Doc No. X3J16-92-118 WG21-N0195

Clarifications on Relaxation of Return Types of Virtuals.

John Bruns
11/02/92

The original decision extended the language to allow virtual functions to differ in return types. The intent was to provide support for polymorphic behavior, that is when the new return type was indeed a pointer to the same object, but eliminate the loss of static type information.

The restricitions on this relaxation were:

1) The return types must be either pointers to objects or references to objects.

2) The original return type must be an accessible base class (pointer or reference) of the original return type.

This was the intent of the proposal and the wording was left to the editor to include in the draft document with the best possible wording.

Since then a number of questions have come up considering potential extensions to this rule.

1) Private inheritance.

Consider the case:

```
class A {
public:
        virtual A * f();
}

class B : private A {
public:
        virtual B * f();    // should this be legal?
}
```

The issue is that a B * cannot be automatically converted to an A * when the inheritance is private except within the members of B. This conversion needs to take place whenever we call f() on a B (or derived subclass of B) object in an the context of an A * or A &. In this case the class B is responsible for the redefinition and A is an accessible base class, therefore the cast from B * to A * is legal.

Yes

Now consider the classes above together with:

```
class C {
public:
        virtual A * g();
}

class D : public C {
public:
        virtual B * f();    // should this be legal?
}
```

In this case class D is the one attempting to redefine (cast) the

return type for classes related by private inheritance. Unlike the case above, D cannot legally cast a B * into an A *. This should be caught and flagged as a protection error.

2) Const (and Volatile) returns:

```
class A {
public:
        virtual const A * f();
}

class B : public A {
public:
        virtual A * f();    // should this be legal?
}
```

In this case it is trivial to convert the A * (or a B *) into an const A * (or volatile A *). This should obviously be allowed. The reverse, redefining a unrestrained pointer into a const pointer is not type safe and illegal.

3) Const pointers.

Consider the case:

```
class A {
public:
        virtual A * const f();
}

class B : public A {
public:
        virtual B * f();    // should this be legal?
}

class C : public B {
public:
        virtual c * const f();  // what about this??
}
```

In general (X * const p) tells the compiler that the pointer itself cannot be changed but X itself can be, as opposed to (const X * p) where X is constant but p could be changed. Since this is irrelevant in the return type of a function (which itself is never an L-Value), all the above are legal but useless.

Unless I'm missing something, aren't all function returns implicitly const in this sense??

4) Override Derived (not just Derived * or Derived &).

This was discussed in detail at the London meeting and was rejected. It was considered both technically more difficult to implement and semantically unsafe. In particular, it would cause truncation of types when used in a base class context. It was considered that this caused enough trouble with the assignment operator and we shouldn't open another loophole.

5) User defined conversions.

This was also discussed and rejected. It was thought that this was both more difficult and opened up the possiblity of strange and unexpected behavior. The intent of the original proposal was to support

polymorphism and rectify a shortcoming in the type system, not to support new language features.

~ONCLUSION

The original accepted proposal to allow redifinition of return types of virtual functions already covers the cases of private inheritance and const or volatile retuurn types. The issues of overiding actual derived objects and user defined conversions was rejected by the working group and should not be re-opened without significant new evidence supporting them.