Document Number: WG14 N447/X3J11 95-048

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

Title: Classes in C - Part 4: Constructors and Destructors _
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:

Constructors provide a common means for executing initialization code in a compact and standard way.

Destructors provide an even more convenient way to insure that when an object is freed, its internal data structures are properly cleaned up. Frequently, data structures are allocated inside an object or files are opened by an object. These resources can then be systematically released in response to a delete operator.

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 constructors and destructors to classes in the C Standard. Constructors provide a means for initializing newly instantiated objects using a language mechanism rather than any convention for initialize methods. Strictly speaking, however, they provide only minimal additional functionality (because of the way in which virtual method dispatch is handled) over normal

methods.

Destructors provide a means for scheduling cleanup work in an object as it is about to be deleted. This provides a standard and convenient way to assist the delete operator without introducing special conventions that the user of a class needs to be told about.

Proposal: _____

The wording changes are summarized in the following points.

* Constructors and destructors are allowed for class objects, but in constrained situations.

* Static constructors and destructors are not allowed. This is accomplished by forbidding the static allocation of an instance of a class with either a destructor or constructor.

* Automatic destructors are not allowed. Automatic instances of classes with destructors are not allowed.

* A constructor is invoked in an automatic declaration of an object or in a new operator.

* A destructor is invoked as a side effect of the delete operator.

* In a deeply nested class hierarchy, constructors actually modify the identity of an object so that any virtual method dispatch has a predictable outcome (i.e. until a constructor for base class foo is done, the object has type foo, but then the constructor for derived class subfoo runs with the object typed as a subfoo). Similarly, destructors work from the leaf type to through the base types executing destructors and setting the object type as it goes.

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.

6.1.2.4 Storage Duration of Objects

Page 22, AFTER line 22, ADD:

An object whose identifier is declared with a class type containing a constructor or destructor or an aggregate type, any of whose members or elements have a class type containing a constructor or destructor, shall not have static storage duration.

An object whose identifier is declared with a class type containing a destructor or an aggregate type, any of whose members or elements have a class type containing a destructor, shall not have automatic storage duration.

RATIONALE:

Static constructors and destructors introduce additional semantics and demand support from the runtime that may not be available in C implementations. Since the order of execution of static constructors and destructors is unspecified in C++, they have very limited utility.

Destructors for automatic objects introduce the problem of when and where those destructors get called. In the interest of keeping the amount of code that is generated as overt as possible, automatic destructors are not allowed.

Note that in both of these cases, the wording is not in a constraints section, so violating these restrictions is only undefined behavior. Thus, an implementation could support static constructors and static and automatic destructors without issuing a diagnostic.

6.3.1 Primary Expressions

<<From previous proposal>> Syntax, page 39, line 6, IS:

new type-name

SHOULD BE:

new type-name constructor-call opt

Constraints

The type name appearing in a constructor new expression shall name a class type that contains a constructor. The number of argument expressions shall agree with the number of parameters defined for the constructor in that class.

Example

```
class con {
        int     x, y;
        public:

        con(int a, int b);
};

func()
{
        class con *cp;

        cp = new con(2, 3);
}

con::con(int a, int b)
{
        x = a;
```

```
        y = b;
}
```

In this example a class is declared with a constructor that accepts tow ints. The operator new then includes arguments for the constructor, which is called as a side-effect of the new operator. Inside the constructor, in this case, the arguments are copied to private data members of the class.

This approach gives the constructor the opportunity to validate the arguments before storing them.

RATIONALE:

<< From previous proposal >> Page 39, IS:

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.

ADD:

A new operator with a type name followed by parentheses enclosing an optional argument expression list is a constructor new expression. Such an expression both allocates storage and calls the constructor for the allocated type. If an identity needs to be set for any objects allocated, they are set before the constructor is called.

ADD TO Forward References:

6.5.4.4 Constructor Calls

RATIONALE:

Operator new needs to allow for calls to constructors.

<< From previous proposal >>

6.3.3.5 The delete Operator

ADD TO Semantics:

If the delete operator is applied to a pointer to a class that contains a destructor, or inherits from a class containing a destructor, the destructors are called before the memory of the object is freed. Beginning with the pointed-at class (or with the class corresponding to the identity for the actual object pointed at, if the class contains or inherits a virtual destructor) and proceeding to each base class in turn until all base classes have been exhausted, any destructor in each class is called. If any class in this sequence has an identity, the identity of the object is modified before the

call to each destructor so that the identity is of the class containing the destructor about to be called.

RATIONALE:

Destructors must be called in sequence when an object is deleted. The provision for virtual destructors allows complex data structures to be deleted without specifically knowing what the object type actually is.

<< From previous proposal >> Page 64, before line 1:

6.5.2.4 Class Specifiers

Syntax

IS:

class-declaration:
visibility-specifier opt
class-specifier-qualifier-list
struct-declarator-list ;

SHOULD BE:

class-declaration:
visibility-specifier opt basic-class-declaration ;

basic-class-declaration:
class-specifier-qualifier-list struct-declarator-list
constructor-declaration
destructor-declaration

constructor-declaration:
class-name (parameter-type-list)

destructor-declaration:
virtual opt ~ class-name ( )

class-name:
identifier

<< Previous proposal >> AT THE END OF Constraints ADD:

The class name in a constructor or a destructor declaration shall name the class of the enclosing class-specifier.

At most one constructor shall be declared within a class-specifier.

At most one destructor shall be declared within a class-specifier.

A class-specifier shall not include a member declaration with a class type that contains either a constructor or a destructor, nor with an array type that is an array of such classes.

<< Previous proposal >> Semantics, IS:

> A member of a class may have any object or member function type. In addition, a member may be a bit-field.

SHOULD BE:

> A member of a class may have any object or member function type. In addition, a member may be a bit-field, a constructor or a destructor.

AT THE END OF Semantics ADD:

> The enclosing class name followed by a parameter list enclosed in parentheses is a constructor declaration.
>
> The enclosing class name preceded by an optional virtual keyword and a tilde and followed by an empty pair of parentheses is a destructor declaration. If the virtual keyword is present, it declares a virtual destructor for the class.

RATIONALE:

> Constructor and destructor declarations conform to most, but not all of the rules for a regular function declaration. It is therefore easier to write the syntax to specify the constructor and destructor syntax specifically.

6.5.4 Declarators

Page 65, line 18 IS:

> identifier

SHOULD BE:

> identifier
> identifier :: identifier

<< From previous proposal >> Page 65, line 22 AFTER:

> direct-declarator (parameter-type-list)
> type-qualifier-list opt

ADD:

> constructor-declarator

Page 65, AFTER line 40 ADD:

> constructor-declarator:
> identifier constructor-call
>
> constructor-call:
> (argument-expression-list opt)

Constraints

The identifier declared in a constructor-declarator shall have class type and that class shall contain a constructor, or shall be derived from a class containing a constructor.

Page 66, AFTER line 11 ADD:

A constructor declarator is an identifier followed by a parenthesis enclosed argument list. It declares the identifier outside the parentheses.

RATIONALE:

Automatic constructor declarators are implicit calls to the constructor for the object being defined.

Page 69, line 10, NEW SECTION:

6.5.4.4 Constructor Calls

Constraints

The number of arguments in a constructor call shall agree with the number of parameters. Each argument shall have a type such that its value may be assigned to an object with the unqualified version of the type of its corresponding parameter.

Semantics

A new expression or a identifier in a declarator followed by parentheses () containing a possibly empty, comma-separated list of expressions is a constructor call. The object allocated by the new expression or the object declared by the declarator denotes the object being constructed. The list of expressions specifies the arguments of the constructor.

An argument may be an expression of any object type. In preparing for a constructor call, the arguments are evaluated and each parameter is assigned the value of the corresponding argument.

The arguments are implicitly converted, as if by assignment, to the types of the corresponding parameters. The ellipsis notation in a constructor declaration causes argument type checking to stop after the last declared parameter. The default argument promotions are performed on trailing arguments.

The order of evaluation of the arguments is unspecified, but there is a sequence point before the call.

Recursive constructor calls shall be permitted, both directly and indirectly through any chain of other functions or constructors.

RATIONALE:

Constructors are called in much the same way as functions. Since

constructors must be declared using new style parameter lists, fewer things need to be said about the argument passing.

6.7 External Definitions

<< From previous proposal >> Page 81, line 6, Syntax IS:

external-declaration:
function-definition
declaration
member-function-definition

SHOULD BE:

external-declaration:
function-definition
declaration
member-function-definition
constructor-definition
destructor-definition

<< From previous proposal >> Page 81, line 22-23 IS:

An external definition is an external declaration that is also a definition of a function, an object, or a member function.

SHOULD BE:

An external definition is an external declaration that is also a definition of a function, an object, a member function, a constructor, or a destructor.

NEW SECTION:

6.7.4 Constructor Definitions

Syntax

constructor-definition:
class-name :: class-name (parameter-type-list)
base-constructor-call opt
compound-statement

base-constructor-call:
: class-name (argument-expression-list opt)

Constraints:

The class-name specified in a constructor definition shall be an identifier declared with class type containing a constructor.

The parameter-type-list shall have the same number of parameters as the constructor declaration in the specified class, and the type of each parameter shall have compatible type with its corresponding parameter in the constructor declaration in the class.

The class-name specified in a base-constructor-call shall be an identifier declared with class type containing a constructor. Moreover, the class of the constructor being defined shall be derived from the class named in the base constructor call and for all classes that derive from this base and in turn are bases for the class whose constructor is being defined shall not contain any constructor.

Semantics:

A constructor definition specifies the name of the class for which the constructor is being defined, the identifiers of its parameters and an optional call to a base constructor. The argument list in any base constructor call shall be correct for the base constructor.

For any classes that have corresponding identities, the identity of the object being constructed is modified to indicate the class containing the constructor before it is called.

The base constructor call is executed before the compound-statement of the constructor definition.

On entry to the member function the value of each argument expression shall be converted to the type of its corresponding parameter, as if by assignment to the parameter. Array expressions and function designators as arguments are converted to pointers before the call. A declaration of a parameter as "array of type" shall be adjusted to "pointer to type," and a declaration of a parameter as "function returning type" shall be adjusted to "pointer to function returning type," as in 6.2.2.1. The resulting parameter type shall be an object type.

On entry to the member function the value of 'this' is set to the address of the appropriate class object as specified in 6.3.1, 6.3.2.5.

Each parameter has automatic storage duration. Its identifier is an lvalue. The layout of storage for parameters is unspecified.

The object that 'this' refers to has automatic storage duration and it is an lvalue. The layout of storage for 'this' is unspecified.

6.7.5 Destructor Definitions

Syntax

destructor-definition:
class-name :: ~ class-name ( ) compound-statement

Constraints:

The class-name specified in a destructor definition shall be an identifier declared with class type containing a destructor.

Semantics:

A destructor definition specifies the name of the class for which the destructor is being defined.

On entry to the member function the value of 'this' is set to the address of the appropriate class object as specified in 6.3.3.5.

The object that 'this' refers to has automatic storage duration and it is an lvalue. The layout of storage for 'this' is unspecified.

RATIONALE:

This language simply extends the rules for member functions to include constructors and destructors where appropriate.

•