Document Numbers: X3J16/95-0044
WG21/N0644
Date: January 31, 1995
Reply To: Bill Gibbons
bgibbons@taligent.com

Pointer to Member Cast Issues

Introduction

There are two related pointer to member cast issues which have not yet been resolved.

The first is casting across virtual inheritance; that is, casting a pointer to member such that the class of the pointer to member is virtual base class, or virtually derived from, the previous class. The current working paper requires this to work, and some compilers support it. But it's potentially expensive, and it makes pointers to members larger than they would otherwise need to be.

The second is casting to a base class of the class containing the original member. The ARM was unclear on whether this should work. The issue was discussed in 94-0101/N0488 and at the Kitchener meeting. Although there was considerable support for allowing this cast, it was clear that the issue had not been discussed enough for a vote. Since the existing wording was unclear, the committee voted to clarify the wording in a way that did not support the proposed casts, pending further discussion.

Virtual Base Classes

In the following program:

```
struct B { void f(); };

struct D : virtual B { };

void g(D *d) {
      void (D::*pf)() = &B::f;  // implicit downcast
      (d->*pf)();               // call
}
```

The call through the pointer to member must, using only information stored in the pointer to member itself, cast the pointer "d" from "D*" to "B*", in order to provide the correct "this" parameter for B::f. Of course the relative position of B to D isn't fixed, since a class derived from D may place B at a different offset.

So pointers to members must contain enough information to perform this upcast; this makes them larger than would otherwise be necessary. And when a pointer to member is dereferenced, the code must check whether to do the upcast (it isn't always needed), and perform the upcast if needed.

The check and the upcast can sometimes be optimized out if there are no virtual base classes. The memory for the description of how to upcast cannot be optimized out, since it's possible to declare a pointer to member using a incomplete class.

The problem is even worse with indirect virtual inheritance:

```
struct V { int a; };

struct X : virtual V { };

struct Y : virtual X { };

int X::*pxa = &V::x;  // implicit downcast

int Y::*pya = pxa;    // implicit downcast
```

The “pxa” pointer to member contains a description of how to cast from X* to V*. The second implicit downcast must somehow modify this description so that it works for casting from Y* to V*.

This is a potentially expensive operation.

There are two common implementations of virtual base classes. In the design used by cfront, derived classes contain pointers to their virtual base classes. So the description of how to cast to a virtual base must be the offset to the virtual base pointer within the derived object.

The second downcast must somehow map the offset to the V locator within an X object to the offset to a V locator within a Y object. There is no simple relationship between these two offset values. All the compiler knows is that the offset value in pxa is one of the virtual base offsets in X. For each of these offset values, there is a corresponding offset value in Y. But there is no simple mapping.

One implementation actually generates a helper function containing a switch statement which switches on the old offset, with a case for each possible value, and returns the corresponding new value. All this to implement what appears to be a simple static cast.

In the other common design, each object (of a class with virtual bases) contains a pointer to a table of offsets to the virtual base classes. The content of these tables depends on the actual relative positions of the virtual base classes in the complete object. So like virtual function tables, they must be initialized by the complete class and their contents are not known to the class which uses them.

In this model, each pointer to member contains an ordinal within the virtual base class offset table for the member’s class. Once again these ordinals must be mapped when a pointer to member is cast across virtual inheritance; but since the ordinals are dense, a simple mapping table suffices.

With either implementation, there is extra space in the pointer to member and extra overhead for performing a simple static cast.

Is this feature important enough to justify the overhead? I suggest that it is not. I propose that pointer to member casts across virtual inheritance be ill-formed.

Upcasts Beyond the Member’s Class

Central to the concept of polymorphism in C++ is the ability to store a pointer to any of a set of concrete object types in a pointer to a common more abstract type, i.e. a base class. In the general case, this concept does not extend to pointer to members. But in an important specific case it does.

In many situations it is useful to construct a pair consisting of an object pointer and a pointer to member which can be dereferenced with that object. This pair represents a deferred action on the object, and so is a very convenient form for handling callbacks, remote procedure calls, container classes, etc.

There is a natural representation of a hierarchy of such pairs, namely the class hierarchy itself. It should be possible to cast the pair up to a base class, and dereference the pair in the context of the base class.

There is no problem casting the pointer half of the pair up the hierarchy to a base class.

But when we try to cast the pointer to member half of the pair, we hit some limitations.

```
struct A {  };

struct B : A { virtual void f(); };

struct C : A { virtual void g(); };

void h() {
      // base class pointer to derived object
      A *objptr = new C;

      // base class member pointer to derived member
      void (A::*memptr)() = (void (A::*)()) &C::g;

      // ... handled as a pair ...
      // ... eventually dereferenced as a pair ...
      (objptr->*memptr)();
}
```

Many compilers do not completely support upcasting a pointer to member beyond the class containing the original member, i.e. to a class that does not contain or inherit the original member. There is no conceptual problem with such a cast; after all, we cast the object pointer to a class in which the member did not exist either. Yet the member is still there; it simply isn't visible using the static type of the pointer.

Similarly, when a pointer to member is upcast beyond the class containing the original member, it still refers to the original member.   It should be possible to dereference the pointer to member with an object which contains the member.

When an object pointer is downcast, the author of the code must be certain that the object really is of the derived type. Similarly, when a pointer to member is upcast, the author of the code must be certain that any dereference of the pointer to member is done with an object of the derived type. In both cases the cast must be explicit because it is impossible to statically check its validity.

In the special case of upcasting a paired object pointer and pointer to member, it is easy to verify the program's validity because the pointer always refers to an object suitable for use with the pointer to member.

This is a very useful technique, largely because of its implicit safety. Although there are alternatives, they are not as clean and simple as this more obvious technique.

How difficult is it for compilers to support these upcasts? It's trivial - because they already support them. If a pointer to member is sufficiently general to handle multiple inheritance and virtual functions, it is already general enough to handle the proposed upcasts.

The only reason these upcasts fail with working compilers today is that some compilers optimize out one or both of these parts of a pointer to member dereference:

- Multiple inheritance offset adjustments
- Virtual function calls

If a pointer to member's class does not use multiple inheritance, some compilers assume there will be no offset adjustment. This is a very small optimization; it usually saves only one instruction.

If a pointer to member's class does not contain or inherit virtual functions, some compilers assume that a pointer to member dereference will never be a virtual function call. This is a noticeable optimization on machines which inline virtual function calls, but only for nonpolymorphic classes. On machines where virtual calls are done with calling thunks there is no optimization at all.

Even on compilers which do these optimizations, it's possible to get pointer to member upcasts to work by defeating the optimizations. For example, by adding a dummy non-primary base class with a dummy virtual function (to add multiple inheritance and virtual functions.)

These upcasts are useful. They are obvious. They are impossible to check for at compile time, and they mostly work in existing compilers. So they will be used, whether the standard supports them or not. Given that the cost is so small, it seems reasonable to require that these casts work.

Upcasting Across Virtual Inheritance

These two issues intersect when we consider upcasting a pointer to member beyond the class containing the original member, to a virtual base of that class.

This is a problem because dereferencing the pointer to member would require casting the object pointer down across virtual inheritance - which is a very expensive operation, as it requires a dynamic cast. It would be unreasonable to require an implicit dynamic cast in this context.

So there are two limitations on upcasting pointers to members:

- They must only be dereferenced with objects containing the original member.
- They may not be cast to a virtual base of the class containing the original member.

The second limitation would be subsumed by a rule which disallowed all pointer to member casts across virtual inheritance.

Proposals

<u>1. Disallow casting pointers to members across virtual inheritance</u>. Change 5.2.8/8

from:

... may be converted to ... where B is a base class of D ...

to:

... may be converted to ... where B is a nonvirtual base class of D ...

2. Allow upcasts of pointers to members beyond the original member's class. Change 5.2.8/8

from:

... and class B does not contain or inherit the original member, ...

to:

... and class B is neither a base nor derived class of the class containing the original member, or class B is a virtual base of the class containing the original member, ...

Add to 5.5:

The dynamic type of the object must contain the member to which the pointer refers.

3. Both of the above. The wording is a little simpler. Change 5.2.8/8:

from:

... may be converted to ... where B is a base class of D ...
... and class B does not contain or inherit the original member, ...

to:

... may be converted to ... where B is a nonvirtual base class of D ...
... and class B is neither a base nor derived class of the class containing the original member, ...

Add to 5.5:

The dynamic type of the object must contain the member to which the pointer refers.