What is new in C and POSIX?

Peter Eisentraut

2025-05-14

peter@eisentraut.org
https://peter.eisentraut.org/
@petereisentraut@mastodon.social

peter.eisentraut@enterprisedb.com
https://www.enterprisedb.com/
@edbpostgres@mastodon.social

Agenda

- context and background
- POSIX.1-2008
- POSIX.1-2024
- C11
- C23
- future and next steps

C and POSIX

- C = C compiler + C library
- POSIX = extended C library + shell
- Single UNIX Specification (SUS) = POSIX (w/XSI) + curses
    - predecessor: X/Open Portability Guide (XPG)

What is in C? / What is in POSIX?

C library (examples)	POSIX System Interfaces (examples)	XSI option (examples)
`fread()`, `fwrite()`, `printf()`, `FILE`, `errno`, `setlocale()`, `localeconv()`, `atoi()`, `malloc()`, `getenv()`, `system()`, `bsearch()`, `memcpy()`, `strcpy()`, `strcoll()`, `strtok()`, ...	C library plus ... `read()`, `write()`, `off_t`, `socket()`, `inet_ntop()`, `dlopen()`, `uselocale()`, `nl_langinfo()`, `setenv()`, `fork()`, `strdup()`, `strnlen()`, `strcoll_l()`, `strtok_r()`, `shmget()`, ...	`crypt()`, `gethostid()`, `getresuid()`, `lockf()`, `nice()`, `sync()`, ...

C vs. POSIX: integer types

Standard C:

- `intNN_t`, `uintNN_t` are all optional.
- (Implementation must only provide these types (8, 16, 32, 64) if it can.)
- `[u]int_leastNN_t`, `[u]int_fastNN_t` are required (8–64).
- `intptr_t`, `uintptr_t` are optional.

POSIX:

- `intNN_t`, `uintNN_t` are required (8–32).

XSI:

- `intptr_t`, `uintptr_t` are required.

C vs. POSIX: null pointers

Standard C:

- `0` and `(void *) 0` are *null pointer constants*.
- `NULL` is defined to a null pointer constant.
- So `#define NULL 0` is valid. (but dangerous)

POSIX:

- `NULL` must be defined as `(void *) 0`.
- Additionally, a pointer with all bits set to zero (e.g., `memset()`) is a null pointer.

Where does C come from (nowadays)?

standard: ISO/IEC 9899

working group: ISO/IEC JTC1 SC22 WG14

web: https://www.open-std.org/jtc1/sc22/wg14/

WG14 Graz (Austria), Feb. 2025

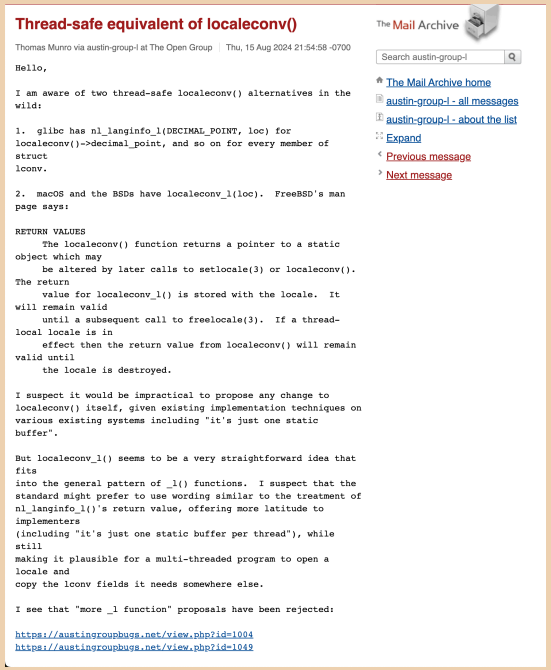
Where does POSIX come from (now)?

standard: IEEE Std 1003.1

working group: Austin Group
(IEEE + The Open Group (+ ISO/IEC)) (since 2001)

web: https://www.opengroup.org/austin/

no field trip photos but …



Thread-safe equivalent of localeconv()

Thomas Munro via austin-group-l at The Open Group | Thu, 15 Aug 2024 21:54:58 -0700

Hello,

I am aware of two thread-safe localeconv() alternatives in the wild:

1. glibc has nl_langinfo_l(DECIMAL_POINT, loc) for localeconv()->decimal_point, and so on for every member of struct
lconv.

2. macOS and the BSDs have localeconv_l(loc). FreeBSD's man page says:

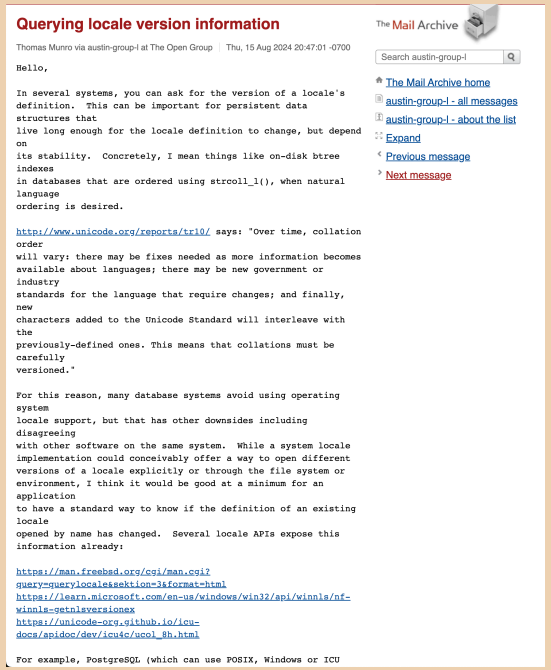
RETURN VALUES
     The localeconv() function returns a pointer to a static object which may
     be altered by later calls to setlocale(3) or localeconv(). The return
     value for localeconv_l() is stored with the locale. It will remain valid
     until a subsequent call to freelocale(3). If a thread-local locale is in
     effect then the return value from localeconv() will remain valid until
     the locale is destroyed.

I suspect it would be impractical to propose any change to localeconv() itself, given existing implementation techniques on various existing systems including "it's just one static buffer".

But localeconv_l() seems to be a very straightforward idea that fits
into the general pattern of _l() functions. I suspect that the standard might prefer to use wording similar to the treatment of nl_langinfo_l()'s return value, offering more latitude to implementers
(including "it's just one static buffer per thread"), while still
making it plausible for a multi-threaded program to open a locale and
copy the lconv fields it needs somewhere else.

I see that "more _l function" proposals have been rejected:

https://austingroupbugs.net/view.php?id=1004
https://austingroupbugs.net/view.php?id=1049



Querying locale version information

Thomas Munro via austin-group-l at The Open Group | Thu, 15 Aug 2024 20:47:01 -0700

Hello,

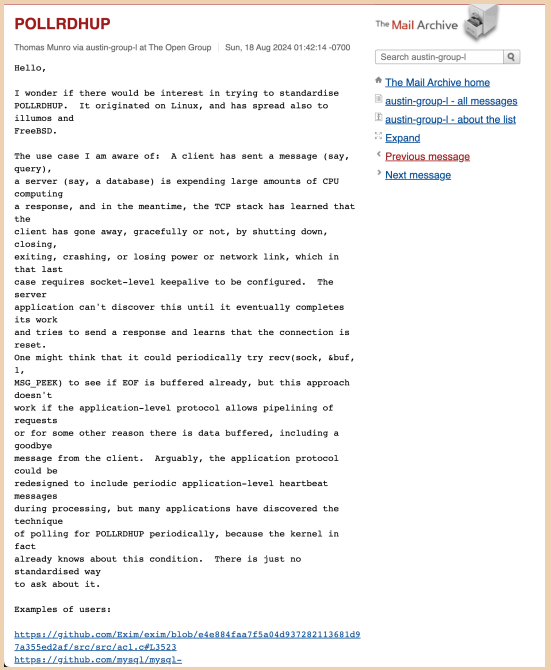
In several systems, you can ask for the version of a locale's definition. This can be important for persistent data structures that
live long enough for the locale definition to change, but depend on
its stability. Concretely, I mean things like on-disk btree indexes
in databases that are ordered using strcoll_l(), when natural language
ordering is desired.

http://www.unicode.org/reports/tr10/ says: "Over time, collation order
will vary: there may be fixes needed as more information becomes available about languages; there may be new government or industry
standards for the language that require changes; and finally, new
characters added to the Unicode Standard will interleave with the
previously-defined ones. This means that collations must be carefully
versioned."

For this reason, many database systems avoid using operating system
locale support, but that has other downsides including disagreeing
with other software on the same system. While a system locale implementation could conceivably offer a way to open different versions of a locale explicitly or through the file system or environment, I think it would be good at a minimum for an application
to have a standard way to know if the definition of an existing locale
opened by name has changed. Several locale APIs expose this information already:

https://man.freebsd.org/cgi/man.cgi?query=querylocale&sektion=3&format=html
https://learn.microsoft.com/en-us/windows/win32/api/winnls/nf-winnls-getnlsversionex
https://unicode-org.github.io/icu-docs/apidoc/dev/icu4c/ucol_8h.html

For example, PostgreSQL (which can use POSIX, Windows or ICU



POLLRDHUP

Thomas Munro via austin-group-l at The Open Group | Sun, 18 Aug 2024 01:42:14 -0700

Hello,

I wonder if there would be interest in trying to standardise POLLRDHUP. It originated on Linux, and has spread also to illumos and
FreeBSD.

The use case I am aware of: A client has sent a message (say, query),
a server (say, a database) is expending large amounts of CPU computing
a response, and in the meantime, the TCP stack has learned that the
client has gone away, gracefully or not, by shutting down, closing,
exiting, crashing, or losing power or network link, which in that last
case requires socket-level keepalive to be configured. The server
application can't discover this until it eventually completes its work
and tries to send a response and learns that the connection is reset.
One might think that it could periodically try recv(sock, &buf, 1,
MSG_PEEK) to see if EOF is buffered already, but this approach doesn't
work if the application-level protocol allows pipelining of requests
or for some other reason there is data buffered, including a goodbye
message from the client. Arguably, the application protocol could be
redesigned to include periodic application-level heartbeat messages
during processing, but many applications have discovered the technique
of polling for POLLRDHUP periodically, because the kernel in fact
already knows about this condition. There is just no standardised way
to ask about it.

Examples of users:

https://github.com/Exim/exim/blob/e4e884faa7f5a04d937282113681d97a355ed2af/src/src/acl.c#L3523
https://github.com/mysql/mysql-

C versions

Revision	Publication	`__STDC_VERSION__`
C2y	future	`> 202311L`
C23	ISO/IEC 9899:2024	`202311L`
C17	ISO/IEC 9899:2018	`201710L`
C11	ISO/IEC 9899:2011	`201112L`
C99	ISO/IEC 9899:1999	`199901L`
C95	ISO/IEC 9899:1990/Amd 1:1995	`199409L`
C90	ISO/IEC 9899:1990	—
C89	ANSI X3.159-1989	—
K&R	*The C Programming Language*	—

POSIX versions

POSIX name	IEEE standard	ISO standard	TOG SBS issue	SUS version	*nix mark
POSIX.1-2024	IEEE Std 1003.1-2024	pending (FDIS)	Issue 8	?	?
(POSIX.1-2017)	IEEE Std 1003.1-2017	(corrigenda)			
POSIX.1-2008	IEEE Std 1003.1-2008	ISO/IEC 9945:2009	Issue 7	SUSv4	UNIX V7
	IEEE Std 1003.1-2004	(corrigenda)			
POSIX.1-2001	IEEE Std 1003.1-2001	ISO/IEC 9945:2002	Issue 6	SUSv3	UNIX 03
… pre Austin Group / timelines bifurcate ↓ …					

read online:

- POSIX.1-2024: https://pubs.opengroup.org/onlinepubs/9799919799/
- POSIX.1-2017: https://pubs.opengroup.org/onlinepubs/9699919799/
- POSIX.1-2008: https://pubs.opengroup.org/onlinepubs/9699919799.orig/

What is new in POSIX.1-2008

- locales: `locale_t`, `newlocale()`, `freelocale()`, `strcoll_l()`, various `*_l()` [req. by PG17] ✅
- various: `strnlen()` ✅, ...

- "at" functions: `fchmodat()`, `linkat()`, `mkdirat()`, `openat()`, `renameat()`, ...
- use file descriptors: `dprintf()`, `dirfd()`, `fopendir()`, ...
- memory as file: `fmemopen()`, `open_memstream()`, ...
- various: `mkdtemp()` ✅, `scandir()`, `stpcpy()`, `strndup()`, `strsignal()` ✅, ...
- removed `mktemp()`, removed `usleep()`

Conclusion POSIX.1-2008

- currently de-facto required by PostgreSQL
- but many interfaces not available on Windows

What is new in POSIX.1-2024

- all new C17 library functions
- gettext: `bindtextdomain()`, `gettext()`, `ngettext()`, … ✅
- `ppoll()` ✅
- `strlcat()`, `strlcpy()` ✅
- `qsort_r()` (= `qsort_arg()`) (✅)
- `memmem()`
- `getlocalename_l()`

(cont'd.)

What is new in POSIX.1-2024, cont'd.

- `posix_close()`
- `asprintf()`
- `mkostemp()`
- `dup3()`, `accept4()`, `pipe2()` (`*_CLOEXEC`, `*_CLOFORK`)
- `beNNtoh()`, `htobeNN()` (16, 32, 64)
- `getresuid()`, `getresgid()`
- `secure_getenv()`
- `str2sig()`, `sig2str()`

What is old in POSIX.1-2024

removed:

- `gettimeofday()` ❌ (use `clock_gettime()`)
- `isascii()` ❌
- `getitimer()` ❌/`setitimer()` ❌ (use `timer_gettime()`, `timer_settime()`)
- `ulimit()` (use `getrlimit()`/`setrlimit()`)

Conclusion POSIX.1-2024

- standardizes a lot of existing practices
- not widely available yet

What is new in C11

- alignment ➡️
- noreturn ➡️
- static assertions ➡️
- benign typedef redefinitions ➡️
- anonymous structures and unions ➡️
- generic selection ➡️
- threads ➡️
- atomic types ➡️

(cont'd.)

What is new in C11, cont'd.

- improved Unicode support
- improved floating-point support
- improved complex values support
- bounds checking (Annex K)
- VLAs optional
- removed `gets()`

C keywords: `thing` vs. `_Thing`

- `thing`: new keyword, bad
- `_Thing`: no name collision, but ugly
- `_Thing` +
  `#define thing _Thing` +
  `<stdthing.h>`: better

C11: alignof

```
#include <stdalign.h>

size_t x = alignof(int);      // probably 4
size_t y = alignof(double);   // maybe 8
```

(also C++11)

C11: alignas

```
#include <stdalign.h>

/* page_buffer must be adequately aligned */
union
{
    char        buf[QUEUE_PAGESIZE];
    AsyncQueueEntry align;
}           page_buffer;

alignas(AsyncQueueEntry) char page_buffer[QUEUE_PAGESIZE];
```

(also C++11)

C11: alignas

```
#include <stdalign.h>

typedef union PGIOAlignedBlock
{
#ifdef pg_attribute_aligned
   pg_attribute_aligned(PG_IO_ALIGN_SIZE)
#endif
   char        data[BLCKSZ];
   double      force_align_d;
   int64       force_align_i64;
} PGIOAlignedBlock;

typedef struct PGIOAlignedBlock
{
   alignas(PG_IO_ALIGN_SIZE) char data[BLCKSZ];
} PGIOAlignedBlock;
```

(also C++11)

C11: noreturn

```
#include <stdnoreturn.h>

noreturn extern void ReThrowError(ErrorData *edata);
```

(clashes with `__attribute__((noreturn))`/`__declspec(noreturn)`!
⇒ `pg_noreturn` in PG18)

(not C++)

C11: static_assert

```
#include <assert.h>

static_assert(2 * TAR_BLOCK_SIZE <= BLCKSZ,
              "BLCKSZ too small for 2 tar blocks");
```

(also C++11)

Note: This is a *declaration*, not a *statement*.

C11: benign typedef redefinitions

```
typedef struct PlannerInfo PlannerInfo;

// redefinition now ok
typedef struct PlannerInfo PlannerInfo;
warning: redefinition of typedef 'PlannerInfo'

// conflict, still error
typedef struct IndexPath PlannerInfo;
error: conflicting types for 'PlannerInfo'; have 'struct IndexPath'
```

(also C++)

C11: anonymous structs and unions

```
typedef struct ReorderBufferTXN
{
    bits32      txn_flags;
    ...
    union
    {
        TimestampTz commit_time;
        TimestampTz prepare_time;
        TimestampTz abort_time;
    }           xact_time;
    Snapshot    base_snapshot;
    ...
} ReorderBufferTXN;
```

(Note: C++ has anonymous unions (since C++98) but *not* anonymous structures.)

C11: generic selection

```
uint32 pg_nextpower2_32(uint32 num);
uint64 pg_nextpower2_64(uint64 num);

#define pg_nextpower2(num) \
    _Generic((num), \
        int32: pg_nextpower2_32, \
        uint32: pg_nextpower2_32, \
        int64: pg_nextpower2_64, \
        uint64: pg_nextpower2_64)(num)
```

(not C++)

C11: threads

```c
#include <threads.h>

thread_local int status;

int do_something(void* s)
{
    // do something interesting
    thrd_exit(0);
}

int main()
{
    thrd_t tid;

    if (thrd_create(&tid, do_something, "extra data") != thrd_success)
    {
        // handle error
        return 1;
    }
    thrd_join(tid, NULL);
    return 0;
}
```

(and much more)

(also C++11)

C11: atomic types

```
#include <stdatomic.h>

_Atomic(int) x;  // makes new type
_Atomic int  y;  // qualified int
atomic_int   z;  // same

t = atomic_load_explicit(y, memory_order_relaxed);
atomic_store_explicit(x, r1, memory_order_relaxed);
```

(and much more)

(also C++11)

Path to C11: Compiler support

alignment	gcc 4.7 (g++ 4.8)	clang 3.2 (clang++ 3.0)	VS 2019 16.8 (C++ VS 2015)	✅
noreturn	gcc 4.7	clang 3.3	VS 2019 16.8	✅
static assertions	gcc 4.6 (g++ 4.3)	clang yes (clang++ 2.9)	VS 2019 16.8 (C++ VS 2015)	✅
typedef redefinition	gcc 4.6	clang 3.1	(VS 2019 ok)	✅
anonymous unions	gcc 4.6	clang yes	(VS 2019 ok)	✅
generic selection	gcc 4.9	clang yes	VS 2019 16.8	❌
threads (+ library)	gcc 4.9	clang 3.3	VS 2022 17.8	❌
atomic types	gcc 4.9	clang yes	VS experimental	❌

- Also supported by: Oracle Developer Studio 12.6, ICC ≥17.0.0, (some version of) AIX
- RHEL 8 = gcc 8, RHEL 7 = gcc 4.8, Debian oldstable = gcc 10, Debian oldoldstable = gcc 8; buildfarm min.: gcc 4.8.5, clang 3.9.1
- min. MSVC: PG16 = VS 2015, PG12 = VS 2013, PG8.3 = VS 2005

Path to C11: Possible proposal

PostgreSQL 18+1:

- require C11 (w/o generic, threads, atomics)
- (requires MSVC 2019)

PostgreSQL 18+x:

- require full C11 (threads?)
- (when support for RHEL 7 is dropped)

What is new in C23

language:

- `#elifdef`, `#elifndef`
- `#embed` ➡️
- `__has_include()`
- `nullptr` — unambiguous null pointer constant
- `typeof()` ➡️
- `auto` type inference: `auto x = 5;`
- `constexpr` — compile-time constants
- binary literals: `0b10101010`
- digit separators: `100'000'000`
- now real keywords: `true`, `false`, `alignas`, `alignof`, `bool`, `static_assert`, `thread_local`
- empty initialization: `{}`
- [[attributes]] ➡️
- removed K&R function declarations

(cont'd.)

What is new in C23, cont'd.

language:

- requires two's complement signed integers (from POSIX)
- fixed(?) `intmax_t`
- bit-precise integer types: `_BitInt(N)`
- decimal floating-point types: `_DecimalNN`
- typed enums: `enum flags : unsigned int { ... };`

(cont'd.)

What is new in C23, cont'd.

library:

- `memset_explicit()`
- `memccpy()` (from POSIX)
- `strdup()`, `strndup()` (from POSIX)
- checked integer operations: `ckd_add()`, `ckd_sub()`, `ckd_mul()` (`<stdckdint.h>`)
- bit utility functions: `stdc_count_ones()`, `stdc_leading_ones()`, etc. (`<stdbit.h>`)
- qualifier-preserving standard library functions ➡️
- `unreachable()`

C23: #embed

```
#define FILE_DH2048 \
"-----BEGIN DH PARAMETERS-----\n\
MIIBCAKCAQEA//////////JD9qiIWjCNMTGYouA3BzRKQJOCIpnzHQCC76mOxOb\n\
IlFKCHmONATd75UZs806QxswKwpt8l8UN0/hNW1tUcJF5IW1dmJefsb0TELppjft\n\
awv/XLb0Brft7jhr+1qJn6WunyQRfEsf5kkoZlHs5Fs9wgB8uKFjvwWY2kg2HFXT\n\
mmkWP6j9JM9fg2VdI9yjrZYcYvNWIIVSu57VKQdwlpZtZww1Tkq8mATxdGwIyhgh\n\
fDKQXkYuNs474553LBgOhgObJ4Oi7Aeij7XFXfBvTFLJ3ivL9pVYFxg5lUl86pVq\n\
5RXSJhiY+gUQFXKOWoqsqmj//////////wIBAg==\n\
-----END DH PARAMETERS-----\n"
```

```
const char FILE_DH2048[] = {
#embed "dhp2048.pem"
    , '\0'
};
```

(also C++26)

C23: typeof

```
extern void *copyObjectImpl(const void *from);

#ifdef HAVE_TYPEOF
#define copyObject(obj) ((typeof(obj)) copyObjectImpl(obj))
#else
#define copyObject(obj) copyObjectImpl(obj)
#endif
```

(≈decltype in C++11)

C23: typeof — find the problem

```
extern void *copyObjectImpl(const void *from);
#define copyObject(obj) ((typeof(obj)) copyObjectImpl(obj))

typedef struct SomeNode { int type; Oid data; } SomeNode;

SomeNode *
transformStuff(const SomeNode *node)
{
    SomeNode *result = copyObject(node);

    result->data = FOOOID;

    return result;
}
```

C23: typeof — find the problem

```
extern void *copyObjectImpl(const void *from);
#define copyObject(obj) ((typeof(obj)) copyObjectImpl(obj))

typedef struct SomeNode { int type; Oid data; } SomeNode;

SomeNode *
transformStuff(const SomeNode *node)
{
    SomeNode *result = copyObject(node);

    result->data = FOOOID;

    return result;
}
```

```
warning: initializing 'SomeNode *' (aka 'struct SomeNode *') with an expression of
type 'typeof (node)' (aka 'const struct SomeNode *') discards qualifiers
```

C23: typeof_unqual

```
extern void *copyObjectImpl(const void *from);

#ifdef HAVE_TYPEOF_UNQUAL
#define copyObject(obj) ((typeof_unqual(*(obj)) *) copyObjectImpl(obj))
#else
#define copyObject(obj) copyObjectImpl(obj)
#endif
```

C23: [[attributes]]

`[[attribute]]` syntax instead of `__attribute__((...))`, `__declspec(...)`,...

```
[[noreturn]] extern void ReThrowError(ErrorData *edata);

[[nodiscard]] extern List *lappend(List *list, void *datum);

[[deprecated("use SPI_prepare_extended")]]
extern SPIPlanPtr SPI_prepare_cursor(const char *src, int nargs,
    Oid *argtypes, int cursorOptions);

[[maybe_unused]] int npages = 0;
```

also: `[[fallthrough]]`, `[[unsequenced]]`, `[[reproducible]]`

implementation extensions: `[[clang::always_inline]] foo();`

👉 unsupported or unrecognized attributes are **silently ignored**

(also C++11, C++14, C++17)

C23: qualifier-preserving functions

```
// pre-C23
char *strstr(const char *haystack, const char *needle);

static int find_foo(const char *in)
{
    char *p = strstr(in, "foo");  // oops, lost const

    ...
}
```

C23: qualifier-preserving functions

```
// C23 new
/*QChar*/ *strstr(/*QChar*/ *haystack, const char *needle);

static int find_foo(const char *in)
{
    // return type is now actually const char * here
    const char *p = strstr(in, "foo");

    ...
}
```

C23: qualifier-preserving functions

sample implementation (incomplete)

```
#define strstr(haystack, needle) \
    _Generic((haystack), \
        const char *: (const char *) strstr(haystack, needle), \
        char *: (char *) strstr(haystack, needle) \
    )
```

C23: qualifier-preserving functions

```
/*QVoid*/ *bsearch(const void *key, /*QVoid*/ *ptr,
                   size_t count, size_t size,
                   int (*comp)(const void*, const void*));

/*QVoid*/ *memchr(/*QVoid*/ *ptr, int ch, size_t count);

/*QChar*/ *strchr(/*QChar*/ *str, int ch);

/*QChar*/ *strpbrk(/*QChar*/ *dest, const char *breakset);

/*QChar*/ *strrchr(/*QChar*/ *str, int ch);

/*QChar*/ *strstr(/*QChar*/ *haystack, const char *needle);
```

(also `w*` variants)

C23 implementation support

- compilers:
  - (gcc 14)
  - gcc 15 (default!)
  - clang 18
- libraries tbd.
- some features have existed as extensions

PostgreSQL's path to C23 😲

could optionally use some features, such as:

- typeof, typeof_unqual
- some attributes
- <stdbit.h>
- <stdckdint.h>

PostgreSQL 30 to require C23. 🤯

Conclusion

- PostgreSQL base: C99 (C11?) + ≈POSIX.1-2008
- New versions: C23 + POSIX.1-2024
- Information is public.
- Participation and changes are possible.

Bye / Questions / Contact

peter@eisentraut.org
https://peter.eisentraut.org/
@petereisentraut@mastodon.social

peter.eisentraut@enterprisedb.com
https://www.enterprisedb.com/
@edbpostgres@mastodon.social

Overtime: C2y: better octal literals

```
// The following literals all specify the same number.

int literal_octal_prefered          = 0o52;
int literal_octal_to_be_deprecated  = 052;
int literal_decimal                 = 42;
int literal_hex                     = 0x2A;
int literal_binary                  = 0b00101010;
```

(N3353)

Overtime: C2y: named loops

```
outer_ij:
for (int i = 0; i < IK; ++i)
{
  for (int j = 0; j < JK; ++j)
  {
    if (cond)
      break outer_ij;

    ...
  }
}
```

(N3355)

Overtime: C2y: if declarations

```
if (int x = get())
{
  // x is non-0 here
}
else
{
  // x is 0 here
}
```

(N3356)

Overtime: C2y: case ranges

```
switch (n)
{
  case 1 ... 10:
    something();
    break;
}
```

(N3370)

Overtime: C2y: defer

```
{
    void *p = malloc(10);
    defer free(p);
    ...
    something(p);
    ...
}
```

(N3489)