“New” Value Terminology

Bjarne Stroustrup

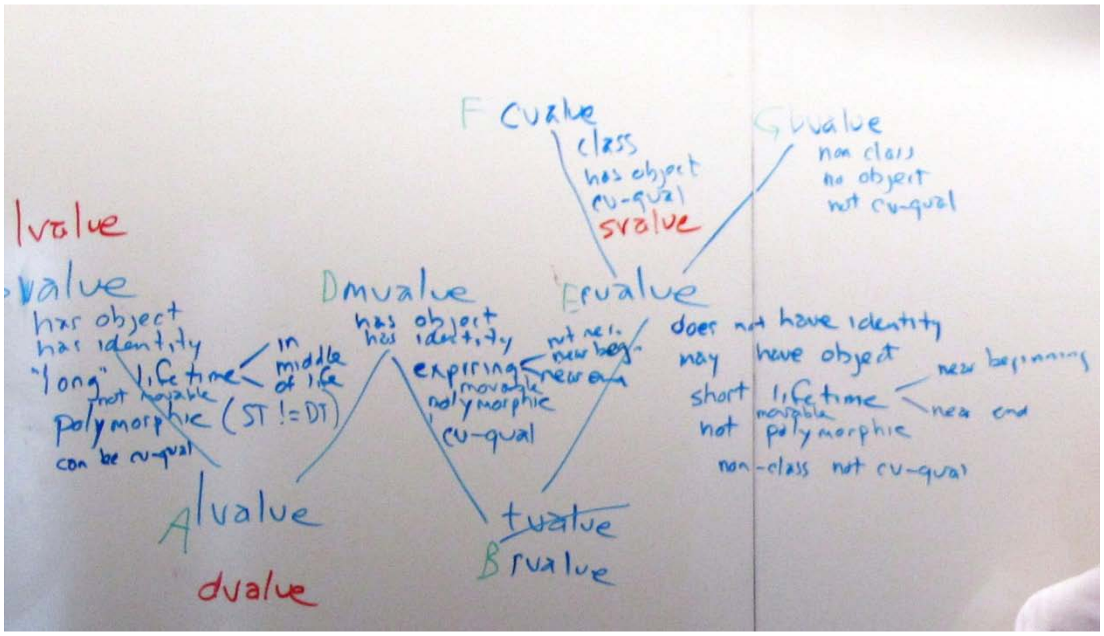
Background

The terms “lvalue” and “rvalue” are deep in C++’s genes. They were introduced by Christopher Strachey for CPL [Strachey,196?], the ancestor to BCPL. Dennis Ritchie used “lvalue” to describe C (e.g. see [K&R,1978]), but left out “rvalue”, considering “lvalue” and “not lvalue” sufficient. I did the same for early definitions of C++ (e.g. see [Stroustrup,1986] and [Ellis,1989]). The terms “lvalue” and “rvalue” are “all over” the draft C++ standard. Clearly, this is not terminology to mess with without serious reason and great care. So – despite appearances and rumors to the contrary – we didn’t. Key terms, such as “lvalue” and “rvalue” retain their conventional meaning in the FCD. Furthermore, I claim that the terminology used in the FCD [Becker,2010] and presented in [Miller,2010a] does not represent “random personal preferences” but is derived from an analysis of fundamental properties and selected from a very constrained set of options. Anyone doing a proper analysis should come to a very similar set of concepts and names for those.

Precise wording for specification is hard, so to aid precision and to resolve some known specification problems (related to rvalue references) Mike Miller wrote a paper with the unpromising title *Rvalue References as ‘Funny’ Lvalues* proposing a changed terminology [Miller,2010]. I did not consider “messing with terminology” late in the standards process a good idea and when Daniel Krügler pointed out that the definition of “rvalue” in that proposal differed from the use of “rvalue” in the standard library, I considered the proposal dead.

The model

However, the majority of the CWG disagreed and insisted that some changed and/or novel terminology was necessary to address known problems and to get the specification consistent. This led to a long (days) confused discussion with many suggestions – some novel and interesting. Eventually, I got sufficiently worried to ask a series of questions about what characterized the various terms, which Steve Adamczyk and others answered. The problems being addressed and their suggested resolutions had already received years of consideration, so my questions had solid answers in the context of the WP. By lunchtime, Wednesday March 13, the whiteboard in the CWG looked like this (the handwriting is mostly Steve’s):



Now I was seriously worried. Clearly we were headed for an impasse or a mess or both. I spent the lunchtime doing an analysis to see which of the properties (of values) were independent. There were only two independent properties:

- "has identity" – i.e. and address, a pointer, the user can determine whether two copies are identical, etc.
- "can be moved from" – i.e. we are allowed to leave to source of a "copy" in some indeterminate, but valid state

This led me to the conclusion that there are exactly three kinds of values (using the regex notational trick of using a capital letter to indicate a negative – I was in a hurry):

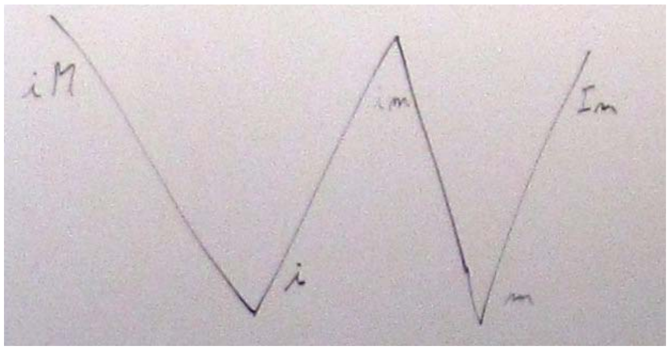
- iM: has identity and cannot be moved from
- im: has identity and can be moved from (e.g. the result of casting an lvalue to a rvalue reference)
- Im: does not have identity and can be moved from

The fourth possibility ("IM": doesn't have identity and cannot be moved) is not useful in C++ (or, I think) in any other language.

In addition to these three fundamental classifications of values, we have two obvious generalizations that correspond to the two independent properties:

- i: has identity
- m: can be moved from

This led me to put this diagram on the board (my handwriting):



Please note that the labels on the diagram (representing the three fundamental kinds of values and the two generalizations) are not names; they are abbreviations of lists of properties claimed to be primitive (in the mathematical sense).

Naming

I observed that we had only limited freedom to name: The two points to the left (labeled “iM” and “i”) are what people with more or less formality have called “lvalues” and the two points on the right (labeled “m” and “Im”) are what people with more or less formality have called “rvalues.” This must be reflected in our naming. That is, the left “leg” of the W should have names related to “lvalue” and the right “leg” of the W should have names related to “rvalue.” I note that this whole discussion/problem arise from the introduction of rvalue references and move semantics. These notions simply don’t exist in Strachey’s world consisting of just rvalues and lvalues. Someone observed that the ideas that

- Every value is either an lvalue or an rvalue
- An lvalue is not an rvalue and an rvalue is not an lvalue

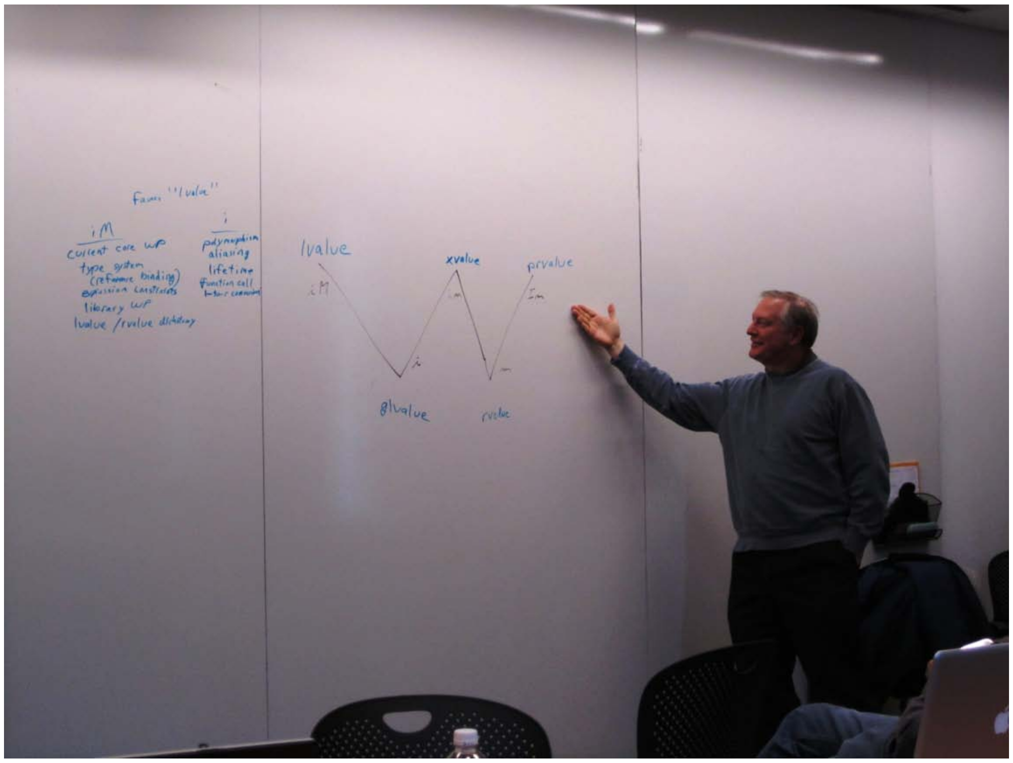
are deeply embedded in our consciousness, very useful properties, and traces of this dichotomy can be found all over the draft standard. We all agreed that we ought to preserve those properties (and make them precise). This further constrained our naming choices. I observed that the standard library wording uses “rvalue” to mean “m” (the generalization), so that to preserve the expectation and text of the standard library the right-hand bottom point of the W should be named “rvalue.”

This led to a focused discussion of naming. First, we needed to decide on “lvalue.” Should “lvalue” mean “iM” or the generalization “i”? Led by Doug Gregor, we listed the places in the core language wording where the word “lvalue” was qualified to mean the one or the other. A list was made and in most cases and in the most tricky/brittle text “lvalue” currently means “iM”. This is the classical meaning of lvalue because “in the old days” nothing was moved; “move” is a novel notion in C++0x. Also, naming the top-left point of the W “lvalue” gives us the property that every value is an lvalue or an rvalue, but not both.

So, the top left point of the W is “lvalue” and the bottom right point is “rvalue.” What does that make the bottom left and top right points? The bottom left point is a generalization of the classical lvalue, allowing for move. So it is a “generalized lvalue.” We named it “glvalue.” You can quibble about the abbreviation, but (I think) not with the logic. We assumed that in serious use “generalized lvalue” would somehow be abbreviated anyway, so we had better do it immediately (or risk confusion). The top right point of the W is less general than the bottom right (now, as ever, called “rvalue”). That point represent the original pure notion of an object you can move from because it cannot be referred to again (except by a destructor). I liked the phrase “specialized rvalue” in contrast to “generalized lvalue” but “pure rvalue” abbreviated to “prvalue” won out (and probably rightly so). So, the left leg of the W is “lvalue” and “glvalue” and the right leg is “prvalue” and “rvalue.” Incidentally, every value is either a glvalue or a prvalue, but not both.

This leaves the top middle of the W: “im”; that is, values that have identity and can be moved. We really don’t have anything that guides us to a good name for those esoteric beasts. They are important to people working with the (draft) standard text, but are unlikely to become a household name. We didn’t find any real constraints on the naming to guide us, so we picked ‘x’ for the center, the unknown, the strange, the xpert only, or even x-rated.

Here is Steve showing off the final product:



Note the classification of current uses of “lvalue” (with various classifications) in the pre-Pittsburgh WP to the left (mostly in Doug’s handwriting).

In Pittsburgh (and a few places later), I had the opportunity to explain this terminology to several people. It can be done in a few minutes. I think it has prvalue (Groan!). Members of the CWG reports that the new terminology has already helped clarify and resolve issues.

Summary

The (supposedly new) terminology in the FCD (and presented in [Miller,2010a]) is logical (derived from fundamental language properties), in accordance with the history of C++ and all of its ancestors, in accordance with the pre-Pittsburgh wording, consistent with the terminology used to specify the standard library, and solves a few specification problems that several members of the CWG deemed important.

References

- [Becker,2010] The FCD.
- [K&R,1978] B. Kernighan and D. Ritchie: *The C programming Language*. Prentice-Hall 1978.
- [Miller,2010] William M. Miller: *Rvalue References as "Funny" Lvalues (Rev.2)*. WG21 10-0020=N3030.
- [Miller,2010a] William M. Miller: *A Taxonomy of Expression Value Categories*.10-0045=N3055
- [Strachey, 1963]: D.W. Barron, et al: *The main features of CPL*. The Computer Journal. 6 (2): 134. (1963)
- [Stroustrup,1986] B. Stroustrup: The C++ Programming Language. Addison-Wesley. 1986.
- [Ellis,1989]: M. Ellis and B. Stroustrup: *The Annotated C++ Reference Manual*. Addison-Wesley. 1989. ("The ARM").