Document Number: J16/04-0165 = WG21 N1725
Date: 2004-11-08
Reply to: William M. Miller
Edison Design Group, Inc.
wmm@edg.com

Copy Elision in Exception Handling

I. The Problem

Clause 15 has two passages dealing with copy elision in exception handling. 15.1¶5 says,

> If the use of the temporary object can be eliminated without changing the meaning of the program except for the execution of constructors and destructors associated with the use of the temporary object (12.2), then the exception in the handler can be initialized directly with the argument of the throw expression. When the thrown object is a class object, and the copy constructor used to initialize the temporary copy is not accessible, the program is ill-formed (even when the temporary object could otherwise be eliminated). Similarly, if the destructor for that object is not accessible, the program is ill-formed (even when the temporary object could otherwise be eliminated).

In a sometimes overlapping, sometimes differing passage, 15.3¶17 says,

> If the use of a temporary object can be eliminated without changing the meaning of the program except for execution of constructors and destructors associated with the use of the temporary object, then the optional name can be bound directly to the temporary object specified in a *throw-expression* causing the handler to be executed. The copy constructor and destructor associated with the object shall be accessible even when the temporary object is eliminated.

Part of the difference is terminological. For instance, 15.1¶5 refers to "the exception in the handler," while the "optional name" in 15.3¶17 is a reference to the *exception-declaration* that is part of a handler (15¶1). There are substantive differences, however: 15.3¶17 appears to deal only with the case in which the *exception-declaration* has a name, and then only if the operand of the *throw-expression* is a temporary. By contrast, 15.1¶5 seems to apply to all *exception-declaration*s and appears to allow something like the return value optimization of 12.8¶15, where copying a local automatic object can be avoided by constructing it directly into the return value.

Another issue is that 12.8¶15 would appear to be an exhaustive catalog of the circumstances in which constructor and destructor side-effects can be ignored in copy elision. However, these two passages in clause 15 specify additional contexts in which this kind of optimization may be applied.

Finally, clause 15 actually specifies two distinct copy operations that are performed during exception handling: the copy from the operand of the *throw-expression* to the exception object (15.1¶3) and the copy from the exception object to the object declared by the *exception-*

declaration (15.3¶16). These operations are logically distinct and separately susceptible to optimization, although these elisions can be combined to achieve the effect described in these passages. Both 15.1¶5 and 15.3¶17 blur this distinction and thus provide an insufficient specification for these optimizations.

II. Proposed Resolution

For consistency, both within clause 15 and with clause 12, I believe the principal specification of copy elision in exception handling ought to be in 12.8¶15, with references to that section in clause 15.

1. Change 12.8¶15 as follows:

> When certain criteria are met, an implementation is allowed to omit the copy construction of a class object, even if the copy constructor and/or destructor for the object have side effects. In such cases, the implementation treats the source and target of the omitted copy operation as simply two different ways of referring to the same object, and the destruction of that object occurs at the later of the times when the two objects would have been destroyed without the optimization. *[Footnote: Because only one object is destroyed instead of two, and one copy constructor is not executed, there is still one object destroyed for each one constructed.]* This elision of copy operations is permitted in the following circumstances (which may be combined to eliminate multiple copies):
>
> - in a `return` statement in a function with a class return type, when the expression is the name of a non-volatile automatic object with the same cv-unqualified type as the function return type, the copy operation can be omitted by constructing the automatic object directly into the function's return value
>
> - **in a *throw-expression*, when the operand is the name of a non-volatile automatic object, the copy operation from the operand to the exception object (15.1) can be omitted by constructing the automatic object directly into the exception object**
>
> - when a temporary class object that has not been bound to a reference (12.2) would be copied to a class object with the same cv-unqualified type, the copy operation can be omitted by constructing the temporary object directly into the target of the omitted copy
>
> - **when the *exception-declaration* of an exception handler (clause 15) declares an object of the same type (except for cv-qualification) as the exception object (15.1), the copy operation can be omitted by treating the *exception-declaration* as an alias for the exception**

object if the meaning of the program will be unchanged except for the execution of constructors and destructors for the object declared by the *exception-declaration*

[*Example: ...*

2. Change 15.1¶5 as follows:

~~If the use of the temporary object can be eliminated without changing the meaning of the program except for the execution of constructors and destructors associated with the use of the temporary object (12.2), then the exception in the handler can be initialized directly with the argument of the throw expression.~~ When the thrown object is a class object, ~~and~~ the copy constructor ~~used to initialize the temporary copy~~ **and the destructor shall be** ~~is not~~ accessible, ~~the program is ill-formed (even when the temporary object could otherwise be eliminated~~ **even if the copy operation is elided (12.8**). ~~Similarly, if the destructor for that object is not accessible, the program is ill-formed (even when the temporary object could otherwise be eliminated).~~

3. Change 15.3¶17 as follows:

~~If the use of a temporary object can be eliminated without changing the meaning of the program except for execution of constructors and destructors associated with the use of the temporary object, then the optional name can be bound directly to the temporary object specified in a *throw-expression* causing the handler to be executed.~~ The copy constructor and destructor associated with the object shall be accessible even ~~when the temporary object is eliminated~~ **if the copy operation is elided (12.8)**.