ISO/IEC JTC 1/SC 22/WG14 N1280

2008-02-10 Douglas Walls

Attribute Names, Use Existing Practice

Don't introduce new features into C1X that are gratuitously different than existing practice. When the committee was discussing the C1X programming language standard charter N1250, I and others emphasized this point. It is captured in principle #13 of the C1X charter.

13. Unlike for C9X, the consensus at the London meeting was that there should be no invention, without exception. Only those features that have a history and are in common use by a commercial implementation should be considered. Also there must be care to standardize these features in a way that would make the Standard and the commercial implementation compatible.

The committee discussions noted this was one of the factors contributing to the slow adoption of the C99 standard. The purpose of this paper is to emphasize the point with some examples of where we did this in C99 and a similar example of where the committee appears to be doing it again with C1X.

I'll cite two examples where complaints were voiced that C99 was gratuitously different with existing practice at the time.

- Macros with a variable number of arguments. C99 introduced an identifier that was used in the macro definition to represent the macro's variable arguments `__VA_ARGS__`. gcc was cited in the papers that lead to the introduction of this feature in C99. However, gcc allowed the user to define the identifier used in the macro definition to represent the macro's variable arguments.
  - An example of how gcc defined macros with a variable number of arguments:
    `#define debug(format, args...) fprintf (stderr, format, args)`
    `args` is user defined, i.e. it can be spelled as the programmer chooses
  - An example of how C99 defined macros with a variable number of arguments:
    `#define debug(format, ...) fprintf (stderr, format, __VA_ARGS__)`
- `__func__` predefined identifier. Many implementations, gcc being one, defi ned this predefined identifier as `__FUNCTION__`. It has the same semantics as the `__func__` predefined identifier introduced into C99.

So where is the committee doing this with C1X. Look at N1273, Attributes, you will see the paper advocating the use of `stdc_` as a prefix for attribute names:

"One further syntactic point was discussed, relating to namespace reservation. It was generally agreed that the attributes themselves have names, and individual implementations should be free to extend that namespace. Existing practice has many more names than described in the table above, and it was agreed that any solution proposed by the committee that outlawed existing practice would not be well received. Therefore, it is proposed that the names used for standardized C attributes be prepended with “stdc_”; the remainder of the attribute namespace should remain as it is now: open to vendor invention. Does this prefix make the name too long? For example “stdc_warn_unused_result”. Any other suitable prefix would be acceptable."

Current implemenation spellings	**Proposed spelling in N1273**
`noreturn`	`stdc_noreturn`
`pure`	`stdc_pure`
`deprecated`	`stdc_deprecated`
`aligned/align`	`stdc_align`
...	...

Though well intentioned, I think this is actually misguided and in direct conflict with principle #13 of the C1X charter. There is no need to do this. The committee is not proposing to introduce attributes with different semantics that outlaw existing practice. Quite the contrary, the committee is trying to standardize existing practice. The committee should follow that existing practice by choosing the same names for the attributes the committee intends to standardize. By choosing to standardize the same attribute, with or without minor semantic differences, the committee will appear to again be making gratuitous changes from existing practice.

Existing programs will not be changed to the new spelling for the attribute just because it has become a part of C1X. The new spellings will be used only if the code must be ported to a different implementation that only supports the C1X spelling. The more common scenario will be that the implementation the code is being ported to already supports the current spelling, and there is no need or motivation to switch to the C1X spelling. The committee should use the current spelling when it defines these attributes in C1X. By doing so these programs approach standards conformance without changes.

I understand the committee is likely to introduce some minor semantic differences between how an attribute is currently implemented and how the committee defines it in C1X. Most of those differences will be the unintended consequences of the exiting implementation in dealing with edge cases that were not well defined and were not considered during the implementation. For the very few and rare cases where it is actually an important difference, the implementation will provide a means for the code base to chose to use either their old semantics or C1X semantics. Implementations have been coping with semantic differences like this for decades, typically by providing compile time flags to choose the semantics the program requires. The committee is not be outlawing existing practice when this occurs.