p1412 - On user-declared and user-defined special member functions in safety-critical code

Peter Sommerlad

2019-01-21

Document Number:	p1412
Date:	2019-01-21
Project:	Programming Language C++ / Programming Language Vulnerabilities C++
Audience:	SG12 / ISO SC22 WG23, SG20, Misra C++

1 Introduction

Why.

TODO: Howard's table

rule of zero and related

context of audience.

forces to be covered.

Properties to be discussed.

Potential dangers.

2 Forces for Safety in Source Code

As a pattern (book) author I would like to introduce so-called "forces" that are used in a pattern's problem description to denote design constraints that influence the pattern's solution. Often such forces are not absolute and a pattern make conscious trade-offs. That is also a reason, why often conflicting patterns for a problem exist that resolve to different solutions.

Here I collect forces that in my observation have influenced existing programming guidelines.

— Simplicity

— Familiarity

3 Howard Hinnant's special member function overview

Table 1 — Howard Hinnant's special member functions table

	What the compiler provides for class X						
user declares	`X()`	`~X()`	`X(X const&)`	`=(X const&)`	`X(X &&)`	`=(X &&)`	OK?
nothing	`=default`	`=default`	`=default`	`=default`	`=default`	`=default`	OK
`X(T)`	not decl	`=default`	`=default`	`=default`	`=default`	`=default`	OK
`X()`	*declared*	`=default`	`=default`	`=default`	`=default`	`=default`	(OK)
`~X()`	`=default`	*declared*	`=default`	`=default`	not decl	not decl	**BAD**
`X(X const&)`	not decl	`=default`	*declared*	`=default`	not decl	not decl	**BAD**
`=(X const&)`	`=default`	`=default`	`=default`	*declared*	not decl	not decl	**BAD**
`X(X&&)`	not decl	`=default`	`=delete`	`=delete`	*declared*	not decl	**BAD**
`=(X&&)`	`=default`	`=default`	`=delete`	`=delete`	not decl	*declared*	(BAD)

Table 2 — Safe and Sane combinations of Special Member Functions (TODO)

	declared or provided						
type category	`X()`	`~X()`	`X(X const&)`	`=(X const&)`	`X(X &&)`	`=(X &&)`	
value/aggregate	`=default`	`=default`	`=default`	`=default`	`=default`	`=default`	OK
value	not decl	`=default`	`=default`	`=default`	`=default`	`=default`	OK
`X()`	*declared*	`=default`	`=default`	`=default`	`=default`	`=default`	(OK)
OO	`=default`	*declared*	`=default`	`=default`	not decl	not decl	**BAD**
`X(X const&)`	not decl	`=default`	*declared*	`=default`	not decl	not decl	**BAD**
`=(X const&)`	`=default`	`=default`	`=default`	*declared*	not decl	not decl	**BAD**
`X(X&&)`	not decl	`=default`	`=delete`	`=delete`	*declared*	not decl	**BAD**
`=(X&&)`	`=default`	`=default`	`=delete`	`=delete`	not decl	*declared*	(BAD)

3.1 Items to be discussed

Things I am unsure

— Are there further useful and safe exceptions?

4 Categories of safe user-defined classes

4.1 Plain Value Types

4.2 Monomorphic Object Types (better name) - Encapsulation Types

4.3 Polymorphic Object Types - Class Hierarchies

4.4 Resource Managing Types

5 Bibliography

Core Guidelines

MISRA

Rule of Zero

Howard's table