# Affine Combination: Divide by `0`?

*by Alex Cohn*

comment on [http://videocortex.io/2018/Affine-Space-Types](http://videocortex.io/2018/Affine-Space-Types)

# The challenge

```
Point pt1{1,2};
Point pt2{2,2};
std::cout << (pt1 + pt2)/2 << std::endl;
```

# The challenge

```
Point pt1{1,2};
Point pt2{2,2};
std::cout << (pt1 + pt2)/2 << std::endl;
```

```
{1.5, 2}
```

# Linear Combination

```
template <class Point>
class Combination {
public:
    Combination(Point const& pt);
    Combination<Point> operator+=(Point const& pt);
    Combination<Point> operator+(Combination<Point> const& other) const;
    Combination<Point> operator+=(Combination<Point> const& other);
    Combination<Point> operator*(double weight) const;
    Combination<Point> operator*=(double weight);
}
<Point> Combination<Point> operator+(Point const& one, Point const& two);
<Point> Combination<Point> operator*(double weight, Point const& t);
```

# Affine Combination

```
template <class Point>
class Combination {
public:
    Combination(Point const& pt);
    Combination<Point> operator+=(Point const& pt);
    Combination<Point> operator+(Combination<Point> const& other) const;
    Combination<Point> operator+=(Combination<Point> const& other);
    Combination<Point> operator*(double weight) const;
    Combination<Point> operator*=(double weight);
private:
    Point accum;
    double sumOfWeights;
    Point affineCombination() const;
}
```

# SumOfWeights::N

```
<Point> Point Combination::affineCombination() const {
   typedef boost::pfr::tuple_element_t<0, Point> expected_field_t;
   Point ret;
   auto ret_ptr = reinterpret_cast<expected_field_t*>(&ret);
   boost::pfr::for_each_field(accum, [&ret_ptr, this](auto field) {
       static_assert(std::is_same< decltype(field), expected_field_t >() );
       0[ret_ptr++] = static_cast< decltype(field) >( field / this->sumOfWeights );
   });
   return ret;
}

enum SumOfWeights { N };
<Point> Point Combination::operator/(SumOfWeights) const { return affineCombination(); }
```

# struct Point

```
struct Point {
   double x;
   double y;
};

std::ostream& operator<<(std::ostream& os, Point const& pt) {
   os << "{" << x << "," << y << "}";
   return pt.operator<<(os);
};

// copied from OpenCV, but cannot stay: we can't override this for our purposes
Point operator + (const Point& a, const Point& b) {…};
```

# Usage examples

```
Point pt1{1,2}, pt2{2,2};
std::cout << "(pt1 + pt2)/N " << pt1 << ", " << pt2 << " -> " << (pt1 + pt2)/N << std::endl;

auto c = pt1 + pt2;
c += Point{0,0};
c += 2*Point{1,1};

std::cout << "c/N           " << pt1 << ", " << pt2 << " -> " << c/N << std::endl;
std::cout << "10*pt1/N      " << pt1 << " -> " << 10*pt1/N << std::endl;
```

# Usage without special class

```
<T> T Combination::operator/(double w) const { return affineCombination(); }

Point pt1{1,2}, pt2{2,2};
std::cout << "(pt1 + pt2)/2 " << pt1 << ", " << pt2 << " -> " << (pt1 + pt2)/2 << std::endl;
std::cout << "10*pt1/N      " << pt1 << " -> " << 10*pt1/2 << std::endl;
std::cout << "({1,2,3} + {3,2,1})/N -> " << (Point3D{1,2,3} + Point3D{3,2,1})/0 << std::endl;
```

# chrono

```
auto tp1 = std::chrono::system_clock::now();
auto tp2 = tp1 + 2ms;
std::cout << "tp1           " << tp1 << std::endl;
std::cout << "tp2           " << tp2 << std::endl;
std::cout << "(tp1+tp2)/N   " << (tp1+tp2)/N << std::endl;
```

# chrono

```
auto tp1 = std::chrono::system_clock::now();
auto tp2 = tp1 + 2ms;
std::cout << "tp1           " << tp1 << std::endl;
std::cout << "tp2           " << tp2 << std::endl;
std::cout << "(tp1+tp2)/N   " << (tp1+tp2)/N << std::endl;

std::cout << "using count() " <<
             (tp1.time_since_epoch().count() + tp2.time_since_epoch().count())/2 << std::endl;
```

# chrono

```
auto tp1 = std::chrono::system_clock::now();
auto tp2 = tp1 + 2ms;
std::cout << "tp1           " << tp1 << std::endl;
std::cout << "tp2           " << tp2 << std::endl;
std::cout << "(tp1+tp2)/N   " << (tp1+tp2)/N << std::endl;

std::cout << "using count() " <<
             (tp1.time_since_epoch().count() + tp2.time_since_epoch().count())/2 << std::endl;

Combination<decltype(tp1), long long> tpc(tp1);
tpc += tp2;
```

# time_point

```
private:
// some Points are not constexpr aggregate initializable
template<class Clock, class Duration>
Point affineCombination(std::chrono::time_point<Clock, Duration> const&) const {
   auto v = static_cast<typename Duration::rep>(
      accum.time_since_epoch().count() / sumOfWeights);
   return Point(Duration {v});
}
```

# How do we know time_point

```
template<class T>
class is_time_point {
   template<typename U> static auto test(U const* u) -> decltype(u->time_since_epoch(),
std::true_type());
   template<typename> static std::false_type test(...);

public:
   static constexpr bool value = decltype(test<T>(new T()))::value;
};
```

# We assume member .x

```
T affineCombination() const {
    T ret;
    typedef decltype(ret.x) V;
    auto accum_ptr = reinterpret_cast<const V *>(&accum);
    …
```

# Missing: reflection ?

```
#include "boost/pfr/precise.hpp"
```

```
boost::pfr::for_each_field(accum, [](auto field) {});
typedef boost::pfr::tuple_element_t<0, T> V;
```

# Thank you!

https://gist.github.com/alexcohn/e38642f772d7bbfa62baaca0fd1ad0da

*Special thanks to:*

```
#include "boost/pfr/precise.hpp"
```

https://github.com/apolukhin/magic_get by Anton Polukhin (**Yandex**)

CppCon 2016: https://www.youtube.com/watch?v=abdeAew3gmQ