N3507
2025-02-27

Pitch for `#dialect` directive

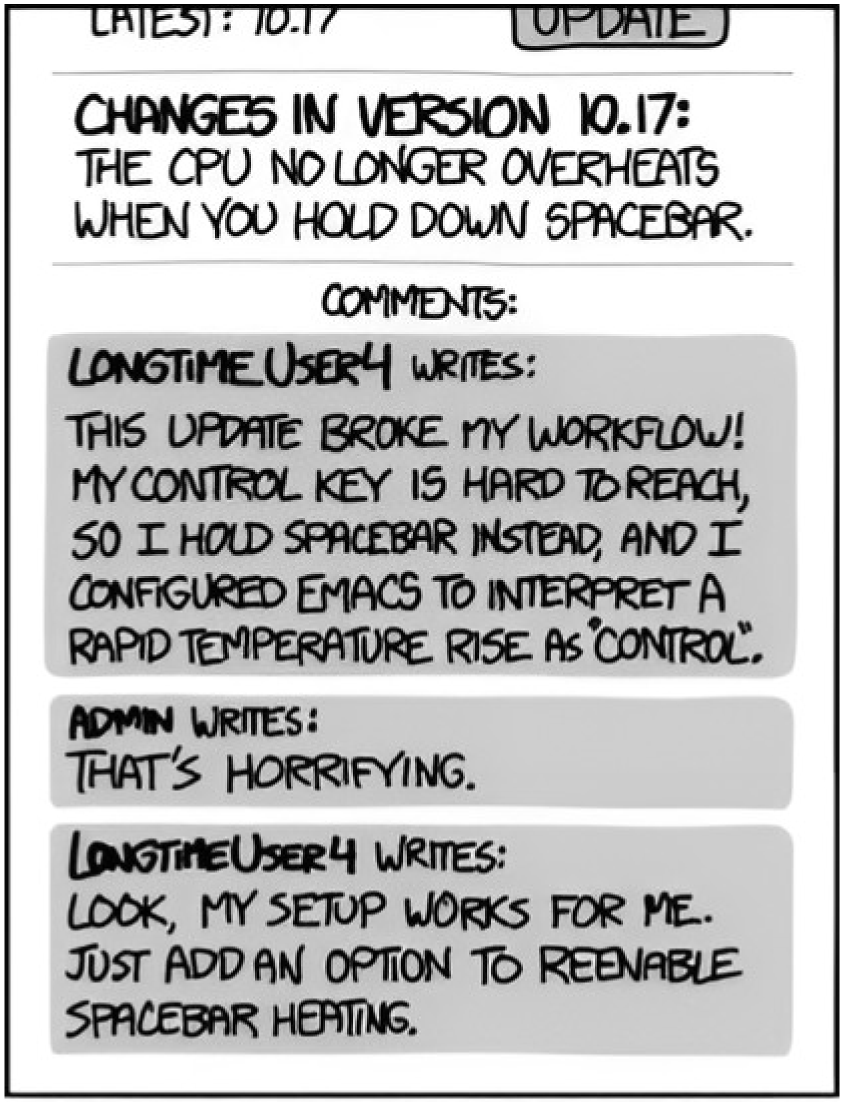
Jakub Łukasiewicz

Longer version: N3407

All examples, semantics, syntax, etc. used through the slides are to convey the **rough idea** behind the suggested mechanism, and will be subject to substantial changes in the actual design meant for real world usage.

PROBLEM

One’s bug is another’s feature


LATEST: 10.17
UPDATE
CHANGES IN VERSION 10.17:
THE CPU NO LONGER OVERHEATS WHEN YOU HOLD DOWN SPACEBAR.
COMMENTS:
LONGTIME USER4 WRITES:
THIS UPDATE BROKE MY WORKFLOW! MY CONTROL KEY IS HARD TO REACH, SO I HOLD SPACEBAR INSTEAD, AND I CONFIGURED EMACS TO INTERPRET A RAPID TEMPERATURE RISE AS "CONTROL".
ADMIN WRITES:
THAT'S HORRIFYING.
LONGTIME USER4 WRITES:
LOOK, MY SETUP WORKS FOR ME. JUST ADD AN OPTION TO REENABLE SPACEBAR HEATING.
EVERY CHANGE BREAKS SOMEONE'S WORKFLOW.

https://xkcd.com/1172/

- Removal of old style functions:
  - The upcoming gnu23 C standard is overdoing it with type-safety. (...) we historically have method tables for generic calls, which keeps the code small and easy to understand.
    - https://github.com/rsyslog/rsyslog/pull/5514/commits/08501b9
  - Recently I tried to build some of the old curses tetris games from the AUR but observed that gcc gives error messages that were formerly not there.
    - https://bbs.archlinux.org/viewtopic.php?id=295780
- Removing implicit int after a quarter of a century allowing it:
  - Build failure with gcc-14: error: type defaults to ‘int’ in declaration of ‘FILE_LICENCE’ [-Wimplicit-int] #1289 (…) everything, works when I just switch to gcc-13.
    - https://github.com/ipxe/ipxe/issues/1289
- General stance on busywork:
  - Fixing style violations while working on a real change as a preparatory clean-up step is good, but otherwise avoid useless code churn for the sake of conforming to style.
    - https://web.git.kernel.org/pub/scm/git/git.git/tree/Documentation/CodingGuidelines
- Compatibility issues with other dialects and forks:
  - Got a bunch of C code that uses *operator* as a variable name. Since this is a C++ keyword, it will be tough to import this into C++ code.
    - https://www.reddit.com/r/C_Programming/comments/1hy63n2/

Everybody wants talk with C, many without translator!

ANSI C
ISO C99
ISO C11
ISO C23

GCC

BERKELEY UPC UNIFIED PARALLEL C

eC

C*

cc65

OpenCilk

Z88DK

C+-

GO

TrapC

ZIG

Objective-C

Fil-C

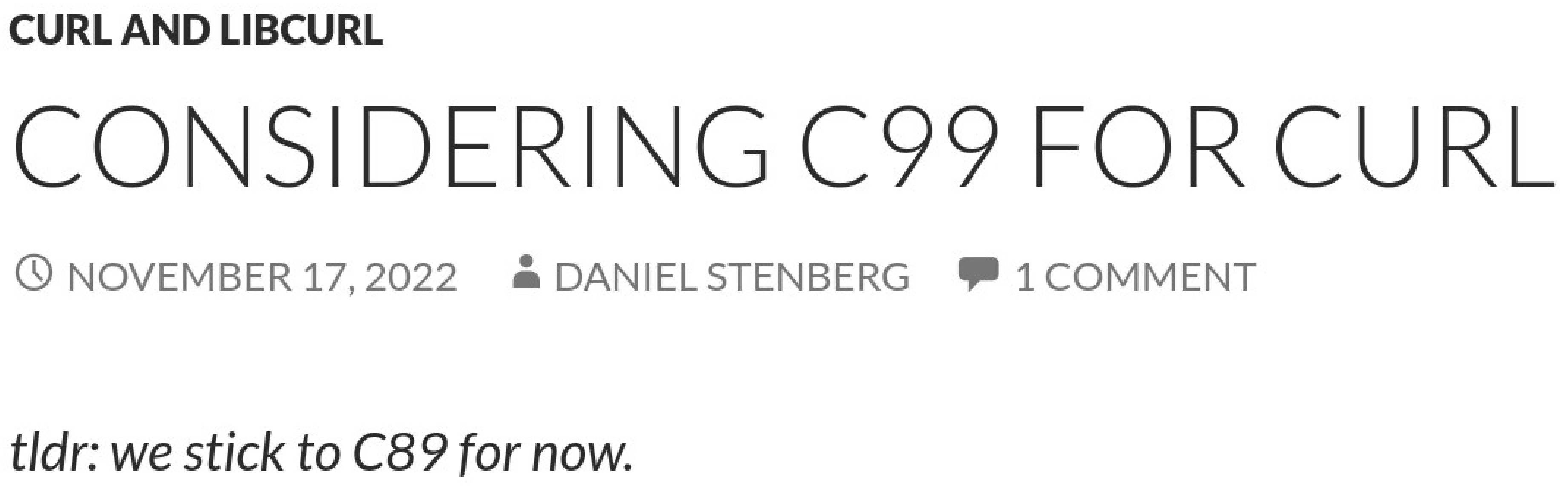
CompCert

CYCLONE

OpenCL™ C

Checked C

Folks like their old C standards



CURL AND LIBCURL

CONSIDERING C99 FOR CURL

NOVEMBER 17, 2022 DANIEL STENBERG 1 COMMENT

tldr: we stick to C89 for now.

https://daniel.haxx.se/blog/2022/11/17/considering-c99-for-curl/

SOLUTION

An in-code marker providing the compiler with the metadata about what dialect was assumed by the original author

Its primary goal would be to enable more targeted diagnostics, thus let users silence or lower severity of messages only for code marked as conforming to older standard, instead of whole project.

Syntax candidates:

- `#dialect GNU23`
- `#dialect C ver-min:99 ver-max:17`
- `#bind dialect C 89`
- `#pragma STDC dialect TrapC`
- `#version …`
- `#lang …`
- `#std …`
- `#features …`
- `#quirks …`
- `dialect "C>=99" { ... }`

et cetera

Rough idea for rules

For the sake of simplicity the examples across this document use rather loose set of rules for the directive. The final feature will be surely defined more robust*, something akin to:

- Unless specified in other way, if preceded only by comments, `#if` and `#define` directives, the first `#dialect` directive sets base dialect for the translation unit.
  - If no dialect is set explicitly before the first non comment, `#if` or `#define`, then value of dialect is implementation defined.
- No dialect based on higher version of C standard than the one set as the base dialect shall be used within the TU.
- `#dialect` directive shall appear only on file scope.
- Dialect value shall be carried into included files.
- After finishing processing of included file, dialect value from before inclusion shall be restored.

The mental model for the directive’s effect

#dialect

provides the translator with metadata

on which it might choose to act

not directly controls it

* Although guidelines how to act for dialects of standard C shall be provided

PRIOR ART

C

- feature test macros
- `#pragma`
  - `#pragma STDC FENV*` [Annex F]
  - `#pragma STDC SAFETY` [N3395]
  - `#pragma * diagnostics` / `#pragma warn` / …
- `-std`/`--standard`/`--cNN`/... compiler flags
- `#version` [N3176]
- `#bind` [N3190]

Ada	`pragma Ada_`*`N`*
CMake	`cmake_minimum_required`
D	`version`
GLSL	`#version`
HTML/XML	`<DOCTYPE>`
Perl	`use v`*`N`*
Python	via name of executable
Racket	`#lang`
Shell script	via name of executable
UserCSS	`@preprocessor`
VimL	`vim9script`

BENEFITS

- What problems does it solve?
- What improvements and fixes it enables?

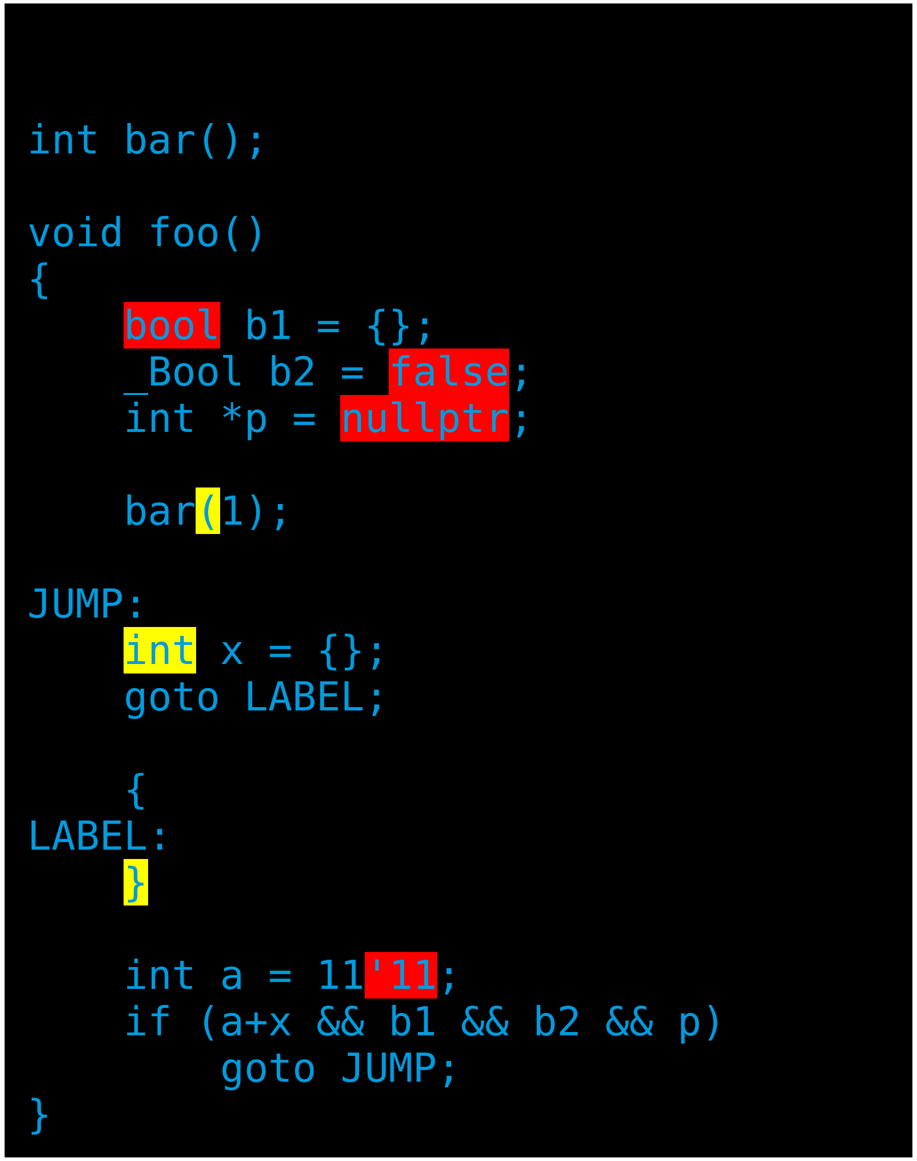
Disclaimer:
Examples on the following slides aren’t meant to be perfectly accurate, but to convey the idea.

(tiny) step towards simpler build systems

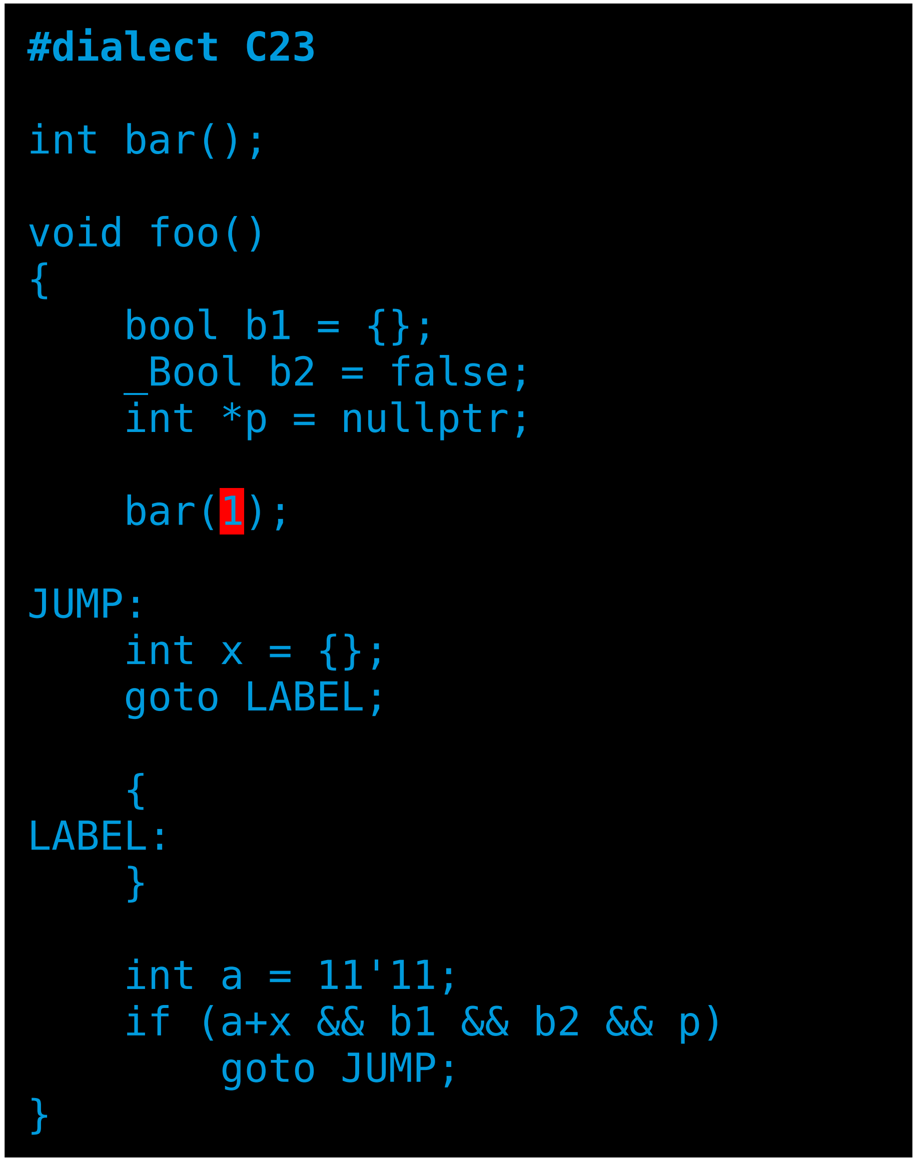
Especially for techniques like:

- unitary builds,
- X-Files,
- subtree dependencies,
- etc.

Correct diagnostics for single files



```
int bar();

void foo()
{
    bool b1 = {};
    _Bool b2 = false;
    int *p = nullptr;

    bar(1);

JUMP:
    int x = {};
    goto LABEL;

    {
LABEL:
    }

    int a = 11'11;
    if (a+x && b1 && b2 && p)
        goto JUMP;
}
```



```
#dialect C23

int bar();

void foo()
{
    bool b1 = {};
    _Bool b2 = false;
    int *p = nullptr;

    bar(1);

JUMP:
    int x = {};
    goto LABEL;

    {
LABEL:
    }

    int a = 11'11;
    if (a+x && b1 && b2 && p)
        goto JUMP;
}
```

(default linting by **clangd 19.1.6**)

No need for _UglyKeywords

```
#dialect C23
#include <lib.h>
const auto my_lib_countof = countof;


#dialect C2y
void func()
{
   auto n = countof (int[12]);
   auto l = my_lib_countof(/* args */);
}
```

Diagnostics when assigning null to pointers-to-non-optional

```
/* old_lib.c */
void foo(int *p)
{
   if (p) {
      *p = 12;
   }
}
```

```
/* new_lib.c */
void bar(int *p)
{
   *p = 12;
}
```

```
#define NULL ((optional void *)0)

#dialect C23
#include <old_lib.h>

#dialect C2y
#include <new_lib.h>

void func()
{
   foo(NULL); // warning
   bar(NULL); // error
}
```

Removal of `0` as null (N3426 alt.1)

```
/* old_lib.h */

#dialect C99

struct Foo {
    char *name;
    int number;
};

static inline void func()
{
    struct Foo x = { 0 };
    int *p = 0;
}
```

```
#dialect C2y
#include <old_lib.h> // fine

int main()
{
    func();
    struct Foo y = {};
    struct Foo z = {0}; // err
    int *ptr = 0;       // err
}
```

Avoids repeat of mess like with implicit `int`;
benefits available immediately for new, and refactored code.

Fixing/improving decay to pointer?

```
#dialect C99
void foo99(int *arr, int n);
void bar99(int arr[], int n);

#dialect C2y
void foo2y(int *ptr, int n);
void bar2y(int arr[], int n);

void foo()
{
    int a[] = { 1, 2, 3 };
    int *p = nullptr;

    foo99(a, countof a);  // no error, maybe warning
    bar99(a, countof a);
    foo99(p, 0);
    bar99(p, 0);           // no error, maybe warning

    foo2y(a, countof a);  // error, or at least a warning
    bar2y(a, countof a);
    foo2y(p, 0);
    bar2y(p, 0);           // error, or at least a warning
}
```

Make `#dialect` work like people **think** `extern "C"` works?

Possibility of an extension in C++:

```
/* foo.h */

#dialect C2y

// not needed: #if __cplusplus \ extern "C"

static void foo(size_t n, int arr[n]) // VM type!
{
   char *operator = malloc(89); // no cast
   printf("%zu\n", lengthof arr);
}
```

```
#dialect GNU++3a
#include <foo.h>  // actually compiles as C


auto example::func() -> void
{
   foo();
}
```

Standardizing popular extensions with different syntax

GNU C

```
asm (
 "idivl  %[divsrc]"
 : "=a" (quotient), "=d" (rem)
 : "d" (hi), "a" (lo),
 [divsrc] "rm" (divisor)
 :
);
```

ISO C

```
asm ??
```

Microsoft style

(in Intel’s "classic" compiler?)

```
asm {
     mov   edx, hi;
     mov   eax, lo;
     idiv  divisor
     mov   quotient, eax
     mov   tmp, edx;
}
```

Code examples from:
https://stackoverflow.com/a/35959859

Avoiding conflicts with extensions

What if C adds trap mechanism but not like e.g. from TrapC (N3423) dialect?

```
#dialect C3a

int func(size_t n)
{
    trap VLA_OVERFLOW { return 1; }
    int arr[n];
    // work on arr...
    return 0;
}
```

(similar to trap from POSIX shell)

Avoiding conflicts when defining ambiguous behaviours differently from what implementations did

For example N3203 standardizes order of expression evaluation.
While unlikely, what if some niche implementation had already defined it, but in reverse?

```
int G = 0;
int f() { return ++G; }
int g() { return (G *= 3); }

#dialect C2y

int s(int a, int b)
{
    return a + b;
}

int main()
{
    return s(f(), g());  // weird-cc notices these can have side effects
    /* warning:
     *  order of evaluation is left to right in C2y;
     *  weird-cc for previous standard used right to left
     */
}
```

and **more**

POLLS

- Does WG14 want any mechanism mitigating breaking changes?
  - Would WG14 consider a *proposal* for something along the lines of #dialect directive?

Thank you!

Acknowledgments:

- Eskil Steenberg (N3176), Ori Bernstein
- Bartosz Zielonka, and MTP team
- Joshua Cranmer for phrase "C89-spelled C2y"
- All WG14 participants in discussion on mailing list
- SWC audience