Typing++ for C++

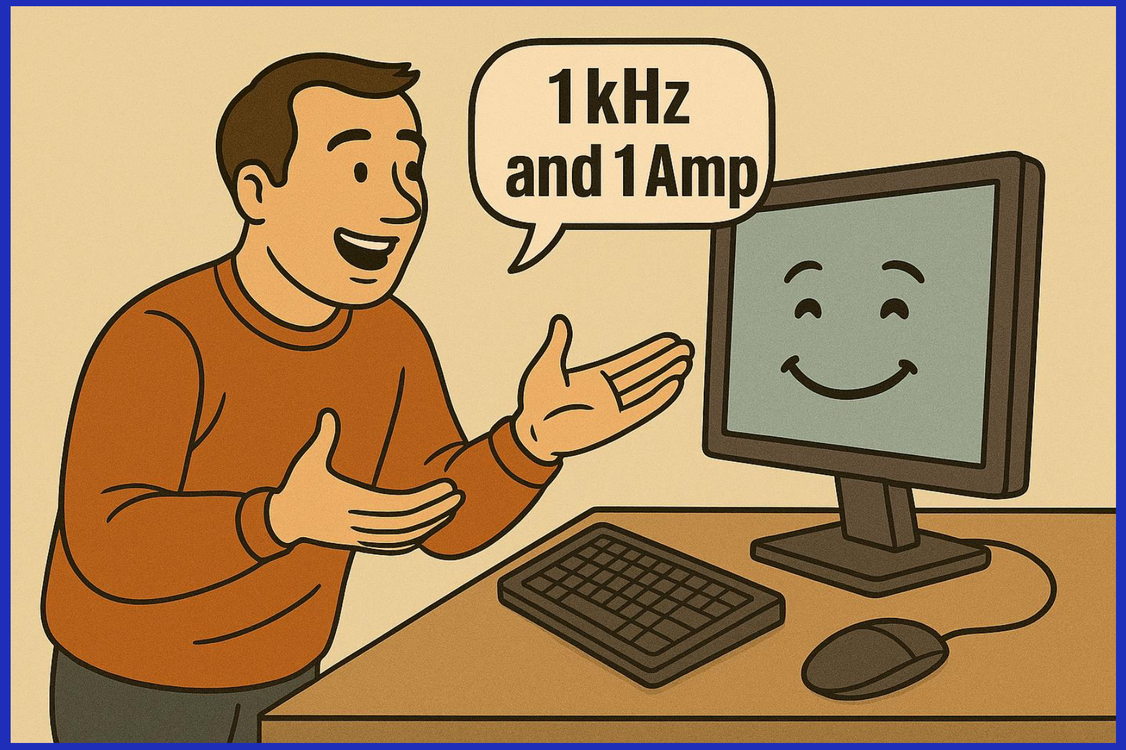
Making the Compiler Do the Thinking

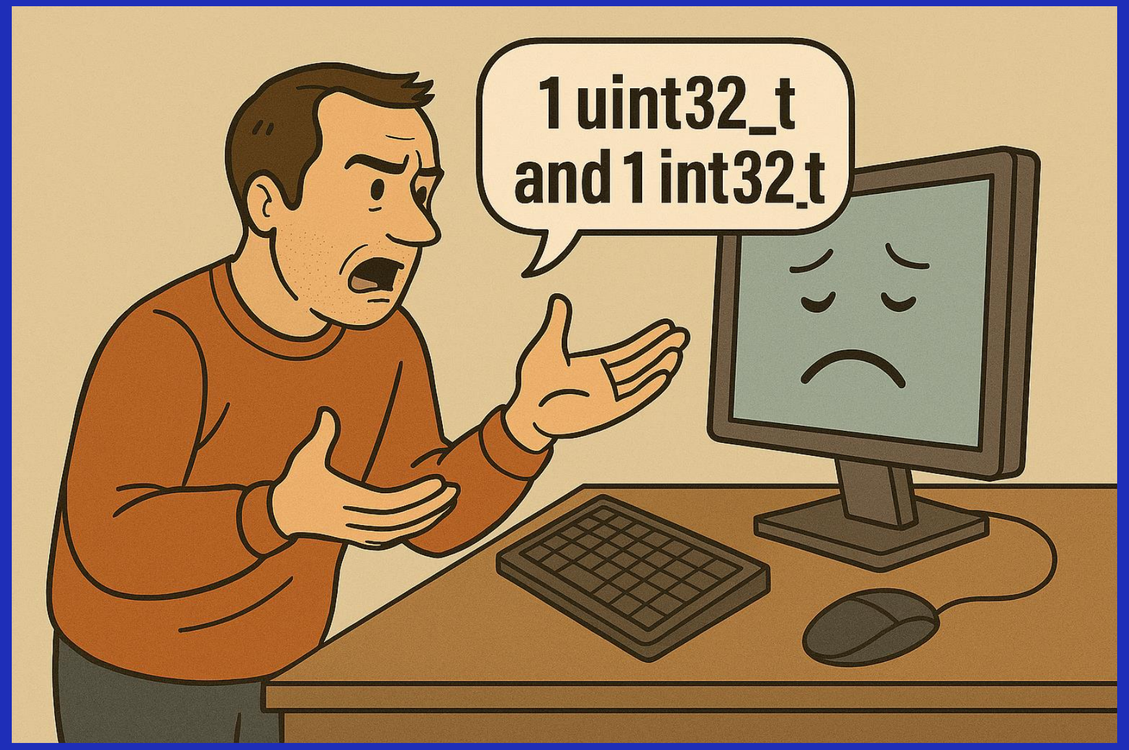
chris.gearing@mobileye.com

Imagine life without types

Images created by Microsoft Copilot

Often with C++, we don’t provide type information





Images created by Microsoft Copilot

What are of Types?

- MHz, KHz, Hz
- Km/h
- Kg, grams
- metres
- Celsius
- Etc.

- Frame IDs
- Bus Address – CAN, Flexray
- Hardware Indexes
- Version Numbers
- Baud Rate
- PDU IDs
- Anything and everything…

Firstly… I’m not the first

- There are many excellent type libraries
  - Boost
  - Nholthaus
  - bernedom/SI
  - mp-units
  - Au
- But for me, they fail the
  - *Keep It Stupidly Simple Stupid* test

Images created by Microsoft Copilot

TTL_strong_type.h Simple Class

- In the TTL Library C++ code is a file called TTL_strong_type.h
  - https://github.com/KhronosGroup/OpenCL-TTL

- You use it like this.

```
enum class TTL_MyType : uint32_t;     // This just needs to be a unique.
using MyType = TTL_StrongType<uint32_t, TTL_MyType>;
```

- Now *MyType* behaves just like a uint32_t
  - Produces exactly the same object code as a uint32_t
  - But possible operations are type consistent.

How simple

- Essentially, it is the underlying type with limited operations

```
template <typename T, typename UNIQUE_ENUM_CLASS>
struct TTL_StrongType {
    /* Construction from a fundamental value. */
    constexpr TTL_StrongType(T value) : value(value) {}

private:
    /* The actual fundamental value. */
    T value;

    /* The unique part */
    static constexpr UNIQUE_ENUM_CLASS unique = UNIQUE_ENUM_CLASS(0);
}
```

What is type consistent

- From the comments in the file

```
/**
 * It is acceptable to add 2 things of the same type. The rules
 * of the underlying value are used for the addition.
 *
 * 2 kmh + 2 kmh = 4 kmh
 */
constexpr TTL_StrongType operator+(TTL_StrongType const &rhs) const {
    return TTL_StrongType(value + rhs.value);
}
```

What is type consistent

- From the comments in the file

```
/**
 * It is generally acceptable to divide by the type.
 *
 * 2 KHz / 2 KHz = 1
 * 1000KHz / 500 KHz = 2
 */
constexpr UNDERLYING operator/(TTL_StrongType rhs) const {
    return value / rhs.value;
}
```

What is type consistent

- From the comments not in the file

```
/**
 * It is generally not acceptable to multiply something by itself
 * and therefor this operator does not exist in the code.
 *
 * 2 KHz * 2 KHz = KHz^2
 */
```

```
/**
 * It is generally not acceptable to add to another type
 *
 * 2 KHz + 2 Km/h = Gives something that is really improbable
 */
```

Looking at some code

```
KHz  CalcRealFreqKHz(uint32_t divider,  KHz  host_freq_khz,
                     KHz  target_freq_khz) {
    return(host_freq_khz / target_freq_khz) / divider;
}
```

```
error: could not convert '(host_freq_khz.StrongType<unsigned int,
StrongTypeUniqueID::KHz>:: operator/(target_freq_khz) / divider)'
from 'unsigned int' to 'KHz' {aka 'StrongType<unsigned int,
StrongTypeUniqueID::KHz>’}

|   return (host_freq_khz / target_freq_khz) / divider;
|   ~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~
                         |
                    unsigned int
```

Why do we get this error

```
KHz    CalcRealFreqKHz(uint32_t divider,    KHz    host_freq_khz,
                       KHz    target_freq_khz) {
    return(host_freq_khz / target_freq_khz) / divider;
}
```

```
KHz    CalcRealFreqKHza(scalar,              KHz,
                        KHz){
    return (KHz / KHz) / scalar;
}
```

Because KHz != KHz/KHz/scalar, and the compiler has enough information to know that

Compiler Detection Of Impossible Code

```
T1 CalcRealFreqKHz(T2 v1, T3 v2, T4 v3, T5 v4) {
    return (v1 op1 v2) op2 (v3 op3 v4);
}
```

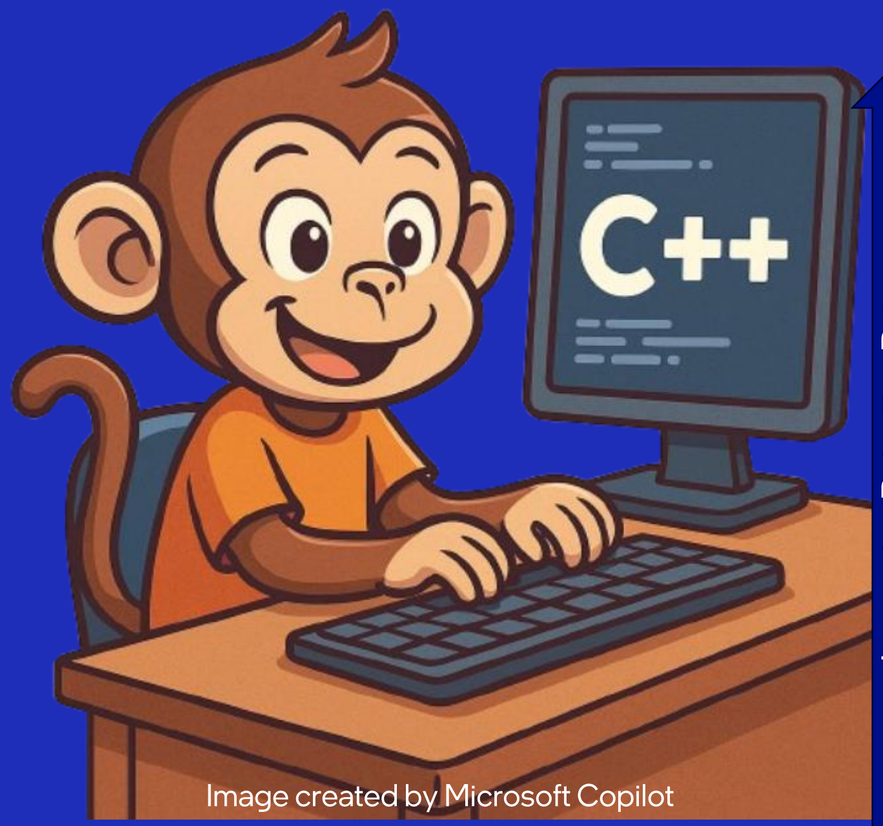
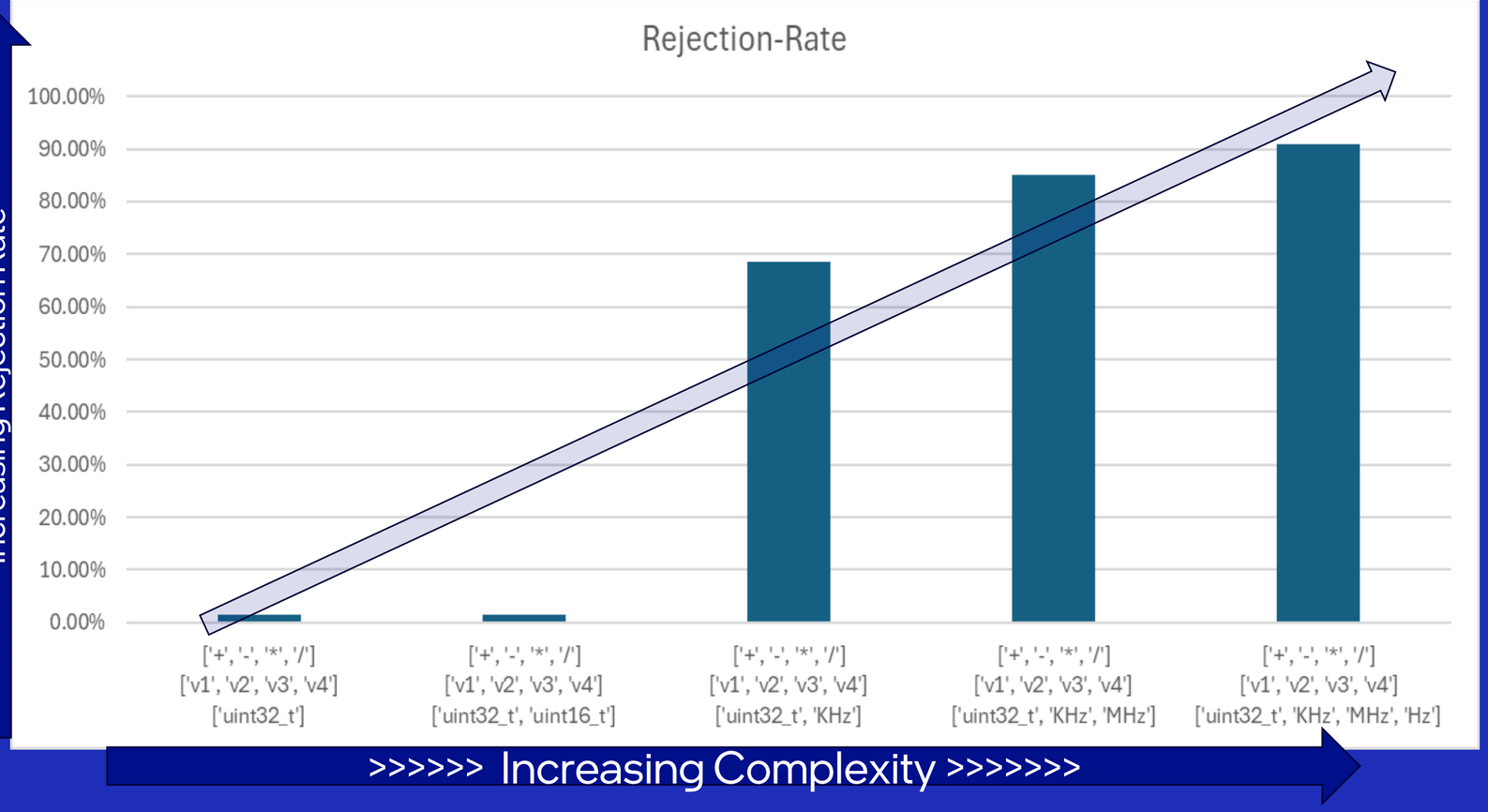


Image created by Microsoft Copilot

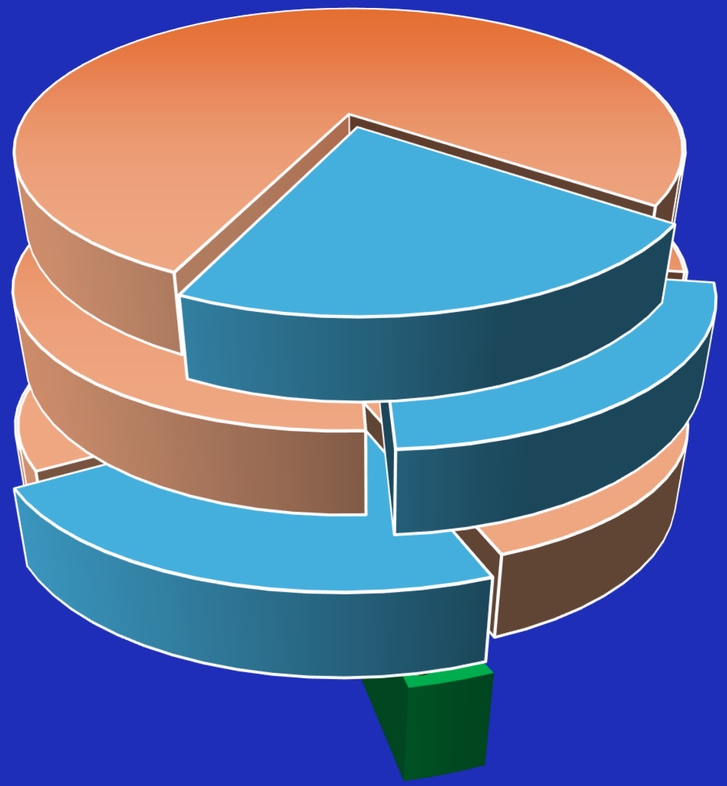


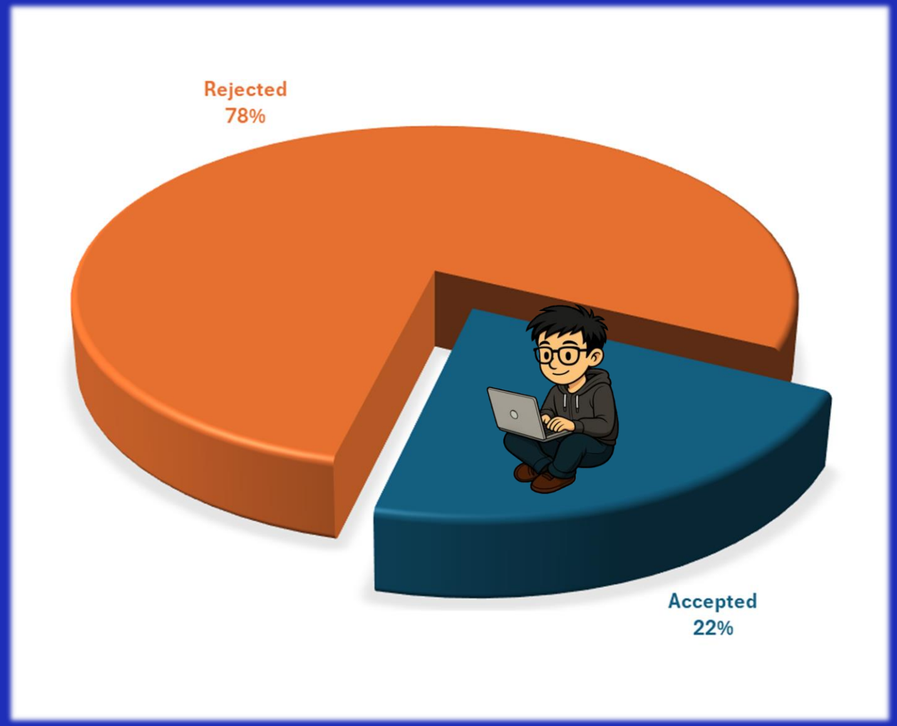
https://www.online-python.com/aKxHNRd2UF

Compiler Detection Of Impossible Code

- Use types and the developer's ability to write good code is increased 4-fold

- Types
- Static Analysis
- Unit Tests
- Code Review
- Sanitizers
- ....





Whole Systems

- The more types you can include in your system, the more robust it will become
- TTL_StrongType is great for scalar values, which most are
- Typed primitives produce a great basis to build more complex types upon
- The system becomes more type-safe

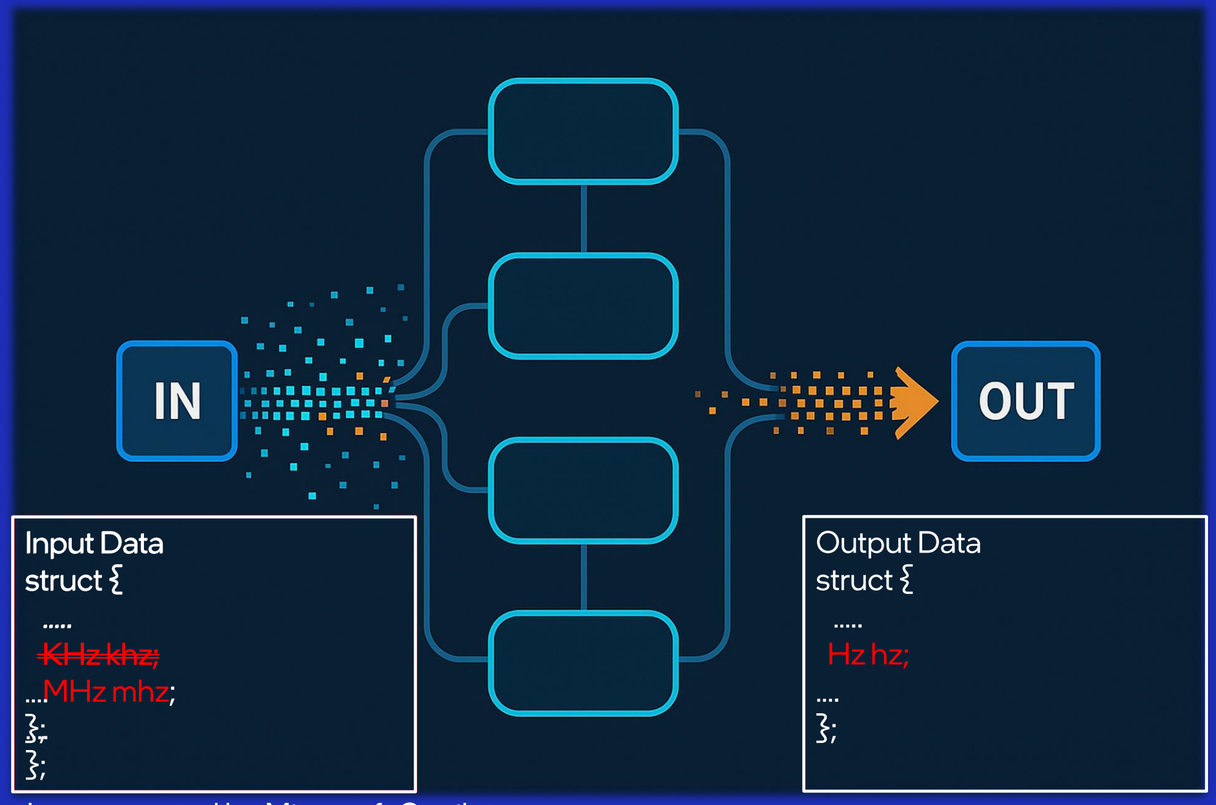


Image created by Microsoft Copilot

One Caveat - Do Not Over Do Typing

- I have seen suggestions that typing should be used to try to eradicate even more errors.

```
RealFrequencyKHz CalcRealFreqKHz(uint32_t divider,
                                 HostFrequencyKHz host_freq_khz,
                                 TargetFrequencyKHz target_freq_khz);
```

- Whilst this might make it even harder to break the code, it will make it much harder to change the code, because you are adding type and usage information.

- Use natural types that are the type of the data, not the usage of the data.

What Next

- Try adding some simple typing to your code
  - Go to https://github.com/KhronosGroup/OpenCL-TTL
  - And try it, a simple example will take 10 minutes
    - https://godbolt.org/z/vjaEx89ea
- Once you try it, look at the other typing libraries
  - Email me if you have suggestions or observations
  - The TTL example is just one way
- Strong Typing makes coding better by any measure

Thankyou
+
Any Questions?

chris.gearing@mobileye.com

Disclaimer