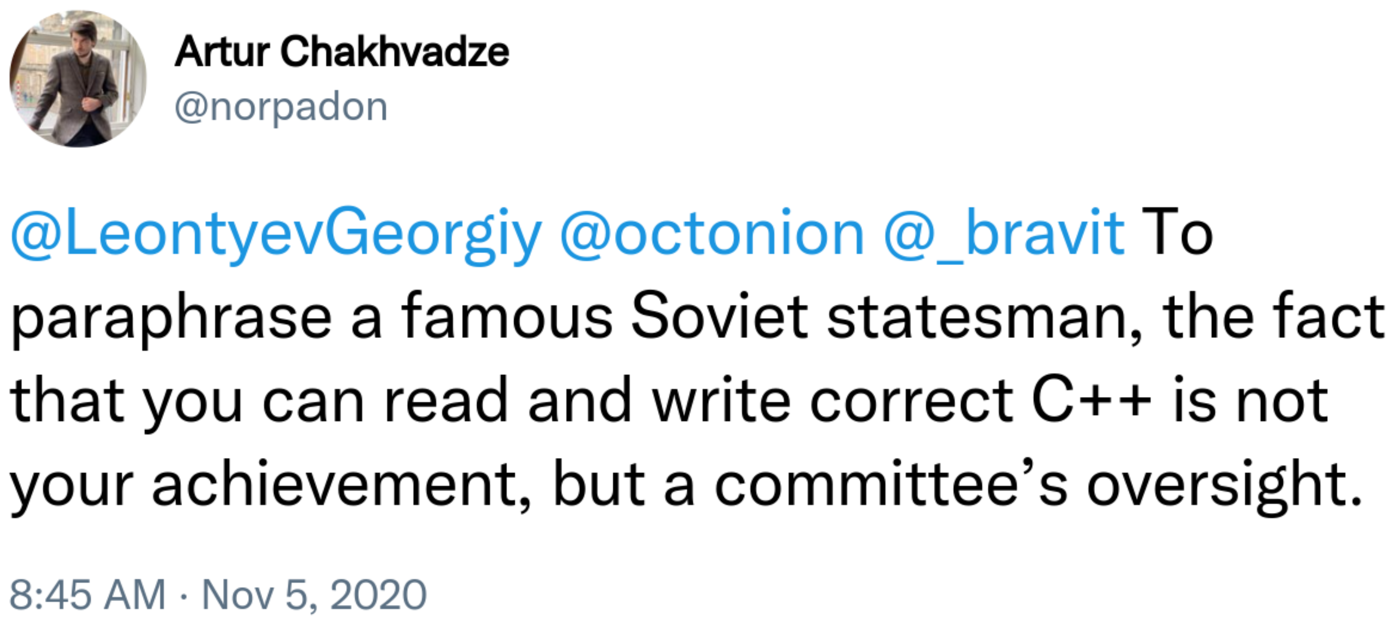
Стандарты C



Artur Chakhvadze
@norpadon

@LeontyevGeorgiy @octonion @_bravit To paraphrase a famous Soviet statesman, the fact that you can read and write correct C++ is not your achievement, but a committee's oversight.

8:45 AM · Nov 5, 2020

История стандартов C

The Development of the C Language, Dennis M. Ritchie

History of C - cppreference.com

Language Standards Supported by GCC (Using GCC)

- **1978**: K&R C

```
main(argc, argv)
int argc;
char** argv;
{
    printf("Hello world!\n");
    return 0;
}
```

- **1989**: ANSI X3.159-1989 / ISO/IEC 9899:1990, `-std=c89` / `-std=c90`
  - codified existing practices
  - new features: `volatile`, `enum`, `signed`, `void`, locales
  - from C++: `const`, function prototypes
- **1995**: ISO/IEC 9899 AM1, `-std=c95`
  - `__STDC_VERSION__`
  - wide and multibyte character support (`wctype.h`, `wchar.h`, additions and changes to stream I/O, etc)
  - digraphs, `iso646.h`
- **1999**: ISO/IEC 9899:1999, `-std=c99`
  - new features:
    * `bool`, `long long`
    * `stdint.h`, `inttypes.h`
    * `restrict`
    * compound literals
    * variable length arrays
    * flexible array members

    * designated initializers
    * variadic macros
    * complex numbers
    * trailing comma in enumerations

```
enum color
{
    RED,
    GREEN,
    BLUE,   /* invalid in C90, valid in C99 */
}
```

    * STDC_* pragmas
    * `__func__`
    * hexadecimal floating point format (%a), hh and `ll` length specifiers, monetary formatting in `lconv`, null return of `tmpnam`, null pointer in `setvbuf`, POSIX-like `strftime` specifiers
    * `isblank`
    * `va_copy`
    * `snprintf`
    * `_Exit`
    * `fenv.h`
    * `tgmath.h`

- from C++
    * `inline`
    * mix declarations and code, declarations in the init clause of the for loop

```
/* invalid in C90, valid in C99 */
int main()
{
    printf("Hello!\n");
    float s = 0.0;
    for(int i = 0; i < 3; i++)
    {
       printf("type in number: ");
       float x = 0.0;
       scanf("%f", &x);
       s += x;
    }
    printf("sum = %f\n", s);
}
```

    * // comments
    * universal character names in source code

- removed implicit functions and implicit `int`

```
int main()
{
    int a = 9;
    printf("%d\n", a); /* invalid in C99, no <stdio.h> */
    return fun();      /* invalid in C99, no fun declaration/definition*/
}

b = 11;                /* invalid in C99, no type in declaration */

int fun()
{
    printf("%d\n", b); /* invalid in C99, no <stdio.h> */
    return 0;
}
```

- **2011**: ISO/IEC 9899:2011, `-std=c11`
  - thread-aware memory model, `stdatomic.h`, `threads.h`
  - type-generic functions
  - `alignas` / `alignof`
  - `noreturn`
  - `_Static_assert`
  - extensions to complex and imaginary types
  - anonymous structures and unions
  - exclusive file open mode (`"x"`)
  - `quick_exit`
  - removed `gets`
  - bounds-checking interfaces (`strcpy_s` etc)
  - unicode: `char16_t`, `char32_t`, and `uchar.h`
- **2017**: ISO/IEC 9899:2018, `-std=c17`
  - "bugfix release", the only difference between the options is the value of `__STDC_VERSION__`
- **development**: C2X, `-std=c2x`
  - predefined boolean constants
  - nullptr and nullptr_t
  - decimal IEEE-754 floating-point types: `_Decimal32`, `_Decimal64`, `_Decimal128`
  - from C++:
    - ⋆ single-argument `_Static_assert`
    - ⋆ attributes `nodiscard`, `maybe_unused`, `deprecated`, `fallthrough`
    - ⋆ binary integer constant
    - ⋆ removal of old-style function definitions
    - ⋆ unnamed parameters in function definitions
  - library changes: POSIX functions `memccpy`, `strdup` / `strndup`, `asctime_r`, `ctime_r`, `gmtime_r`, `localtime_r`, `strftime` / `wcsftime` extensions; `timespec_getres`

Обзор стандарта C23

Some obscure C features

Lambdas, Nested Functions, and Blocks, oh my!

Why the C Language Will Never Stop You from Making Mistakes, перевод

C Portability Lessons from Weird Machines, H. Rabinowitz, Portable C

GNU расширения

Extensions to the C Language Family (Using GCC)

`-std=gnu90` / `-std=gnu99` / `-std=gnu11` / `-std=gnu17`

Designated initializers

```
int a[6] = { [4] = 29, [2] = 15 };

struct point { int x, y; };
struct point p = { .y = 2, .x = 1 };
```

Function attributes

Declaring Attributes of Functions (Using GCC)

Common Function Attributes (Using GCC)

```
int fn () __attribute__ ((warn_unused_result));
int foo ()
{
  if (fn () < 0) return -1;
  fn ();
  return 0;
}
```

```
static int max(int x, int y) __attribute__((always_inline));
static int max(int x, int y)
{
    return x > y ? x : y; // always inline if possible
}
```

Inline assembly

```
#define DebugBreak() asm("int $3")
```

Builtins

- Vector extensions
- __atomic Builtins
- Interger Overflow Builtins
- Target Builtins
- Other Builtins

```
if (__builtin_expect(ptr != NULL, 1))
  foo (*ptr);
```

Case ranges

```
case 'A' ... 'Z':
case 65 ... 90:
```

Другие расширения

- Clang language extensions
- Microsoft extensions to C and C++
    - Unnamed fields (Using GCC)

Unspecified & Undefined Behaviour

With Undefined Behavior, Anything is Possible

Неуточнённое поведение - Википедия

```
#include <stdio.h>

int f() {
  printf("In f\n");
  return 3;
}

int g() {
  printf("In g\n");
  return 4;
}

int sum(int i, int j) {
  return i + j;
}

int main() {
  return sum(f(), g());
}
```

Undefined behavior

```
int arr[4] = {0, 1, 2, 3};
int *p = arr + 5;  // undefined behavior for indexing out of bounds
p = 0;
int a = *p;        // undefined behavior for dereferencing a null pointer
int i = 5;
i = ++i + ++i;     // undefined behavior for unsequenced modification and use of i
```

Appendix J.1 и J.2 Стандарта

Undefined behavior can result in time travel, перевод

Что каждый программист на C должен знать об Undefined Behavior, часть 2, часть 3

Ещё раз о неопределённом поведении или «почему не стоит забивать гвозди бензопилой»

Undefined behavior, and the Sledgehammer Principle

Undefined Behavior in 2017 - Embedded in Academia

A Guide to Undefined Behavior in C and C++, part 2, part 3

- **Spatial Memory Safety Violations**. Accessing out-of-bounds storage and even creating pointers to that storage are UB in C.
- **Temporal Memory Safety Violations**. Any use of a memory location after its lifetime has ended. This includes addresses of automatic variables outliving these variables; use-after-free, where a dangling pointer is accessed for reading or writing; and double free.
- **Integer Overflow**. Integers can overflow in both directions. Signed integer overflow is UB; this includes `INT_MIN / -1`, `INT_MIN % -1`, negating `INT_MIN`, shift with negative exponent, left-shifting a one past the sign bit, and (sometimes) left-shifting a one into the sign bit. Division by zero and shift by >= bitwidth are UB in both the signed and unsigned flavors.
- **Strict Aliasing Violations**. The "strict aliasing rules" in C standard allow the compiler to assume that if two pointers refer to different types, they cannot point to the same storage. This enables nice optimizations but risks breaking programs that take a flexible view of types.
- **Alignment Violations**. RISC-style processors have tended to disallow memory accesses where the address is not a multiple of the size of the object being accessed. On the other hand, C programs that use unaligned pointers are undefined regardless of the target architecture.
- **Loops that Neither Perform I/O nor Terminate**. A loop in C code that neither performs I/O nor terminates is undefined and can be terminated arbitrarily by the compiler.
- **Data Races**. A data race occurs when a piece of memory is accessed by more than one thread, at least one of the accesses is a store, and the accesses are not synchronized using a mechanism such as a lock. Data races are UB in modern flavor of C (there is no semantics in older versions since those standards do not address multithreaded code).
- **Unsequenced Modifications**. In C, "sequence points" constrain how early or late a side-effecting expression such as `x++` can take effect. Unsequenced modifications of the same value, or an unsequenced modification and use of the same value, results in UB.

Warnings

Options to Request or Suppress Warnings

- `-w` Inhibit all warning messages.
- `-Werror` Make all warnings into errors.
- `-Wpedantic` / `-pedantic` Issue all the warnings demanded by strict ISO C and ISO C++; reject all programs that use forbidden extensions, and some other programs that do not follow ISO C.
- `-Wall` This enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning). Note that some warning flags are not implied by -Wall. Some of them warn about constructions that users generally do not consider questionable, but which occasionally you might wish to check for; others warn about constructions that are necessary or hard to avoid in some cases, and there is no simple way to modify the code to suppress the warning. Some of them are enabled by -Wextra but many of them must be enabled individually.
- `-Wextra` This enables some extra warning flags that are not enabled by `-Wall`.

Нестандартные стандартные библиотеки

Comparison of C/POSIX standard library implementations for Linux

- MSVCRT
- FreeBSD libc
- GNU libc
- musl libc
- uClibc, uClibc-ng
- dietlibc
- Newlib
- klibc