



Revise spelling of keywords

Jens Gustedt

► To cite this version:

Jens Gustedt. Revise spelling of keywords: proposal for C23. [Research Report] N2934, ISO JTC1/SC22/WG14. 2022. hal-02167870v4

HAL Id: hal-02167870

<https://inria.hal.science/hal-02167870v4>

Submitted on 6 Oct 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

2022-2-15

Revise spelling of keywords proposal for C23

Jens Gustedt
INRIA and ICube, Université de Strasbourg, France

Over time C has integrated some new features as keywords (some genuine, some from C++) but the naming strategy has not been entirely consistent: some were integrated using non-reserved names (**const**, **inline**) others were integrated in an underscore-capitalized form. For some of them, the use of the lower-case form then is ensured via a set of library header files. The reason for this complicated mechanism had been backwards compatibility for existing code bases. Since now years or even decades have gone by, we think that it is time to switch and to use the primary spelling.

This is a revision of papers N2368 and N2392 where we reduce the focus to the list of keywords that found consensus in the WG14 London 2019 meeting. Other papers will build on this for those keywords or features that need more investigation.

Changes in v3:

- Remove the requirement for implementations to have these keywords as macro names and adapt title and contents accordingly.
- Update Annex B.

Changes in v4:

- Move the changes for **false** and **true** to paper N2458 (now N2935).

Changes in v5:

- Realize the proposed changes as separate change instructions.
- Make the possibility of having the keywords as macros as a choosable option for WG14.

Changes in v6:

- Remove the option to disallow redefinition of the keywords by user code, WG14 had no consensus on this.
- Because of the objections from N2850, make the addition of **thread_local** a separate option.

Changes in v7: Small changes to put the proposed text in the final form as voted with large majority in favor by WG14.

- Don't mark headers as obsolescent, WG 14 doesn't want that.
- Don't have **thread_local** as a separate option, but include it directly back into the set.

1. INTRODUCTION

Several keywords in current C2x have weird spellings as reserved names that have ensured backwards compatibility for existing code bases. As a reply to a previous paper in this series, replacing the keywords

_Alignas **_Alignof** **_Bool** **_Static_assert** **_Thread_local**

by the forms that are so far provided by some standard headers

alignas **alignof** **bool** **static_assert** **thread_local,**

respectively, has found consensus.

The new keywords **false** and **true** also found consensus, but their possible use in the preprocessor needs more provisions than given here. They are thus moved to N2935.

2. PROPOSED MECHANISM OF INTEGRATION

Many code bases use in fact the underscore-capitalized form of the keywords and not the compatible ones that are provided by the library headers. Therefore we need a mechanism that makes a final transition to the new keywords seamless. We propose the following:

- Allow for the keywords to also be macros, such that implementations may have an easy transition.
- Don't allow user code to change such macros.
- Allow the keywords to result in other spellings when they are expanded in with `#` or `##` operators.
- Keep the alternative spelling with underscore-capitalized identifiers around for a while.

With this in mind, implementing these new keywords is in fact almost trivial for any implementation that is conforming to C17.

- 5 predefined macros (7 when adding **false** and **true**) have to be added to the startup mechanism of the translator. They should expand to similar tokens as had been defined in the corresponding library headers.
- If some of the macros are distinct to their previous definition, the library headers have to be amended with `#ifndef` tests. Otherwise, the equivalent macro definition in a header should not harm.

Needless to say that on the long run, it would be good if implementations would switch to full support as keywords, but there is no rush, and some implementations that have no need for C++ compatibility might never do this.

3. CONSIDERATIONS FOR THREAD LOCAL

Paper N2850 raised an issue that C's and C++'s thread local variables are not compatible. This is because in C++ the property if a thread local variable needs a constructor (or not) at thread initialization time is not necessarily known in all TU that see a declaration of that variable. C does not have constructors and so it only allows initialization of thread local variables with constant expressions.

It was thus suggested that implementations that combine both languages may distinguish between **thread_local** for the C++ variant and **_Thread_local** for the interoperable C variant. Discussion in the C/C++ compatibility group revealed that even as of today this approach is doomed to fail, because C already has the compatibility macro **thread_local** in `<threads.h>`. Thus even as today, implementations combining both languages have to address this problem differently, for example by enhancing their linkers with the knowledge if a thread local variable has a constant expression as initializer. The preferred solution that has recently be discussed with the C – C++ liaison group is to make that choice for the C or C++ ABI by using the established feature of specifying C linkage with **extern "C"**.

In a previous version of this paper we factored out the changes for **thread_local** as a separate option, but WG 14 was still in favor to make the change for that feature, too.

4. REFERENCE IMPLEMENTATION

To add minimal support for the proposed changes, an implementation would have to add definitions that are equivalent to the following lines to their startup code:

```
#define alignas      _Alignas
#define alignof      _Alignof
#define bool         _Bool
#define static_assert _Static_assert
#define thread_local _Thread_local
```

At the other end of the spectrum, an implementation that implements all new keywords as first-class constructs and also wants to provide them as macros (though they don't have to) can simply have definitions that are the token identity:

```
#define alignas      alignas
#define alignof      alignof
#define bool         bool
#define static_assert static_assert
#define thread_local thread_local
```

5. MODIFICATIONS TO THE STANDARD TEXT

This proposal implies a large number of trivial modifications in the text, namely simple text processing that replaces the occurrence of one of the deprecated keywords by its new version. These modifications are not by themselves interesting:

CHANGE 1. *In the whole document, replace all occurrences of the tokens*

_Alignas _Alignof _Bool _Static_assert _Thread_local

by their new forms

alignas alignof bool static_assert thread_local,

respectively.

5.1. Changes to the language clauses

This invalidates the previous alphabetic order of keywords in 6.4.1 and A.1.2:

CHANGE 2. *Reorder the lists of keywords in 6.4.1 and A.1.2 alphabetically.*

Since we want to enable user code to continue to use the existing spellings, we introduce them as “alternative spellings”.

CHANGE 3. *Add a new paragraph after 6.4.1 p2:*

2’ The following table provides alternate spellings for certain keywords. These can be used wherever the keyword can.

FOOTNOTE[These alternative keywords are obsolescent features and should not be used for new code.]

<u>keyword</u>	<u>alternative spelling</u>
alignas	<i>_Alignas</i>
alignof	<i>_Alignof</i>
bool	<i>_Bool</i>
static_assert	<i>_Static_assert</i>
thread_local	<i>_Thread_local</i>

The spelling of these keywords and their alternate forms inside expressions that are subject to the # and ## preprocessing operators is unspecified.

FOOTNOTE [The intent of these specifications is to allow but not to force the implementation of the correspondig feature by means of a predefined macro.]

5.2. Interaction with legacy code

Clause 6.10.8 Predefined macro names

CHANGE 4. Add the following to p2:

*None of these macro names, nor the identifiers **defined** or **__has_c_attribute**, shall be the subject of a **#define** or a **#undef** preprocessing directive. Any other predefined macro names shall begin with a leading underscore followed by an uppercase letter or a second underscore or shall be any of the identifiers **alignas**, **alignof**, **bool**, **static_assert**, or **thread_local**.*

5.3. Changes to the library clauses

Since the new keywords have previously been macros defined by headers, we have to update these headers.

Clause 7.2 <assert.h>

CHANGE 5. Remove p3: ~~The macro **static_assert**...~~

Clause 7.15 <stdalign.h>

CHANGE 6. Replace the content of clause 7.15 by

The header <stdalign.h> provides no content.

Clause 7.18 <stdbool.h>

CHANGE 7. Replace p1 by

The header <stdbool.h> defines ~~four~~three macros.

CHANGE 8. Remove p2: ~~The macro **bool**...~~

6. QUESTION FOR WG14

QUESTION. Does WG14 want to integrate the changes as proposed in N2934 into C23?

Acknowledgement

We thank Joseph Myers, JeanHeyd Meneide and Aaron Ballmann for feedback and discussions.