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

N2718 v5
P2311R1

2021-1-20

Make false and true first-class language features
proposal for C23

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.
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.
Changes in v3: It was then observed in a discussion on the reflector, that the possible use of these predefined constants in the preprocessor needs some more precautions.
Changes in v4: Now that the type change has been integrated into C23, it remains to integrate the new keywords properly into all translation phases.
Changes in v5:
— Make it clear that the constants count as integer constant expressions.
— Synchronize the handling in the preprocessor with C++.
— Explicitly mark the macro `__bool_true_false_are_defined` as obsolescent and keep it as last remaining content in `<stdbool.h>`.

1. INTRODUCTION

The integration of Boolean constants **false** and **true** as proper language constructs, is meant to provide a better feedback to programmers for the use of these constants by the translantor or from debuggers. In particular, diagnostics will hopefully be provided when they are used in arithmetic or used contrary to the intent, *e.g* as null pointer constants.

2. IMPACT

A possible impact of changing **false** and **true** to keywords could be the use of these constants in preprocessing conditional expressions. Currently preprocessing arithmetic sees the existing macros from `<stdbool.h>` as signed values, and thus the result of expressions is merely consistent between the preprocessor and the rest of the language. When changing to keywords we should ensure that **false** and **true** may still be used in the preprocessor with the same semantics as before. This is done by enforcing the following:

— Other than for other keywords, **true** is automatically rewritten to pp-token `1` in preprocessor arithmetic.
— This ensures that preprocessor arithmetic uses signed values for these constants, such that results of such arithmetic remain the same between C17 and C2x.

3. REFERENCE IMPLEMENTATION

To add minimal support for the proposed changes, an implementation that does not yet want to implement **false** and **true** as full-featured keywords would have to add definitions that are equivalent to the following lines to their startup code:

```
#define false          ((bool)+0)
#define true           ((bool)+1)
```

Notice that these do not use the literals `0U` or `1U` because with that arithmetic with these constants in the preprocessor would be performed as unsigned integers. This would have the consequence that something like `-true` would result to `UINTMAX_MAX` in the preprocessor and `-1` otherwise.

4. CHANGES

Predefined constants need a little bit more effort for the integration, than the other keywords in N2654, because up to now C did not have named constants on the level of the language.

4.1. Syntax

We propose to integrate these constants by means of a new syntax term `predefined constant`. The text itself is then integrated as a clause with a subclause. This strategy is chosen because we expect another named constant, namely **`nullptr`**, to be added to C23, once the details for that have been sorted out.

CHANGE 1. *Add* **`false`** *and* **`true`** *into the alphabetic order of 6.4.1.*

CHANGE 2. *Add a new syntax item* predefined-constants *to the end of 6.4.4 p1, Constants.*

CHANGE 3. *Add a new clause 6.4.4.5 and subclause as follows.*

> ***6.4.4.5 Predefined constants***
>
> ***Syntax***
>
> *1 predefined-constant:*
> **`false`**
> **`true`**
>
> ***Description***
>
> *2 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`** *are integer constant expressions of type* **`bool`** *with value* 0 *for* **`false`** *and* 1 *for* **`true`**. *When used as operands of arithmetic operators, the effect is as if the values are promoted to type* **`int`**.
>
> *FOOTNOTE[Therefore, when used for arithmetic in translation phase 4,* **`false`** *and* **`true`** *are signed values and the result of such arithmetic will be consistent with results of later translation phases.]*

CHANGE 4. *Add to the end of* `6.6 p5`*:*

> *Predefined constants are constant expressions and count in the following as constants of their respective type and value and as constants with other properties as required in their respective clauses.*

4.2. Interaction with legacy code

There is still some code in field that redefines these keywords. When compiler versions for C23 come out, it would be important that there is no silent redefinition of types or values depending on which headers are included and in which order. Therefore we think that it is important to impose diagnostics whenever user code tries to undefine or redefine these new keywords. But how to do this is clearly a question of policy, so as for N2654 we leave the way to address these problems to the appreciation of WG14.

CHANGE 5 (OPTIONAL). *If the corresponding change for other new keywords from N2654 was adopted, add* **false** *and* **true** *to the list of tokens that should not be subject to* **#define** *or* **#undef**.

4.3. The bool type

Definitions of the **bool** type should now directly refer to the constants and make no fuzz about zero or non-zero values anymore.

CHANGE 6. *In 6.2.5 (Types) make the following change to p2:*

> *An object declared as type* **bool** *is large enough to store the values ~~0~~***false** *and ~~1~~***true**.

CHANGE 7. *In 6.3.1.2 (Boolean type) make the following change to p1:*

> *When any scalar value is converted to* **bool**, *the result is ~~0~~***false** *if the value compares equal to 0;(FNT) otherwise, the result is ~~1~~***true**.

The current state of conversion to the type **bool** makes several implicit references back and forth between conversions and the equality operator.[1] We think that the changes proposed here, give an opportunity to improve that situation.

CHANGE 8 (ALTERNATIVE). *In 6.3.1.2 (Boolean type) make the following change to p1 and remove the corresponding footnote:*

> *When any scalar value is converted to* **bool**, *the result is ~~0~~***false** *if the value ~~compares equal to 0~~is a zero (for arithmetic types) or null (for pointer types)~~;(FNT)~~ otherwise, the result is ~~1~~***true**.

4.4. Preprocessing

The tokens **false** and **true** need a specific exception during preprocessing, such that constructs such as the following do not have surprising results.

```
#if true
...
#endif
```

CHANGE 9. *In 6.10.1 p7, amend the follwing partial phrase:*

> *... all remaining identifiers other than* **false** *or* **true** *(including those lexically identical to keywords) are replaced with the pp-number 0, ...*

Because in a transition phase these new keywords might still have macro definitions, we also add them to the list for which the spelling after preprocessing is unspecified.

CHANGE 10. *In 6.4.1 p2' (as of N2654) make the following changes:*

[1] The process of converting a **long** to **bool** is *e.g* as follows: `1L` ⟹ `(1L == 0)` ⟹ `(1L == 0L)` ⟹ **false**.

The spelling of these keywords, ~~and~~ their alternate forms, and of **`false`** *and* **`true`** *inside expressions that are subject to the* **`#`** *and* **`##`** *preprocessing operators is unspecified.*

4.5. Changes to library clauses

Clause 7.18 `<stdbool.h>`

This header now holds no reasonable contents and should be removed after a time of adjustment.

CHANGE 11. *Replace the content of clause 7.18 by*

The obsolescent header `<stdbool.h>` *provides the obsolescent macro* **`__bool_true_false_are_defined`** *which expands to the integer constant 1.*

Also update the corresponding entry for future library directions:

CHANGE 12. *Replace the content of clause 7.31.12 by*

The header `<stdbool.h>` *and the macro* **`__bool_true_false_are_defined`** *are obsolescent features.*

Clause 7.26 `<threads.h>`

This header has several functions or macros that return **`bool`** values.

CHANGE 13. *In 7.17.5.1, 7.17.7.4 and 7.17.8.1 change the specification of return values to the keywords* **`false`** *and* **`true`** *where appropriate.*

5. QUESTIONS FOR WG14

QUESTION 1. *Does WG14 want to integrate changes 1 – 4, 6, 7, 9 – 13 as proposed in N2718 into C23?*

QUESTION 2. *Does WG14 want to integrate change 5 as proposed in N2718 for C23 similar to change 5 in N2654?*

QUESTION 3. *Does WG14 want to use the alternative change 8 instead of change 7 as proposed in N2718 for C23?*

Acknowledgement

We thank JeanHeyd Meneide and Aaron Ballmann as well as the C/C++ liaison study group for feedback and discussions.