

Forwarding and inherited constructors

Michel Michaud and Michael Wong

Cégep Saint-Jean-sur-Richelieu and IBM

mm@gdzid.com

michaelw@ca.ibm.com

Document number: N1898=05-0158

Date: 2005-10-06

Project: Programming Language C++, Evolution Working Group

Reply-to: Michael Wong (michaelw@ca.ibm.com)

Revision: 2

Abstract

This paper presents a design for forwarding constructors and for inherited constructors.

The discussion is based on the earlier papers, especially the paper “Initialization and initializers” by Bjarne Stroustrup and Gabriel Dos Reis” [N1890] and on discussions in the Evolution Working Group. Proposals for forwarding constructors appeared in **Delegating Constructors** [N1445, 1581,1618] and **Inheriting Constructors** [N1583].

1 Forwarding Constructors

As we add many constructors to a class, the chance that two constructors do something very similar increases significantly. One example from [N1581] is:

```
class X {  
    void CommonInit();  
    Y y_;  
    Z z_;  
public:  
    X();  
    X( int );  
    X( W );  
};  
X::X() : y_(42), z_(3.14) { CommonInit(); }  
X::X( int i ) : y_(i), z_(3.14) { CommonInit(); }  
X::X( W e ) : y_(53), z_( e ) { CommonInit(); }
```

Saying exactly the same thing many times is sloppy and a maintenance hazard. This particular example is not too bad in practice, but in general we need something better. The proposal for forwarding constructors [N1581] comes to our rescue:

```
class X {
    X( int, W& );
    Y y_;
    Z z_;
public:
    X();
    X( int );
    X( W& );
};
X::X( int i, W& e ) : y_(i), z_(e) { /*Common Init*/ }
X::X() : X( 42, 3.14 ) { SomePostInitialization(); }
X::X( int i ) : X( i, 3.14 ) { OtherPostInitialization(); }
X::X( W& w ) : X( 53, w ) { /* no post-init */ }
```

For a double forwarding situation, we can use an example along the lines of class X above, but also including handling of exceptions:

```
X::X(U& u) try: X(W(u)) { /* */}
catch (...) { /* would catch all exceptions from called constructors */}
```

This proposed feature fits in well with the language.

2 Inherited constructors

One of the most frequently requested convenience features is “let me inherit the constructors from my base class. Except for a quirk of naming, we already have that! Consider:

```
class Base {
public:
    Base(int);
    Base();
    Base(double);

    void f(int);
    void f ();
    void f (double);

    // ...
};
```

```
class Derived : public Base {
public:
    using Base::f;           // lift Base's f into Derived's scope
    void f(char);           // provide a new f
    void f(int);            // prefer this f to Base::f(int);

    using Base::Base;        // proposed syntax to lift Base constructors
                             // into Derived's scope
    Derived(char);           // provide a new constructor
    Derived(int);            // prefer this constructor to Base::Base(int);

    // ...
};
```

Little more than a historical accident prevents using this to work for a constructor as well as for an ordinary member function. Had a constructor been called “ctor” or “constructor” rather than being referred to by the name of their class, this would have worked. We propose this as the mechanism for inheriting constructors.

3 Acknowledgements

Obviously, much of this initializer list and constructor design came from earlier papers and discussions.