Omitting empty constructor bodies

I. Introduction

This is a proposal to allow to avoid writing constructor bodies in constructor definition if the body is empty.

II. Motivation And Scope

The motivation of this proposal is to simplify the object constructor definition whenever all the construction work is already done before the constructor body scope starts and hence the constructor body is empty.

III. Examples

There's some simple objects with few members with all of them initialized in constructor member initializer list, and therefore there is no work left to do in constructors body:

```
template <typename T>
struct point3d {
    T x{}, y{}, z{};
    point3d() = default;
    point3d(T xx, T yy, T zz) : x{xx}, y{yy}, z{zz} {}
    //                  empty constructor body ----> ^^
};

struct ip {
    std::uint32_t address{0x7f000001};

    ip(std::uint32_t A, std::uint32_t B,
       std::uint32_t C, std::uint32_t D) :
        address{D + (C << 8u) + (B << 16u) + (A << 24u)}
    {}
//  ^^ <---- empty constructor body
    ip(std::uint32_t value) :
        address{value}
    {}
//  ^^ <---- empty constructor body
};
```

Even if the constructor body is empty, it is required. This paper proposes to avoid the constructor body if it is empty:

```
template <typename T>
struct point3d {
    T x{}, y{}, z{};
    point3d() = default;
    point3d(T xx, T yy, T zz) : x{xx}, y{yy}, z{zz};
    //                   no constructor body ----> ^
};

struct ip {
    std::uint32_t address{0x7f000001};
    ip() = default;
    ip(std::uint32_t A, std::uint32_t B,
       std::uint32_t C, std::uint32_t D) :
            address{D + (C << 8u) + (B << 16u) + (A << 24u)};
//                                  no constructor body ----> ^
    ip(std::uint32_t value) :
        address{value};
//                     ^ <---- no constructor body
};
```

An alternative to this new syntax sugar could be to also avoid the semicolon ';' at the end of constructor definition:

```
template <typename T>
struct point3d {
    T x{}, y{}, z{};
    point3d() = default;
    point3d(T xx, T yy, T zz) : x{xx}, y{yy}, z{zz}
//    no constructor body ----> ^^^^^^^^^^^^^^^^^^^
};

struct ip {
    std::uint32_t address{0x7f000001};
    ip() = default;
    ip(std::uint32_t A, std::uint32_t B,
       std::uint32_t C, std::uint32_t D) :
        address{D + (C << 8u) + (B << 16u) + (A << 24u)}
//      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//      no constructor body
    ip(std::uint32_t value) :
        address{value}
//      ^^^^^^^^^^^^^ <---- no constructor body
};
```

But this could lead to confusing code, which is the opposite of the goal of this proposal. The constructor body omission shall be used only in -obviously- definition, and wouldn't have any effect in declaration:

```
template <typename T>
struct point3d {
    T x{}, y{}, z{};
    point3d(); // Ctor declaration.
    point3d(T xx, T yy, T zz); // Ctor declaration
};

point3d::point3d(); // Ctor definition
//                ^ <---- no constructor body
point3d::point3d() : x{xx}, y{yy}, z{zz}; // Ctor definition
//            no constructor body ----> ^

struct ip {
    std::uint32_t address{0x7f000001};
    ip(); // Ctor declaration
    ip(std::uint32_t A, std::uint32_t B,
       std::uint32_t C, std::uint32_t D); // Ctor declaration
    ip(std::uint32_t value); // Ctor declaration
};

ip::ip(); // Ctor definition
//      ^ <---- no constructor body
ip::ip() : // Ctor definition
    address{D + (C << 8u) + (B << 16u) + (A << 24u)};
//                         no constructor body ----> ^
ip::ip() : address{value}; // Ctor definition
//                        ^ <---- no constructor body
```

IV. Impact On The Standard

This proposal shouldn't break any existing code.

V. Standardese

Add the following paragraph to §12.1.1.3:

(1.3) — in a declaration at namespace scope or in a friend declaration, the *id-expression* is a *qualified-id* that names a constructor (3.4.3.1).

The *class-name* shall not be a *typedef-name*. In a constructor declaration, each *decl-specifier* in the optional *decl-specifier-seq* shall be **`friend`**, **`inline`**, **`explicit`**, or **`constexpr`**. [*Example:*

```
struct S {
  S();        // declares the constructor
};
S::S() { }    // defines the constructor
```

—end example] the body of the constructor can be omitted in definition if it is empty [*Example:*

```
struct S {
  S();        // declares the constructor
};
S::S();       // defines the constructor, body is empty
```

—end example]