Project: Programming Language C++, Library Evolution Working Group
Document number: P0298R0
Date: 2016-05-27
Reply-to: Neil MacIntosh neilmac@microsoft.com

A byte type definition

Contents

Changelog

Introduction

The core of this proposal is to introduce a distinct type implementing the concept of *byte* as specified in the C++ language definition. The proposal suggests a very simple definition of that type, named `std::byte`. It is argued that C++17 is expressive enough for a simple library definition, as opposed to a keyword (however uglified). An additional small fix is suggested to allow `std::byte` to embody an aliasing property for access to arbitrary object representations. Taken altogether, `std::byte` is just as "builtin" as if it was designated by dedicated keyword (which would have to be ugly or incur source breaking changes.); an illustration of C++'s expressive power.

Motivation and Scope

Many programs require byte-oriented access to memory. Today, such programs must use either the `char`, `signed char`, or `unsigned char` types for this purpose. However, these types perform a "triple duty". Not only are they used for byte addressing, but also as arithmetic types, and as character types. This multiplicity of roles opens the door for programmer error – such as accidentally performing arithmetic on memory that should be treated as a byte value – and confusion for both programmers and tools.

Having a distinct *byte* type improves type-safety, by distinguishing byte-oriented access to memory from accessing memory as a character or integral value. It improves readability. Having the type would also make the intent of code clearer to readers (as well as tooling for understanding and transforming programs). It increases type-safety by removing ambiguities in expression of programmer's intent, thereby increasing the accuracy of analysis tools.

A separate paper – P0257 – describes wording changes to the Standard that would allow a `std::byte` type to be used as a means for accessing object storage, analogous to `unsigned char`.

Design Decisions

std::byte is not an integer and not a character

The key motivation here is to make *byte* a distinct type – to improve program safety by leveraging the type system. This leads to the design that `std::byte` is not an integer type, nor a character type. It is a distinct type for accessing the bits that ultimately make up object storage.

As its underlying type is `unsigned char`, to facilitate bit twiddling operations, convenience conversion operations are provided for mapping a byte to an unsigned integer type value. They are provided through the function template:

```
namespace std {

template <class IntegerType> // constrained appropriately
  IntegerType to_integer(byte b);

}
```

This supports code that, for example, wants to produce a integer hash of an object to take a sequence of `std::byte`, convert them to integers and then perform arithmetic operations on them as needed.

Conversely, conversions in the other direction, e.g. from an integer value to `std::byte` type is now made simpler, and safer with the relaxation of enum value construction rule in C++17. Now, you can just write `std::byte{i}` when `i` is a non-negative integer value no more than `std::numeric_limits<unsigned char>::max()`.

During a previous review by CWG at the Spring 2015 meeting in Jacksonville, FL, John Spicer suggested an alternative definition that would leave much to "implementation defined" to avoid dependency on `<cstddef>`. That dependency is not a substantial issue in practice, for any interesting program. So, we do not recommend that path as it would add complexity to the wording and definition of `std::byte`.

std::byte is storage of bits

A *byte* is a collection of bits, as described in 1.7/1. `std::byte` would provide operations that allow manipulation of the bits that it contains. The definition suggested below is kept deliberately simple with a minimum interface.

As illustration, these operations would be declared as follows:

```
namespace std {

// IntType would be constrained to be true for is_integral_t<T>
template <class IntType>
  constexpr byte operator<<(byte b, IntType shift);
template <class IntType>
  constexpr byte operator>>(byte b, IntType shift);

constexpr byte operator|(byte l, byte r);
constexpr byte operator&(byte l, byte r);
constexpr byte operator~(byte b);
constexpr byte operator^(byte l, byte r);

}
```

Their semantics would be the simple ones you might expect for a non-numeric type that is a collection of bits. Right-shift will shift bits rightwards, filling trailing places with zero bits. Left-shift will shift bits leftwards, filling trailing places with zero bits. These semantics are different from the equivalent operations on `char` or `unsigned char`, because `std::byte` is not a `char` or `unsigned char` (even though that may be its underlying storage).

Similarly, `std::byte` can be compared, as comparing and ordering instances is a sensible and useful operation. Given its underlying storage type, the comparison operators would give the same results as if performed on the underlying type.

Because `std::byte` uses `unsigned char` as its underlying type, initialization from an integer literal is possible, which is a convenience feature.

```
// example of initializing a byte
```

```
byte b = { 0x01 };
```

Implementation Alternatives

Two possible definitions of `std::byte` are presented here:

```
// Alternative A:
namespace std {
    enum class byte : unsigned char {};
}
```

and

```
// Alternatve B:
namespace std {
    using byte = /* implementation defined */;
}
```

We find alternative A easier to present and explain to ordinary programmers. For all practical purposes, it has the right semantics. That definition needs no more complication that needs to be hidden behind implementation-defined weasel words.

Proposed Wording Changes

The following proposed changes are relative to N4567 [1]. Additions are highlighted here in green and deletions in red (the deletions are merely for grammatical purposes).

Alternative A

18.2 Types [support.types]

Table 30 – Header <cstddef> synopsis

Type	Name(s)
Macros:	NULL offsetof
Types:	ptrdiff_t size_t max_align_t nullptr_t byte

10 The definition of `byte` is:

```
namespace std {
    enum class byte : unsigned char {};
}
```

Alternative B

18.2 Types [support.types]

Table 30 – Header <cstddef> synopsis

Type	Name(s)
Macros:	NULL offsetof
Types:	ptrdiff_t size_t max_align_t nullptr_t byte

10 The type `byte` is an implementation-defined distinct type with the same size, signedness and alignment as `unsigned char`, called the underlying type. It supports all the same behavior as a scoped enumeration type that has an underlying type of `unsigned char` and that has no enumerators defined.

Acknowledgements

Gabriel Dos Reis originally suggested the definition of byte as a library type by using a scoped enumeration and also provided valuable review of this proposal.

References

[1] Richard Smith, "Working Draft: Standard For Programming Language C++", N4567, 2015, [Online], Available: http://open-std.org/JTC1/SC22/WG21/docs/papers/2015/n4567.pdf

[2] Neil MacIntosh, "A byte type for increased type safety", P0257, 2016, [Online].