C++11 Select

sha@ase.au.dk
V0.9

Introduction

In this exercise different pre-selected elements from the new C++11 standard will be broad to use. The exercise is constructed such that the emphasis is on reflection and on one providing one's own examples on how this should be used.

Exercise 1 Moving and *Rule of 5*

In C++11 a new concept has emerged, namely the moving concept. This concept add two additional methods to a given class.

Below are some questions to consider; for some code snippets are needed.

- What is it that makes this addition an improvement.
- What is their signature?
- How are they used and when?
- One can force their usage, when is this a good idea and when is this a bad idea?
- How do you force their usage and more importantly, what happens in the code that makes this feasible?

Exercise 2 Lambda, auto & decltype and new loop facilities

Inspect the file `C++11Select/NewBasicsFacilities/newalgo.cpp`. Consider the code and change all places where you think that one of *lambda*, *auto* and *decltype* would be a more suitable solution. Remember to argue as to why each and every place you do choose to alter or choose *not* to alter.

Exercise 3 Variadic templates

Redo the `TypeList` concept with variadic templates. This means:

- Create an appropriate structure named `TypeList` (see slides).
- Provide an example how this is used
- Implement the *Contains* concept and show that it works.

Exercise 4 Threading

Inspect the file `C++11Select/Threading/async_algo.cpp`. In this file you will find an algorithm typedefed into 3 different names. The assumption is that you in fact have 3 different algorithms to work with.

The idea is that you are to start 3 different threads in parallel (how is up to you) and in those threads you are to copy the data from `data` to your algorithm meaning that the algorithm is to

be constructed in its own thread in `std::shared_ptr` . After the respective thread has completed the construction of the algorithm it should call the method `doAlgo()` to carry out the actual processing. To retrieve the result from these parallel processed algorithms, you are to use a `future` that contains a `std::shared_ptr<...>` to the respective algorithm.

Having acquired access to each of the processed algorithm (in `main()`) , determine which was quickest (in this particular run) and print out the time it used (use method `processingTime()` as well as its name.