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, 18.04.2023

FS2023

OST
Ostschweizer
Fachhochschule

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

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

- **Problem: Literal values lack a dimension**

    - Can result in hard to detect bugs

    - Especially when they are implicitly convertible

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

auto isFasterThanWalking(Speed speed) -> bool;
```

```cpp
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)

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

auto isFasterThanWalking(Speed speed) -> bool;
```

```cpp
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**

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

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

```cpp
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**

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

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

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

- **Specialize `ConversionTraits` class template**

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


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

```cpp
template <typename Unit>
auto isFasterThanWalking(Speed<Unit> speed) -> bool {
  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**

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