Introductory C++ exercises

Rupert Nash

r.nash@epcc.ed.ac.uk

The files for this are on Github.

To check out the repository run:

```
git clone https://github.com/EPCCed/2019-04-16-ModernCpp.git.git
cd 2019-04-16-ModernCpp/practicals/01
```

Array

The array is a fundamental data structure, especially for processing large amounts of data, as it allows the system to take advantage of the cache hierarchy.

Recall the array examples from the lecture - in `2019-04-16-ModernCpp/practicals/-array` is a basic implementation and a (hopefully) working test program.

Compile this and run it for a few problem sizes. What is the scaling?

We need to take a decision about copying - do we wish to allow implicit copying which for large arrays is very slow? If not, should we add an *explicit* method to do this? What would its signature be? How would we tell the compiler not to allow this?

Libraries

While we've taken an RAII approach here, it comes with some overhead: we had to implement (or delete) five functions.

A more idiomatic approach is to wrap the resource into a class that does nothing but manage a resource, then it can be used elsewhere and the compiler will produce correct implicit constructors, destructor and assignment operators with no boilerplate code!

See one of the below for an in-depth discussion: * http://en.cppreference.com/w/cpp/language/rule_of_three * http://scottmeyers.blogspot.co.uk/2014/03/a-concern-about-rule-of-zero.html

Fortunately the standard library includes several "smart pointers" that will do this for you for memory! They can be accessed using the `<memory>` header.

They are:

- `std::unique_ptr` - this uniquely owns the pointed-to object. The object is deleted (can be customised) when the smart pointer destructs or you assign a new value. You cannot copy a `unique_ptr`. *This should be your default pointer type!*
- `std::shared_ptr` - this shares ownership of the pointed-to object. All the child `shared_ptr`s point to the same object. The object will be deleted when *all* the pointers are either destructed or assigned a new value.
- `std::weak_ptr` - much like a `shared_ptr` but it doesn't own a share of the object. It can become invalid. Used to break reference cycles.

(There also exists a `std::auto_ptr`. This is deprecated and has been removed from C++17, so do not use it.)

Have a look at the reference and re-implement `Array` using either a unique or shared pointer.