San José State University
Department of Computer Engineering

CMPE 180-92
Data Structures and Algorithms in C++

Fall 2017

Instructor: Ron Mak

Assignment #9

Assigned:	Friday, October 20
Due:	Thursday, October 26 at 5:30 PM
CodeCheck:	http://codecheck.it/files/1710212432aqyuus4pbpu4yq1jsll3ryj3g
Canvas:	Assignment #9. STL Vector and List
Points:	100 with extra credit

STL Vector and List

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

You will see whether a vector or a 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. For extra credit, you can make a few small tweaks to significantly reduce the overhead.

Test suite

Your program will run a suite of tests for the following operations on two types of STL containers, a vector and a list. Run each test several times with an increasing number of data nodes: 100; 500; 1000; 5000; 10,000; and 50,000 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.

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

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

Sample output

```
=======
Prepend
=======
         |---------------Vector-------------|   |----------------List-------------|
   Size         Time  Creates    Copies Destroys        Time  Creates    Copies Destroys
    100         0 ms      100       227      227        0 ms      100       100      100
    500         0 ms      500     1,011    1,011        0 ms      500       500      500
  1,000         2 ms    1,000     2,023    2,023        0 ms    1,000     1,000    1,000
  5,000        65 ms    5,000    13,191   13,191        0 ms    5,000     5,000    5,000
 10,000       231 ms   10,000    26,383   26,383        1 ms   10,000    10,000   10,000
 50,000     5,509 ms   50,000   115,535  115,535        6 ms   50,000    50,000   50,000

======
Append
======
         |---------------Vector-------------|   |----------------List-------------|
   Size         Time  Creates    Copies Destroys        Time  Creates    Copies Destroys
    100         0 ms      100       227      227        0 ms      100       100      100
    500         0 ms      500     1,011    1,011        0 ms      500       500      500
  1,000         0 ms    1,000     2,023    2,023        0 ms    1,000     1,000    1,000
  5,000         0 ms    5,000    13,191   13,191        0 ms    5,000     5,000    5,000
 10,000         1 ms   10,000    26,383   26,383        1 ms   10,000    10,000   10,000
 50,000         3 ms   50,000   115,535  115,535        7 ms   50,000    50,000   50,000

===
Get
===
         |---------------Vector-------------|   |----------------List-------------|
   Size         Time  Creates    Copies Destroys        Time  Creates    Copies Destroys
    100         0 ms        0    10,000   10,000        1 ms        0    10,000   10,000
    500         0 ms        0    10,000   10,000        5 ms        0    10,000   10,000
  1,000         0 ms        0    10,000   10,000       10 ms        0    10,000   10,000
  5,000         0 ms        0    10,000   10,000       51 ms        0    10,000   10,000
 10,000         1 ms        0    10,000   10,000       96 ms        0    10,000   10,000
 50,000         3 ms        0    10,000   10,000      493 ms        0    10,000   10,000

======
Remove
======
         |---------------Vector-------------|   |----------------List-------------|
   Size         Time  Creates    Copies Destroys        Time  Creates    Copies Destroys
    100         0 ms        0         0      100        0 ms        0         0      100
    500         0 ms        0         0      500        0 ms        0         0      500
  1,000         1 ms        0         0    1,000        0 ms        0         0    1,000
  5,000        28 ms        0         0    5,000       14 ms        0         0    5,000
 10,000       120 ms        0         0   10,000       64 ms        0         0   10,000
 50,000     2,960 ms        0         0   50,000    1,711 ms        0         0   50,000

======
Insert
======
         |---------------Vector-------------|   |----------------List-------------|
   Size         Time  Creates    Copies Destroys        Time  Creates    Copies Destroys
    100         0 ms      100       227      227        0 ms      100       100      100
    500         1 ms      500     1,011    1,011        1 ms      500       500      500
  1,000         5 ms    1,000     2,023    2,023        6 ms    1,000     1,000    1,000
  5,000       137 ms    5,000    13,191   13,191      157 ms    5,000     5,000    5,000
 10,000       542 ms   10,000    26,383   26,383      668 ms   10,000    10,000   10,000
 50,000    13,414 ms   50,000   115,535  115,535   19,113 ms   50,000    50,000   50,000

Done! Total time: 45.4937 seconds
```

In the sample output, **`Size`** is the number of data nodes, **`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, and **`Destroys`** is the number of calls to the **`Node`** destructor.

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.

Source files

Source file **`STLVectorList.cpp`** contains the **`main`**.

`Node.h` and **`Node.cpp`**: The data nodes for the containers, each with a **`long value`** data member. During each test, count how many times each **`Node`** constructor, copy constructor, and destructor function is called. Therefore, class **`Node`** has these private static data members:

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

and these public static member functions:

```
static long get_constructor_count();
static long get_copy_count();
static long get_destructor_count();
static void reset();
```

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, 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.

`SortedVector.h` and **`SortedVector.cpp`**: Private member **`vector<Node> data`** is the container. Public member functions **`prepend`**, **`append`**, **`remove`**, and **`insert`** perform the operations described above. Public member function **`at`** returns the node at the given index position in the vector. You may want to add helper member functions.

`SortedList.h` and `SortedList.cpp`: The sorted linked list and the sorted vector have similar attributes. Private member `list<Node> data` is the container. Public member functions `prepend`, `append`, `remove`, and `insert` perform the operations described above. Public member function `at` returns the node at the given index position in the list. You may want to add helper member functions.

Tip: Unlike a vector node, you cannot directly access a list node. Take advantage of reverse iterators. If the node you want to access is closer to the beginning of the list, use a regular (forward) iterator to reach it. However, if the node you want to access is closer to the end of the list, use a reverse iterator to reach it. Unfortunately, STL member functions like `erase` only work with a forward iterator. To convert a reverse iterator that points to a node to a forward 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.)

`STLVectorList.cpp` contains the `main`, which calls function `run_test_suite`. This function 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 the `timed_test` functions to run the vector test and the list test for different data sizes. As shown in the sample output 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: The `chrono` functions require you to compile with `-std=c++0x`.

`TestSuite.cpp` contains all the functions that implement the operation tests for both an STL vector and an STL list. Functions `vector_gets` and `list_gets` each accesses `GET_COUNT` nodes at random index positions. Similarly, functions `vector_remove` and `list_remove` each removes nodes at random index positions until the container is empty. Functions `vector_inserts` and `list_inserts` each inserts nodes with random values up to the specified size. Both the vector and the list must remain sorted.

CodeCheck limitations

This program may run longer than CodeCheck allows. Therefore, your submission to CodeCheck should only test with data sizes of 100; 500; 1,000; 5,000; and 10,000. Include the remaining size 50,000 only outside of CodeCheck.

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

Submission into Canvas

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. Include as part of your submission a separate text file containing the output with all the data sizes 100; 500; 1,000; 5,000; 10,000; and 50,000. 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.

Rubric

Your program will be graded according to these criteria:

Criteria	Max points
Good output	**20**
• Timings	• 10
• Counts	• 10
Good program design	**80**
• Class `Node` constructors and destructor with call counting.	• 5
• `SortedVector::prepend`	• 5
• `SortedVector::append`	• 5
• `SortedVector::remove`	• 5
• `SortedVector::insert`	• 5
• `SortedVector::at`	• 5
• `SortedList::prepend`	• 5
• `SortedList::append`	• 5
• `SortedList::remove`	• 5
• `SortedList::insert`	• 5
• `SortedList::at`	• 5
• Test suite	
○ `vector_prepends` and `list_prepends`	• 5
○ `vector_appends` and `list_appends`	• 5
○ `vector_gets` and `list_gets`	• 5
○ `vector_removes` and `list_removes`	• 5
○ `vector_inserts` and `list_inserts`	• 5

Extra credit #1 (up to 10 points each, total 20 points)

Make some minor tweaks to your program to

- Reduce the vector counts for the **Prepend**, **Append**, and **Insert** tests.
- Reduce all the counts to zero for the **Get** test for both the vector and the list.

Zip your modified program together with a text file of your new output and a brief explanation of your changes.
Submit to Canvas: **Extra credits/Assignment #9: Extra credit #1**.

Extra credit #2 (up to 10 points each, total 50 points)

For each of the five tests, plot the run times of the STL vector and the STL list in an individual graph for comparison. Do the times grow linearly? Exponentially? Submit your graphs (up to five of them) to Canvas: **Extra credits/Assignment #9: Extra credit #2**.

Tip: Modify your program to output only the elapsed times, for example, for sizes 5,000 through 50,000 by increments of 5,000. You can use Excel or any other tool to generate the graphs.

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.