Problem Set 4

Due before midnight on Sunday, November 6, 2011.

1: Assignment Goals

(a) To explore the time efficiency of various mechanisms for maintaining a list of elements.
(b) To understand a simple testing framework.
(c) To learn how to set up and use static tables.
(d) To learn how to use member function pointers.
(e) To learn how to use template classes.
(f) To explore some template classes in the standard C++ library.

2: Assignment

This assignment builds on the class demonstration 15-Runtime, which gives a testing framework for experimentally obtaining the running time of programs. It uses the framework to time the same two code blocks that were timed in demo 14-StopWatch. The first code block allocates an array of 20 million doubles, fills it with random numbers, and then deletes it. The second code block does the same but uses a linked list instead of an array to store the numbers.

In each of the following problems, you are asked to extend the code in 15-Runtime, which has been copied into /c/cs427/assignments/ps4 on the Zoo for your convenience. The extensions are cumulative, so when all problems have been completed, your code should incorporate all of the required extensions. However, if you get stuck on one of the extensions, by all means go ahead with the others and come back to it later. When you submit your code, make clear in your report any required extensions that have not been implemented.

After each extension, compile and run your program, test it for correctness, and test it for memory leaks with valgrind before continuing.

Problem 1: Add a new static constant

Declare numTests to be a static const int in class Tester. Initialize it to the number of tests in the testSuite array. This initialization number should be compile-time computed using the sizeof operator in the same way that num is computed in the first line of Tester::runAllTests(). Delete that line of Tester::runAllTests() and replace the use of num by the new static constant numTests.
Problem 2: Use the data in the stored list

The test subject functions \texttt{test1()} and \texttt{test2()} do not produce any output, so there is no way to tell if they actually run correctly. Change each of these functions to return an \texttt{unsigned int} result, and modify the framework accordingly. Change \texttt{runOneTest()} to print out a space and the return value from the test \textit{in hex} on the same line right after printing the run time. This hex value should begin with the characters “0x” followed by 8 hex digits. Leading zeros should be printed, so the number 0x3f should print as 0x0000003f.

The number that \texttt{test1()} and \texttt{test2()} should return is a checksum of all of the random numbers in the list. The checksum should be computed by making a second pass over the stored list. (Do not compute the checksum in the same loop that stores the array – the idea is to check the numbers that are later fetched \textit{from} the stored list.) The checksum has type \texttt{unsigned int}, which you may assume is a 32-bit register. We define \texttt{checksum} of a list of integers recursively:

\[
\texttt{checksum}(a_1, a_2, \ldots, a_n) = \begin{cases} 
0 & \text{if } n = 0; \\
\text{rot5}(\texttt{checksum}(a_1, \ldots, a_{n-1})) \oplus a_n & \text{if } n > 0.
\end{cases}
\]

(It follows that \texttt{checksum}(a_1) = a_1.) Here, \texttt{\oplus} denotes bitwise exclusive-or (the ‘^’ operator of C++) and \texttt{unsigned int rot5(unsigned int)} is the function that performs a circular left rotation its argument by 5 bits. (This means that the five bits shifted off of the left end of the argument are placed in the low-order five bits of the result.) You will need to use the bitwise shifting and masking operators of C++ to do this. For example,

\[
\text{rot5}(0x10203040) == 0x04060802
\]

If everything is done correctly, and if each test function resets the seed of the random number generator so the generated sequence of numbers is the same, then each test function should return the same checksum. Make sure that \texttt{test1()} and \texttt{test2()} give the same checksum before continuing.

Problem 3: Measure a Java-style Cell class

Add a new \texttt{test3()} function that is identical to \texttt{test2()} except that the type of \texttt{Cell::data} is \texttt{int*} instead of \texttt{int}. This will require some changes to the definition of the \texttt{Cell} class, but you should not change the calling program (with the big \texttt{for}-loop). Make sure that any storage allocated by \texttt{Cell} is deleted when the test function returns.\footnote{This modification reflects what happens with generic containers in Java such as \texttt{ArrayList<Integer>}, where the cell \textit{points to} the actual data rather than \textit{containing} it.}

Problem 4: Measure a FlexArray

Add a new \texttt{test4()} function that is similar to \texttt{test1()} except that it uses a \texttt{FlexArray} with element type \texttt{int} instead of a C-style array.

Problem 5: Measure an STL Vector

Add a new \texttt{test5()} function that is similar to \texttt{test4()} except that it uses the standard \texttt{Vector} template class instantiated to element type \texttt{int}. Note that \texttt{Vector} is similar to, but much more general than, a \texttt{FlexArray}. How would you expect the running times to compare?
Problem 6: Measure an STL List

Add a new test6() function that is similar to test4() except that it uses the standard List template class instantiated to element type int. Note that List is implemented as a doubly-linked list. How would you expect the running times of test2() and test6() to compare?

3: Deliverables

You should submit this assignment using the submit script that you will find in /c/cs427/bin/ on the Zoo. Remember to submit the following items:

(a) All source code and header files, including copies of any files from the ps4 assignment directory that your code needs in order to run.

(b) A makefile and any other files necessary to build your project. (If you’re using Eclipse with the default settings, the makefiles will be found in the directory Debug.)

(c) A brief report named report.txt (or any other common format such as .pdf) describing the code you wrote, the experiments you ran, and the results obtained. You can also put any other information here that the TA should know about your program.