The pieces fit, so why won't it work?

Arno Lepisk

arno.lepisk@hiq.se

@arno_l f00ale

ClubHiQ 2019-11-27

Background

- Using a typed language is one way of catch bugs in programs
- But using a typed language doesn't catch all type related bugs.
- Problem is independent of language, but my examples are in C++.

Example

```
double calcTotalPrice(double amount, double
{
  return amount * price;
}
```

Problem?

```
auto mytotal = calcTotalPrice(10, 5);
```

```
auto mytotal = calcTotalPrice(10, myprice);
```

```
auto mytotal = calcTotalPrice(myprice, myam
```

Problem!

```
double calcTotalPrice(double amount, double
{
  double discount = 0;
  if(amount > 5) discount = 0.1;
  return amount * price * (1-discount);
}
```

Typedef

```
using price_t = double;
using amount_t = double;
using totalprice_t = double;
```

Typedef

```
totalprice_t calcPrice(amount_t, price_t);
```

```
auto tot = calcPrice(amount_t(3), price_t(1
auto tot = calcPrice(price_t(3), amount_t(1
auto tot = calcPrice(amount_t(3), amount_t(
```

This doesn't help!

A simple wrapper

```
struct price_t { double val; };
struct amount_t { double val; };
struct total_t { double val; };
```

```
total_t calcPrice(amount_t a, price_t p) {
    return total_t{a.val*p.val};
}
```

Wrapper use

```
auto tot = calcPrice(amount_t{3}, price_t{1
~~auto tot = calcPrice(price_t{3}, amount_t{1~~
```

```
error: could not convert 'price_t{3.0e+0}'
       from 'price_t' to 'amount_t'
   |  auto tot = calcPrice(price_t{3},amount
   |                       ^~~~~~~
   |                       |
   |                       price_t
```

Make the wrapper nicer

```
class price_t {
private:
  double value;
public:
  explicit price_t(double v) : value(v) {}
  double get() const { return value; }
};
```

dito for `amount_t` and `total_t`.

A lot to write...

Take 2 - Generic wrapper

```
template<typename Tag, typename T>
class type {
private:
  T value;
public:
  explicit type(T v) : value(std::move(v))
  const T & get() const { return value; }
};
```

Tag types

```
using price_t  = type<struct price_tag, dou
using amount_t = type<struct amount_tag, do
using total_t  = type<struct total_tag, dou
```

Usage

```
total_t calcPrice(amount_t a, price_t p) {
    return total_t(a.get()*p.get());
}
```

Add some operators

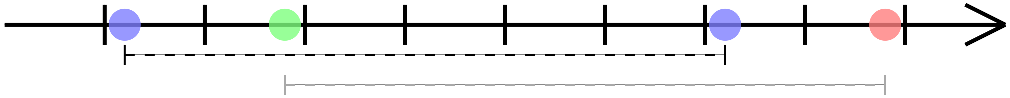
```
total_t operator*(const amount_t & p,
                  const price_t & a) {
    return total_t(a.get() * p.get());
}
```

```
total_t calcPrice(amount_t a, price_t p) {
    return a*p;
}
```

What are the types representing?

This decides which operations are sensible

- Multiply an amount with a unit price?
- Add two amounts? Subtract them?
- Multiply two amounts??
- Add an amount with a price???



Time point and duration

```
auto today = std::chrono::system_clock::now
auto two_days = std::chrono::hours(48);

auto day_after_tomorrow = today + two_days;
```

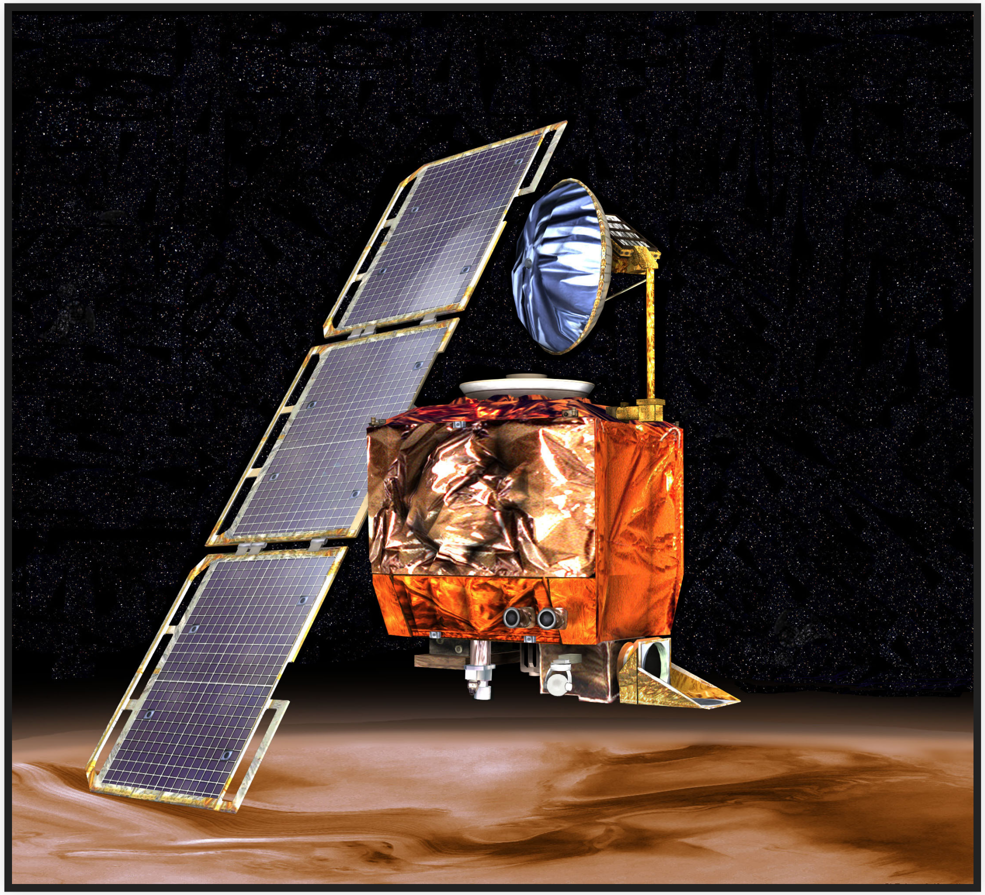
Scaled types

```
auto d1 = std::chrono::hours(1);

auto d2 = std::chrono::hours(1) +
    std::chrono::minutes(30);

auto d3 = 1h + 30min;
```

Mars Climate Orbiter



We can use a scaled unit for lengths

```
void setLength(const units::Meter<double> &
    std::cout << l << std::endl;
}
```

```
setLength(3_m);     -> 3 m
setLength(3000_mm); -> 3 m
setLength(10_ft);   -> 3.048 m
```

Combined units

- length + length → length
- length · length → area
- length / time → speed
- length / time^2 → acceleration
- mass · acceleration → force

Example

```
print(3_m * 4_m);                   → 12m^2
print(SqMeter<double>(3_m*10_ft));  → 9.144
print(12_m / 1.5_s);                → 8 m/s

print(10_kg * g<double>);           → 98.2
print(10_kg * g<double>*5_m);       → 491 J
print(10_kg * g<double>*5_m/.2_s);  → 2455

print(230_V * 10_A);                → 2300
print(2.8_V / 16_mA);               → 0.175
```

How does it work?

Dimensional analysis

- To add / subtract the dimensions have to be the same
- Multiply: add the powers of dimensions
- Divide: subtract powers of dimensions

Math!

$$\text{Length} : (1, 0, 0)$$
$$\text{Time} : (0, 0, 1)$$
$$\text{Velocity} = \text{Length}/\text{Time} :$$
$$(1, 0, 0) - (0, 0, 1) = (1, 0, -1)$$

Back to prices

- per piece
- per weight
- per volume
- per length
- per time
- per …

Conclusion

- Typed languages are good
- Think about the types you use
- Much can be done without a (runtime) performance penalty!

Arno Lepisk

arno.lepisk@hiq.se

@arno_l f00ale