

Proposal for C2x
WG14 N2572

Title: What we think we reserve
Author, affiliation: Aaron Ballman
Date: 2020-09-30
Proposal category: Modifying existing normative requirements
Target audience: Users

Abstract: The C standard normatively reserves identifiers and some of the reservations impose onerous requirements on programmers. The most severe requirements are generally unknown to programmers, not checked by tools, and demonstrate a disconnect between the C standards committee and the language as it is used by programmers.

Reply-to: Aaron Ballman (aaron@aaronballman.com)

Document No: N2572

Revises Document No: N2493

Date: 2020-09-30

Summary of Changes

N2572

- Added proposed wording
- Added an open question about double underscore reservations

N2493

- Switched to the idea of a potentially reserved identifier

N2409

- Original proposal

Introduction and Rationale

C does not have a syntactic feature for reserving identifiers. Instead, the standard makes sweeping identifier reservations using lexical patterns, such as identifiers starting with an underscore followed by an uppercase letter, and the C committee expects the entire C community to know and adhere to the reservations to avoid breaking code when adding new language or library features. However, some of the current reservations result in onerous requirements on the programming community that are often not reliably checked by implementations and tools, or honored by users.

What makes a reserved identifier?

Identifier reservations are unfortunately split into two different places within the standard. 7.1.3p1 gives what looks to be an exhaustive list of reserved identifiers, and 7.1.3p2 goes on to state: No other identifiers are reserved. However, you need to read p1 carefully to note that 7.31 Future Library Directions also includes a list of reserved identifiers that are reserved under entirely different circumstances. For instance, 7.1.3 talks about reserving identifiers only if their associated header is included, while 7.31p1 reserves identifiers regardless of what headers are included (if any).

7.1.3 Reserved Identifiers

Identifiers with two leading underscores or a leading underscore followed by a capital letter. However, this only applied in cases where the identifier is not lexically identical to a keyword.

Reserved	Unreserved
<code>int __foobar, _Foobar</code>	<code>#define _Generic(x)</code>

Identifiers that begin with an underscore at file scope.

Reserved	Unreserved
<code>int _foobar;</code>	<code>int func(void) { int _foobar; }</code>

Macro names and identifiers with external linkage that are specified in the C standard library clauses.

Reserved	Unreserved
<pre>#include <locale.h> int func(void) { const char *localeconv; }</pre>	<pre>int func(void) { const char *localeconv; }</pre>

This proposal does not propose any changes to these reserved identifiers.

7.31 Future Library Directions

The individual reservations make claims as to what kinds of identifiers are reserved (macro names, function names, etc.) and what header file is impacted. However, p1 makes it clear that all identifiers reserved from this subclause are reserved identifiers regardless of what header files are included, meaning that these rules apply to all C code. Further, reserving an identifier pattern for a given use has limited practical effect on the context under which the identifier is reserved. Reserving an identifier for any use effectively reserves it for all uses in a practical sense. For instance, reserving something for use as a macro name or enumeration constant practically ensures that the name cannot also be used as the identifier in a function declaration, and vice versa. In effect, these identifiers are reserved for all uses in C regardless of what header files (if any) are included, and so the identifier reservations are being listed below by pattern rather than by header or entity.

- is, to, str, mem, wcs, atomic_, memory_, memory_order_, cnd_, mtx_, thrd_, or tss_ followed by a lowercase letter
- E, FE_, LC_, SIG, SIG_, ATOMIC_, or TIME_ followed by an uppercase letter
- E followed by a number
- PRN or SCN followed by a lowercase letter or the uppercase letter X
- Identifiers starting with uint or int and ending with _t, or UINT or INT and ending with _MAX, _MIN, or _C
- cerf, cerfc, cexp2, cexpm1, clog10, clog1p, clog2, clgamma, ctgamma, optionally followed by f or l

While many of these reservations seem reasonable or even necessary, they have some far-reaching consequences for introducing undefined behavior in user programs. Consider the following examples:

```
enum structure { // reserved
    isomorphic, // reserved
    nonisomorphic
};
void memorize_secret( // reserved
    const char *string // reserved
);
struct toxicology { // reserved
    enum condition {
        cnd_clean, // reserved
        cnd_dirty // reserved
    } cnd;
};
```

```
};
#define ENTOMOLOGY 1 // reserved
#define SIGNIFICANT_RESULTS 1 // reserved
#define TIME_TO_EAT 1 // reserved
#define ATOMIC_WEIGHT .000001f // reserved
#define INTERESTING_VALUE_MIN 0 // reserved
```

While these identifiers may seem contrived, it does not stretch the imagination to believe that programmers will accidentally use reserved identifiers with relative frequency without realizing it. A survey of the most egregious prefix patterns demonstrates that there are a considerable number of English words prohibited from use in C currently: e (33,921 words), to (3,810 words), is (3,267 words), str (1,643 words), sig (470 words), and mem (231 words). A survey of compilers and static analyzers were unable to identify a single tool warning users about all forms of reserved identifiers, including ones from the Future Library Directions, though all of the tools surveyed were able to warn about varying subsets of the reserved identifiers. The tools surveyed were: Clang, Microsoft Visual Studio, GCC, ICC, CodeSonar, CppCheck, Coverity, QAC, and two unnamed static analysis tools (not all tools can be listed by name due to Terms of Service requirements).

Code in the Wild

Code in the wild seems to ignore the reservations from 7.31. It is trivial to find examples of identifiers in popular C projects that violate the reserved identifiers restrictions from 7.31. A brief survey of a few popular C projects doing a simple regular expression search over header files finds the following examples, with the reserved identifiers highlighted in red for clarity:

sqlite (<https://sqlite.org/index.html>)

```
int (*strlike)(const char*,const char*,unsigned int);
#define EP_Reduced 0x002000 /* Expr struct EXPR_REDUCEDSIZE bytes only */
void *token; /* id that may be used to recursive triggers */
```

Windows 10 SDK (<https://developer.microsoft.com/en-us/windows/downloads/windows-10-sdk>)

```
#define ERROR_SUCCESS 0L
typedef struct tagRECT
{
    LONG left;
    LONG top;
    LONG right;
    LONG bottom;
} RECT;
```

ReactOS (<https://github.com/reactos/reactos>)

```
extern int iso9660_level;
extern int iso9660_namelen;
struct directory_entry {
    ...
    unsigned int total_rr_attr_size;
    ...
};
struct chmcTopicEntry {
```

```

    UInt32 tocidx_offset;
    ...
};
#define POW2(stride) (!((stride) & ((stride)-1)))

```

libuv (<https://github.com/libuv/libuv>)

```

#define container_of(ptr, type, member) \
    ((type *) ((char *) (ptr) - offsetof(type, member)))
typedef enum {
    TCP = 0,
    UDP,
    PIPE
} stream_type;
# define ENABLE_EXTENDED_FLAGS 0x80

```

libiconv (<https://www.gnu.org/software/libiconv/>)

```

static inline int streq8 (const char *s1, const char *s2, char s28);
#define isxbase64(ch) ((ch) < 128 && ((xbase64_tab[(ch)>>3] >> (ch&7)) & 1))
#define EXPR_SIGNED(e) (_GL_INT_NEGATE_CONVERT (e, 1) < 0)

```

Proposal

The goal of the future language and library reservations is to alert C programmers of the potential for future standards to use a given identifier as a keyword, macro, or entity with external linkage so that WG14 can add features with less fear of conflict with identifiers in user's code. However, the mechanism by which this is accomplished is overly restrictive – it introduces unbounded runtime undefined behavior into programs using a future language/library reserved identifier despite there not being any actual conflict between the identifier chosen and the current release of the standard. While it may be appealing to ignore this as "harmless" undefined behavior because implementations would not change runtime behavior to benefit from this latitude, it does still pose a burden for users. For instance, coding standards will often have a blanket prohibition against instances of undefined behavior (such as Rule 1.3 in MISRA C:2012 [0] or MSC15-C in the CERT C Secure Coding Standard [1]).

Instead of making the future language/library identifiers be reserved identifiers, causing their use to be runtime unbounded undefined behavior per 7.1.3p1, we propose introducing the notion of a *potentially reserved identifier* to describe the future language and library identifiers (but not the other kind of reservations like `__name` or `_Name`). These potentially reserved identifiers would be an informative (rather than normative) mechanism for alerting users to the potential for the committee to use the identifiers in a future release of the standard. Once an identifier is standardized, the identifier stops being potentially reserved and becomes fully reserved (and its use would then be undefined behavior per the existing wording in C17 7.1.3p2). These potentially reserved identifiers could either be listed in Annex A/B (as appropriate), Annex J, or within a new informative annex. Additionally, it may be reasonable to add a recommended practice for implementations to provide a way for users to discover use of a potentially reserved identifier. By using an informative rather than normative restriction, the committee can continue to caution users as to future identifier usage by the standard without adding undue burden for developers targeting a specific version of the standard.

It is worth noting that some of the reserved identifiers in the Future Library Directions subclause are reserved for use by the implementation rather than solely for potential future standardization. Such an

identifier would continue to be a reserved identifier rather than converted to a potentially reserved identifier.

Open Questions

WG21 has slightly different rules for reserved identifiers in that they reserve identifiers which contain a double underscore rather than reserving identifiers which begin with a double underscore. This can potentially lead to accidentally using a reserved identifier in C++ within a header file originally targeting a C compiler. Does WG14 want to modify our reserved identifier rules to make identifiers containing double underscores be reserved in C as well?

Proposed Wording

The wording proposed is a diff from the committee draft of WG14 N2478 with WG14 N2448 applied. Green text is new text, while ~~red~~ text is deleted text.

Insert 6.4.2.1p5-8 in the Semantics section:

5 Some identifiers are reserved.

- All identifiers that begin with a double underscore (`__`) or begin with an underscore (`_`) followed by an uppercase letter are reserved for any use, except those identifiers which are lexically identical to keywords^{x)}.

- All identifiers that begin with an underscore are reserved for use as identifiers with file scope in both the ordinary and tag name spaces.

Other identifiers may be reserved, see 7.1.3.

6 If the program declares or defines an identifier in a context in which it is reserved (other than as allowed by 7.1.4), the behavior is undefined.

7 If the program defines a reserved identifier or attribute token described in 6.7.11.1 as a macro name, or removes (with `#undef`) any macro definition of an identifier in the first group listed above or attribute token described in 6.7.11.1, the behavior is undefined.

8 Some identifiers may be *potentially reserved*. A potentially reserved identifier is an identifier which is not reserved but is anticipated to become reserved by a future version of this document.

Insert a Recommended Practice section before the Implementation Limits section:

9 Implementations are encouraged to issue a diagnostic message when a potentially reserved identifier is declared or defined for any use to bring attention to a potential conflict when porting a program to a future revision of this document.

Add a footnote for p5:

x) Allows a reserved identifier that matches the spelling of a keyword to be used as a macro name by the program.

Add forward references to 7.1.3, 7.1.4, and 6.7.11.1 at the end of 6.4.2.1.

Modify 7.1.3p1:

Each header declares or defines all identifiers listed in its associated subclause, and optionally declares or defines identifiers listed in its associated future library directions subclause and identifiers which are always reserved either for any use or for use as file scope identifiers.

~~— All identifiers that begin with an underscore and either an uppercase letter or another underscore are always reserved for any use, except those identifiers which are lexically identical to keywords.¹⁹³⁾~~

~~— All identifiers that begin with an underscore are always reserved for use as identifiers with file scope in both the ordinary and tag name spaces.~~

— All potentially reserved identifiers (including ones listed in the future library directions) that are provided by an implementation are reserved for any use. No other potentially reserved identifiers are reserved.^{y)}

— Each macro name in any of the following subclauses (including the future library directions) is reserved for use as specified if any of its associated headers is included; unless explicitly stated otherwise (see 7.1.4).

— All identifiers with external linkage in any of the following subclauses (including the future library directions) and `errno` are always reserved for use as identifiers with external linkage.¹⁹⁴⁾

— Each identifier with file scope listed in any of the following subclauses (including the future library directions) is reserved for use as a macro name and as an identifier with file scope in the same name space if any of its associated headers is included.

Add a footnote for p1:

^{y)} A potentially reserved identifier becomes a reserved identifier when an implementation begins using it or a future standard reserves it, but is otherwise available for use by the programmer.

Remove footnote 193: ~~Allows identifiers spelled with a leading underscore followed by an uppercase letter that match the spelling of a keyword to be used as macro names by the program.~~

Remove 7.1.3p2-3.

Modify 7.27.1p2:

Drafting note: this change is intended to clarify the intent of the standard. Footnote 334 coupled with 7.31.16p1 suggests that implementations may add additional `TIME_` macros, but the standard does not state this explicitly as it does elsewhere. This change is intended to be editorial. See related changes to 7.31.16p1.

The feature test macro `__STDC_VERSION_TIME_H__` expands to the token `yyyymmL`. The other macros defined are `NULL` (described in 7.19);

`CLOCKS_PER_SEC`

which expands to an expression with type `clock_t` (described below) that is the number per second of the value returned by the clock function; and

`TIME_UTC`

which expands to an integer constant greater than 0 that designates the UTC time base. *Additional time base macro definitions, beginning with `TIME_` and an uppercase letter, may also be specified by the implementation.*³³⁴⁾

Modify footnote 334:

334) See “future library directions” (7.31.16). Implementations can define additional time bases, but are only required to support a real time clock based on UTC.

Modify 7.31.1p1:

The function names

<big list of function names>

and the same names suffixed with **f** or **l** are potentially reserved identifiers and may be added to the declarations in the <complex.h> header.

Modify 7.31.2p1:

Function names that begin with either **is** or **to**, and a lowercase letter are potentially reserved identifiers and may be added to the declarations in the <ctype.h> header.

Modify 7.31.3p1:

Drafting note: the intent is not to change the reservation behavior but to better explain the reservation; so this is an editorial change.

Macros that begin with **E** and a digit or **E** and an uppercase letter may be added to the macros defined in the <errno.h> header by a future revision of this document or by an implementation.

Modify 7.31.4p1:

Drafting note: the intent is not to change the reservation behavior but to better explain the reservation; so this is an editorial change.

Macros that begin with **FE_** and an uppercase letter may be added to the macros defined in the <fenv.h> header by a future revision of this document or by an implementation.

Modify 7.31.5p1:

Macros that begin with **DBL_**, **DEC32_**, **DEC64_**, **DEC128_**, **DEC_**, **FLT_**, or **LDBL_** and an uppercase letter are potentially reserved identifiers and may be added to the macros defined in the <float.h> header.

Modify 7.31.6p1-2:

1 Macros that begin with either **PRI** or **SCN**, and either a lowercase letter or **X** are potentially reserved identifiers and may be added to the macros defined in the <inttypes.h> header.

2 Function names that begin with **str**, or **wcs** and a lowercase letter are potentially reserved identifiers and may be added to the declarations in the <inttypes.h> header.

Modify 7.31.7p1:

Drafting note: the intent is not to change the reservation behavior but to better explain the reservation; so this is an editorial change.

Macros that begin with **LC_** and an uppercase letter may be added to the macros defined in the <locale.h> header by a future revision of this document or by an implementation.

Modify 7.31.8p1:

Drafting note: the FP_ macros can be expanded by the implementation, but the same is not true for MATH_ macros, so p1 is split into two paragraphs.

Macros that begin with **FP_ or MATH_** and an uppercase letter may be added to the macros defined in the <math.h> header *by a future revision of this document or by an implementation.*

Insert a new 7.31.8p2:

Macros that begin with or MATH_ and an uppercase letter are potentially reserved identifiers and may be added to the macros defined in the <math.h> header.

Modify the existing 7.31.8p3-4:

3 Function names that begin with **is** and a lowercase letter *are potentially reserved identifiers and* may be added to the declarations in the <math.h> header.

4 The function names

<big list of function names>

and the same names suffixed with **f, l, d32, d64, or d128** *are potentially reserved identifiers and* may be added to the <math.h> header. The **cr** prefix is intended to indicate a correctly rounded version of the function.

Modify 7.31.9p1:

Drafting note: the intent is not to change the reservation behavior but to better explain the reservation; so this is an editorial change.

Macros that begin with either **SIG** and an uppercase letter or **SIG_** and an uppercase letter may be added to the macros defined in the <signal.h> header *by a future revision of this document or by an implementation.*

Modify 7.31.10p1:

Macros that begin with **ATOMIC_** and an uppercase letter *are potentially reserved identifiers and* may be added to the macros defined in the <stdatomic.h> header. Typedef names that begin with either **atomic_** or **memory_** and a lowercase letter *are potentially reserved identifiers and* may be added to the declarations in the <stdatomic.h> header. Enumeration constants that begin with **memory_order_** and a lowercase letter *are potentially reserved identifiers and* may be added to the definition of the `memory_order` type in the <stdatomic.h> header. Function names that begin with **atomic_** and a lowercase letter *are potentially reserved identifiers and* may be added to the declarations in the <stdatomic.h> header.

Modify 7.31.12p1:

Typedef names beginning with **int** or **uint** and ending with **_t** *are potentially reserved identifiers and* may be added to the types defined in the <stdint.h> header. Macro names beginning with **INT** or **UINT** and ending with **_MAX, _MIN, _WIDTH, or _C** *are potentially reserved identifiers and* may be added to the macros defined in the <stdint.h> header.

Modify 7.31.14p1:

Function names that begin with **str** or **wcs** and a lowercase letter **C** are potentially reserved identifiers and may be added to the declarations in the <stdlib.h> header.

Modify 7.31.15p1:

Function names that begin with **str**, **mem**, or **wcs** and a lowercase letter are potentially reserved identifiers and may be added to the declarations in the <string.h> header.

Modify 7.31.16p1:

Drafting note: the intent is not to change the reservation behavior but to better explain the reservation; so this is an editorial change. See related changes to 7.27.1.

Macros beginning with **TIME_** and an uppercase letter may be added to the macros in the <time.h> header by a future revision of this document or by an implementation.

Modify 7.31.17p1:

Function names, type names, and enumeration constants that begin with either **cnd_**, **mtx_**, **thrd_**, or **tss_**, and a lowercase letter are potentially reserved identifiers and may be added to the declarations in the <threads.h> header.

Modify 7.31.18p1:

Function names that begin with **wcs** and a lowercase letter are potentially reserved identifiers and may be added to the declarations in the <wchar.h> header.

Modify 7.31.19p1:

Function names that begin with **is** or **to** and a lowercase letter are potentially reserved identifiers and may be added to the declarations in the <wctype.h> header.

Acknowledgements

I would like to recognize the following people for their help in this work: Alex Gilding, Tom Honermann, Christof Meerwald, Clive Pygott, Robert Seacord, and David Svoboda.

References

[0] MISRA C 2012: Guidelines for the Use of the C Language in Critical Systems: March 2013. MIRA Limited, 2013.

[1] Seacord, Robert C. The CERT C Coding Standard: 98 Rules for Developing Safe, Reliable, and Secure Systems. Addison-Wesley, 2014.