Department I - C Plus Plus

Modern and Lucid C++ Advanced for Professional Programmers

Week 7 – Bonus: Tags for Value Types

Thomas Corbat / Felix Morgner

Rapperswil, 06.04.2020

FS2020

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

IFS
INSTITUTE FOR
SOFTWARE

```
struct Speed {
  constexpr explicit Speed(double kmh)
    : kmh { kmh } {}
  double kmh;
};
```

```
Speed v{1.0}; //Unit?
```

- **Problem: Literal values lack a dimension**
  - Can result in hard to detect bugs
  - Especially when they are implicitly convertible

```
struct Speed {
  constexpr Speed(double value) {...}
};

bool isFasterThanWalking(Speed speed);
```

```
ASSERT(isFasterThanWalking(10.0));
ASSERT(isFasterThanWalking(2.8));
ASSERT(isFasterThanWalking(6.2));
```

- **Possible Approach: Add factory functions for disambiguation**
  - Tedious to call in many places
  - Difficult to extend (Open Closed Principle is violated)

```
struct Speed {
  static Speed fromKmh(double value);
  static Speed fromMph(double value);
  static Speed fromMps(double value);
private:
  Speed(double value);
};

bool isFasterThanWalking(Speed speed);
```

```
ASSERT(isFasterThanWalking(Speed::fromKmh(10.0)));
ASSERT(isFasterThanWalking(Speed::fromMph(2.8)));
ASSERT(isFasterThanWalking(Speed::fromMps(6.2)));
```

- **Create a tag type for the unit**

```
struct Kph;
struct Mph;
struct Mps;
```

- **Create a quantity type template for speed**

```
template <typename Unit>
struct Speed {
  constexpr explicit Speed(double value)
    : value{value}{};
  constexpr explicit operator double() const {
    return value;
  }
private:
  double value;
};
```

- **Add a `speedCast` function**

```
template<typename Target, typename Source>
constexpr Speed<Target> speedCast(Speed<Source> const & source) {
  return Speed<Target>{ ConversionTraits<Target, Source>::convert(source) };
}
```

- **Create a `ConversionTraits` class template**

```
template<typename Target, typename Source>
struct ConversionTraits {
  constexpr static Speed<Target> convert(Speed<Source> sourceValue) = delete;
};
```

- **Specialize `ConversionTraits` class template**

```
template<typename Same>
struct ConversionTraits<Same, Same> {
  constexpr static Speed<Same> convert(Speed<Same> sourceValue) {
    return sourceValue;
  }
};

static constexpr double mphToKphFactor { 1.609344 };
template<>
struct ConversionTraits<tags::Kph, tags::Mph> {
  constexpr static Speed<tags::Kph> convert(Speed<tags::Mph> sourceValue) {
    return Speed<tags::Kph>{static_cast<double>(sourceValue) * mphToKphFactor};
  }
};
//...
```

```
template <typename Unit>
bool isFasterThanWalking(Speed<Unit> speed) {
  return velocity::speedCast<Kph>(speed) > Speed<Kph>{5.0};
}
```

- **Requires comparison operations, i.e >**
  - They can be implemented using boost
- **Arbitrary Speed objects can be compared with an == operator template**

```
template <typename LeftTag, typename RightTag>
constexpr bool operator==(Speed<LeftTag> const & lhs,
                          Speed<RightTag> const & rhs) {
  return lhs == speedCast<LeftTag>(rhs);
}
```