Document Number: X3J16/92-0125
WG21/N0202
Date: November 5, 1992
Project: Programming Language C++
Reply to: Tom Pennello
tom@metaware.com

Trivial default constructors

Abstract. We specify when control flow may bypass a declaration of a class object or an array of class objects. When a class has constructors bears on this issue, and so we point out inconsistencies in the draft in this regard and repair them. The basic result is that control flow can bypass a class object declaration when there is "nothing to do" in initializing that object. We make precise the notion of "nothing to do". This retains a measure of conformance with C.

Motivation

The current draft prohibits control transfer around the declaration of an initialized object to a point where that object is still in scope. Transfer around an uninitialized object is OK; the presumption is that nothing happens at the site of the declaration to provide the object some initial value, so it's OK to jump around the initialization. The question is whether a declaration of a class object contains an implicit initialization.

```
struct s {};
struct t { t(); };      // User-declared ctor.
struct u : virtual s{};
goto L1; int x1 = 1;    L1:     // Error.
goto L2; int x2;        L2:     // OK.
goto L3; s x3;          L3:     // OK?
goto L4; t x4;          L4:     // Error.  User ctor skipped.
goto L5; u x5;          L5:     // OK?
```

The draft actually says that the goto L3 and L5 are both incorrect, because a class with no user-declared constructor has a compiler- declared default constructor ("one is generated", in the words of the draft); since this compiler-declared constructor is then used in the declaration, the declared object is thus initialized and so its declaration cannot be skipped. One may conclude from the words in the draft that all declared class objects are initialized.

We intend to segregate compiler-declared default constructors that do "interesting" work from those that don't, and allow transfers of control over objects initialized with a non-interesting compiler-declared default constructor.

"Interesting" work includes initializing pointers to virtual function tables and virtual base classes. For a class with such initialization requirements and with no user-declared constructor, an implicit ("compiler-generated") default constructor performs the initialization tasks; if the user declares a constructor, the initialization code is inserted into that constructor's definition.

Inconsistencies in the draft

Certain portions of the draft imply that a class always has one or more constructors,

and other portions imply that a class might not have constructors. Here are the statements from the draft:

> 12.1[P4] A default constructor for a class X is a constructor of class X that can be called without an argument. A default constructor will be generated for a class X only if no constructor has been declared for class X.

Thus if a class has no user-declared constructor, the compiler declares one (presuming that's included in the notion of "generated", whatever that means). Or it might mean that that the default constructor is "generated" only if called; if never called the class "doesn't have constructors", and if called, the class then "has constructors". Furthermore:

> 12.1[P5] A copy constructor for a class X is a constructor whose first argument is of type X& or const X& and whose other arguments, if any, all have defaults, so that it can be called with a single argument of type X. For example, X::X(const X&) and X::X(X&,int=0) are copy constructors. A copy constructor is generated if and only if no copy constructor is declared in the class definition.

Thus we have that both a default constructor and a copy constructor are "generated". We interpret this to mean they are declared, or at least that a class "has" a constructor as a result of the generation.

Here is other language in the draft that seems to contradict the implication:

> 12.6[P1] An object of a class with no constructors, ...
>
> 12.6.1[P3] Arrays of objects of a class with constructors use constructors in initialization (f12.1) just like individual objects.
>
> 12.6.1[P4] An object of class M can be a member of a class X only if (1) M does not have a constructor, or ...
>
> 8.4.1[P1] An aggregate is an array or an object of a class (f9) with no constructors (f12.1), ...

These three excerpts indicate that a class can have no constructors. But the class "struct s{};" has two generated constructors, so we intrepret the class to "have" constructors.

If we believe that a class always has a constructor, every declaration of a class object is an initialization. Either

- = {...} is provided (where allowed)
- a parenthesized expn list is provided, in which case a specific constructor is called
- the default constructor is invoked

Interestingly, = {...} isn't allowed for a class with constructors, and now because all classes have constructors, = {...} isn't even an option.

To prevent the incompatibility we define the notion of "trivial compiler-defined default constructor", and rule that initialization via trivial compiler-defined default constructor is not considered an initialization for the purposes of control transfer around a declaration. Furthermore, we should fix the rule for initialization

by {...} to allow initialization of classes with trivial default constructors. The intent is to allow the C fragment

```
struct S {};
goto L; { S x; L: }
```

to work.

The declaration of an object of a class (or array thereof) requiring non-trivial initialization is considered a declaration with an initialization, even though the declaration may lack an initializer. That is, having a default constructor invoked is not an initialization unless the class requires non-trivial initialization.

Editing proposal:

Insert the following somewhere; perhaps near the discussion of default constructors?

(Recursive) Definition of non-trivial initialization and non-trivial implicitly-declared default constructor.

> A class having a user-declared constructor or having a non-trivial implicitly-declared default constructor is said to require non-trivial initialization.
>
> An implicitly-declared default constructor is non-trivial iff either
>
> - the class has direct virtual bases or virtual functions
>
> - the class has direct bases or members of a class (or array thereof) requiring non-trivial initialization.

Then we change the statements about the "generation" of constructors, saying instead that implicit constructors are declared where the user doesn't:

Change

12.1[P4] A default constructor for a class X is a constructor of class X that can be called without an argument. A default constructor will be generated for a class X only if no constructor has been declared for class X.

12.1[P5] ... A copy constructor is generated if and only if no copy constructor is declared in the class definition.

to

12.1[P4] A default constructor for a class X is a constructor of class X that can be called without an argument. If no constructor has been declared for class X, a default constructor is implicitly declared. The definition for an implicitly declared constructor is supplied if that constructor is called. The definition may violate other constraints; see ??? [is there a discussion of this? such as attempting to call a private base constructor.].

12.1[P5] ... If no copy constructor is declared, a copy constructor is implicitly declared. The definition for an implicitly declared copy constructor is supplied if that copy constructor is called. The definition may violate other constraints; see 12.8[P5].

Now we change the uses of the implicit constructors in other text.
Change the text

12.6 Initialization

1 An object of a class with no constructors, no private or protected members, no virtual functions, and no base classes can be initialized using an initializer list; see f8.4.1. An object of a class with a constructor must either be initialized or have a default constructor (f12.1). The default constructor is used for objects that are not explicitly initialized.

to

12.6 Initialization

1 An object of a class (or array thereof) with no private or protected non-static data members and that does not require non-trivial initialization can be initialized using an initializer list; see f8.4.1. An object of a class (or array thereof) with a user-declared constructor must either be initialized or have a default constructor (f12.1) (whether user- or compiler-declared). The default constructor is used for an object (or array thereof) that is not explicitly initialized.

(Note: I relaxed the constraint so that if a class has all public non-static data members it still is in the running for {} initialization. Previously a private member function would prohibit its {} init.)

Change

12.6.1[P3] Arrays of objects of a class with constructors use constructors in initialization (f12.1) just like individual objects. If there are fewer initializers in the list than elements in the array, the default constructor (f12.1) is used. If there is no default constructor the initializer- clause must be complete.

12.6.1[P4] An object of class M can be a member of a class X only if (1) M does not have a constructor, or (2) M has a default constructor, or (3) X has a constructor and if every constructor of class X specifies a ctor-initializer (f12.6.2) for that member. In case 2 the default constructor is called when the aggregate is created.

to

12.6.1[P3] Arrays of objects of a class use constructors in initialization (f12.1) just as do individual objects. If there are fewer initializers in the list than elements in the array, a default constructor must be declared (whether by the compiler or the user), and it is used; otherwise, the initializer-clause must be complete.

12.6.1[P4] An object of class M can be a member of a class X only if (1) M has a default constructor or (2) X has a user-declared constructor and every user-declared constructor of class X specifies a ctor-initializer (f12.6.2) for that member. In case 1 the default constructor is called when the aggregate is created.

Change

5.3.3[P9](new) If a class has a constructor an object of that class can be created by new only if suitable arguments are provided or if the class has a default constructor (f12.1).

5.3.3[P10] ... Arrays of objects of a class with constructors can be created by operator new only if the class has a default constructor (f12.1).

to

5.3.3[P9](new) An object of a class can be created by new only if suitable arguments are provided to the class's constructors, or if the class has a default constructor (f12.1).

(Note: this means that "struct s{}; s x; s y(x);" is allowed on the grounds that class s has an implicitly declared copy constructor, to which the argument x is being provided.)

5.3.3[P10] ... Arrays of objects of a class can be created by operator new only if the class has a default constructor (f12.1).

(Note: C-style structs have an implicitly-declared default constructor.)

Change

12.6.2[P3] ...If a constructor of the most derived class does not specify a mem-initializer for a virtual base class then that virtual base class must have a default constructor or no constructors.

to

12.6.2[P3] ...If a constructor of the most derived class does not specify a mem-initializer for a virtual base class then that virtual base class must have a default constructor.

Change

8.4.1[P1] An aggregate is an array or an object of a class (f9) with no constructors (f12.1), no private or protected members (f11), ...

8.4.1[P8] The initializer for a union with no constructor is either a single expression of the same type, ...

to

8.4.1[P1] An aggregate is an array or an object of a class (f9) with no user-declared constructors (f12.1), no private or protected members (f11), ...

8.4.1[P8] The initializer for a union with no user-declared constructor is either a single expression of the same type, ...

I have used a text search to find all the places in the draft that say "no constructor" but I may have missed some; they should still be edited.

There are a few other places in the draft where mention is made of the "generated copy constructor"; these should be replaced by the "implicitly declared copy constructor". Same goes for the generated default constructor.

Acknowledgements. Thanks to Randy Swan who provided the on-line ASCII text of the draft, without which the writing of this paper would have been quite tedious.