An in-line defaulted destructor should keep the copy- and move-operations

Document Number:	**P2917 R0**
Date:	2023-06-14
Project:	ISO JTC1/SC22/WG21: Programming Language C++
Reply-to:	Andreas Fertig ⟨ isocpp@andreasfertig.com ⟩
Audience:	EWG

Contents

1 Introduction

Defining a class with a defaulted `virtual` destructor requires users to also default the move- and copy-operations. Otherwise, the class is not moveable as of today, and the behavior for the copy-operations is deprecated and on the list of being removed [P2863R0] §6.8, which was also asked on the reflector [DepCopyOps]. These additional defaulted special members are boilerplate code. This aims to reduce the lines of code required for a class with a defaulted destructor.

2 Motivation

Since C++11, the language has move semantics and with `=default` a way to ask for the compiler-provided implementation of a special member function.

C++11 changed the rules for the implicit declaration of copy functions [depr.impldec] as such as that they are no longer provided in case the class has a user-declared destructor.

The author thinks that the interaction between the copy and move operations is correct. However, the destructor is special. The destructor is the only special member function always present independently of the other special member functions. We only lose the destructor if the class contains a member without a destructor.

This paper proposes to decouple the destructor from the other special member functions. A class with an explicitly defaulted destructor on its first declaration (whether `virtual` or not) would then still be copy- and move-able.

Currently

```cpp
class IUSART {
public:
  virtual ~IUSART() = default;
  IUSART(const IUSART&) = default;
  IUSART& operator=(const IUSART&)
                                = default;
  IUSART(IUSART&&) = default;
  IUSART& operator=(IUSART&&) = default;

   virtual void Open() =0;
   virtual void Close() =0;
};
```

With proposal

```cpp
class  IUSART {
public:
  virtual ~IUSART() = default;
  virtual void Open() =0;
  virtual void Close() =0;
};
```

```cpp
class Apple {
public:
  ~Apple() = default;
  Apple(const Apple&) = default;
  Apple& operator=(const Apple&)
                                = default;
  Apple(Apple&&) = default;
  Apple& operator=(Apple&&) = default;
};
```

```cpp
class Apple {
public:
  ~Apple() = default;
};
```

Unchanged

```cpp
class Apple {
public:
  ~Apple();
  Apple(const Apple&) = default;
  Apple& operator=(const Apple&)
                                = default;
  Apple(Apple&&) = default;
  Apple& operator=(Apple&&) = default;
};

Apple::~Apple() = default;
```

```cpp
class Apple {
public:
  ~Apple();
  Apple(const Apple&) = default;
  Apple& operator=(const Apple&)
                                = default;
  Apple(Apple&&) = default;
  Apple& operator=(Apple&&) = default;
};

Apple::~Apple() = default;
```

3 The design

3.0.1 Why only for in-class defaulted destructors?

Allowing this behavior only for in-class defaulted destructors seems to be the most consistent with the behavior of the compiler for an unmodified class. Allowing also out-of-line destructors could make the implementation hard.

4 Implementation

This proposal was not implemented yet.

5 Proposed wording

This wording is based on the working draft [N4950] and was not yet reviewed by a Core expert.

Change **[class.copy.ctor]** 11.4.5.3:

6 If the class definition does not explicitly declare a copy constructor, a non-explicit one is declared *implicitly*. If the class definition declares a move constructor or move assignment operator, the implicitly declared copy constructor is defined as deleted; otherwise, it is defined as defaulted (dcl.fct.def). The latter case is deprecated if the class has a user-declared copy assignment operator or a user-declared destructor other than an explicitly defaulted destructor on its first declaration (depr.impldec).

8 If the definition of a class X does not explicitly declare a move constructor, a non-explicit one will be implicitly declared as defaulted if and only if

- X does not have a user-declared copy constructor,
- X does not have a user-declared copy assignment operator,
- X does not have a user-declared move assignment operator, and
- X does not have a user-declared destructor other than an explicitly defaulted destructor on its first declaration.

Change **[class.copy.assign]** 11.4.6:

2 If the class definition does not explicitly declare a copy assignment operator, one is declared *implicitly*. If the class definition declares a move constructor or move assignment operator, the implicitly declared copy assignment operator is defined as deleted; otherwise, it is defined as defaulted (dcl.fct.def). The latter case is deprecated if the class has a user-declared copy constructor or a user-declared destructor (depr.impldec) other than an explicitly defaulted destructor on its first declaration.

4 If the definition of a class X does not explicitly declare a move assignment operator, one will be implicitly declared as defaulted if and only if

- X does not have a user-declared copy constructor,
- X does not have a user-declared move constructor,
- X does not have a user-declared copy assignment operator, and

- X does not have a user-declared destructor other than an explicitly defaulted destructor on its first declaration.

Bibliography

[P2863R0] Alisdair Meredith: *"Review Annex D for C++26"*, P2863R0, 2023-05-15.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/p2863r0.html

[N4950] Thomas Köppe: *"Working Draft, Standard for Programming Language C++"*, N4950, 2023-05-10.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/n4950.pdf

[DepCopyOps] Deprecation of defaulted copy constructor and copy assignment
http://lists.isocpp.org/core/2023/02/13934.php