Doc No: SC22/WG21/N1494
J16/03-0077
Date: 09/09/2003
Project: JTC1.22.32
Reply to: Daniel F. Gutson
danielgutson@hotmail.com

Pure implementation method declaration

1. The Problem

When declaring a method intended to implement a specific abstract method in a base class, there is no way to warrant the connection other than the name and signature matching.
If, by maintenance, mistake, or lack of documentation, someone changes the method in the base/abstract class, the binding is lost and the code will still compile while no notification will be issued at all.
Additionally, there is no language-standard way of evaluating the impact of changing an interface or base class method (i.e., all the ´disconnections´ it would cause to the derived classes).
However, it is quite common to make changes to base classes, and also -in ´historical´ terms- to start making a class, then generalizing it with a base class, and then modifying the base class.

Finally, there is no standard way to expliciting in the code the programmer's intention for specifying when a method is a ´pure´ implementation, expecting that the method is defined in a base class.

The consequences of not addressing this problem are:
- need for additional documentation
- static analysis tool for evaluating the impact of changing a base class, or a time demanding analysis looking for all the derived classes where the method is implemented
- the possibility of inserting a silent change of undesirable behavior

The categories where this proposal fits are:
* improve support for systems programming
* improve support for library building
* make C++ easier to teach and learn (see below)

2. The Proposal

This paper proposes to add the implementation's counterpart of the abstract method declaration:

```
virtual ... > 0;
virtual ... >= 0;
```

When a method is declared as

```
virtual method(....) > 0;
```

means that
- the method will be implemented in this class
- the method MUST be declared in some base class.

Methods declared in this way could be named ´pure implementation methods´.

When a method is declared as

```
virtual method(....) >= 0;
```

means that the method is semantically a combination of the

```
      virtual ... = 0
and
      virtual ... > 0
declarations, viz:
- the method can be not implemented in this class
- the class cannot be instantiated
- the method MUST be declared in some base class.

These declarations neither change run time behavior nor code
generation, but compile-time checking only.
A virtual ... > 0 methods behaves exactly the same way as a
virtual ... method, plus the additional checking mentioned above.
Similarly, a virtual ... >= 0 behaves as a virtual ... = 0 method, plus
the checking of existence in a base class.

When the checking rules are not accomplished, a compiler error shall
be generated.

A method declared as virtual = 0 or virtual only in a base class,
can be declared either as virtual >0 or virtual >=0 in a derived class.
A method declared as virtual >= 0 in a (non-base) class, can be declared
as either virtual >= 0 or virtual >0 in a derived class.
A method declared as virtual >0 in a (non-base) class, can only be
declared as virtual >0 in a more derived class.
```

2.1 Basic Cases

```
struct Base
{
      virtual void f() = 0;
};

class Der : public Base
{
      virtual void f() > 0
      { /* do something */ }

      virtual void g() > 0;   // error: g does not exist in Base
};
```

2.2 Advanced Cases

This proposal is applicable to virtual inheritance also.

```
struct Base
{
      virtual void f() = 0;
      virtual int g() const = 0;
};

class DerF : virtual public Base
{
      virtual void f() > 0
      { /*do something */ }
};
```

```
class DerG : virtual public Base
{
      virtual int g() const >= 0;
};

class BottomDer : public DerF, public DerG
{
      virtual int g() const > 0
      { /*return something*/ }
};
```

3. Interactions and Implementability

3.1 Interactions
Declarations forms proposed in this paper may interact with CV-qualifiers as well, being a semantic orthogonal feature. Being a new grammatical form, no existing code is broken and backward compatibility is maintained.

This feature performs an idiomatic closure with the virtual =0 idiom, as far as the implementation counterpart is provided. It helps to learn C++ due to its declaration consistency at the language scope.

3.2 Implementability
This feature applies to the compiler-time phase only, and can be implemented by querying the compiler tables as an additional checking.