Maria Javier: mj2729
Design with C++
Prof. Bjarne Stroustrup
HW5

Results and Analysis

In this assignment, I specifically wanted to look at the difference between using templates versus non-generic code. As I was trying to decide how to organize the template, I at first thought that asking the user for both the unknown variable that would contain what needed to be followed by a comparator function followed by iterators could work as an implementation similar to qsort. I decided against due to the fact that the design is just cumbersome and the user would have to have a firm understanding of whatever type they wanted to evaluate. This implementation would also be constricted to sequences that implement iterators. Integers are not included in this implementation. I then tried to configure the type at compile time using typeid().name() which failed miserably due to the fact that it has undefined behavior across different compilers. I found out that this function I decided to opt for one argument which is the most essential one, the variable to evaluate of typename T. I make a copy of this variable, reverse it, and then compare it with the original value passed in. The reverse() function provided by the standard template library requires iterators so this still does not include the case where an integer is passed into the function. In order to deal with I depend on template specialization and make a special case just for integers. The drawback to this is that some might consider this kind of implementation as truly generic but it is a quick and easy way to deal with this case. According to the results below, there was no overhead.

Asides from design, I tested whether or not there was a way to improve non-generic performance compared to generic template use. Preliminary results are shown below. Clearly here, template use is several milliseconds faster than non-generic use. I wasn't able to implement a faster non-generic version that could beat these numbers. I find it interesting how the generic template version seems to be faster considering I use the exact same algorithm for both cases.

Arg Type	Non-Generic (ms)	Generic (ms)
Int	0.01344	0.005183
String	0.00281	0.002422
Vector<Int>	0.001838	0.001622
Vector<string>	0.005337	0.003055

Based off of the research that I have done templates are evaluated at compile time and performs type checking when doing so. The compiler when seeing a call to anything that refers to the template replacing that piece of code with the right type and then proceeding to call it. The benefits of templates include having smaller smarter source files with the advantage of deducing

types at compile time. Here, vector seems to work slightly faster for generic types which makes sense since they exist in memory contiguously.

In the future I would like to reduce my template size to just a one liner by utilizing the standard template library in run tests in order to see if a smaller implementation is faster or produces the same results.

Machine
Macbook Pro (Retina, 2015)
Processor 2.7 Ghz Intel Core i5

Compiler
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Thread Model: posix
C++11

Resources

1. http://www.cplusplus.com/doc/oldtutorial/templates/
2. http://en.cppreference.com/w/cpp/language/template_specialization
3. https://stackoverflow.com/questions/21393179/how-does-c-partial-compilation-with-templates-work
4. http://users.cis.fiu.edu/~weiss/Deltoid/vcstl/templates#T1