San José State University
Department of Computer Engineering

CMPE 180-92

Data Structures and Algorithms in C++

Spring 2018

Instructor: Ron Mak

Assignment #9

Assigned:	Monday, March 26
Due:	Thursday, April 5 at 5:30 PM
CodeCheck:	http://codecheck.it/files/180326075292xpg4lwjt8ir9ctwna56cp98
Canvas:	Assignment #9. STL Vector and List
Points:	200

STL Vector and List

This assignment will give you practice with the vector and the linked list containers from the Standard Template Library (STL), subclasses, iterators, and exceptions. By running similar tests on a sorted subclass of each container, you will compare their performance with respect to execution time and the number of calls to the constructor, copy constructor, assignment operator, and destructor functions.

You will see whether a sorted vector or a sorted linked list container performs better for each test, and you will discover how much overhead is caused by calls to the constructor and destructor functions.

Test suite

Your program will run a suite of tests for the following operations on two types of containers, a sorted vector and a sorted list. The former is a subclass of the STL vector template class, and the latter is a subclass of the STL (doubly-linked) list template class. Run each test several times with an increasing number of data nodes: 100, 500, 1000; 5000, and 10,000 nodes.

Prepend: Insert nodes one at a time at the beginning of the container.

Append: Add nodes one at a time to the end of the container.

Get: Access nodes at random positions in the container.

Insert: Insert nodes at random positions one at a time into the container while maintaining sort order.

Remove: Remove nodes at random positions one at a time from the container.

Reverse: Reverse the order of the sorted nodes of the container.

Online C++ references

Plan to consult online C++ references. Links you may find especially useful:

- http://www.cplusplus.com/reference/vector/vector/
- http://www.cplusplus.com/reference/list/list/
- http://www.cplusplus.com/reference/iterator/

In particular, note that member function **erase**, which removes an element from a container, takes as a parameter an iterator that points to the element to remove.

Program structure

The nodes

Class **Node** represents the data nodes for the containers, each with a **long value** data member. During each test, count how many times each constructor, copy constructor, overloaded assignment operator, and destructor function is called for all nodes. Therefore, class **Node** has these private static data members:

```
static long constructor_count;
static long copy_count;
static long assign_count;
static long destructor_count;
```

and these public static member functions:

```
static long get_constructor_count();
static long get_copy_count();
static long get_assign_count();
static long get_destructor_count();
```

Static data members and functions belong to their class, not to individual objects. A static data member acts like a global variable. For example, use static data member **constructor_count** to count how many times the **Node** constructor is called for all **Node** objects. To call a public static member function from a non-member function, you must use the scope resolution operator, such as **Node::get_constructor_count()** and **Node::reset()**. The latter function resets all three counters to 0.

The sorted container classes

Container class **SortedVector** is a subclass of the STL template class **vector<Node>** and container class **SortedList** is a subclass of the STL template class **list<Node>**. Each subclass adds the constraint that the nodes must be sorted by the nodes' **value** fields. Each container class implements the public member functions **prepend()**, **append()**, **insert()**, **remove()**, and **reverse()** to perform the operations described above. Each also implements helper member functions **check()** and **check_reversed()** verify that the container's nodes are sorted and reverse sorted, respectively.

If you override a parent class's member function, you can still call it by using the scope resolution operator. For example, inside the subclass member function **`SortedVector::insert()`**, you can call **`vector<Node>::insert()`**.

Class **`SortedList`** overloads the subscript **`[]`** operator to enable accessing a node in the linked list using a subscript. But unlike a vector node, you cannot directly access a list node. You must "chase links" from either the head end or the tail end of the list to arrive at the desired node. Take advantage of reverse iterators. If the node you want to access is closer to the head of the list, use a regular (forward) iterator to reach it. However, if the node you want to access is closer to the tail of the list, use a reverse iterator to reach it. Unfortunately, STL member functions like **`erase`** only work with a regular iterator. To convert a reverse iterator that points to a node to a regular iterator that points to the same node, see http://stackoverflow.com/questions/4407985/why-can-i-not-convert-a-reverse-iterator-to-a-forward-iterator. (You can convert.) ***Tip:*** Implement a helper function that returns a regular iterator that points to the desired indexed node.

Reversing the order of a container

For this assignment, do not reverse the order of the nodes of a sorted container by simply copying the contents (i.e., the value) of the nodes in place. Instead, we want to exercise each container by removing and inserting nodes using iterators.

For example, suppose a sorted container contains the nodes **A B C**. Follow these steps:

- Set a regular (forward) iterator to point to the second node, **B**.
- Insert a copy of this node at the beginning of the container: **B A B C**
- Remove the node pointed to by the iterator: **B A C**
- Advance the iterator to point to the next node, **C**.
- Insert a copy of this node at the beginning of the container: **C B A C**
- Remove the node pointed to by the iterator: **C B A**
- Advance the iterator. It's off the end of the container, so you're done reversing.

The tests

`TestSuite.h` and **`TestSuite.cpp`** implement the operation tests for both container classes **`SortedVector`** and **`SortedList`**. Functions **`vector_gets()`** and **`list_gets()`** each accesses **`GET_COUNT`** nodes at random index positions. Each function throws an exception if a wrong node was accessed. Functions **`vector_inserts()`** and **`list_inserts()`** each inserts nodes with random values up to the specified size. The containers must remain sorted, and each function throws an exception if a container becomes unsorted. Functions **`vector_reverse()`** and **`list_reverse()`** reverse the sort order of containers. Each throws an exception if a container is not properly reverse sorted. Functions **`vector_remove()`** and **`list_remove()`** each removes nodes at random index positions of a container until the container is empty.

`TestDriver.cpp` contains **`main()`**, which calls function **`run_test_suite()`**. This function in turn calls **`run_test_functions()`** for each of the tests described above, passing the name of the test and the two test functions, one for the vector and one for the list. Function **`run_test_functions()`** calls function **`timed_test()`** to run the vector test and the list test for different data sizes. As shown in the sample output below for each test, function **run_test_functions()** records and prints the elapsed time and the counts of calls to the **Node** constructor, copy constructor, and destructor functions.

There are two versions of function **timed_test()**, one for a vector and one for a list. Each function receives a test function **f()** as a parameter. Function **timed_test()** runs the test function **f()** under a timer, and then it returns the elapsed time in milliseconds.

Note 1: The **chrono** functions require you to compile with **-std=c++0x**.

Note 2: For this assignment, do not reserve space for the vector. We want to see how many constructor and destructor calls result from C++ expanding a vector's size.

Submission into Canvas

Because of random numbers, the different timings, and possibly different counts, CodeCheck will not compare your output.

When you're satisfied with your program in CodeCheck, click the "Download" link at the very bottom of the Report screen to download a signed zip file of your solution. Submit this signed zip file into Canvas. You can submit as many times as you want until the deadline, and the number of submissions will not affect your score. Only your last submission will be graded.

Submit into Canvas: **Assignment #9. STL Vector and List.**

Note: You must submit the signed zip file that you download from CodeCheck, or your submission will not be graded. Do not rename the zip file.

Sample output

In the sample output below, **Size** is the number of **Node** objects, **Time** is the elapsed time in milliseconds required to execute the test for that size, **Creates** is the number of calls to the **Node** constructor, **Copies** is the number of calls to the **Node** copy constructor, **Assigns** is the number of calls to the overloaded **Node** assignment operator, and **Destroys** is the number of calls to the **Node** destructor.

Be sure that you understand and can explain all the vector and list counts! If you reserved space for the vector, what affect would that have on its counts? For each size of the test, how much space should you reserve?

```
=======
Prepend
=======
        |--------------------Vector-------------------|  |---------------------List--------------------|
  Size      Time  Creates   Copies     Assigns Destroys      Time  Creates   Copies     Assigns Destroys
   100      0 ms      100      227       4,823      227      0 ms      100      100           0      100
   500      0 ms      500    1,011     124,239    1,011      0 ms      500      500           0      500
 1,000      3 ms    1,000    2,023     498,477    2,023      0 ms    1,000    1,000           0    1,000
 5,000     74 ms    5,000   13,191  12,489,309   13,191      0 ms    5,000    5,000           0    5,000
10,000    298 ms   10,000   26,383  49,978,617   26,383      1 ms   10,000   10,000           0   10,000

======
Append
======
        |--------------------Vector-------------------|  |---------------------List--------------------|
  Size      Time  Creates   Copies     Assigns Destroys      Time  Creates   Copies     Assigns Destroys
   100      0 ms      100      227           0      227      0 ms      100      100           0      100
   500      0 ms      500    1,011           0    1,011      0 ms      500      500           0      500
 1,000      0 ms    1,000    2,023           0    2,023      0 ms    1,000    1,000           0    1,000
 5,000      0 ms    5,000   13,191           0   13,191      0 ms    5,000    5,000           0    5,000
10,000      1 ms   10,000   26,383           0   26,383      1 ms   10,000   10,000           0   10,000

===
Get
===
        |--------------------Vector-------------------|  |---------------------List--------------------|
  Size      Time  Creates   Copies     Assigns Destroys      Time  Creates   Copies     Assigns Destroys
   100      0 ms        0        0           0        0      1 ms        0        0           0        0
   500      0 ms        0        0           0        0      5 ms        0        0           0        0
 1,000      0 ms        0        0           0        0      9 ms        0        0           0        0
 5,000      0 ms        0        0           0        0     54 ms        0        0           0        0
10,000      1 ms        0        0           0        0     96 ms        0        0           0        0

======
Remove
======
        |--------------------Vector-------------------|  |---------------------List--------------------|
  Size      Time  Creates   Copies     Assigns Destroys      Time  Creates   Copies     Assigns Destroys
   100      0 ms        0        0       2,478      100      0 ms        0        0           0      100
   500      0 ms        0        0      60,156      500      0 ms        0        0           0      500
 1,000      1 ms        0        0     256,191    1,000      0 ms        0        0           0    1,000
 5,000     39 ms        0        0   6,207,613    5,000     16 ms        0        0           0    5,000
10,000    162 ms        0        0  24,782,497   10,000     64 ms        0        0           0   10,000

======
Insert
======
        |--------------------Vector-------------------|  |---------------------List--------------------|
  Size      Time  Creates   Copies     Assigns Destroys      Time  Creates   Copies     Assigns Destroys
   100      0 ms      100      227       2,455      227      0 ms      100      100           0      100
   500      1 ms      500    1,011      62,090    1,011      1 ms      500      500           0      500
 1,000      5 ms    1,000    2,023     255,091    2,023      6 ms    1,000    1,000           0    1,000
 5,000    146 ms    5,000   13,191   6,365,209   13,191    181 ms    5,000    5,000           0    5,000
10,000    600 ms   10,000   26,383  25,309,881   26,383    719 ms   10,000   10,000           0   10,000

=======
Reverse
=======
        |--------------------Vector-------------------|  |---------------------List--------------------|
  Size      Time  Creates   Copies     Assigns Destroys      Time  Creates   Copies     Assigns Destroys
   100      0 ms        0       99      14,751       99      0 ms        0       99           0       99
   500      2 ms        0      499     373,751      499      0 ms        0      499           0      499
 1,000      9 ms        0      999   1,497,501      999      0 ms        0      999           0      999
 5,000    236 ms        0    4,999  37,487,501    4,999      1 ms        0    4,999           0    4,999
10,000    908 ms        0    9,999 149,975,001    9,999      3 ms        0    9,999           0    9,999

Done! Total time: 3.67917 seconds
```

Rubric

Your program will be graded according to these criteria:

Criteria	Max points
Good output	**20**
• Timings	• 10
• Counts	• 10
Good program design	**180**
• Class `Node` with call counting.	• 10
• `SortedVector::prepend()`	• 10
• `SortedVector::append()`	• 10
• `SortedVector::remove()`	• 10
• `SortedVector::insert()`	• 10
• `SortedVector::reverse()` using iterator	• 10
• `SortedList::prepend()`	• 10
• `SortedList::append()`	• 10
• `SortedList::remove()`	• 10
• `SortedList::insert()`	• 10
• `SortedList::reverse()` using iterator	• 10
• `SortedList::operator[]()`	• 10
• Test suite	
○ `vector_prepends()` and `list_prepends()`	• 10
○ `vector_appends()` and `list_appends()`	• 10
○ `vector_gets()` and `list_gets()`	• 10
○ `vector_removes()` and `list_removes()`	• 10
○ `vector_inserts()` and `list_inserts()`	• 10
○ `vector_reverse()` and `list_reverse()`	• 10

Academic integrity

You may study together and discuss the assignments, but what you turn in must be your individual work. Assignment submissions will be checked for plagiarism using Moss (http://theory.stanford.edu/~aiken/moss/). **Copying another student's program or sharing your program is a violation of academic integrity.** Moss is not fooled by renaming variables, reformatting source code, or re-ordering functions.

Violators of academic integrity will suffer severe sanctions, including academic probation. Students who are on academic probation are not eligible for work as instructional assistants in the university or for internships at local companies.