the sequence, I declare a small array of characters,

// and I put the character 'x' into each spot of these arrays.

// Later, if I notice that one of the x's has been changed, or if

// I notice an 'x' inside of the sequence, then the most

// likely reason was that one of the sequence's member functions accessed

// the sequence's array outside of its legal indexes.

char prefix[4] = {'x', 'x', 'x', 'x'};

sequence test;

char suffix[4] = {'x', 'x', 'x', 'x'};

// Within this function, I create several different sequences using the

// items in these arrays:

double items1[1] = { 30 };

double items2[2] = { 10, 30 };

double items3[3] = { 10, 20, 30 };

size\_t i; // for-loop control variable

char \*char\_ptr; // Variable to loop at each character in a sequence's memory

// Build a sequence with three items 10, 20, 30, and remove the middle,

// and last and then first.

cout << "Using attach to build a sequence with 10,30." << endl;

test.attach(10);

test.attach(30);

cout << "Insert a 20 before the 30, so entire sequence is 10,20,30." << endl;

test.insert(20);

if (!correct(test, 3, 1, items3)) return 0;

cout << "Remove the 20, so entire sequence is now 10,30." << endl;

test.start();

test.advance();

test.remove\_current();

if (!correct(test, 2, 1, items2)) return 0;

cout << "Remove the 30, so entire sequence is now just 10 with no cursor.";

cout << endl;

test.start();

test.advance();

test.remove\_current();

if (!correct(test, 1, 1, items2)) return 0;

cout << "Set the cursor to the start and remove the 10." << endl;

test.start();

test.remove\_current();

if (!correct(test, 0, 0, items2)) return 0;

// Build a sequence with three items 10, 20, 30, and remove the middle,

// and then first and then last.

cout << "Using attach to build another sequence with 10,30." << endl;

test.attach(10);

test.attach(30);

cout << "Insert a 20 before the 30, so entire sequence is 10,20,30." << endl;

test.insert(20);

if (!correct(test, 3, 1, items3)) return 0;

cout << "Remove the 20, so entire sequence is now 10,30." << endl;

test.start();

test.advance();

test.remove\_current();

if (!correct(test, 2, 1, items2)) return 0;

cout << "Set the cursor to the start and remove the 10," << endl;

cout << "so the sequence should now contain just 30." << endl;

test.start();

test.remove\_current();

if (!correct(test, 1, 0, items1)) return 0;

cout << "Remove the 30 from the sequence, resulting in an empty sequence." << endl;

test.start();

test.remove\_current();

if (!correct(test, 0, 0, items1)) return 0;

// Build a sequence with three items 10, 20, 30, and remove the first.

cout << "Build a new sequence by inserting 30, 10, 20 (so the sequence\n";

cout << "is 20, then 10, then 30). Then remove the 20." << endl;

test.insert(30);

test.insert(10);

test.insert(20);

test.remove\_current();

if (!correct(test, 2, 0, items2)) return 0;

test.start();

test.remove\_current();

test.remove\_current();

// Just for fun, fill up the sequence, and empty it!

cout << "Just for fun, I'll empty the sequence then fill it up, then\n";

cout << "empty it again. During this process, I'll try to determine\n";

cout << "whether any of the sequence's member functions access the\n";

cout << "array outside of its legal indexes." << endl;

for (i = 0; i < test.DEFAULT\_CAPACITY; i++)

test.insert(0);

for (i = 0; i < test.DEFAULT\_CAPACITY; i++)

test.remove\_current();

// Make sure that the character 'x' didn't somehow get into the sequence,

// as that would indicate that the sequence member functions are

// copying data from before or after the sequence into the sequence.

char\_ptr = (char \*) &test;

for (i = 0; i < sizeof(sequence); i++)

if (char\_ptr[i] == 'x')

{

cout << "Illegal array access detected." << endl;

return POINTS[3] / 4;

}

// Make sure that the prefix and suffix arrays still have four

// x's each. Otherwise one of the sequence operations wrote outside of

// the legal boundaries of its array.

for (i = 0; i < 4; i++)

if ((suffix[i] != 'x') || (prefix[i] != 'x'))

{

cout << "Illegal array access detected." << endl;

return POINTS[3] / 4;

}

// All tests passed

cout << "All tests of this third function have been passed." << endl;

return POINTS[3];

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// int test4()

// Performs some tests of resize.

// Returns POINTS[4] if the tests are passed. Otherwise returns 0.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int test4()

{

sequence test;

size\_t i;

char bytes[sizeof(sequence)];

char newbytes[sizeof(sequence)];

size\_t mismatches;

cout << "I will now resize a sequence to a larger capacity, and then\n";

cout << "attach that many items. The sequence should NOT need to\n";

cout << "resize itself under this situation." << endl;

test.resize(2\*test.DEFAULT\_CAPACITY);

test.attach(0);

memcpy(bytes, (char \*) &test, sizeof(sequence));

// At this point, I should be able to insert 2\*DEFAULT\_CAPACITY-1

// more items without calling resize again. Therefore, at most 1 byte

// of the object will change (the least significant byte of used).

for (i = 1; i < 2\*test.DEFAULT\_CAPACITY; i++)

test.attach(i);

test.start();

memcpy(newbytes, (char \*) &test, sizeof(sequence));

for (i = 0; i < 2\*test.DEFAULT\_CAPACITY; i++)

{

if (test.current() != i)

{

cout << " sequence does not contain correct items." << endl;

return 0;

}

test.advance();

}

test.start();

mismatches = 0;

for (i = 0; i < sizeof(sequence); i++)

if (bytes[i] != newbytes[i])

mismatches++;

if (mismatches > 1)

{

cout << " sequence was resized when it should not be." << endl;

return 0;

}

else

cout << " Test passed." << endl;

cout << "Now I will call resize(1) for the sequence, but the actual\n";

cout << "sequence should not change because the sequence already has \n";

cout << test.DEFAULT\_CAPACITY\*2 << " items." << endl;

memcpy(bytes, (char \*) &test, sizeof(sequence));

test.resize(1);

mismatches = 0;

for (i = 0; i < sizeof(sequence); i++)

if (bytes[i] != newbytes[i])

mismatches++;

if (mismatches > 0)

{

cout << " sequence was resized when it should not be." << endl;

return 0;

}

else

cout << " Test passed." << endl;

// All tests passed

cout << "All tests of this fourth function have been passed." << endl;

return POINTS[4];

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// int test5()

// Performs some tests of the copy constructor.

// Returns POINTS[5] if the tests are passed. Otherwise returns 0.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int test5()

{

sequence original; // A sequence that we'll copy.

double items[2\*original.DEFAULT\_CAPACITY];

size\_t i;

// Set up the items array to conatin 1...2\*DEFAULT\_CAPACITY.

for (i = 1; i <= 2\*original.DEFAULT\_CAPACITY; i++)

items[i-1] = i;

// Test copying of an empty sequence. After the copying, we change the original.

cout << "Copy constructor test: for an empty sequence." << endl;

sequence copy1(original);

original.attach(1); // Changes the original sequence, but not the copy.

if (!correct(copy1, 0, 0, items)) return 0;

// Test copying of a sequence with current item at the tail.

cout << "Copy constructor test: for a sequence with cursor at tail." << endl;

for (i=2; i <= 2\*original.DEFAULT\_CAPACITY; i++)

original.attach(i);

sequence copy2(original);

original.start();

original.advance();

original.remove\_current(); // Removes 2 from the original, but not the copy.

if (!correct

(copy2, 2\*original.DEFAULT\_CAPACITY, 2\*original.DEFAULT\_CAPACITY-1, items)

)

return 0;

// Test copying of a sequence with cursor near the middle.

cout << "Copy constructor test: for a sequence with cursor near middle." << endl;

original.insert(2);

for (i = 1; i < original.DEFAULT\_CAPACITY; i++)

original.advance();

// Cursor is now at location [DEFAULT\_CAPACITY] (counting [0] as the first spot).

sequence copy3(original);

original.start();

original.advance();

original.remove\_current(); // Removes 2 from the original, but not the copy.

if (!correct

(copy3, 2\*original.DEFAULT\_CAPACITY, original.DEFAULT\_CAPACITY, items)

)

return 0;

// Test copying of a sequence with cursor at the front.

cout << "Copy constructor test: for a sequence with cursor near middle." << endl;

original.insert(2);

original.start();

// Cursor is now at the front.

sequence copy4(original);

original.start();

original.advance();

original.remove\_current(); // Removes 2 from the original, but not the copy.

if (!correct

(copy4, 2\*original.DEFAULT\_CAPACITY, 0, items)

)

return 0;

// Test copying of a sequence with no current item.

cout << "Copy constructor test: for a sequence with no current item." << endl;

original.insert(2);

while (original.is\_item())

original.advance();

// There is now no current item.

sequence copy5(original);

original.start();

original.advance();

original.remove\_current(); // Removes 2 from the original, but not the copy.

if (!correct

(copy5, 2\*original.DEFAULT\_CAPACITY, 2\*original.DEFAULT\_CAPACITY, items)

)

return 0;

// All tests passed

cout << "All tests of this fifth function have been passed." << endl;

return POINTS[5];

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// int test6()

// Performs some tests of the assignment operator.

// Returns POINTS[6] if the tests are passed. Otherwise returns 0.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int test6()

{

sequence original; // A sequence that we'll copy.

double items[2\*original.DEFAULT\_CAPACITY];

size\_t i;

// Set up the items array to conatin 1...2\*DEFAULT\_CAPACITY.

for (i = 1; i <= 2\*original.DEFAULT\_CAPACITY; i++)

items[i-1] = i;

// Test copying of an empty sequence. After the copying, we change the original.

cout << "Assignment operator test: for an empty sequence." << endl;

sequence copy1;

copy1 = original;

original.attach(1); // Changes the original sequence, but not the copy.

if (!correct(copy1, 0, 0, items)) return 0;

// Test copying of a sequence with current item at the tail.

cout << "Assignment operator test: for a sequence with cursor at tail." << endl;

for (i=2; i <= 2\*original.DEFAULT\_CAPACITY; i++)

original.attach(i);

sequence copy2;

copy2 = original;

original.start();

original.advance();

original.remove\_current(); // Removes 2 from the original, but not the copy.

if (!correct

(copy2, 2\*original.DEFAULT\_CAPACITY, 2\*original.DEFAULT\_CAPACITY-1, items)

)

return 0;

// Test copying of a sequence with cursor near the middle.

cout << "Assignment operator test: for a sequence with cursor near middle." << endl;

original.insert(2);

for (i = 1; i < original.DEFAULT\_CAPACITY; i++)

original.advance();

// Cursor is now at location [DEFAULT\_CAPACITY] (counting [0] as the first spot).

sequence copy3;

copy3 = original;

original.start();

original.advance();

original.remove\_current(); // Removes 2 from the original, but not the copy.

if (!correct

(copy3, 2\*original.DEFAULT\_CAPACITY, original.DEFAULT\_CAPACITY, items)

)

return 0;

// Test copying of a sequence with cursor at the front.

cout << "Assignment operator test: for a sequence with cursor near middle." << endl;

original.insert(2);

original.start();

// Cursor is now at the front.

sequence copy4;

copy4 = original;

original.start();

original.advance();

original.remove\_current(); // Removes 2 from the original, but not the copy.

if (!correct

(copy4, 2\*original.DEFAULT\_CAPACITY, 0, items)

)

return 0;

// Test copying of a sequence with no current item.

cout << "Assignment operator test: for a sequence with no current item." << endl;

original.insert(2);

while (original.is\_item())

original.advance();

// There is now no current item.

sequence copy5;

copy5 = original;

original.start();

original.advance();

original.remove\_current(); // Removes 2 from the original, but not the copy.

if (!correct

(copy5, 2\*original.DEFAULT\_CAPACITY, 2\*original.DEFAULT\_CAPACITY, items)

)

return 0;

cout << "Checking correctness of a self-assignment x = x;" << endl;

original.insert(2);

original = original;

if (!correct

(original, 2\*original.DEFAULT\_CAPACITY, 1, items)

)

return 0;

// All tests passed

cout << "All tests of this sixth function have been passed." << endl;

return POINTS[6];

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// int test7()

// Performs some basic tests of insert and attach for the case where the

// current capacity has been reached.

// Returns POINTS[7] if the tests are passed. Otherwise returns 0.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int test7()

{

sequence testa, testi;

double items[2\*testa.DEFAULT\_CAPACITY];

size\_t i;

// Set up the items array to conatin 1...2\*DEFAULT\_CAPACITY.

for (i = 1; i <= 2\*testa.DEFAULT\_CAPACITY; i++)

items[i-1] = i;

cout << "Testing to see that attach works correctly when the\n";

cout << "current capacity is exceeded." << endl;

for (i = 1; i <= 2\*testa.DEFAULT\_CAPACITY; i++)

testa.attach(i);

if (!correct

(testa, 2\*testa.DEFAULT\_CAPACITY, 2\*testa.DEFAULT\_CAPACITY-1, items)

)

return 0;

cout << "Testing to see that insert works correctly when the\n";

cout << "current capacity is exceeded." << endl;

for (i = 2\*testi.DEFAULT\_CAPACITY; i >= 1; i--)

testi.insert(i);

if (!correct

(testi, 2\*testi.DEFAULT\_CAPACITY, 0, items)

)

return 0;

// All tests passed

cout << "All tests of this seventh function have been passed." << endl;

return POINTS[7];

}

int run\_a\_test(int number, const char message[], int test\_function(), int max)

{

int result;

cout << endl << "START OF TEST " << number << ":" << endl;

cout << message << " (" << max << " points)." << endl;

result = test\_function();

if (result > 0)

{

cout << "Test " << number << " got " << result << " points";

cout << " out of a possible " << max << "." << endl;

}

else

cout << "Test " << number << " failed." << endl;

cout << "END OF TEST " << number << "." << endl << endl;

return result;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// int main()

// The main program calls all tests and prints the sum of all points

// earned from the tests.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main()

{

int sum = 0;

cout << "Running " << DESCRIPTION[0] << endl;

sum += run\_a\_test(1, DESCRIPTION[1], test1, POINTS[1]);

sum += run\_a\_test(2, DESCRIPTION[2], test2, POINTS[2]);

sum += run\_a\_test(3, DESCRIPTION[3], test3, POINTS[3]);

sum += run\_a\_test(4, DESCRIPTION[4], test4, POINTS[4]);

sum += run\_a\_test(5, DESCRIPTION[5], test5, POINTS[5]);

sum += run\_a\_test(6, DESCRIPTION[6], test6, POINTS[6]);

sum += run\_a\_test(7, DESCRIPTION[7], test7, POINTS[7]);

cout << "Your sequence implementation has scored\n";

cout << sum << " points out of the " << POINTS[0];

cout << " points based on this test program.\n";

return EXIT\_SUCCESS;

}