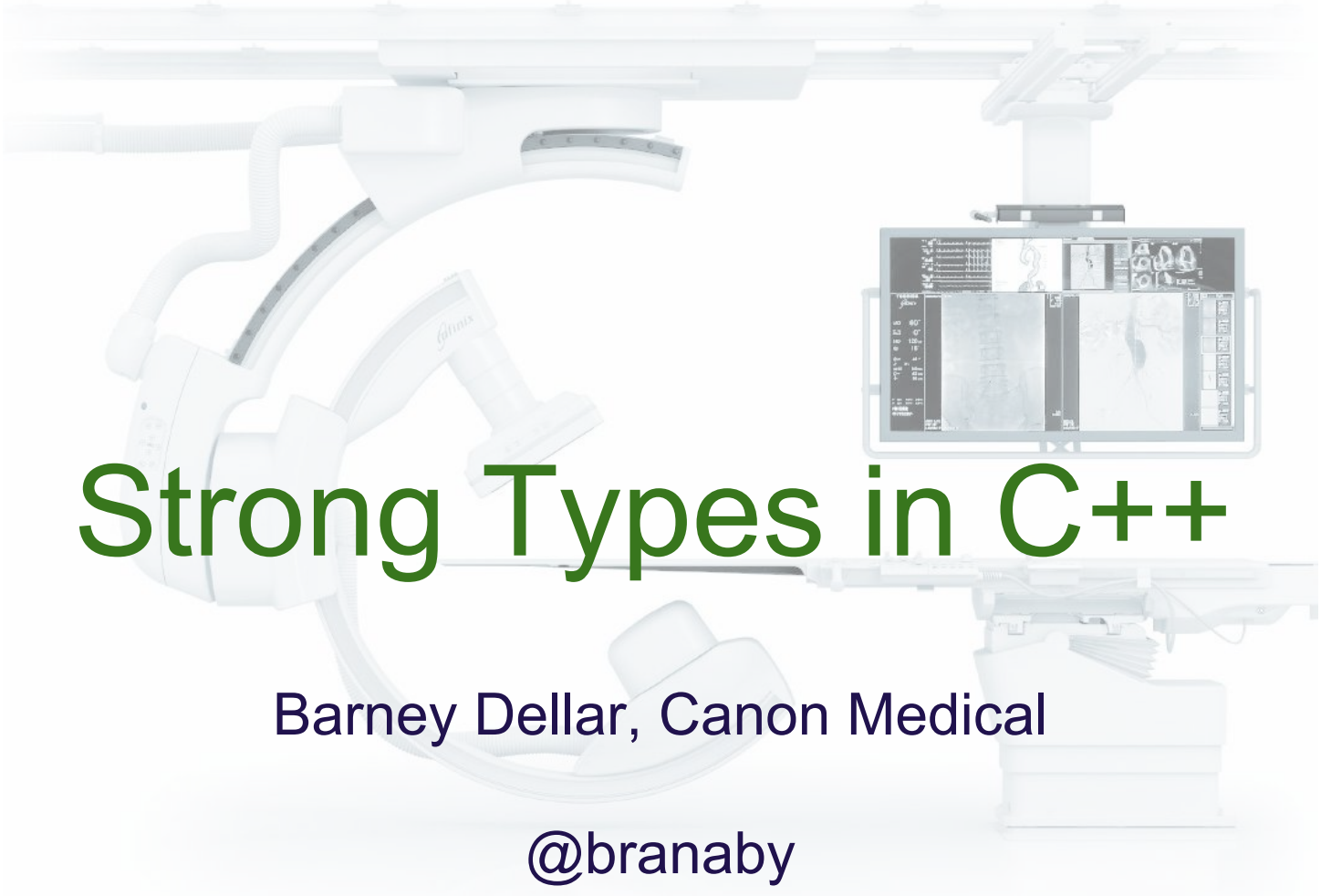


Strong Types in C++

Barney Dellar, Canon Medical

@branaby

So, what is a strong type??

What is a type??

What is a type??

What is a type??

What is a type??

JavaScript

```
let x = 5;
x = 'five';
x = 5.0;
```

```
const separateByComma = function (a, b) {
    return a + ', ' + b;
};

console.log(separateByComma('x', 'y'));
```

```
const separateByComma = function (a, b) {
    return a + ', ' + b;
};

console.log(separateByComma('x', 'y'));

// Prints "x, y"
```

```
const separateByComma = function (a, b) {
    return a + ', ' + b;
};

console.log(separateByComma(1, 2));
```

```
const separateByComma = function (a, b) {
    return a + ', ' + b;
};

console.log(separateByComma(1, 2));

// Prints "1, 2"
```

```
x = 5

x = 'five'

x = 5.0
```

```
def separate_by_comma(a, b):
    return a + ', ' + b


print(separate_by_comma('x', 'y'))
```

```
def separate_by_comma(a, b):
    return a + ', ' + b

print(separate_by_comma('x', 'y'))

# prints "x, y"
```

```
def separate_by_comma(a, b):
    return a + ', ' + b


print(separate_by_comma(1, 2))
```

```
Traceback (most recent call last):
  File "C:\Users\bpde\Desktop\test.py", line 6, in <module>
    print(separate_by_comma(1, 2))
  File "C:\Users\bpde\Desktop\test.py", line 2, in
separate_by_comma
    return s1 + ', ' + s2
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
Traceback (most recent call last):
  File "C:\Users\bpde\Desktop\test.py", line 6, in <module>
    print(separate_by_comma(1, 2))
  File "C:\Users\bpde\Desktop\test.py", line 2, in
separate_by_comma
    return s1 + ', ' + s2
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
Traceback (most recent call last):
  File "C:\Users\bpde\Desktop\test.py", line 6, in <module>
    print(separate_by_comma(1, 2))
  File "C:\Users\bpde\Desktop\test.py", line 2, in
separate_by_comma
    return s1 + ', ' + s2
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
Traceback (most recent call last):
  File "C:\Users\bpde\Desktop\test.py", line 6, in <module>
    print(separate_by_comma(1, 2))
  File "C:\Users\bpde\Desktop\test.py", line 2, in
separate_by_comma
    return s1 + ', ' + s2
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
Traceback (most recent call last):
  File "C:\Users\bpde\Desktop\test.py", line 6, in <module>
    print(separate_by_comma(1, 2))
  File "C:\Users\bpde\Desktop\test.py", line 2, in
separate_by_comma
    return s1 + ', ' + s2
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
std::string SeparateByComma(std::string a, std::string b) {
    return a + ", " + b;
}

std::cout << SeparateByComma("x", "y");
```

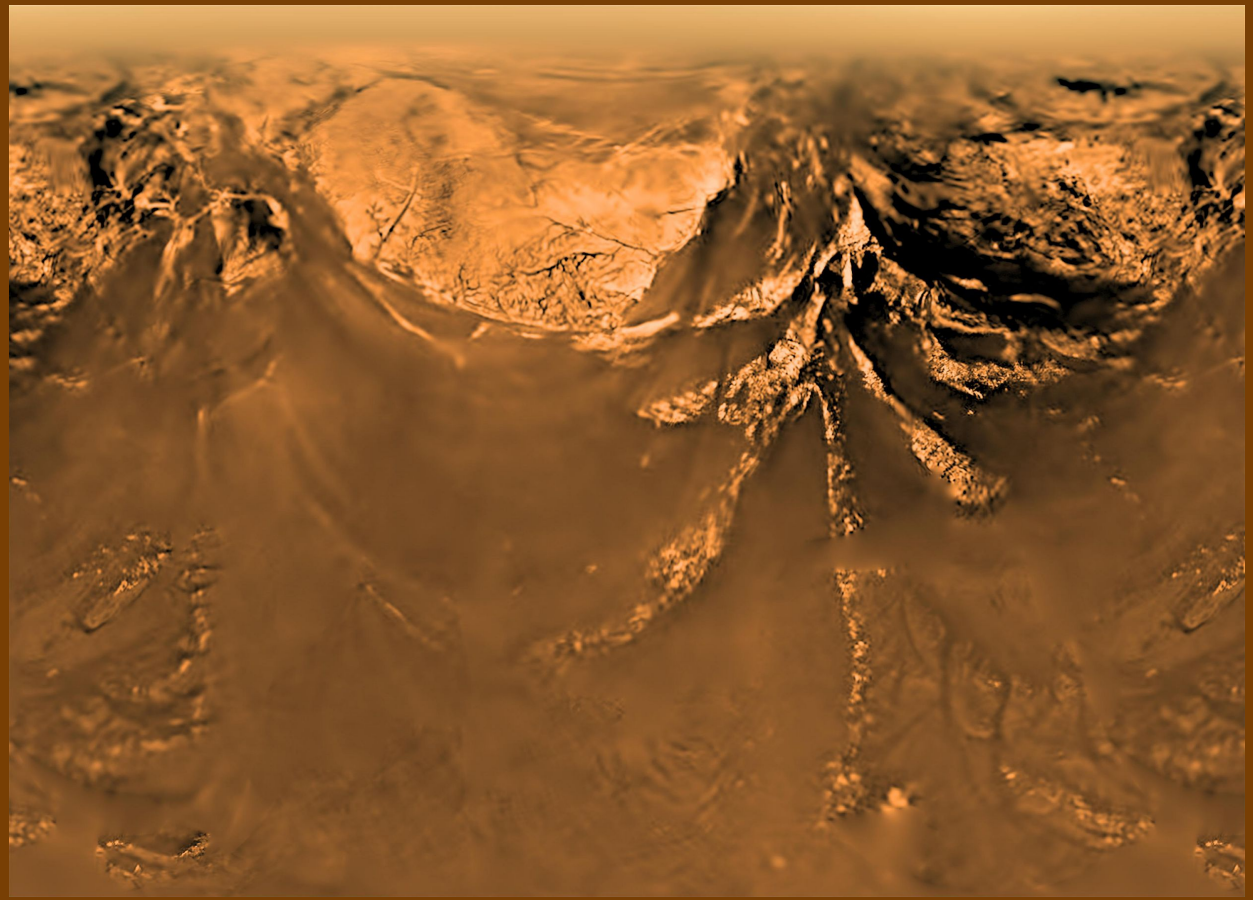
```
std::string SeparateByComma(std::string a, std::string b) {
    return a + ", " + b;
}

std::cout << SeparateByComma("x", "y");

// Prints "x, y"
```

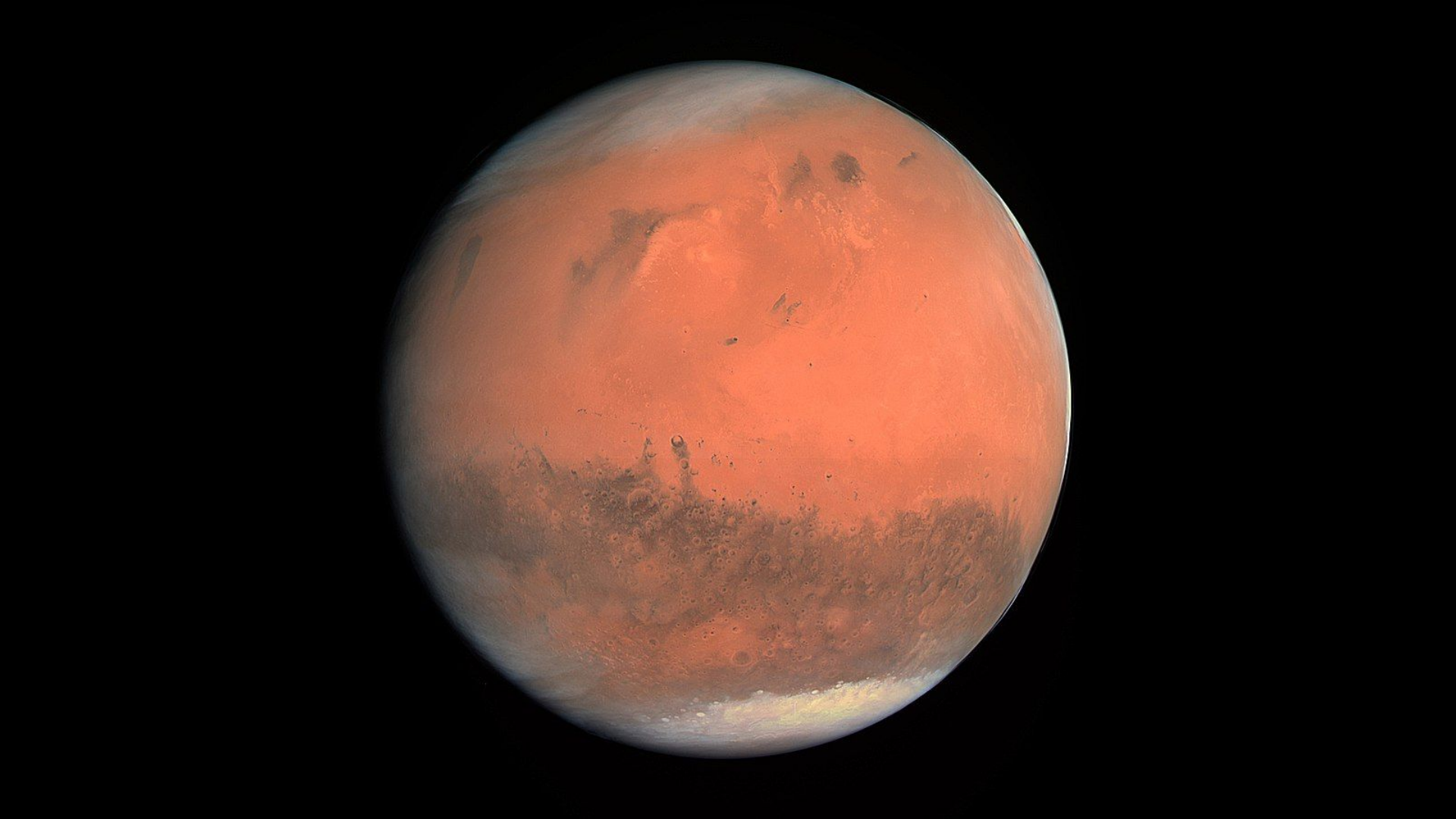
```
std::string SeparateByComma(std::string a, std::string b) {
    return a + ", " + b;
}

std::cout << SeparateByComma(1, 2);
```

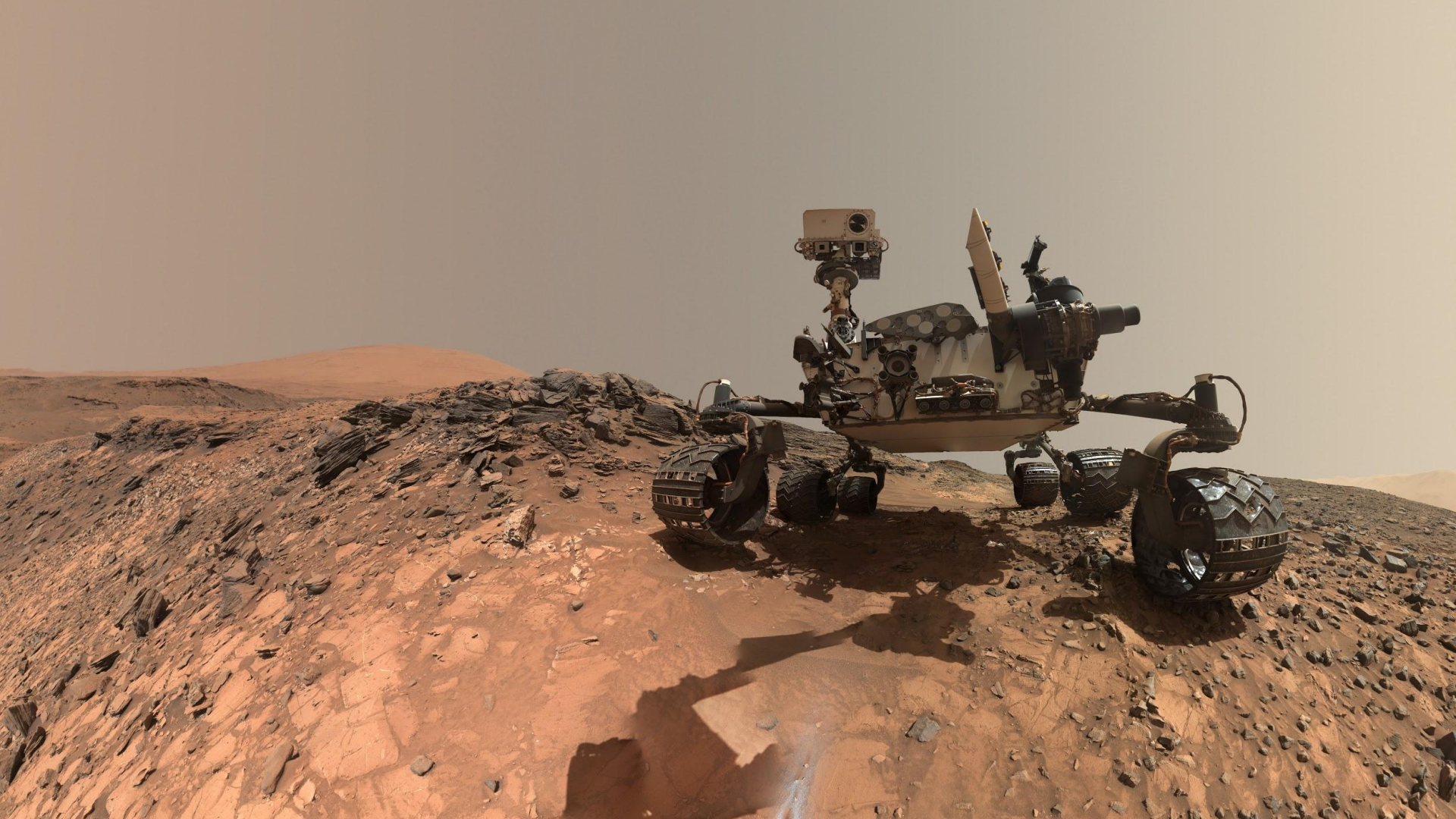
```
error C2664: 'std::string
SeparateByComma(std::string,std::string)': cannot
convert argument 1 from 'int' to 'std::string'
```

C++ is Strongly Typed :-)

C++ is Strongly Typed :-)







DAILY NEWS 25 July 2018

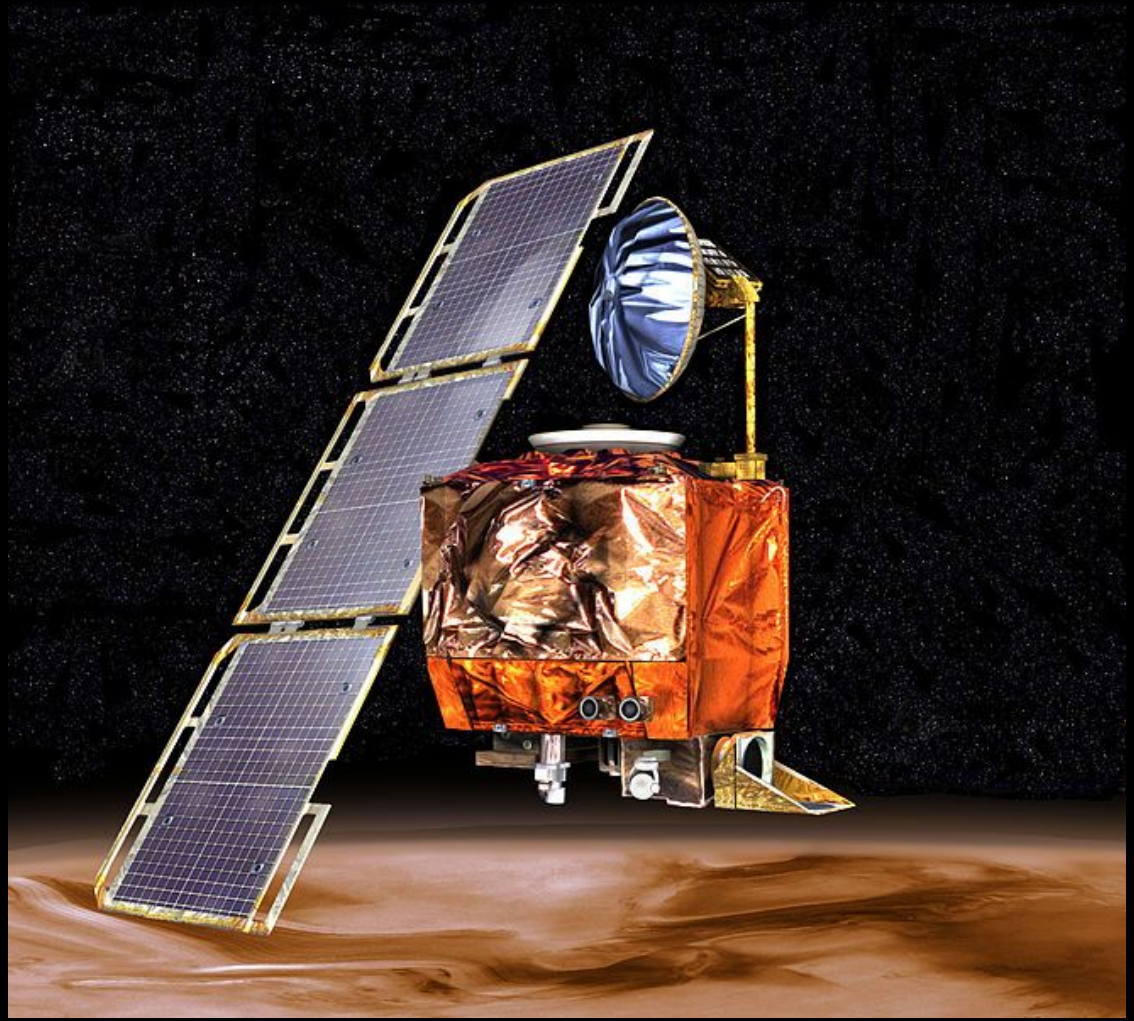
Massive lake of water found beneath Mars' south pole could host life

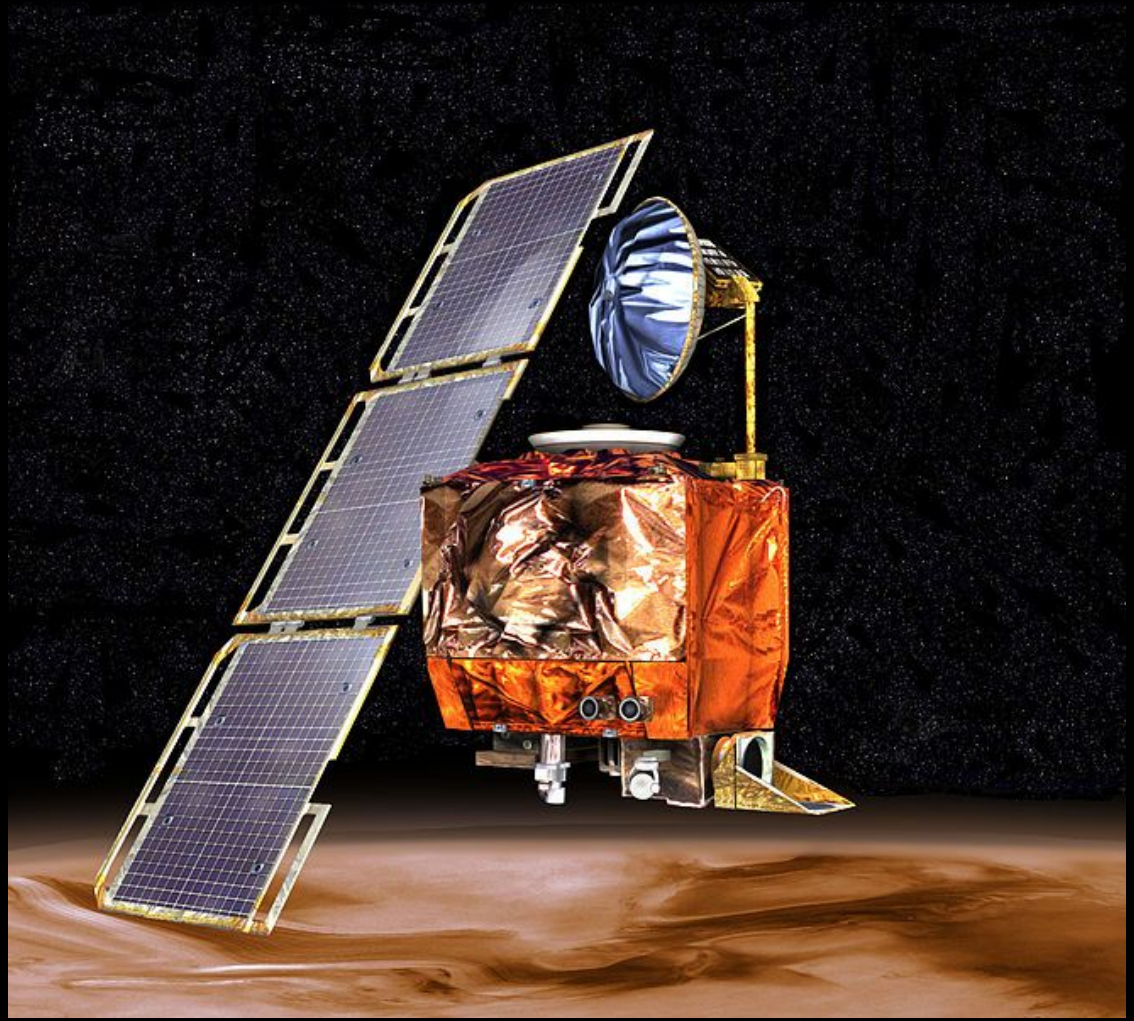
An artist's impression of Mars Express probing the Red Planet
ESA, INAF. Graphic rendering by Davide Coero Borga – Media INAF

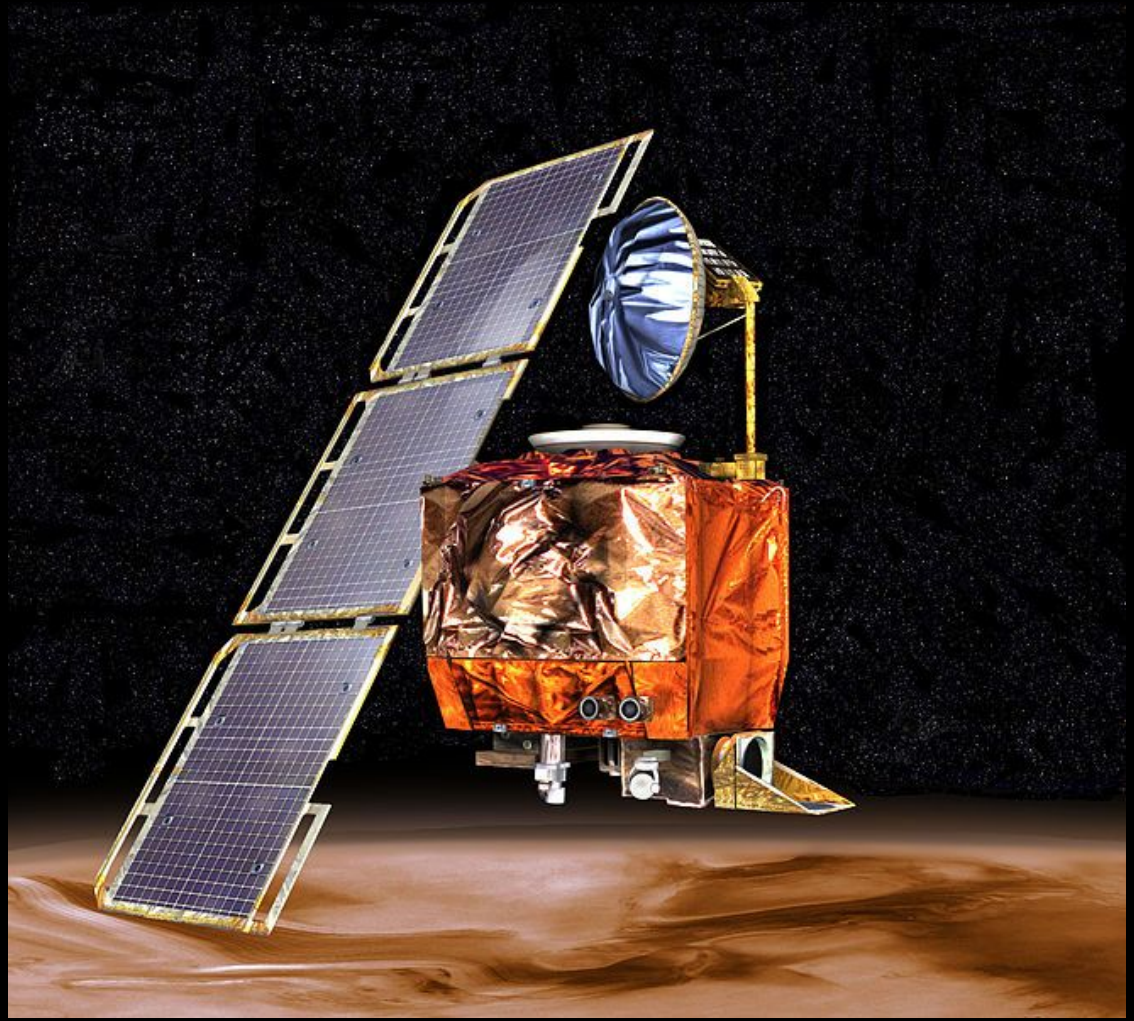
DAILY NEWS 25 July 2018

Massive lake of water found beneath Mars' south pole could host life

An artist's impression of Mars Express probing the Red Planet
ESA, INAF. Graphic rendering by Davide Coero Borga – Media INAF



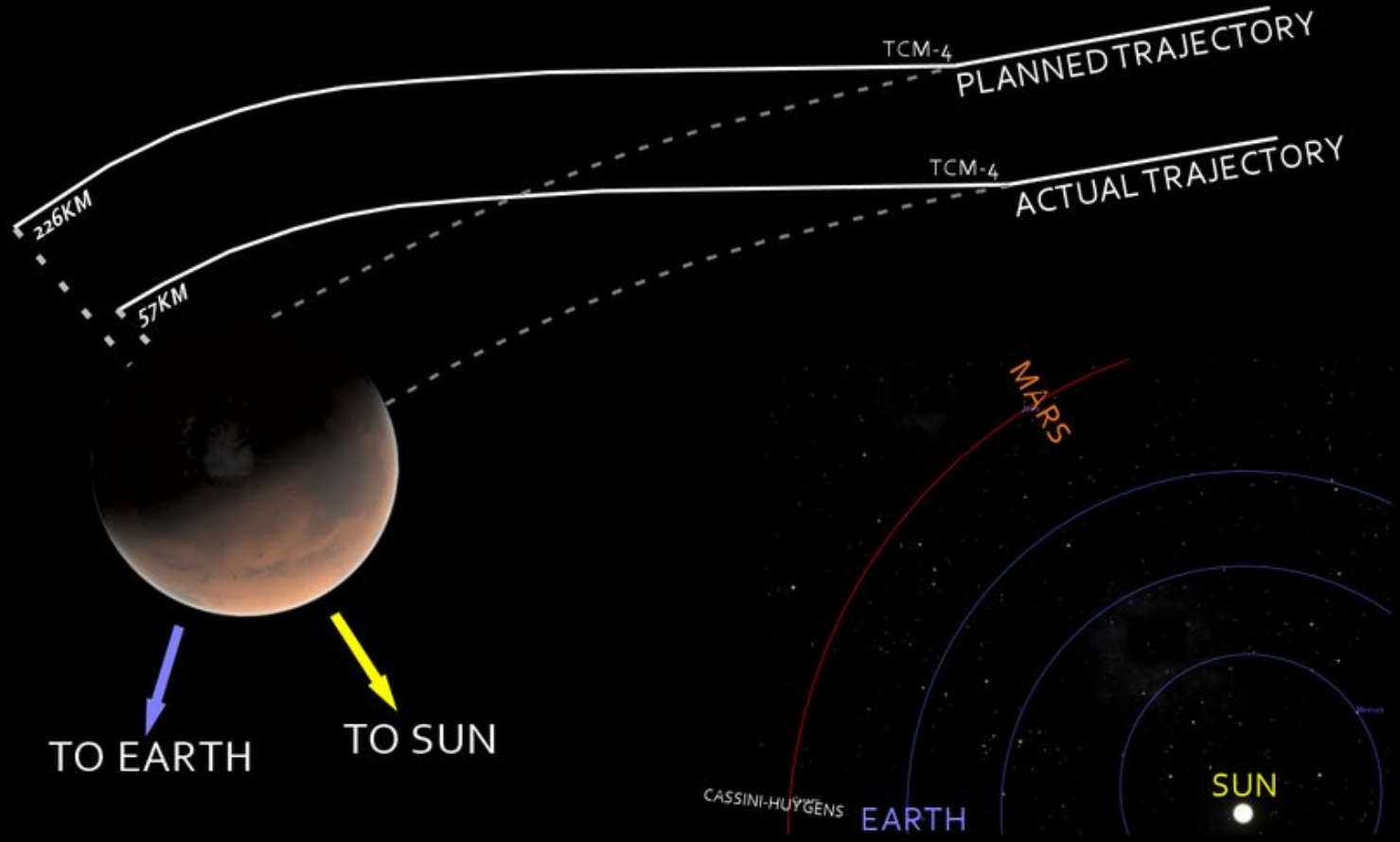




“Metric system used by NASA for many years”

“Metric system used by NASA for many years”

```
// Not actual Lockheed Martin code
double Impulse()
{
    ...
    return impulseInPoundForceSeconds;
}

// Not actual NASA code
double impulseInNewtonSeconds = Impulse();
double CalculateAngularMomentumDesaturation(impulseInNewtonSeconds);
```


TCM-4
PLANNED TRAJECTORY
TCM-4
ACTUAL TRAJECTORY
226KM
57KM
TO EARTH
TO SUN
MARS
CASSINI-HUYGENS
EARTH
SUN

```
// Not actual Lockheed Martin code
double Impulse()
{
    ...
    return impulseInPoundForceSeconds;
}

// Not actual NASA code
double impulseInNewtonSeconds = Impulse();
double CalculateAngularMomentumDesaturation(impulseInNewtonSeconds);
```

```
// Not actual Lockheed Martin code
double Impulse()
{
    ...
    return impulseInPoundForceSeconds;
}

// Not actual NASA code
double impulseInNewtonSeconds = Impulse();
double CalculateAngularMomentumDesaturation(impulseInNewtonSeconds);
```

```
// Not actual Lockheed Martin code
double Impulse()
{
    ...
    return impulseInPoundForceSeconds;
}

// Not actual NASA code
double impulseInNewtonSeconds = Impulse();
double CalculateAngularMomentumDesaturation(impulseInNewtonSeconds);
```

Types as meaning

Types as meaning

Types as meaning

```
double CalculateForce(double mass);
```

```
double CalculateForce(double mass);
```

```
Newton CalculateForce(Kilogram mass);
```

```
Newton CalculateForce(Kilogram mass);

Pound mass = GetMass();
PoundForce force = CalculateForce(mass); // Error!
```

Kilogram	floating point number
Pound	floating point number
Newton	floating point number
PoundForce	floating point number

```
using Kilogram = double;
```

```
using Kilogram = double;
using Pound = double;
```

```
Newton CalculateForce(Kilogram mass);


Pound mass = GetMassInPounds();
auto force = CalculateForce(mass);
```

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

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

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

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

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

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

```
template <typename UnderlyingType>
class StrongType
{
public:
    explicit StrongType(UnderlyingType const& value) : value_(value) {}
    UnderlyingType get() { return value_; }
private:
    UnderlyingType value_;
};
```

```cpp
template <typename UnderlyingType>
class StrongType
{
public:
    explicit StrongType(UnderlyingType const& value) : value_(value) {}
    UnderlyingType get() { return value_; }
private:
    UnderlyingType value_;
};

using Kilogram = StrongType<double>;
```

```cpp
template <typename UnderlyingType>
class StrongType
{
public:
    explicit StrongType(UnderlyingType const& value) : value_(value) {}
    UnderlyingType get() { return value_; }
private:
    UnderlyingType value_;
};

using Kilogram = StrongType<double>;
using Pound = StrongType<double>;
```

```cpp
template <typename UnderlyingType>
class StrongType
{
public:
    explicit StrongType(UnderlyingType const& value) : value_(value) {}
    UnderlyingType get() { return value_; }
private:
    UnderlyingType value_;
};

using Kilogram = StrongType<double>;
using Pound = StrongType<double>;
```

```cpp
template <typename UnderlyingType, typename PhantomTag>
class StrongType
{
public:
    explicit StrongType(UnderlyingType const& value) : value_(value) {}
    UnderlyingType get() { return value_; }
private:
    UnderlyingType value_;
};

struct KilogramParameter{};
using Kilogram = StrongType<double, KilogramParameter>;
```

```
template <typename UnderlyingType, typename PhantomTag>
class StrongType
{
public:
    explicit StrongType(UnderlyingType const& value) : value_(value) {}
    UnderlyingType get() { return value_; }
private:
    UnderlyingType value_;
};

struct KilogramParameter{};
using Kilogram = StrongType<double, KilogramParameter>;
```

```
template <typename UnderlyingType, typename PhantomTag>
class StrongType
{
public:
    explicit StrongType(UnderlyingType const& value) : value_(value) {}
    UnderlyingType get() { return value_; }
private:
    UnderlyingType value_;
};

struct KilogramParameter{};
using Kilogram = StrongType<double, KilogramParameter>;

struct PoundParameter{};
using Pound = StrongType<double, PoundParameter>;
```

```
template <typename UnderlyingType, typename PhantomTag>
class StrongType
{
public:
    explicit StrongType(UnderlyingType const& value) : value_(value) {}
    UnderlyingType get() { return value_; }
private:
    UnderlyingType value_;
};

using Kilogram = StrongType<double, struct KilogramParameter>;
using Pound = StrongType<double, struct PoundParameter>;
```

```
template <typename UnderlyingType, typename PhantomTag>
class StrongType
{
public:
    explicit StrongType(UnderlyingType const& value) : value_(value) {}
    UnderlyingType get() { return value_; }
private:
    UnderlyingType value_;
};

using Kilogram = StrongType<double, struct KilogramParameter>;
using Pound = StrongType<double, struct PoundParameter>;
```

```
template <typename UnderlyingType, typename PhantomTag>
class StrongType
{
public:
    explicit StrongType(UnderlyingType const& value) : value_(value) {}
    UnderlyingType get() { return value_; }
private:
    UnderlyingType value_;
};

using Kilogram = StrongType<double, struct KilogramParameter>;
using Pound = StrongType<double, struct PoundParameter>;
```

Unit Conversion

```cpp
using namespace std::chrono;
void PrintSeconds(seconds s)
{
    std::cout << s.count() << '\n';
}

int main()
{
    hours hrs = 4h;
    PrintSeconds(hrs); // Prints 14400
}
```

```
std::ratio<3, 2>
```

```
std::ratio<60>;
```

```
std::ratio<60>;

std::ratio<60, 1>;
```

```
template <typename UnderlyingType, typename PhantomTag>
class StrongType
{
public:
    explicit StrongType(UnderlyingType const& value) : value_(value) {}
    UnderlyingType get() { return value_; }
private:
    UnderlyingType value_;
};
```

```
template <typename UnderlyingType, typename PhantomTag, typename Ratio>
class StrongType
{
public:
    explicit StrongType(UnderlyingType const& value) : value_(value) {}
    UnderlyingType get() { return value_; }
private:
    UnderlyingType value_;
};
```

```cpp
struct MassTag {};

using Gram = StrongType<double, MassTag, std::ratio<1>>;

using Kilogram = StrongType<double, MassTag, std::kilo>;
```

```
// In StrongType class definition
template <typename OtherRatio>
operator StrongType<UnderlyingType, PhantomTag, OtherRatio>()
const
{
    return StrongType<UnderlyingType, PhantomTag, OtherRatio>(
        get() *
        Ratio::num / Ratio::den *
        OtherRatio::den / OtherRatio::num
    );
}
```

```
// In StrongType class definition
template <typename OtherRatio>
operator StrongType<UnderlyingType, PhantomTag, OtherRatio>()
const
{
    return StrongType<UnderlyingType, PhantomTag, OtherRatio>(
        get() *
        Ratio::num / Ratio::den *
        OtherRatio::den / OtherRatio::num
    );
}
```

```
using Gram = StrongType<double, MassTag, std::ratio<1>>;

using Kilogram = StrongType<double, MassTag, std::kilo>;
```

```
template <typename UnderlyingType, typename PhantomTag, typename Ratio>
class StrongTypeImpl {
    ...
}

template <typename UnderlyingType, typename PhantomTag>
using StrongType = StrongTypeImpl<UnderlyingType, PhantomTag, std::ratio<1>>;
```

```
using Gram = StrongType<double, struct MassTag>;

using Kilogram = MultipleOf<Gram, std::kilo>;
```

Conversion from Pounds to Kilograms

```
using Pound = MultipleOf<Kilogram, std::ratio<56699, 125000>>;
```

Adding in base functionality

```
Gram sum = Gram(g1.get() + g2.get());
```

```
Gram sum = Gram(g1.get() + g2.get());
```

```
Gram sum = Gram(g1.get() + g2.get());
```

```
Gram sum = g1 + g2;
```

```
Gram sum = g1 + g2;
```

```
Gram sum = g1 + g2;
```

```
using Gram = StrongType<double, MassTag, Addable, Subtractable>;
```

Curiously Recurring Template Pattern!

```
template <typename Strong_Type, template<typename> class crtpType>
struct crtp
{
    Strong_Type const& underlying() const { return static_cast<Strong_Type const&>(*this); }
};

template <typename Strong_Type>
struct Addable : crtp<Strong_Type, Addable>
{
    Strong_Type operator+(Strong_Type const& other) const {
        return Strong_Type(this->underlying().get() + other.get());
    }
};

template <typename UnderlyingType, typename PhantomTag>
class StrongType : Addable<StrongType<UnderlyingType, PhantomTag>>
{
    ...
};
```

```cpp
template <typename Strong_Type, template<typename> class crtpType>
struct crtp
{
    Strong_Type const& underlying() const { return static_cast<Strong_Type const&>(*this); }
};

template <typename Strong_Type>
struct Addable : crtp<Strong_Type, Addable>
{
    Strong_Type operator+(Strong_Type const& other) const {
        return Strong_Type(this->underlying().get() + other.get());
    }
};

template <typename UnderlyingType, typename PhantomTag>
class StrongType : Addable<StrongType<UnderlyingType, PhantomTag>>
{
    ...
};
```

```
template <typename Strong_Type, template<typename> class crtpType>
struct crtp
{
    Strong_Type const& underlying() const { return static_cast<Strong_Type const&>(*this); }
};

template <typename Strong_Type>
struct Addable : crtp<Strong_Type, Addable>
{
    Strong_Type operator+(Strong_Type const& other) const {
        return Strong_Type(this->underlying().get() + other.get());
    }
};

template <typename UnderlyingType, typename PhantomTag>
class StrongType : Addable<StrongType<UnderlyingType, PhantomTag>>
{
    ...
};
```

```
template <typename Strong_Type, template<typename> class crtpType>
struct crtp
{
    Strong_Type const& underlying() const { return static_cast<Strong_Type const&>(*this); }
};

template <typename Strong_Type>
struct Addable : crtp<Strong_Type, Addable>
{
    Strong_Type operator+(Strong_Type const& other) const {
        return Strong_Type(this->underlying().get() + other.get());
    }
};

template <typename UnderlyingType, typename PhantomTag>
class StrongType : Addable<StrongType<UnderlyingType, PhantomTag>>
{
    ...
};
```

```
template <typename Strong_Type, template<typename> class crtpType>
struct crtp
{
    Strong_Type const& underlying() const { return static_cast<Strong_Type const&>(*this);
};

template <typename Strong_Type>
struct Addable : crtp<Strong_Type, Addable>
{
    Strong_Type operator+(Strong_Type const& other) const {
        return Strong_Type(this->underlying().get() + other.get());
    }
};

template <typename UnderlyingType, typename PhantomTag>
class StrongType : Addable<StrongType<UnderlyingType, PhantomTag>>
{
    ...
};
```

```cpp
template <typename Strong_Type, template<typename> class crtpType>
struct crtp
{
    Strong_Type const& underlying() const { return static_cast<Strong_Type const&>(*this); }
};

template <typename Strong_Type>
struct Addable : crtp<Strong_Type, Addable>
{
    Strong_Type operator+(Strong_Type const& other) const {
        return Strong_Type(this->underlying().get() + other.get());
    }
};

template <typename UnderlyingType, typename PhantomTag>
class StrongType : Addable<StrongType<UnderlyingType, PhantomTag>>
{
    ...
};
```

```cpp
template <typename Strong_Type>
struct Addable : crtp<Strong_Type, Addable>
{
    Strong_Type operator+(Strong_Type const& other) const {
        return Strong_Type(this->underlying().get() + other.get());
    }
};
template <typename Strong_Type>
struct Subtractable : crtp<Strong_Type, Subtractable>
{
    Strong_Type operator-(Strong_Type const& other) const {
        return Strong_Type(this->underlying().get() - other.get());
    }
};
template <typename Strong_Type>
struct Multipliable : crtp<Strong_Type, Multipliable>
{
    Strong_Type operator*(Strong_Type const& other) const {
        return Strong_Type(this->underlying().get() * other.get());
    }
};
```

```
template <typename Strong_Type>
struct Comparable : crtp<Strong_Type, Comparable>
{
    bool operator<(Strong_Type const& other) const { return this->underlying().get() < other.get(); }
    bool operator>(Strong_Type const& other) const { return other.get() < this->underlying().get(); }
    bool operator<=(Strong_Type const& other) const { return !(other.get() < this->underlying().get()); }
    bool operator>=(Strong_Type const& other) const { return !(*this < other); }
    bool operator==(Strong_Type const& other) const { return !(*this < other) && !(other.get() <
this->underlying().get()); }
    bool operator!=(Strong_Type const& other) const { return !(*this == other); }
};
```

```cpp
template <typename Strong_Type>
struct Printable : crtp<Strong_Type, Printable>
{
    void print(std::ostream& os) const { os << this->underlying().get(); }
};
```

```cpp
template <typename Strong_Type>
struct Printable : crtp<Strong_Type, Printable>
{
    void print(std::ostream& os) const { os << this->underlying().get(); }
};

using MyType = StrongType<double, struct MyTag>;

MyType m{5};
std::cout << m; // Doesn’t work :-(
```

```
template <typename UnderlyingType, typename PhantomTag>
std::ostream& operator<<(
    std::ostream& os,
    StrongType<UnderlyingType, PhantomTag> const& object
) {
    ...
}
```

```cpp
template <typename Strong_Type>
struct Printable : crtp<Strong_Type, Printable>
{
    void print(std::ostream& os) const { os << this->underlying().get(); }
};

template <typename UnderlyingType, typename PhantomTag>
std::ostream& operator<<(
    std::ostream& os,
    StrongType<UnderlyingType, PhantomTag> const& object
) {
    object.print(os);
    return os;
}
```

```cpp
template <typename Strong_Type>
struct Printable : crtp<Strong_Type, Printable>
{
    void print(std::ostream& os) const { os << this->underlying().get(); }
};

template <typename UnderlyingType, typename PhantomTag>
std::ostream& operator<<(
    std::ostream& os,
    StrongType<UnderlyingType, PhantomTag> const& object
) {
    object.print(os);
    return os;
}

using MyType = StrongType<double, struct MyTag>;

MyType m{5};
std::cout << m; // Works now :-)
```

```
using MyType = StrongType<double, struct MyPhantom, Addable, Subtractable>;
```

```
template <
    typename UnderlyingType,
    typename PhantomTag,
    template<typename> class... Skills
>
class StrongType :
    public Skills<StrongType<UnderlyingType, PhantomTag, Skills...>>...
{
    ...
};
```

```
template <
    typename UnderlyingType,
    typename PhantomTag,
    template<typename> class... Skills
>
class StrongType :
    public Skills<StrongType<UnderlyingType, PhantomTag, Skills...>>...
{
    ...
};
```

```
template <
    typename UnderlyingType,
    typename PhantomTag,
    template<typename> class... Skills
>
class StrongType :
    public Skills<StrongType<UnderlyingType, PhantomTag, Skills...>>...
{
    ...
};
```

```
template <
    typename UnderlyingType,
    typename PhantomTag,
    template<typename> class... Skills
>
class StrongType :
    public Skills<StrongType<UnderlyingType, PhantomTag, Skills...>>...
{
    ...
};
```

```
template <
    typename UnderlyingType,
    typename PhantomTag,
    template<typename> class... Skills
>
class StrongType :
    public Skills<StrongType<UnderlyingType, PhantomTag, Skills...>>...
{
    ...
};

using XType = StrongType<double, struct MyPhantom, Addable, Subtractable>;
using YType = StrongType<double, struct MyPhantom, Multipliable>;
```

Defining literals

```
Gram operator"" _grams(unsigned long long g)
{
    return Gram(g);
}
```

```
Gram operator"" _grams(unsigned long long g)
{
    return Gram(g);
}

Gram g = 3_grams;
```

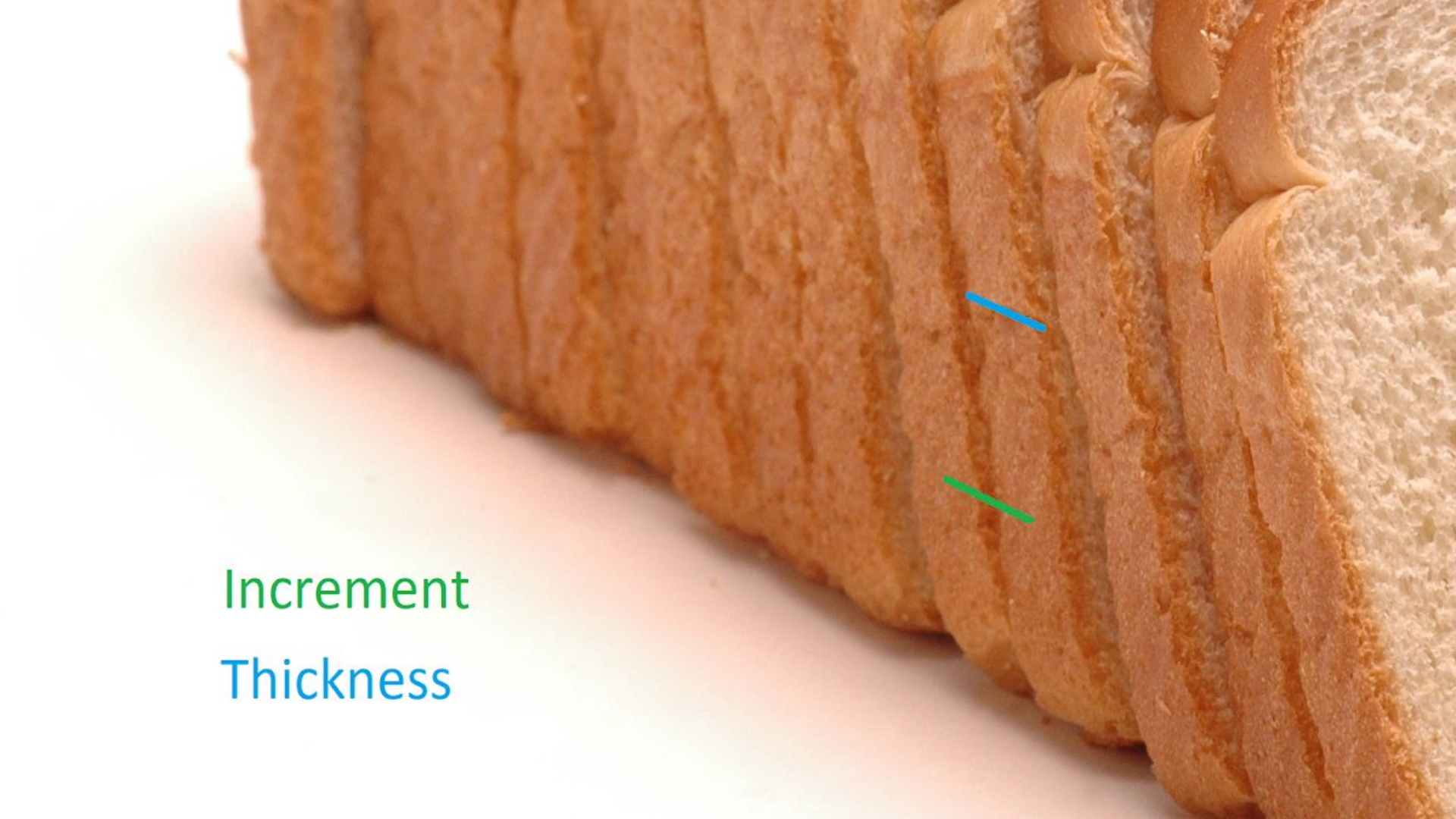
```
Gram operator"" _grams(unsigned long long g)
{
    return Gram(g);
}

Kilogram k = 10_grams;
Pound p = 2_grams;
```

```
Kilogram operator"" _kilograms(unsigned long long k)
{
    return Kilogram(k);
}

Pound operator"" _pounds(unsigned long long p)
{
    return Pound(p);
}
```

Show how it all falls together

```
using Gram = StrongType<double, MassTag, Addable, Printable>;

using Kilogram = MultipleOf<Gram, std::kilo>;
using Pound = MultipleOf<Kilogram, std::ratio<56699, 125000>>;

Kilogram operator"" _kilograms(unsigned long long k) {return Kilogram(k);}
Pound operator"" _pounds(unsigned long long p) {return Pound(p);}

void printMassInGrams(Gram mass) {
    std::cout << mass << "g";
}

printMassInGrams(2_kilograms + 2_pounds);
// Prints 2907.185g :-)
```

Real-World use-cases


Increment
Thickness

```
void Initialise(bool enabled, double thickness, double increment);
```

```
void Initialise(bool enabled, double thickness, double increment);
```

```
void Initialise(bool enabled, double thickness, double increment);
```

```
// Header file:
void Initialise(bool enabled, double thickness, double increment);

// Cpp file:
void MyClass::Initialise(bool enabled, double increment, double thickness) {
    //...
}
```

```
struct StepValues {
    bool enabled;
    double thickness;
    double increment;
};

void Initialise(StepValues);
```

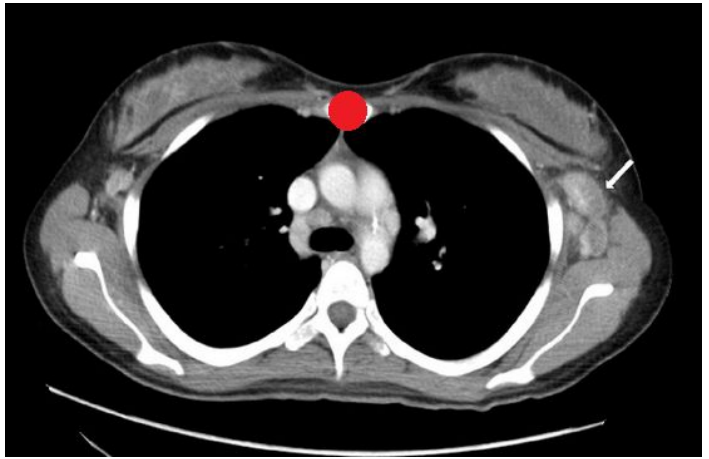
```
using Enabled = StrongType<bool, struct EnabledTag>;
using Thickness = StrongType<double, struct ThicknessTag>;
using Increment = StrongType<double, struct IncrementTag>;

struct StepValues {
    Enabled enabled;
    Thickness thickness;
    Increment increment;
};

void Initialise(StepValues);
```

```
void Initialise(Enabled, Thickness, Increment);
```

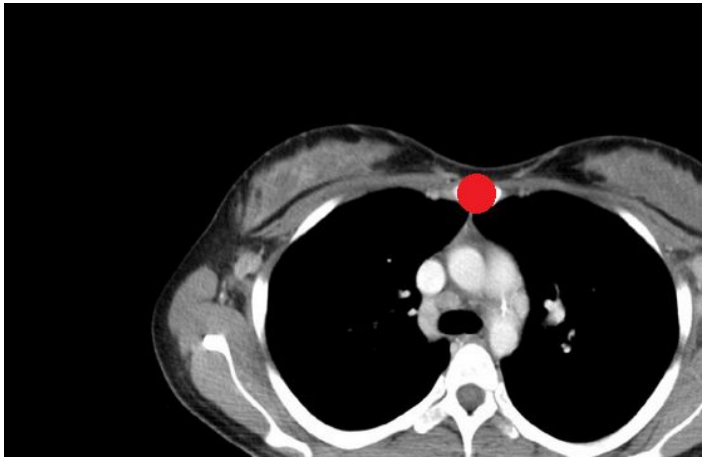
```
void Initialise(Enabled, Thickness, Increment);
```

Real-World use-cases


ONE
Aquilion
10
sec
10
sec




ONE
Aquilion
10
sec
10
sec



```
class Vector {
public:
    Vector(float x, float y, float z);
    float X() const;
    float Y() const;
    float Z() const;

    Vector operator +(const Vector& other) const;
    Vector operator -(const Vector& other) const;
    float Dot(const Vector& other) const;
    Vector Cross(const Vector& other) const;
    void Norm();
    float Mag();
    ...
private:
    float data[3];
};
```

```
class Matrix {
public:
    ...
    Vector Transform(const Vector& vector) const;
    Vector Transform33(const Vector& vector) const;
    ...
private:
    float data[12];
};
```

```
class TransformManager
{
public:
    ...
    Matrix GetTransform(Space from, Space to) const;
    ...
};
```

```
Vector p_patient = ...;
Vector q_patient = ...;

Vector pq_patient = q_patient - p_patient;

Matrix transform_matrix = ...;

Vector p_view = transform_matrix.Transform(p_patient);
Vector pq_view = transform_matrix.Transform33(pq_patient);

Vector broken = pq_view + q_patient; // What is this?

double m = pq_patient.Mag();
double p = pq_view.Mag();
```

```
Vector p_patient = ...;
Vector q_patient = ...;

Vector pq_patient = q_patient - p_patient;

Matrix transform_matrix = ...;

Vector p_view = transform_matrix.Transform33(p_patient);
Vector pq_view = transform_matrix.Transform(pq_patient);

Vector broken = pq_view + q_patient; // What is this?

double m = pq_patient.Mag();
double p = pq_view.Mag();
```

```
Vector p_patient = ...;
Vector q_patient = ...;

Vector pq_patient = q_patient - p_patient;

Matrix transform_matrix = ...;

Vector p_view = transform_matrix.Transform(p_patient);
Vector pq_view = transform_matrix.Transform33(pq_patient);

Vector broken = pq_view + q_patient; // What is this?

double m = pq_patient.Mag();
double p = pq_view.Mag();
```

```
Vector p_patient = ...;
Vector q_patient = ...;

Vector pq_patient = q_patient - p_patient;

Matrix transform_matrix = ...;

Vector p_view = transform_matrix.Transform(p_patient);
Vector pq_view = transform_matrix.Transform33(pq_patient);

Vector broken = pq_view + q_patient; // What is this?

double m = pq_patient.Mag();
double p = pq_view.Mag();
```

```
Vector p_patient = ...;
Vector q_patient = ...;

Vector pq_patient = q_patient - p_patient;

Matrix transform_matrix = ...;

Vector p_view = transform_matrix.Transform(p_patient);
Vector pq_view = transform_matrix.Transform33(pq_patient);

Vector broken = pq_view + q_patient; // What is this?

double m = pq_patient.Mag();
double p = pq_view.Mag();
```

```
Vector p_patient = ...;
Vector q_patient = ...;

Vector pq_patient = q_patient - p_patient;

Matrix transform_matrix = ...;

Vector p_view = transform_matrix.Transform(p_patient);
Vector pq_view = transform_matrix.Transform33(pq_patient);

Vector broken = pq_view + q_patient; // What is this?

double m = pq_patient.Mag();
double p = pq_view.Mag();
```

```
Vector p_patient = ...;
Vector q_patient = ...;

Vector pq_patient = q_patient - p_patient;

Matrix transform_matrix = ...;

Vector p_view = transform_matrix.Transform(p_patient);
Vector pq_view = transform_matrix.Transform33(pq_patient);

Vector broken = pq_view + q_patient; // What is this?

double m = pq_patient.Mag();
double p = pq_view.Mag();
```

Understandable RayTracing in 256 lines of bare C++

https://github.com/ssloy/tinyraytracer

```
Vec3f cast_ray(
    const Vec3f &orig,
    const Vec3f &dir,
    const std::vector<Sphere> &spheres,
    const std::vector<Light> &lights,
    size_t depth = 0
);
```

Understandable RayTracing in 256 lines of bare C++

https://github.com/ssloy/tinyraytracer

```
Vec3f cast_ray(
    const Vec3f &orig,
    const Vec3f &dir,
    const std::vector<Sphere> &spheres,
    const std::vector<Light> &lights,
    size_t depth = 0
);
```

Understandable RayTracing in 256 lines of bare C++

https://github.com/ssloy/tinyraytracer

```
Vec3f cast_ray(
    const Vec3f &orig,
    const Vec3f &dir,
    const std::vector<Sphere> &spheres,
    const std::vector<Light> &lights,
    size_t depth = 0
);
```

```
Patient::Point p_patient = ...;
Patient::Point q_patient = ...;

Patient::Vector pq_patient = q_patient - p_patient;

TransformManager helper = ...;

View::Point p_view = p_patient.ConvertTo<View>(helper);
View::Vector pq_view = pq_patient.ConvertTo<View>(helper);

auto broken = pq_view + q_patient; // Does not compile :-)

Millimetres m = pq_patient.Mag();
Pixels p = pq_view.Mag();
```

```
Patient::Point p_patient = ...;
Patient::Point q_patient = ...;

Patient::Vector pq_patient = q_patient - p_patient;

TransformManager helper = ...;

View::Point p_view = p_patient.ConvertTo<View>(helper);
View::Vector pq_view = pq_patient.ConvertTo<View>(helper);

auto broken = pq_view + q_patient; // Does not compile :-)

Millimetres m = pq_patient.Mag();
Pixels p = pq_view.Mag();
```

```
Patient::Point p_patient = ...;
Patient::Point q_patient = ...;

Patient::Vector pq_patient = q_patient - p_patient;

TransformManager helper = ...;

View::Point p_view = p_patient.ConvertTo<View>(helper);
View::Vector pq_view = pq_patient.ConvertTo<View>(helper);

auto broken = pq_view + q_patient; // Does not compile :-)

Millimetres m = pq_patient.Mag();
Pixels p = pq_view.Mag();
```

```
Patient::Point p_patient = ...;
Patient::Point q_patient = ...;

Patient::Vector pq_patient = q_patient - p_patient;

TransformManager helper = ...;

View::Point p_view = p_patient.ConvertTo<View>(helper);
View::Vector pq_view = pq_patient.ConvertTo<View>(helper);

auto broken = pq_view + q_patient; // Does not compile :-)

Millimetres m = pq_patient.Mag();
Pixels p = pq_view.Mag();
```

```
Patient::Point p_patient = ...;
Patient::Point q_patient = ...;

Patient::Vector pq_patient = q_patient - p_patient;

TransformManager helper = ...;

View::Point p_view = p_patient.ConvertTo<View>(helper);
View::Vector pq_view = pq_patient.ConvertTo<View>(helper);

auto broken = pq_view + q_patient; // Does not compile :-)

Millimetres m = pq_patient.Mag();
Pixels p = pq_view.Mag();
```

```
error: You can't add points to vectors

...

error: You can't add vectors from different spaces
```

Conclusion

Conclusion

MCO Root Cause

The MCO MIB has determined that the root cause for the loss of the MCO spacecraft was the failure to use metric units in the coding of a ground software file, “Small Forces,” used in trajectory models. Specifically, thruster performance data in English units instead of metric units was used in the software application code titled SM_FORCES (small forces). The output from the SM_FORCES application code as required by a MSOP Project Software Interface Specification (SIS) was to be in metric units of Newton-seconds (N-s). Instead, the data was reported in English units of pound-seconds (lbf-s). The Angular Momentum Desaturation (AMD) file contained the output data from the SM_FORCES software. The SIS, which was not followed, defines both the format and units of the AMD file generated by ground-based computers. Subsequent processing of the data from AMD file by the navigation software algorithm therefore, underestimated the effect on the spacecraft trajectory by a factor of 4.45, which is the required conversion factor from force in pounds to Newtons. An erroneous trajectory was computed using this incorrect data.

Conclusion

MCO Root Cause

The MCO MIB has determined that the root cause for the loss of the MCO spacecraft was the failure to use metric units in the coding of a ground software file, “Small Forces,” used in trajectory models. Specifically, thruster performance data in English units instead of metric units was used in the software application code titled SM_FORCES (small forces). The output from the SM_FORCES application code as required by a MSOP Project Software Interface Specification (SIS) was to be in metric units of Newton-seconds (N-s). Instead, the data was reported in English units of pound-seconds (lbf-s). The Angular Momentum Desaturation (AMD) file contained the output data from the SM_FORCES software. The SIS, which was not followed, defines both the format and units of the AMD file generated by ground-based computers. Subsequent processing of the data from AMD file by the navigation software algorithm therefore, underestimated the effect on the spacecraft trajectory by a factor of 4.45, which is the required conversion factor from force in pounds to Newtons. An erroneous trajectory was computed using this incorrect data.

Conclusion

"Culture Eats Strategy for Breakfast"

Peter Drucker

Conclusion

- Types add **meaning**

Conclusion

- Types add **meaning**

Conclusion

- Types add **meaning**
- Types allow us to **understand** our code

Conclusion

- Types add **meaning**
- Types allow us to **understand** our code
- Types allow **others** to understand our code

Conclusion

- Types add **meaning**
- Types allow us to **understand** our code
- Types allow **others** to understand our code
- Strong typing allows the **compiler** to catch bugs

Conclusion

- Types add **meaning**
- Types allow us to **understand** our code
- Types allow **others** to understand our code
- Strong typing allows the **compiler** to catch bugs
- Strong types make an API **easy** to use well, and **difficult** to use badly

Conclusion

- Types add **meaning**
- Types allow us to **understand** our code
- Types allow **others** to understand our code
- Strong typing allows the **compiler** to catch bugs
- Strong types make an API **easy** to use well, and **difficult** to use badly
- Strong types tell us **what something is**, not how it’s implemented

Conclusion

- Types add **meaning**
- Types allow us to **understand** our code
- Types allow **others** to understand our code
- Strong typing allows the **compiler** to catch bugs
- Strong types make an API **easy** to use well, and **difficult** to use badly
- Strong types tell us **what something is**, not how it’s implemented
- You can write code closer to the **problem domain**

Conclusion

- Types add **meaning**
- Types allow us to **understand** our code
- Types allow **others** to understand our code
- Strong typing allows the **compiler** to catch bugs
- Strong types make an API **easy** to use well, and **difficult** to use badly
- Strong types tell us **what something is**, not how it’s implemented
- You can write code closer to the **problem domain**
- With a well-written library, you can **easily** create strong types

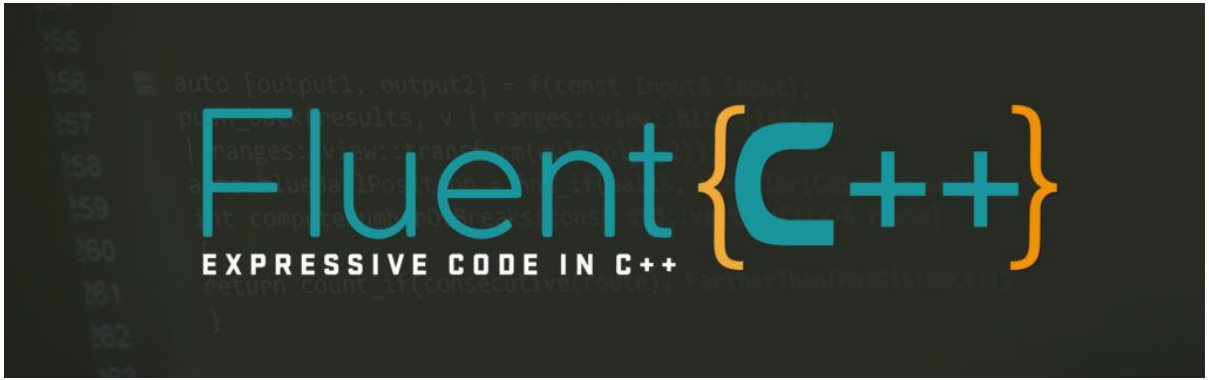
Conclusion

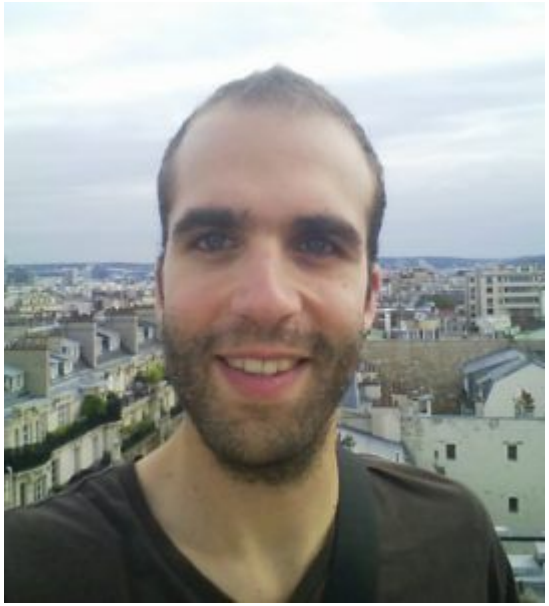
- Types add **meaning**
- Types allow us to **understand** our code
- Types allow **others** to understand our code
- Strong typing allows the **compiler** to catch bugs
- Strong types make an API **easy** to use well, and **difficult** to use badly
- Strong types tell us **what something is**, not how it's implemented
- You can write code closer to the **problem domain**
- With a well-written library, you can **easily** create strong types
- **Other people** have already written these libraries for you **:-)**

Conclusion

- Types add **meaning**
- Types allow us to **understand** our code
- Types allow **others** to understand our code
- Strong typing allows the **compiler** to catch bugs
- Strong types make an API **easy** to use well, and **difficult** to use badly
- Strong types tell us **what something is**, not how it's implemented
- You can write code closer to the **problem domain**
- With a well-written library, you can **easily** create strong types
- **Other people** have already written these libraries for you **:-)**

Jonathan Boccara





https://www.fluentcpp.com/posts/#strong-types

https://github.com/joboccara/StrongType

Jonathan Müller

https://foonathan.net/blog/2016/10/11/type-safe.html

https://github.com/foonathan/type_safe

- Types add **meaning**
- Types allow us to **understand** our code
- Types allow **others** to understand our code
- Strong typing allows the **compiler** to catch bugs
- Strong types make an API **easy** to use well, and **difficult** to use badly
- Strong types tell us **what something is**, not how it's implemented
- You can write code closer to the **problem domain**
- With a well-written library, you can **easily** create strong types
- **Other people** have already written these libraries for you **:-)**

Thanks :-)

@branaby