(ab)using the C++ type system for fun and profit(?)

Arno Lepisk

arno@lepisk.se / arno.lepisk@hiq.se

Version 1.1

Example 1

Original

```
Car::Car(int seats, int doors) {
  ...
}
```

```
int seats = 5;
int doors = 4;
auto c1 = Car(seats, doors);
// or...
auto c2 = Car(doors, seats);
```

Example 1

Original

Typedefs

```
using Seats_t = int;
using Doors_t = int;
Car::Car(Seats_t seats, Doors_t doors) {
  ...
}
```

```
Seats_t s = 5;
Doors_t d = 4;
auto c1 = Car(s, d);
auto c2 = Car(d, s);
```

Example 1

Original

Typedefs

Wrapper type

```
template<typename T, uint32_t Id>
struct StrictType {
  T val;
  constexpr explicit StrictType(const T & v) : val{v} {}
  constexpr bool operator == (const StrictType & o)
    const { return val == o.val; }
  constexpr bool operator <  (const StrictType & o)
    const { return val <  o.val; }
};

constexpr uint32_t simplehash(const char * str) {
  return *str + (*str ? 31 * simplehash(str+1) : 0);
}

#define STRICTTYPE(name, type) \
  using name = StrictType<type, simplehash(#name)>;
```

Example 1

Original

Typedefs

Wrapper type

Wrapper use

```
STRICTTYPE(Seats_t, int);
STRICTTYPE(Doors_t, int);
Car::Car(Seats_t seats, Doors_t doors) { /* ... */ }

Seats_t s{5};
Doors_t d{4};
auto c1 = Car(s, d);
// does not compile:
// auto c2 = Car(d, s);
```

```
test/step1.cpp: In function 'int main()':
test/step1.cpp: error: no matching function for call to
'Car::Car(Doors_t&, Seats_t&)'
   auto c2 = Car(d, s);
                     ^
test/step1.cpp: note: candidate: Car::Car(Seats_t, Doors_t)
   Car(Seats_t, Doors_t) {}
   ^~~
test/step1.cpp: note:   no known conversion for argument 1 from
'Doors_t {aka StrictType<int, 2700692506>}' to
'Seats_t {aka StrictType<int, 2700738339>}'
```

Example 1

Original

Typedefs

Wrapper type

Wrapper use

Alt version

```
template<typename T, typename Tag>
struct StrictType {
  T val;
  constexpr explicit StrictType(const T & v) : val{v} {}
  constexpr bool operator == (const StrictType & o)
    const { return val == o.val; }
  constexpr bool operator <  (const StrictType & o)
    const { return val <  o.val; }
};

#define STRICTTYPE(name, type) \
  struct name ## _tag; \
  using name = StrictType<type, name ## _tag>;
```

Example 2

Original

```
void Car::setLength(double len);
```

Example 2

Original

Wrapper

```
template<typename T, typename Scale>
struct Length {
  T val;
  explicit constexpr Length(const T & v) : val{v} {}
  // ...
};

template<typename T> using Meter =
    Length<T, ratio<1>>;
template<typename T> using Centimeter =
    Length<T, std::centi>;
template<typename T> using Inch =
    Length<T, ratio<254, 10000>>;
template<typename T> using Foot =
    Length<T, ratio<12*254, 10000>>;
```

Example 2

Original

Wrapper

Ratio

```
template<typename T>
constexpr T gcd(T n, T d) {
  return n < 1 ? d : (d == 0 ? n : gcd(d, n % d));
}
// std::gcd is in C++1z (C++17)

template<auto n, auto d = 1>
using ratio = std::ratio<n/gcd(n,d), d/gcd(n,d)>;

// or ratio = std::ratio<n,d>::type
```

Example 2

Original

Wrapper

Ratio

Add constructor

```
template<typename T, typename Scale>
struct Length {
  T val;
  explicit constexpr Length(const T & v) : val{v} {}

  template<typename Os>
  constexpr Length(const Length<T, Os> & o) :
    val{o.val * Os::num / Os::den * Scale::den / Scale::num}
    {}
};
```

Example 2

Original

Wrapper

Ratio

Add constructor

Addition

```
template<typename T, typename S1, typename S2>
auto operator+(const Length<T, S1> & l1,
               const Length<T, S2> & l2) {
  const auto gn = gcd(S1::num, S2::num);
  const auto gd = gcd(S1::den, S2::den);
  return Length<T, std::ratio<gn,S1::den*S2::den/gd>>{
    (l1.val * S1::num * S2::den + l2.val * S2::num * S1::den)
    / (gn * gd)
  };
}
```

Example 2

Original

Wrapper

Ratio

Add constructor

Addition

Use

```
template<typename T, typename S>
std::ostream & operator<<(std::ostream & os,
                         const Length<T,S> & l) {
  os << l.val << "(" << S::num << "/" << S::den << ")m";
  return os;
}

template<typename T>
std::ostream & operator<<(std::ostream & os,
                         const Centimeter<T> & l)
  { os << l.val << "cm"; return os; }
// analogous for Meter, Foot, Inch

void Car::setLength(Centimeter<double> cm) {
  std::cout << cm << std::endl;
}
```

Example 2

Original

Wrapper

Ratio

Add constructor

Addition

Use

Result

```
print(Centimeter<double>{423});
print(Meter<double>{4.13});
print(Foot<double>{12});
print(Inch<double>{150});

print(Meter<double>{3} + Centimeter<double>{12});
print(Meter<double>{3} + Inch<double>{5});
print(Meter<double>{1} - Foot<double>{1});
```

```
423cm -> 423cm
4.13m -> 413cm
12ft -> 365.76cm
150in -> 381cm

312cm -> 312cm
15635(1/5000)m -> 312.7cm
869(1/1250)m -> 69.52cm
```

Example 3

The problem

```
void setArea(double);
void setSpeed(double);
void setPower(double);
```

```
void setArea(SquareMeter<double>);
void setSpeed(KilometerPerHour<double>);
void setPower(Watt<double>);
```

Example 3

The problem

Type

```
template<typename T, int L, int M, int Ti, int I,
         typename Scale>
struct unit {
  T val;
  explicit constexpr unit(const T & v) : val{v} {}

  template<typename Os>
  constexpr unit(const unit<T, L, M, Ti, I, Os> & o) :
    val{o.val * Os::num / Os::den * Scale::den / Scale::num}
    {}
};

template<typename T, typename Scale> using Length  =
  unit<T, 1, 0, 0, 0, Scale>;
template<typename T, typename Scale> using Mass    =
  unit<T, 0, 1, 0, 0, Scale>;
template<typename T, typename Scale> using Time    =
  unit<T, 0, 0, 1, 0, Scale>;
template<typename T, typename Scale> using Current =
  unit<T, 0, 0, 0, 1, Scale>;
```

Example 3

The problem

Type

cont

```
template<typename T, typename Scale> using Area     =
  unit<T, 2, 0, 0, 0, Scale>;
template<typename T, typename Scale> using Velocity =
  unit<T, 1, 0, -1, 0, Scale>;
template<typename T, typename Scale> using Energy   =
  unit<T, 2, 1, -2, 0, Scale>;
template<typename T, typename Scale> using Voltage  =
  unit<T, 2, 1, -3, -1, Scale>;
template<typename T, typename Scale> using Power    =
  unit<T, 2, 1, -3, 0, Scale>;
```

Example 3

The problem

Type

cont

Operations

```
template<typename T1, int L1, int M1, int Ti1, int I1,
         typename Scale1,
         typename T2, int L2, int M2, int Ti2, int I2,
         typename Scale2>
auto operator*(
      const unit<T1, L1, M1, Ti1, I1, Scale1> & u1,
      const unit<T2, L2, M2, Ti2, I2, Scale2> & u2) {
  return unit<decltype(u1.val*u2.val),
              L1+L2, M1+M2, Ti1+Ti2, I1+I2,
              std::ratio_multiply<Scale1, Scale2>>{
                u1.val*u2.val
              };
}
```

Example 3

The problem

Type

cont

Operations

Units

```
template<typename T> using Meter = Length<T, ratio<1>>;
template<typename T> using Foot =
  Length<T, ratio<3048,10000>>;
template<typename T> using Kilometer = Length<T, std::kilo>;
template<typename T> using Mile =
  Length<T, ratio<1609'344,1'000>>;

template<typename T> using Second = Time<T, ratio<1>>;
template<typename T> using Hour = Time<T, ratio<3600>>;
template<typename T> using SquareMeter = Area<T, ratio<1>>;

template<typename T> using KilometerPerHour =
  Velocity<T, ratio<1000,3600>>;

template<typename T> using Joule = Energy<T, ratio<1>>;
template<typename T> using Ampere = Current<T, ratio<1>>;
template<typename T> using Volt = Voltage<T, ratio<1>>;

template<typename T> using Watt = Power<T, ratio<1>>;
```

Example 3

The problem

Type

cont

Operations

Units

Literals

```
auto operator""_m(long double v)
  { return Meter<double>(v); }
auto operator""_km(long double v)
  { return Kilometer<double>(v); }
auto operator""_ft(long double v)
  { return Foot<double>(v); }
auto operator""_mi(long double v)
  { return Mile<double>(v); }

auto operator""_s(long double v)
  { return Second<double>(v); }
auto operator""_h(long double v)
  { return Hour<double>(v); }

auto operator""_A(long double v)
  { return Ampere<double>(v); }
auto operator""_V(long double v)
  { return Volt<double>(v); }
```

Example 3

The problem

Type

cont

Operations

Units

Literals

Use

```
setArea(3.0_m*2.5_m);
setArea(2.0_km*0.5_km);
setArea(2.0_m*1.5_ft);

setSpeed(KilometerPerHour<double>{50});
setSpeed(14.0_m/1.0_s);
setSpeed(35.0_mi/1.0_h);

setPower(Watt<double>{40});
setPower(230.0_V * 3.0_A);
```

```
7.5m²
1e+06m²
0.9144m²

50km/h
50.4km/h
56.327km/h

40W
690W
```

Thanks for listening

arno@lepisk.se / arno.lepisk@hiq.se