# Advanced Programming  Mini-project 2

Martin Zimmer Kristensen
Aalborg University

October 28, 2016

**Static and Dynamic Polymorphism** In comparing So here it a writes the A nonbreaking Spaces. the writability of static polymorphism and dynamic polymorphism in the two versions the first provides concise code through templates that work on generic types. The latter requires that a virtual function is defined, possibly overloaded for each type, and overridden by derived objects such that the appropriate functions can be selected at run-time. This is additional work compared to a templated version where one template can support the same arguments as several overloaded "equivalents".

In comparing the readability of static polymorphism and dynamic polymorphism the latter is more "spelled-out" as the virtual functions clearly define what derived classes should implement and which overloaded functions will be called for different parameter types. With static polymorphism, in the way I have implemented it in the library (not using a design pattern such as CRTP), it is not as clear how the calls to templated functions will be deduced.

Performance-wise the static version performs at about twice the speed of the dynamic version when compiled with no optimization flags. However, when optimization flag -O2 is used they are close-to-equal performance-wise. Therefore, it seems, that the compiler is able to do some optimization that brings the dynamic version on par with the static version. Without the optimization flag, however, the static version would achieve a better performance. Therefore, I expect that there are some cases where static polymorphism will always be faster than dynamic polymorphism depending on the solution and how well the compiler is able to optimize.

**Library Code Size** The implementation of the first assignment's library code size is 18.605 characters and the implementation of the second assignment's library code size is 15.712 characters. However, the comparison is a little vague as the libraries differ in the number of types that are supported, the comments, and in the design approach. It is, however, interesting to note that the second library has smaller code size despite being more complete in the number of types that it supports. The code size, therefore, indicates that using templates allowed me to write less code to support more types.

The executable of the dynamic version when compiled using the -O2 flag is 141K whereas the static version is 219K. This is to be expected as the second version supports more types. If no optimization is used the dynamic version is only 508K compared to the 2M size of the static version. Without optimization the static version is a lot larger which is possibly due to the amount of functions that have to be created by the compiler from the templates. With optimization the size difference is not that significant but I expect that there might be cases when it is difficult for the compiler to optimize and a templated version of a program will grow larger than a non-templated version.

**Concurrency Features** When no optimization is used the concurrent version performs slightly better than the single-threaded. With the -O2 flag the single-threaded version performs slightly better than the concurrent version. I believe this is caused by the additional overhead of creating threads. When the single-threaded version performs better it is because the overhead of creating threads is significant compared to the actual execution times of each statistics. Therefore, the concurrent version is slightly faster when no optimization is used as the execution times of each statistic is longer therefore the additional overhead of creating the threads is less significant.

**Conclusion** By comparing the two versions I believe that static polymorphism is faster and has better writability than that of dynamic polymorphism. The performance tests show that it is significantly faster when none are compiled with optimization flags. Templates allowed me to write less code to support more types without having to overload a function for every type that should be supported.

However, in regards to readability I believe that dynamic polymorphism is slightly better. Defining an interface for functions that should be supported by the derived classes provides a clear overview of what functions are used for different parameter types. With static polymorphism this is not as clear. However, as the program grows and more types are supported the dynamic version might be hard to organize as the code size will increase for every overloaded function. In the templated version the code size remains small in comparison which may improve readability in some cases and makes it easier to maintain the program by having to maintain fewer functions.

I think that static polymorphism should be used when templates can improve the writability, when for example many overloaded functions are necessary, without getting too complex or when performance is an important factor. I believe that in most cases static polymorphism should be chosen for its improved writability and performance, and that dynamic polymorphism should be chosen for its improved readability and less complex design. Personally, I preferred writing the templated version of the assignment as the overloading of every function was tedious.