ISO/IEC JTC 1/SC 22/WG14
WG 21, SG 22

N2884 v6
P2310R1

2021-12-31

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 be 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 to the primary spelling.
This is a revsion of papers to 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 N2885).

Changes in v5:

— Realize the proposed changes as separate change instructions.
— Make the possibility of having the keywords as macros as a chosable 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.

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 N2885.

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. 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
```

4. 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**

by their new forms

alignas **alignof** **bool** **static_assert**

respectively.

4.1. 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.]
>
> | keyword | alternative spelling |
> |---|---|
> | **alignas** | **_Alignas** |
> | **alignof** | **_Alignof** |
> | **bool** | **_Bool** |
> | **static_assert** | **_Static_assert** |
>
> 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.]

4.2. 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**, or **static_assert** .

4.3. 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 obsolescent header <stdalign.h> provides no content.

Also update the corresponding entry for future library directions:

CHANGE 7. *Replace the content of clause 7.31.10 by*

The header `<stdalign.h>` is an obsolescent feature.

Clause 7.18 `<stdbool.h>`

CHANGE 8. *Replace p1 by*

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

CHANGE 9. *Remove p2:* ~~The macro **bool** ...~~

4.4. 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.

So the raised issue is not new and the proposed solution to distinguish the two approaches by the syntax of the keywords is currently not valid. Nevertheless we factor the changes for **thread_local** as a separate option, such that WG14 can make an informed decision if we want to make the proposed changes with respect to **thread_local** or not. Regardless of the outcome, the real causes of this issue have to be addressed in C++.

Add `thread_local` to the list of keywords

CHANGE 10 (OPTIONAL). *To the changes 1 – 4 add the new keyword* **thread_local** *as replacing alternate spelling* **_Thread_local**.

Clause 7.26 `<threads.h>`

CHANGE 11 (OPTIONAL). *In p3 remove the partial phrase:* ~~**thread_local** which expands to the keyword **_Thread_local**;~~

5. QUESTIONS FOR WG14

QUESTION 1. *Does WG14 want to integrate changes 1 – 9 as proposed in N2884 into C23?*

QUESTION 2. *Does WG14 want to integrate changes 10 and 11 as proposed in N2884 into C23?*

Acknowledgement

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