ISO/IEC JTC 1/SC 22/WG14 **N2393**

v 2

October 24, 2019

Make false and true first-class language features v.2

proposal for C2x

Jens Gustedt
INRIA and ICube, Université de Strasbourg, France

In its London 2019 meeting, WG14 has found consensus to elevate **false** and **true** to proper keywords. This is only a first step to make these constants first-class language features and to achieve a full compatibility with C++. Therefore we need also to change their type, namely to change them from **int** to **bool**.
Changes in v2: WG14 was not sympathetic to force these keywords also to be macros, so we remove the text corresponding to this idea. WG14 also was not in favor of the parts that proposed to introduce recommended practice and to add future language directions, so these are also removed.

1. INTRODUCTION

The Boolean constants **false** and **true** are a bit ambivalent because in C17 they expand to integer constants `0` and `1` that have type **int** and not **_Bool**. This is unfortunate when they are used as arguments to type-generic macros, because there they could trigger an unexpected expansion, namely for **int** instead of **_Bool**. Since for C++, these constants are of type **bool**, we propose to do it the same.
The integration of these constants as proper language constructs, also allows to provide a better feedback to programmers, where such constants seem to be used erroneously. In particular, diagnostics may be provided when they are used in arithmetic or used contrary to the intent, *e.g* as null pointer constants.

2. IMPACT

The change should not have a big impact on user code. In most contexts where these constants are used (assignment, arithmetic, comparison, non-prototyped function call), **bool** values will be promoted to **int**, anyhow. So in these "regular" contexts the result after promotion would be exactly the same, namely **int** values 0 and 1, respectively. As arguments to function calls that provide a prototype, there is no change either, since the values 0 and 1 are valid for any arithmetic type and so the value and type received by the function are exactly the same.
The change can have marginal impact on existing code, when the constants are used in **sizeof**, **alignas** or **_Generic** expressions. All should be relatively rare. The latter, **_Generic**, is a sought effect of this change, because we think that choosing **bool** for these constants is a much more natural choice and will surprise less. In any case, these usages are compile-time detectable and we expect that quality implementations can provide diagnostics during the transition phase to C2x.

3. CHANGES

On top of the "keywords" paper, the changes for this feature are quite minimal, in essence it is the replacement of the token **int** by **bool** in one place.

Forward references: common definitions <stddef.h> (7.19), the **mbtowc** function (7.22.7.2), Unicode utilities <uchar.h> (7.28).

6.4.4.5 Predefined constants

Syntax

1 *predefined-constant:*
false
true

Description

Some keywords represent constants of a specific value and type.

6.4.4.5.1 The **false** and **true** constants

Description

1 The keywords **false** and **true** represent constants of type ~~**int**~~ **bool** that are suitable for use as are integer literals. Their values are 0 for **false** and 1 for **true**.[86)]

6.4.5 String literals

Syntax

1 *string-literal:*
encoding-prefix$_{\text{opt}}$ " *s-char-sequence*$_{\text{opt}}$ "
encoding-prefix:
u8
u
U
L

s-char-sequence:
s-char
s-char-sequence s-char

s-char:
any member of the source character set except
the double-quote ", backslash \, or new-line character
escape-sequence

Constraints

2 A sequence of adjacent string literal tokens shall not include both a wide string literal and a UTF–8 string literal.

Description

3 A *character string literal* is a sequence of zero or more multibyte characters enclosed in double-quotes, as in "xyz". A UTF–*8 string literal* is the same, except prefixed by **u8**. A *wide string literal* is the same, except prefixed by the letter **L**, **u**, or **U**.

4 The same considerations apply to each element of the sequence in a string literal as if it were in an integer character constant (for a character or UTF–8 string literal) or a wide character constant (for a wide string literal), except that the single-quote ' is representable either by itself or by the escape sequence \', but the double-quote " shall be represented by the escape sequence \".

[86)] Thus, the keywords **false** and **true** are usable in preprocessor directives.

Annex M
(informative)
Change History

M.1 Fifth Edition

1 Major changes in this fifth edition (**__STDC_VERSION__** *yyyymm*`L`) include:

— add a one-argument version of **static_assert**, make it a keyword and deprecate the underscore-capital form

— harmonization with ISO/IEC 9945 (POSIX):

- extended month name formats for **strftime**
- integration of functions: **memccpy**, **strdup**, **strndup**

— harmonization with floating point standard IEC 60559:

- integration of binary floating-point technical specification TS 18661-1
- integration of decimal floating-point technical specification TS 18661-2
- integration of decimal floating-point technical specification TS 18661-4a

— the macro **DECIMAL_DIG** is declared obsolescent

— added version test macros to certain library headers

— added the attributes feature

— added **nodiscard**, **maybe_unused** and **deprecated** attributes

— change **bool**, **false** and **true** to keywords and make them type **bool**

— change **alignas**, **alignof** and **thread_local** to be keywords and deprecate the underscore-capital forms

M.2 Fourth Edition

1 There were no major changes in the fourth edition (**__STDC_VERSION__** `201710L`), only technical corrections and clarifications.

M.3 Third Edition

1 Major changes in the third edition (**__STDC_VERSION__** `201112L`) included:

— conditional (optional) features (including some that were previously mandatory)

— support for multiple threads of execution including an improved memory sequencing model, atomic objects, and thread-local storage (`<stdatomic.h>` and `<threads.h>`)

— additional floating-point characteristic macros (`<float.h>`)

— querying and specifying alignment of objects (`<stdalign.h>`, `<stdlib.h>`)

— Unicode characters and strings (`<uchar.h>`) (originally specified in ISO/IEC TR 19769:2004)

— type-generic expressions

— static assertions

— anonymous structures and unions

— no-return functions