Doc No: X3J16/92-0136
WG21/N0212
Date: December 24, 1992
Project: Programming Language C++
Ref Doc:
Reply to: Philippe Gautron (gautron@chorus.fr)

use const_cast workaround

Qualified Member Function Call

Philippe Gautron
UNIX System Laboratories Europe
c/o Chorus Systèmes, 6 avenue Gustave Eiffel
78180 Montigny Le Bretonneux, France

Member functions may be overloaded on the basis of their constness: the non-const function is applied on non-const objects while the const function is applied on const objects. But although it should be type-safe to directly apply the const function on non-const objects, the current WP doesn't introduce any syntax to support this. The current paper proposes a slight extension to the C++ function call syntax in order to support these semantics.

1 Constant Member Functions

C++ permits overloading a member function on the basis of its constness. For example:

```
class C {
  public:
    void f(void);          // (1)
    void f(void) const;    // (2)
};
```

The first version of `C::f` is called when `f` is applied on a non-const object, the second version when `f` is applied on a const object. For example:

```
void some_function (void) {
    C c1;
    const C c2;

    c1.f();    // calls version (1) of f
    c2.f();    // calls version (2) of f
}
```

Permitting the non-const version of `f` to be applied on a const object should be an obvious error. Conversely, applying the const version of `f` on a non-const object could be safely processed. The current definition of the WP doesn't support any syntax for that. This restriction can be considered as purely gratuitous. Indeed, consider the following code:

```
void some_function (void) {
  // two pointers to member functions of C with no arguments and returning void
    void (C::*pf1) (void);          // (1) pointer to a non-const function
    void (C::*pf2) (void ) const;   // (2) pointer to a const function

    C c1;
    const C c2;

    pf1 = &C::f;    // assigns version (1) of f to pf1
    pf2 = &C::f;    // assigns version (2) of f to pf2

    (c1.*pf1) ();    // applies version (1) of f on c1
    (c2.*pf2) ();    // applies version (2) of f on c2

    (c1.*pf2) ();    // applies version (2) of f on c1
    (c2.*pf1) ();    // error
}
```

The code above is legal (except for the error). We are able to apply the `const` version of `f` on a non-`const` object through the use of pointers to member functions.

2 The Proposal

Semantics similar to those allowed for pointers to member function calls can be directly supported for member function calls through a slight extension of the C++ function call syntax. The proposal is to qualify a function call by explicitly stating that a call to the `const` function is expected.

```
void some_function (void) {
    C c1;

    c1.f() const;    // applies version (2) of f on c1
}
```

If there is no declaration of `const` member function, a qualified member function call should be then an error. Conversely, it should not be an error to qualify a call applied to a `const` object (the same function is called in both cases).

The rationale for this syntax is to be quite homogeneous to the declarations of `const` member function and pointer to member function:

```
class C {
  public:
    void f(void) const;          // a constant member function
};

void (C::*pf) (void ) const;     // a pointer to a constant member function

C c;
c.f() const;                     // a call to a constant member function
```

3 Work Around

A `const` member function can be applied on a non-`const` object by casting away the invoked object. For example:

```
void some_function (void) {
    C c1;

    ((const C) c1).f();     // applies version (2) of f on c1
}
```

This syntax is not elegant and using a cast notation to overcome a language deficiency is not the neatest way to operate.

4 Volatile Member Function Call

I don't know what is exactly the status (nor what could be the meaning) of a volatile member function. But if such a declaration is (or becomes) legal, a similar extension should have to be applied. Conversely, if such a declaration is outlawed, the title of this paper itself should have to be changed to "Constant Member Function Call".

5 Conclusion

This extension could be provide at least for completeness of the language definition. Whether it is a pure extension or just an editorial change is a matter of opinion.