C++11
Making your code simpler since ~~2011~~2015

Simon Heybrock
simon.heybrock@esss.se

European Spallation Source

Motivation

C++11

- Overwhelming number of important and useful features
- Many of them are perceived as complex

This talk

- Highlight some simple but immensely useful bits of C++11
- All code examples from Mantid

Range-based for and auto

```
vector<string> names = function->getParameterNames();
for (size_t i = 0; i < names.size(); ++i) {
  string &name = names[i];
  map<string, double>::iterator miter = parmap.find(name);
  if (miter != parmap.end())
    function->setParameter(name, miter->second);
}
```

Range-based for and auto

```
vector<string> names = function->getParameterNames();
for (size_t i = 0; i < names.size(); ++i) {
  string &name = names[i];
  map<string, double>::iterator miter = parmap.find(name);
  if (miter != parmap.end())
    function->setParameter(name, miter->second);
}
```

Step 1: range-based for (think of Python's for item in list:)

```
for (string &name : function->getParameterNames()) {
  map<string, double>::iterator miter = parmap.find(name);
  if (miter != parmap.end())
    function->setParameter(name, miter->second);
}
```

Range-based for and auto

```
vector<string> names = function->getParameterNames();
for (size_t i = 0; i < names.size(); ++i) {
  string &name = names[i];
  map<string, double>::iterator miter = parmap.find(name);
  if (miter != parmap.end())
    function->setParameter(name, miter->second);
}
```

Step 1: range-based for (think of Python's for item in list:)

```
for (string &name : function->getParameterNames()) {
  map<string, double>::iterator miter = parmap.find(name);
  if (miter != parmap.end())
    function->setParameter(name, miter->second);
}
```

Step 2: auto

```
for (string &name : function->getParameterNames()) {
  auto miter = parmap.find(name);
  if (miter != parmap.end())
    function->setParameter(name, miter->second);
}
```

The `delete` keyword

Old:

- Private copy constructor, not implemented.
- Fails at link-time if called.

```
private:
  /// Private copy constructor - copying is not allowed.
  FunctionDomain1D(const FunctionDomain1D &r);
```

New: use delete

- Public and deleted copy constructor, clearly states intention.
- Fails at compile-time if called.

```
public:
  FunctionDomain1D(const FunctionDomain1D &r) = delete;
```

The `default` keyword

Old:

- Polymorphic class, copy constructors protected
  ⇒ need to write implementation!

```
// Header
protected:
  IMDEventWorkspace(const IMDEventWorkspace &o);

// Source
IMDEventWorkspace::IMDEventWorkspace(
    const IMDEventWorkspace &other)
    : IMDWorkspace(other), MultipleExperimentInfos(other),
      m_fileNeedsUpdating(other.m_fileNeedsUpdating) {}
```

New: use default

```
protected:
  IMDEventWorkspace(const IMDEventWorkspace &o) = default;
```

Replace `typedef` by `using`

Old:

```cpp
typedef boost::shared_ptr<MatrixWorkspace>
    MatrixWorkspace_sptr;
typedef boost::shared_ptr<const MatrixWorkspace>
    MatrixWorkspace_const_sptr;
```

Replace `typedef` by `using`

Old:

```
typedef boost::shared_ptr<MatrixWorkspace>
    MatrixWorkspace_sptr;
typedef boost::shared_ptr<const MatrixWorkspace>
    MatrixWorkspace_const_sptr;
```

New:

```
using MatrixWorkspace_sptr =
    boost::shared_ptr<MatrixWorkspace>;
using MatrixWorkspace_const_sptr =
    boost::shared_ptr<const MatrixWorkspace>;
```

Replace typedef by using

Old:

```
typedef boost::shared_ptr<MatrixWorkspace>
    MatrixWorkspace_sptr;
typedef boost::shared_ptr<const MatrixWorkspace>
    MatrixWorkspace_const_sptr;
```

New:

```
using MatrixWorkspace_sptr =
    boost::shared_ptr<MatrixWorkspace>;
using MatrixWorkspace_const_sptr =
    boost::shared_ptr<const MatrixWorkspace>;
```

Or: using works with templates!

```
template <class T>
using sptr<T> = boost::shared_ptr<T>;
template <class T>
using const_sptr<T> = boost::shared_ptr<const T>;
```

Class member initializers and delegating constructors

```
Peak::Peak()
    : m_detectorID(-1), m_H(0), m_K(0), m_L(0),
      m_intensity(0), m_sigmaIntensity(0), m_binCount(0),
      m_initialEnergy(0.), m_finalEnergy(0.),
      m_GoniometerMatrix(3, 3, true),
      m_InverseGoniometerMatrix(3, 3, true),
      m_runNumber(0), m_monitorCount(0), m_row(-1),
      m_col(-1), m_orig_H(0), m_orig_K(0), m_orig_L(0),
      m_peakShape(new NoShape) {}

Peak::Peak(Geometry::Instrument_const_sptr m_inst,
           Mantid::Kernel::V3D QLabFrame,
           boost::optional<double> detectorDistance)
    : m_H(0), m_K(0), m_L(0), m_intensity(0),
      m_sigmaIntensity(0), m_binCount(0),
      m_GoniometerMatrix(3, 3, true),
      m_InverseGoniometerMatrix(3, 3, true),
      m_runNumber(0), m_monitorCount(0), m_orig_H(0),
      m_orig_K(0), m_orig_L(0), m_peakShape(new NoShape) {
  this->setInstrument(m_inst);
  this->setQLabFrame(QLabFrame, detectorDistance);
}
```

5 more like this!

Class member initializers and delegating constructors

Class member initializers

```
private:
  double m_H = 0.0;
  double m_intensity = 1.0;
  Kernel::Matrix<double> m_GoniometerMatrix{3, 3, true};
```

Class member initializers and delegating constructors

Class member initializers

```
private:
  double m_H = 0.0;
  double m_intensity = 1.0;
  Kernel::Matrix<double> m_GoniometerMatrix{3, 3, true};
```

Delegating constructors

```
Peak::Peak() = default;

Peak::Peak(Geometry::Instrument_const_sptr m_inst,
           Mantid::Kernel::V3D QLabFrame,
           boost::optional<double> detectorDistance)
    : Peak() {
  this->setInstrument(m_inst);
  this->setQLabFrame(QLabFrame, detectorDistance);
}
```

- Can save > 30 lines of code!
- Significantly reduced risk of bugs.

Also worth mentioning

- `override` keyword
- scoped enums (`enum class`)
- `std::unique_ptr` and `std::shared_ptr` (as we know it from `boost`)
- uniform initialization (can get complicated)