Through an error on my part, the semi-final version of Sakkinen's article in 5.1 (pp. 69–110) was printed, instead of the final version. My personal apologies. — P.H. Salus

# *Corrigendum to "A Critique of the Inheritance Principles of C++"*

Markku Sakkinen University of Jyväskylä, Finland

I have noted some errors in Sakkinen [1992] after publication. One of them is so significant that submitting a corrigendum seemed necessary; I am grateful to the editors for agreeing to publish it. At the same time it is convenient to point out and correct the smaller mistakes. Most of these corrections had already been sent in, but by accident did not get into the published version. John Skaller and other participants of the Usenet group 'comp.std.c++' must be acknowledged for pointing out one of the minor errors recently (see below); this lead me indirectly to discover the major one.

The significant change is that Restriction 2 in §5.5 (p. 101)—which I had felt as an unwelcome necessity—should be removed. Indeed, the argument about Example 12 that precedes the restriction is invalid, and the restriction would completely forbid inaccessible fork-join inheritance!

Think about a case in which the restriction is violated: a non-immediate ancestor **A** of class **C** is accessible to two intermediate classes **D** and **E** in the inheritance graph, but there is no class in the inheritance graph to which both **D** and **E** are accessible. The paths from **C** to **A** through **D** and **E** cannot then both be accessible, thus by

the main Rule in §5.4 (p. 98) itself, they correspond to two disjoint **A** subobjects. Therefore it is fully feasible to have different redefinitions of **A**'s virtual functions in **D** and **E**: the restriction is not needed.

The reference given for the BETA language at the end of §3.5 (p. 84), [Madsen 1987], is not the most appropriate one. Another article from the same book should have been referred to: [Kristensen et al. 1987].

There is a slight misunderstanding about the accessibility of constructors of virtual base classes in §3.6 (p. 86). I supposed that the constructors of a virtual base class would be automatically visible to all descendants, ignoring access specifiers. This is not true, but instead, non-immediate descendants may need to declare that class also as a direct base class only in order to be able to invoke its constructor(s). A class may therefore be non-instantiable (abstract) also because it has no access to necessary base class constructors, and not only because of pure virtual functions [Skaller 1992].

It is an open question whether such an additional direct base declaration is always needed in current C++ in order to use explicit initialisers, even if the constructors of the indirect virtual base classes are *accessible* to the most derived class. On the one hand, as stated in Ellis & Stroustrup [1990 §12.6.2, p. 290]:

> *Initializers for immediate base classes [. . .] may be specified in the definition of a constructor.*

Also in the example on p. 294 there is no other reason for such a redundant-looking declaration. On the other hand, the rationale given for the above rule is that multiple initialisations of the same base class subobject are prevented; but for virtual base classes this is already guaranteed by their special rules. Also, the HP compiler (Release 2.1) available to me did allow the constructor of a non-immediate virtual base class to be invoked.

There is a small but irritating clerical error in Example 7 (§4.2, p. 91). Class **CowboyWindow** should be derived from **WWindow** and **CCowboy** instead of **Window** and **Cowboy**. Readers may have guessed that, because the example makes no sense otherwise.

The availability of the new book by Bertrand Meyer [1992] causes modifications to a couple of comparisons between C++ and Eiffel. Current Eiffel seems to allow, by renaming, the possibility that is desired in §4.2, in the last paragraph beginning on p. 92. It is called

'joining' in Meyer [1992 §10]. The types of the inherited routines (functions) that are joined in a subclass do not even need to be identical.

In §4.3, the second last line of p. 93 contains an incorrect comparison: inheriting a class without exporting any features in Eiffel corresponds to **protected** rather than **private** derivation in C++. The disadvantage of Eiffel described in the last paragraph (p. 94) has been corrected in the newest version of Eiffel. The following paragraph should be added to the end of §4.3.

So-called system-level validity checking in current Eiffel [Meyer 1992 §22] will detect this kind of attempted misuse of a FIXED_STACK object: not the assignment, but any invocation of an ARRAY routine. Thus Eiffel can now better than C++ enforce the protection of non-public features toward outside clients. However, Eiffel does not offer any protection toward descendant classes: there is nothing corresponding to **private** derivation (nor private members). The designer of class FIXED_STACK cannot therefore easily cancel the original decision to use ARRAY in the implementation, because some descendant classes may already depend on it. Also, Eiffel's system-level checking is pessimistic: it can reject even programs that would actually be type safe.

## *References*


B. B. Kristensen, O. L. Madsen, B. Møller-Pedersen and K. Nygaard, The BETA Programming Language, *Research Directions in Object-Oriented Programming* (B. Shriver and P. Wegner, Eds.), pages 7–48, Cambridge, MA: MIT Press, 1987.

B. Meyer, *Eiffel: the Language,* Hemel Hempstead, England: Prentice Hall, 1992.

M. Sakkinen, A Critique of the Inheritance Principles of C++, *Computing Systems,* 5(1): 69–110, Winter 1992.

J. Skaller, postings on Usenet (comp.std.c++), 1992.