Proposing the Rule of Five

Document #: WG21 N3578
Date: 2013-03-12
Revises: None
Project: JTC1.22.32 Programming Language C++
Reply to: Walter E. Brown <webrown.cpp@gmail.com>

Contents

1 Introduction

[Mau10], approved at the Batavia meeting, provided initial wording for the conditions[1] under which move operations would be implicitly compiler-generated. This wording affected the implicit declarations of copy operations as well, specifying both the conditions under which those functions are implicitly declared, as well as the conditions determining whether they are implicitly defined as deleted or defaulted. The wording was adjusted at the next (Madrid) meeting via the final resolution of CWG issue 1082 in [Mil11].

Some of the behavior these papers specify is explicitly called out as deprecated. These parts of the feature are summarized in Annex D:

> The implicit definition of a copy constructor as defaulted is deprecated if the class has a user-declared copy assignment operator or a user-declared destructor. The implicit definition of a copy assignment operator as defaulted is deprecated if the class has a user-declared copy constructor or a user-declared destructor. In a future revision of this International Standard, these implicit definitions could become deleted" [depr.impldec]/(cross-references omitted).

This paper proposes to obsolete the deprecated behavior, giving C++14 a true "rule of five" instead of the traditional "rule of three."

2 Discussion

The "rule of three" is an oft-quoted C++ design guideline, attributed to Marshall Cline, dating from 1991. Informally, the guideline recommends that three special functions — the destructor, the copy constructor, and the copy assignment operator — be considered together when designing a class: If there is a reason for the class to define one of them explicitly, it is likely that the class should explicitly define all three.

[1] These conditions correspond to option #2 in [Str10b], a follow-up to an analysis begun in [Str10a].

[KM01] concludes more precisely that “The Rule of Three is really two rules:

- “If a class has a nonempty destructor, it almost always needs a copy constructor and an assignment operator.
- “If a class has a nontrivial copy constructor or assignment operator, it usually needs both of these members and a destructor as well.”

During discussions leading to the status quo, LWG considered at one point actually enforcing this rule, and augmenting it with corresponding restrictions regarding the then-new move special functions. The compromise enshrined in today’s wording (including the deprecation warnings) was reached in an effort to avoid breaking C++03 code while stretching the rules to accommodate move functions.

With several years of warning in place by the time we issue the next standard, we believe it is time to consider achieving the best possible consistency of behavior. We therefore propose to adapt the C++11 wording so as to achieve a simple rule. To be known informally as the “rule of five,” we propose that **no copy function, move function, or destructor be compiler-generated if any of these functions is user-provided**.

3 Proposed wording

Adjust [class.copy]/7 as shown:

If the class definition does not explicitly declare a copy constructor, one is declared *implicitly*. If the class definition declares a copy assignment operator, move constructor, ~~or~~ move assignment operator, or destructor, the implicitly declared copy constructor is defined as deleted; otherwise, it is defined as defaulted (8.4). ~~The latter case is deprecated if the class has a user-declared copy assignment operator or a user-declared destructor.~~ Thus, for the class definition

```
struct X {
  X(const X&, int);
};
```

a copy constructor is implicitly-declared. If the user-declared constructor is later defined as

```
X::X(const X& x, int i =0) { /* ... */ }
```

then any use of **X**’s copy constructor is ill-formed because of the ambiguity; no diagnostic is required.

Adjust [class.copy]/18 as shown:

If the class definition does not explicitly declare a copy assignment operator, one is declared *implicitly*. If the class definition declares a copy constructor, move constructor, ~~or~~ move assignment operator, or destructor, the implicitly declared copy assignment operator is defined as deleted; otherwise, it is defined as defaulted (8.4). ~~The latter case is deprecated if the class has a user-declared copy constructor or a user-declared destructor.~~ The implicitly-declared copy assignment operator for a class **X** will have the form . . .

Remove the entirety of [depr.impldec] (cited in §1 above) from Annex D.

4 Acknowledgments

Many thanks to the readers of early drafts of this paper for their helpful feedback.

5 Bibliography

[DuT12] Stefanus Du Toit: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N3485 (post-Portland mailing), 2012-11-02.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3485.pdf.

[KM01] Andrew Koenig and Barbara E. Moo: "C++ Made Easier: The Rule of Three." 2001-06-01.
http://www.drdobbs.com/c-made-easier-the-rule-of-three/184401400.

[Mau10] Jens Maurer: "Tightening the conditions for generating implicit moves." ISO/IEC JTC1/SC22/WG21 document N3203 (post-Batavia mailing), 2010-11-11.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2010/n3203.htm.

[Mil11] William M. Miller: "Additional Core Language Issue Resolutions for Madrid." ISO/IEC JTC1/SC22/WG21 document N3262 (post-Madrid mailing), 2011-03-25.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3262.htm.

[Str10a] Bjarne Stroustrup: "To move or not to move." ISO/IEC JTC1/SC22/WG21 document N3174 (pre-Batavia mailing), 2010-10-17.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2010/n3174.pdf.

[Str10b] Bjarne Stroustrup: "Moving right along." ISO/IEC JTC1/SC22/WG21 document N3201 (post-Batavia mailing), 2010-10-23.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2010/n3201.pdf.

6 Revision history

Revision	**Date**	**Changes**
1.0	2013-03-12	• Published as N3578.