Document Number: WG14 N445/X3J11 95-046

C9X Revision Proposal
==========

Title: Classes in C - Part 2: Inheritance
Author: Robert Jervis
Author Affiliation: Sun Microsystems, Inc.
Postal Address: 2550 Garcia Ave., Mountain View, CA 94043 USA
E-mail Address: robert.jervis@eng.sun.com
Telephone Number: +1 415 3367964
Fax Number: +1 415 9640946
Sponsor: _____
Date: 1995-04-25
Proposal Category:
__ Editorial change/non-normative contribution
__ Correction
X New feature
__ Addition to obsolescent feature list
__ Addition to Future Directions
__ Other (please specify) _____
Area of Standard Affected:
__ Environment
X Language
__ Preprocessor
__ Library
__ Macro/typedef/tag name
__ Function
__ Header
__ Other (please specify) _____
Prior Art: C++ _____
Target Audience:

These features are useful to a wide range of programmers. The facilities help improve problems of complex data structures that share common components. A class hierarchy frequently replaces unions of structures in C. For example, expression tree nodes in a compiler are frequently described in C as a union of several different structures, each of which share some common prefix of members. Inheritance permits expressing this system of structures as a base class, containing the common members, and derived classes that represent each member of the union.

Inheritance can also be used to share code. A base class can define member functions which are then shared across all the classes derived from the base.

Related Documents (if any): C++ Draft Standard _____

Proposal Attached: X Yes __ No, but what's your interest?
Abstract:

This proposal includes the exact wording changes needed to add inheritance assuming the changes of Part I of CLasses in C have

been accepted for the C Standard. Inheritance in this proposal is a subset of the corresponding C++ feature and is intended to be upward-compatible with it.
Please advise me if I have accidentally introduced incompatibilities in this and the following proposals.

Proposal: _____

The wording changes are summarized in the following points.

* Classes can be declared to have a base class.

* Public members of a base class are available in the derived class as if they had been declared in the derived class.

* Only single inheritance is supported. That is, each derived class can have only one base class.

* Members in a derived class can redeclare the names of any member of its chain of base classes. In effect, the scope of a derived class hides like-named members in its base class.

* No keywords are needed for this proposal.

ISSUES:

These items are details of the proposal where the committee may wish to consider alternatives beside what is presented here.

* In section 6.3.1, scope-qualifiers are limited to class-name qualifiers. The global scope qualifier (e.g. ::foo) is not included in this proposal. This capability is not directly related to classes, although classes create more opportunities to need such qualifiers. The committee may wish to add this extension.

* In section 6.5.2.4, the location of the members of the base class within a derived class is unspecified. The committee may choose to specify that the base class starts at the same address as the derived class object. Nothing prevents an implementation from doing this, and it certainly makes converting pointers between base and derived classes easier. See also 6.2.2.3 for some details.

The following are the specific changes to the Standard. Section numbers are in reference to the International Standard ISO/IEC 9899:1990 Programming Languages - C. Where relevant, changes affecting the TC1 defect reports are stated where they appear in the Standard. The TC2 Defect Reports have not been scanned for possible changes, nor has the normative addendum.

6.1.2.1 Scopes of Identifiers

Page 20, BEFORE line 30, REPLACING THIS TEXT FROM PART 1:

... and resumes for the duration of each member function definition associated with the same class encountered later in the same

translation unit.

WITH:

... and resumes for the duration of each member function definition associated with the same class encountered later in the same translation unit. The scope of a base class also resumes for the duration of the class declaration list and each of the member function definitions of each class encountered later in the same translation unit that publicly inherits from the base.

Page 20, line 35-37 IS:

If an outer declaration of a lexically identical identifier exists in the same name space, it is hidden until the current scope terminates, after which it again becomes visible.

SHOULD BE:

If an outer declaration of a lexically identical identifier exists in the same name space (except within a primary-expression consisting of an identifier with a scope-qualifier that names the scope of the outer declaration), it is hidden until the current scope terminates, after which it again becomes visible.

RATIONALE:

This text extends the concept of class scope to include derived classes. The use of scope qualifiers allows member functions in a derived class to call otherwise hidden member functions in its base class. This idiom is common, since derived classes can then extend the functionality implemented in a base class rather than merely replace it.

6.2.2.3 Pointers

Page 36, line 36 IS:

... shall compare equal to the original pointer.

SHOULD BE:

... shall compare equal to the original pointer. A pointer to a derived class type may be converted to a pointer to any of the classes the derived class inherits and back again; the result shall compare equal to the original pointer. A pointer to a derived class type when converted to a pointer to any of the classes the derived class inherits shall point at the object of the base class type allocated within the original class. A pointer to the allocated object of a base class type when converted to a pointer to the derived class type of which the object is a part shall point at the derived class object.

RATIONALE:

Without making any guarantees about the exact layout of base classes within derived classes, we need to guarantee that converting the pointer correctly moves from one object to another.

This language would certainly be easier to write here if we were to guarantee that a base class object starts at the same address as all derived class objects. Of course, because of multiple inheritance and virtual base classes in C++, this kind of mandate is not really possible. So, rather than introduce a potential incompatibility with C++ I have specified the conversion in a way that still allows the pointer conversions to work properly.

6.3.1 Primary Expressions

Page 39, line 4 IS:

identifier

SHOULD BE:

scope-qualifier opt identifier

Page 39, line 7 AFTER:

(expression)

ADD:

scope-qualifier:
identifier ::

Page 39, line 8 BEFORE:

Semantics

ADD:

Constraints

The identifier in a scope-qualifier shall designate a visible class tag name. The scope-qualifier shall appear in the scope of the named class.

RATIONALE:

Scope qualifiers occur in derived classes in order to refer to members in a base class. A frequent idiom in classes is to redefine a member function in a derived class in order to expand on the code in the base class. For example:

```
class base {
        public:

        void    action(int x);
};
```

```
class derived : public base {
        public:

        void    action(double x);
};

derived::action(double x)
{
        printf("Action called with %g\n", x);
        base::action((int)x - 1);
}
```

In this example, the derived action function does a printf call and then calls the base class version of the action function.

Note that without the scope qualifier, the derived action function could not call the base action function at all. If the scope qualifier were left off the call, the call would be a recursive call to itself, which would quickly crash as the stack overflowed.

Note also that global scope qualifiers are not included. For example:

```
extern int i = 7;

void foo()
{
        int     i = 5;
        int     j;

        j = ::i;
        printf("j = %d\n", j);                  // prints j = 7
}
```

In this example, the global scope qualifier allows one to refer to global variables, even though they are normally hidden.

6.5.2.4 Class Specifiers

CHANGE THE Syntax FROM:

```
class-specifier:
        class identifier opt { class-declaration-list }
        class identifier
```

TO:

```
class-specifier:
        class identifier opt base-class-specifier opt
                                        { class-declaration-list }
        class identifier
```

ADD THE FOLLOWING SYNTAX:

```
base-class-specifier:
```

: visibility opt identifier

ADD TO Constraints:

The identifier in a base-class-specifier shall name a visible class tag name.

ADD TO Semantics:

A base-class-specifier names a base class for the new type being declared. A class type with a base class is said to be derived from its base and is a derived class.

A derived class publicly inherits from another class if the derived class

* has a base-class-specifier that contains the keyword public and

* either names the other class as its base or its base publicly inherits the other class.

A derived class inherits from another class if the derived class

* has a base-class-specifier that names the other class or

* the base inherits the other class.

IN Semantics AFTER:

Within a class object, the non-bit-field members and the units in which bit-fields reside have addresses that increase in the order in which they are declared.

ADD:

In addition, if the class has a base class, an object of the base class type is allocated within the derived class. All references to members of the base class refer to this allocated object.

RATIONALE:

Public base classes make their members visible to their derived classes and this visibility recursively extends to each successive derived class in turn.

Private base classes hide their members from derived classes.

This distinction is made to preserve compatibility with C++ where the default visibility of a base class is private.

6.5.7 Initialization

Page 72, line 20-21 IS:

> Otherwise, the initializer for an object that has aggregate type shall be a brace-enclosed list of initializers for the members of the aggregate, written in increasing subscript or member order; ...

SHOULD BE:

> Otherwise, the initializer for an object that has aggregate type shall be a brace-enclosed list of initializers for the members of the aggregate, written in increasing subscript or member order, where a derived class with a public base class initializes the members of the base class before the members of the derived class; ...

RATIONALE:

> This language is needed to specify how inherited classes are initialized.

•