Document Number: WG14 N446/X3J11 95-047

C9X Revision Proposal

Title: Classes in C - Part 3: Virtual Functions
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: 1995-08-19
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 extend the usefulness of class hierarchies by allowing code to be written that can be extended to include new classes without the calls to virtual functions being affected. Frequently, virtual functions are used in situations where switch statements appear in conventional C programs. Instead of dispersing the behavior of a class into many places in a program, virtual functions can be used to abstract class-specific behavior into them and keep class-independent code free of implementation details.

Virtual functions can also enhance sharing of code, since common non-vritual methods can be used in combination with virtual code to provide extensible libraries. Virtual functions can be defined in user code that then makes use of shared components in library code.

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.

* Member functions can be declared to be virtual.

* Calls to virtual functions use a hidden object identity to resolve the actual function called.

* A call to a virtual function using a pointer calls the function for the actual object pointed at, not the declared type of the pointer.

* The identity of a static or automatic object is assigned when the object is created.

* Operators new and delete have been added to provide a means for setting the hidden identity of dynamically allocated objects. Operator new allocates an object of a specified type. Operator delete is included for symmetry and to provide better compatibility with C++.

* Three keywords are needed for this proposal: delete, new and virtual.

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.

6.1.1 Keywords

ADD THE FOLLOWING KEYWORD TO THE LIST:

virtual

NEW SECTION:

6.1.2.7 Object Identity

An object whose identifier is declared to have a class type which either includes a virtual function or inherits such a class will be assigned a runtime identity, unique to its declared class type. This identity is assigned when storage for that object is reserved. If the object is a member or element of some aggregate type, its identity is assigned when

storage for the aggregate is reserved.

An object allocated by a new operator that names a class type which either includes a virtual function or inherits such a class will be assigned its runtime identity as a side effect of the operator.

The identity corresponding to a class is stored in such a way that among a group of classes deriving from a common base class that has an identity, each class in the group is distinguishable from the others.

Example:

```
class base {
        public:

        void virtual      action();
};

class   derived : public base {
        public:

        void virtual      action();
};

func()
{
        class base        *bp;

        bp = new derived;

        bp->action();             /* Calls derived.action */
}
```

In this example, func creates an object with derived type. This establishes an identity for that memory. THe address of the allocated memory is then assigned to a pointer to the base class (allowed by for inherited classes). Then, using the base pointer, the virtual function for class derived is called, not for base. If the virtual keywords were removed, the call would call the action method for class base.
In this example,

6.3.1 Primary Expressions

ADD TO Syntax, page 39, line 6, AFTER:

```
        string-literal
        ( expression )
```

ADD:

```
        new type-name
```

Page 39, line 17, AFTER:

... or a void expression.

ADD:

> The new operator allocates enough storage to hold an object of the named type. The value of the expression is a pointer to the allocated object and the type of the expression is pointer to the named type. If the named type or any member or element of the type needs an identity, the identity is assigned in the newly allocated storage.
>
> The value of the allocated storage, other than the assigned identities is indeterminate.

RATIONALE:

> Operator new provides a means to setting the virtual table pointers in allocated objects, since the expression explicitly states the type that will be used for the storage. Otherwise, the only way to initialize an allocated object with virtual functions would be to copy an existing and initialized object of the same type into the allocated storage.

6.3.2.5 Member Function Calls

Semantics

<<From the first part on classes>>, IS:

> A postfix expression followed by an arrow ->, an identifier and followed by parentheses () containing a possibly empty, comma-separated list of expressions is a member function call. The value of 'this' in the called member function shall be the value of the the first operand.

ADD:

> If the member function named is a virtual function, the actual function called is the same-named virtual function of the class whose identity is stored in the object pointed to by the first operand. If the object pointed to by the first operand does not have a properly assigned identity, the behavior is undefined.

RATIONALE:

> The power of virtual functions derives from the dynamic dispatch involved. This needs a hidden 'identity' stored with each object that has virtual functions. In C++ implementations, this identity is the virtual function table pointer. The dispatch can then be performed very efficiently on most machines, usually with just a couple memory fetches before the actual call.

6.3.3 Unary Operators

Syntax, Page 43, line 19, AFTER:

sizeof (type-name)

ADD:

delete unary-expression

NEW SECTION, Page 46, ADD:

6.3.3.5 The delete Operator

Constraints

The delete operator shall be applied to an expression with pointer to object type.

Semantics

The delete operator frees any storage allocated previously by the new operator. If the pointer passed to the delete operator was not returned by some previous new operator, the behavior is undefined.

RATIONALE:

While it would be possible to specify that storage allocated by operator new is freed by the library free call, this is not true for C++, so portability of C code to C++ would be harmed. A C implementation could use malloc to implement operator new and free to implement operator delete

6.5.1 Storage-Class Specifiers

Syntax, Page 58, line 8, AFTER:

register

ADD:

virtual

Constraints, Page 58, line 11, AFTER:

... specifiers in a declaration.

ADD:

The virtual specifier shall appear only in declarations of member functions.

Semantics, Page 58, line 20, AFTER:

... storage-class specifier other than extern.

ADD:

The virtual specifier is called a "storage-class specifier" for

syntactic convenience only. A member function declared with the virtual specifier is a virtual function.

When a virtual function is declared in a class derived from a base class that contains a member with the same name, the virtual function shall be delcared with a compatible type with the type of the member in the base class.

RATIONALE:

This provides the syntactic placement for the virtual declaration specifier. The requirement for compatible declarations is needed to ensure that calling conventions are not violated by virtual functions in different classes. The rules could be slightly less strict such as the rules C++ has apparently adopted, but the utility of such a relaxed set of rules is arguable. These rules have the virtue of simplicity.