Inheriting Base-Class Constructors

The term *inheriting constructors* refers to the use of a **`using declaration`** to expose nearly all of the constructors of a base class in the scope of a derived class.

Description

In a class definition, a **`using declaration`** naming a base class's constructor results in the derived class "inheriting" all of the nominated base class's constructors, except for *copy* and *move* constructors. Just like **`using`** declarations of member functions, the nominated base class's constructors will be considered when no matching constructor is found in the derived class. When a base class constructor is selected in this way, that constructor will be used to construct the base class, and the remaining bases and data members of the subclass will be initialized as if by the default constructor (e.g., applying default initializers; see Section 2.1."Default Member Init" on page 318).

```
struct B0
{
    B0() = default;      // public, default constructor
    B0(int)         { }  // public, one argument (implicit) value constructor
    B0(int, int)    { }  // public, two argument value constructor

private:
    B0(const char*) { }  // private, one argument (implicit) value constructor
};

struct D0 : B0
{
    using B0::B0;  // using declaration
    D0(double d);  // suppress implicit default constructor
};

D0 t(1);     // OK, inherited from B0::B0(int)
D0 u(2, 3);  // OK, inherited from B0::B0(int, int)
D0 v("hi");  // Error, Base constructor is declared private.
```

The only constructors that are explicitly *not* inheritable by the derived class are the potentially compiler-generated *copy* and *move* constructors:

```
#include <utility>  // std::move

B0 b1(1);              // OK, base-class object can be created.
B0 b2(2, 3);           // OK, base-class object can be created.
B0 b3(b1);             // OK, base-class object can be copied (from lvalue).
B0 b4(std::move(b1));  // OK, base-class object can be moved (from rvalue).
```