C++ optimization

Justin Allen

English 102

Chabner

For this project I wrote a small program that tests the difference between 5 C++ optimizations and their equivalent, but inefficient counterparts. While this is far from an accurate test, it still gives a general idea about how much time is saved by each of these optimizations. The five tests are: Post-increment vs. Pre-increment (Scheinerman, 2006), un-nested if statements vs. if-else statements ( Dale, Weems, Headington, 1997), if-else statements vs. switch-case (Hubbard, Hubbard, & Baxter ,2000), global counter vs. scoped counter (Schildt, 2001), and initializing variables vs. setting them (Barr, 1999). The test is written so that the more efficient options runs first, and counter a gets set to how many times the code ran in 30 seconds. Counter b is for the less efficient code. A positive difference means the first code ran more times, a negative value indicates the second did.

Post-increment vs. pre-increment is the smallest change on this list, so small that due to outside factors, the test often reads a false negitive. Sometimes it says pre-increment is faster, sometimes it’s post-increment, but when averaged out, pre-increment comes out ahead. The reasoning behind this is that the post-increment has to check to see if the variable is requested by an operation, at which point the operation is given the value before the increment (Perry, 1999). Initializing vs. setting variables is also a very small change that also may return false-negatives, but less often than the increment test. Initializing variables saves the step of the variable getting made to a null value (Barr, 1999). Even when setting 50 doubles (twice the memory space as integers) there is very little difference between the times each ran per second.

The switch and case function is a very clean, nicely formatted function for selection certain cases based on a variable. But for being made for one purpose, it is inefficient at doing such. For those situations, it is better to just use an if-elseif structure (Scheinerman, 2006). Next I compared the if-elseif function to just using the if statement. It’s no surprise that using else if’s instead of more if’s is much more inefficient. When you are only checking on variable, the else if will stop checking after it gets a positive check, skipping anything after it. So when you have one case that is 10 times greater to happen than the others, putting that one first with stop checking after it (Deitel, Deitel, 2005).

The final and often the change that has the largest impact is the global counter vs scoped counter for any type of loop. Scoped variables are created at the beginning of the function and destroyed when the function exits (Dan Saks, 2008). While this saves memory usage, using it for the counters of a loop wastes many operations by creating and destroying that one variable. Just having a global counter for 4 for loops instead of initializing and using 4 different counters for the loops saves millions of operations a second. With C++ you can even use the same counter for a loop inside another loop and aslong as it is entirely inside the loop, it keeps each of them strait (Bulka, Mayhew ,1999).

Many of these are so small that most of the time someone won’t notice the difference. But for any project that is calculation heavy and is always looking for small things to increase performance, some of these can be extremely valuable, and while a 2 million operation/second is a very small difference for today’s computers, my test is on a very small scale and only runs each test for 30 seconds.

Barr, M. (1999). *Programming Embedded Systems in C and C ++*. Sebastopol: O'Reilly Media, Inc..

Bulka, D., & Mayhew, D. (1999). *Efficient C++: Performance Programming Techniques*. New York: Addison-wesley Professional.

Dale, N. B., Weems, C., & Headington, M. R. (1997). *Programming and Problem Solving with C++.* Massachusetts: Jones & Bartlett Publishers, Inc.

Dan Saks.  (2008, May). The yin and yang of dynamic allocation. Embedded Systems Design, 21(5), 12.  Retrieved August 3, 2009, from ProQuest Computing. (Document ID: 1471425411).

Deitel, H. M., & Deitel, P. J. (2005). *C++: How to program*. Upper Saddle River, NJ: Pearson/Prentice Hall.

Hubbard, J. R., Hubbard, J. R., & Baxter, A. Q. (2000). *Programming with C++*. New York: McGraw-Hill.

Perry, G. (1999). *C by Example*. Indianapolis: Que.

Scheinerman, E. (2006). *C++ for Mathematicians: An Introduction for Students and Professionals*. Boca Raton: CRC.

Schildt, H. (2001). *C the Complete Reference*. New York: Mcgraw-Hill Osborne Media.