Addendum to X3J16 / 92–0004 – WG21 / N0082

Extending C++ to allow restricted return types on virtual functions

John Bruns and Dmitry Lenkov
jwb@cup.portal.com dmitry@hpcldnl.hp.com
Jan 31, 1992

THE PROBLEMS WITH THE NAIVE WORK AROUND

One way to approach the problem is to do nothing in the class itself and let the user do any appropriate casting. This ignores the fact that we wish to encapsulate the ugliness and make the class as easy to use as possible. However, the real problem with this approach is that it doesn't work in all cases.

SIMPLE SINGLE INHERITANCE

```
struct A {
      A * clone() { return new A(*this) ;};
...};

struct B : public A {
      A * clone() { return new B(*this); };
... };
```

This works fine, to get a B* in a B context just cast

```
B b;
(B *) b.clone();    // result is a B *
```

Note that the user must know what is a legal cast. This is not a problem with clone but could be with a more complicated function. By definition, the compiler will allow the cast without checking on its validity. This could be tested at run time if the proposal for run time type identification and casting is accepted, although at the cost of more code complexity.

VIRTUAL INHERITANCE

```
ASSUME A from above.

struct B   : public virtual A{
      A * clone() { return new B(*this); };
...};

B b;
(B *) b.clone();  //  ILLEGAL to cast from virtual base
```

Virtual inheritance prevents you from casting from a base class to a derived class. So this also breaks the naive work around. If the proposal for run time type identification and casting is accepted, this would now become a legal statement.

MULTIPLE INHERITANCE

```
ASSUME A from above.

struct B {
      B * clone(){ return new B(*this); };
                  // This can't return A * and make any sense
...};

struct C : public A, public B {
      ??? * clone() { return new C(*this): };
            // HELP This is illegal as there is a name conflict
...};
```

In fact we could use the work around for generalized renaming and make classes AA and BB which each transfer to new virtual functions A_Clone and B_Clone which then transfer to a new C_Clone in C that returns a C *. This is similar to the work around mentioned in the main paper with the added ugliness of the two renaming classes. Furthermore, the name clone() is forever lost to succeeding classes. Since this is the result of a name conflict, run time type identification would not help in this case.

Without MI, it is possible to use brute force casting to get around the type restriction on return types of virtual functions. MI makes it impossible to do so without using complex stratgies.