Proposal for C2x
WG14 N2409

Title:	What we think we reserve
Author, affiliation:	Aaron Ballman, GrammaTech
Date:	2019-08-11
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: N2409
Date: 2019-08-11

Summary of Changes

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 in order 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. Further, some of these reservations are generally unknown to the greater user community, in part due to the fact that the standard is neither free nor easy to read. This results in an untenable situation where the C committee believes certain identifiers are available to be used by the committee because they've been reserved, while it is still possible to result in breaking significant amounts of user code.

Because of this disconnect between the standard and the community, I am proposing we modify our set of reserved names to match industry practice even if it means giving up rights to names we believe we already have rights to. I believe it is to our benefit to have realistic rules for reserving identifiers instead of a semi-antagonistic approach where we can break user code by making use of an identifier that's reserved via an overly-broad mechanism that users are unaware of and isn't generally caught by compilers or tools.

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
`int __foobar, _Foobar`	`#define _Generic(x)`

Identifiers that begin with an underscore at file scope.

Reserved	Unreserved
`int _foobar;`	`int func(void) { int _foobar; }`

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

Reserved	Unreserved

`#include <locale.h>`
`int func(void) {`
`  const char *localeconv;`
`}`
	`int func(void) {`
	`  const char *localeconv;`
	`}`

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 (33921 words), to (3810 words), is (3267 words), str (1643 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, casting into doubt whether WG14 can rely on these reservations to prevent breaking user code. 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:

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 C standard needs to carve out reserved identifiers, so it is not proposed that the standard remove all identifier reservations. However, the standard should only reserve identifiers if there is a reasonable chance that the reservation is sufficiently well-known to users and checked by tools to justify the belief that code will not be broken when the standard adds a new identifier based on that reservation. Even should tooling improve to cover all cases of reserved identifiers, the set of identifiers reserved by the C committee is far too large and intrusive. It is a primary responsibility of the committee not to break existing code [0]. It does not serve the user community or the standards body to claim to reserve identifiers that the committee has no intention of using, such as every identifier starting with the letters is, str, or mem.

For every character added to the identifier pattern reservation, the number of English words that match will be reduced. The standard should either use exact-match identifiers or utilize patterns with both a prefix and suffix of sufficient length and complexity to reduce the likelihood of the pattern including English words or common abbreviations. Additionally, restricting the reservation to only apply when certain header files are included further reduces the risk of accidentally conflicting with existing user code. This paper proposes changing the identifier reservations by:

- retaining what is already reserved by 7.1.3,
- modifying 7.31p1 to only apply when the given header file is included,
- retaining the precise reserved identifiers from 7.31.1,
- modifying the pattern-based reserved identifiers from 7.31.10 to be int or uint followed by a number and ending with _t or INT or UINT followed by a number and ending with _MAX, _MIN, or _C,
- removing or modifying the pattern-based reserved identifiers from 7.31.2, 7.31.3, 7.31.4, 7.31.5, 7.31.6, 7.31.7, 7.31.8, 7.31.12, 7.31.13, 7.31.14, 7.31.15, 7.31.16, and 7.31.17.

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] http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2021.htm