C Cheat Sheet

C23 / ISO/IEC 9899:2024
verified with `clang -std=c23 -Wall -Wextra`

HELLO, WORLD & COMPILING

```
#include <stdio.h>

int main(void)
{
    puts("Hello, World");
    return 0;   // 0 = success
}
```

```
clang -std=c23 -Wall -Wextra -Werror \
      -o hello hello.c
gcc   -std=c23 -O2 -flto -o app *.c -lm
```

Stages: preprocess `-E` → compile `-S` → assemble `-c` → link. Undefined symbols are link-time (`-lm` for `<math.h>`).

TYPES & LITERALS

```
bool  true false          // C23 keywords
char signed/unsigned char  // 1 byte
short int long long long
float double long double
int32_t uint64_t size_t ptrdiff_t
uintptr_t intmax_t        // <stdint.h>
_BitInt(12) unsigned _BitInt(3)
```

```
42  052(oct)  0x2A  0b1010(C23)
1'000'000(C23 sep)  42u 42L 42ULL
3.14  1.6e-19  0x1.8p1  3.14f  3.14L
'A'(int!)  "text"  u8"utf8"  L'w'
nullptr(C23)  NULL
```

Ranges: `<limits.h>` `INT_MAX`, `SIZE_MAX`, `CHAR_BIT`; `<float.h>` `DBL_EPSILON`.

Signed overflow = UB. Unsigned wraps mod 2^N. `sizeof('A')==sizeof(int)`. `1/2==0`.

CONVERSIONS & PROMOTIONS

Anything narrower than `int` promotes to `int`. Then the usual arithmetic conversions: floating beats integer, wider rank wins, and **at equal rank unsigned beats signed**.

```
-1 < 1u          // FALSE: -1 -> UINT_MAX
i < (int)strlen(s)  // cast size_t
(uint8_t)300     // 44, defined
(int)-3.99       // -3, toward zero
```

`_BitInt` never promotes. Build with `-Wsign-compare -Wconversion`.

OPERATORS & PRECEDENCE

`++ -- () [] . ->`	postfix
`++ -- + - ! ~ (T) * & sizeof alignof`	unary, R→L
`* / %`	multiplicative
`+ -`	additive
`<< >>`	shift
`< <= > >=`	relational
`== !=`	equality
`&  ^  |`	bitwise and/xor/or
`&&  ||`	logical, short-circuit
`?:  = op=  ,`	R→L, then comma

`flags & M == M` parses as `flags & (M==M)` — parenthesize. `1 << 2 + 3` is `1 << 5`.

One side effect per expression: `i = i++ + 1` and `a[i] = i++` are UB. Argument order is unspecified.

CONTROL FLOW

```
if (x > 0) { } else if (y) { } else { }

switch (c) {
case 1:
case 2:  n = 1; break;      // stacked
case 3:  n = 2; [[fallthrough]];
default: n = 0; break;      // always
}

for (size_t i = 0; i < n; ++i) { }
for (;;) { break; }
while (cond) { continue; }
do { } while (0);
while ((c = getchar()) != EOF) { }
goto cleanup;   // forward only
cleanup:
    free(buf);
```

No labeled `break`: use `goto`. `break` in a `switch` leaves the switch, not the loop.

FUNCTIONS

```
double area(double r);        // prototype
static int helper(int x) { return x; }
inline int max2(int a, int b);
[[noreturn]] void fatal(const char *m);
[[nodiscard]] int must_check(void);

int main(void) { }
int main(int argc, char *argv[]) { }
void f(size_t n, int a[static 1]);
int  g(size_t r, size_t c, int m[r][c]);
```

```
#include <stdarg.h>
int sum(int n, ...) {
    va_list ap; va_start(ap, n);
    int t = 0;
    for (int i = 0; i < n; ++i)
        t += va_arg(ap, int);
    va_end(ap); return t;
}
```

All pass-by-value. C23: `f()` == `f(void)`. Variadics promote `float`→`double`, `char`/`short`→`int`.

POINTERS

```
int v = 42;
int *p = &v;          // address-of
*p = 99;              // deref (write)
p[2] == *(p + 2)      // indexing IS arith
ptrdiff_t d = q - p;  // %td

const char *pc;       // pointee const
char *const cp;       // pointer const
void *any;            // no cast needed
int (*fp)(int, int) = add;  // fn ptr
void f(int *restrict a, int *restrict b);
```

One-past-the-end may be compared, never dereferenced. Only compare/subtract pointers into the same array. After `free`, set to `nullptr`.

ARRAYS & STRINGS

```
int a[5] = { 1, 2, 3 };   // rest zero
int b[] = { 1, 2, 3 };
int s[10] = { [9] = 1 };  // designated
int g[2][3] = {{1,2,3},{4,5,6}};
size_t n = sizeof a / sizeof a[0];

char buf[16] = "start";   // writable
const char *lit = "text"; // read-only!
strlen(s)   // bytes before NUL, O(n)
sizeof buf  // 16, includes the NUL
```

No bounds checking anywhere. In a parameter an array is a *pointer*: `sizeof` gives the pointer size, so always pass the length.

STRUCTS, UNIONS, ENUMS

```
struct Point { double x, y; };
struct Point p = { .x = 3, .y = 4 };
struct Point q = { };          // C23: zeroed
p.x = 1;  ptr->x = 1;         // (*ptr).x

typedef struct Node {
    int value;
    struct Node *next;        // self-pointer
} Node;

enum Color { RED, GREEN = 10, BLUE };
enum Op : uint8_t { NOP = 0 };  // C23
enum { BUFFER_SIZE = 4096 };    // constant

union Value { int i; float f; };
struct Tagged {                 // sum type
    enum { AS_INT, AS_DBL } kind;
    union { long i; double d; };
};
struct Flags { unsigned a : 1, b : 3; };
struct Buf { size_t len; char data[]; };
```

`offsetof(T,m)`; order members widest-first to cut padding. Never `memcmp` structs (padding) and never `==`.

DYNAMIC MEMORY

```
T *p = malloc(n * sizeof *p);  // uninit
T *z = calloc(n, sizeof *z);   // zeroed
if (p == nullptr) { /* handle */ }

T *grown = realloc(p, m * sizeof *p);
if (grown != nullptr) p = grown;  // temp!

void *a = aligned_alloc(64, 128);
free(p);  p = nullptr;   // free(nullptr) ok
free_sized(p, n);        // C23
```

Own it: one owner per block, free in reverse order of acquisition, pair `create`/`destroy`, guard size multiplications.

Leak · use-after-free · double free · overflow → `-fsanitize=address,undefined`, `valgrind --leak-check=full`.

STORAGE, SCOPE & LINKAGE

Declaration	Duration / linkage
`int x;`	block: automatic, indeterminate
`static int x;`	in fn: static, kept across calls
`static int x;`	file: static, internal linkage
`int x = 1;`	file: static, external — one only
`extern int x;`	declaration only — put in headers
`thread_local int x;`	one per thread, zeroed
`malloc`	allocated, until `free`

Static/thread objects are zero-initialized and need a constant initializer. Automatic objects are **not** — always initialize. `-Wshadow`.

PRINTF / SCANF

`%d %i`	int	`%u %o %x %X`	unsigned
`%f %e %g %a`	double	`%Lf`	long double
`%c`	char	`%s`	char *
`%p`	void *	`%%`	literal %
`%zu`	size_t	`%td`	ptrdiff_t
`%lld`	long long	`%b`	binary (C23)
`%" PRId64 "`	int64_t — `<inttypes.h>`		

```
// flags - + 0 space #, width, .prec, *
printf("[%-8s][%+.2f][%08.3f]", s, f, f);
int need = snprintf(buf, sizeof buf, "%s", s);
if (need < 0 || (size_t)need >= sizeof buf)
    /* truncated */;
```

Never `printf(user_input)` — use `"%s"`. Mismatched conversions are UB: `-Wformat=2`. Prefer `fgets`+`strtol` to `scanf`.

FILES & STREAMS

```
FILE *f = fopen(path, "rb");  // r w a r+ w+ a+ wx
if (f == nullptr) { perror("fopen"); }
fgets(line, sizeof line, f);  // nullptr at end
fread(buf, 1, n, f);          // elements read
fwrite(buf, 1, n, f);
fseek(f, 0, SEEK_END); ftell(f);
rewind(f);
if (fclose(f) != 0) /* full disk! */;

while ((c = fgetc(f)) != EOF) { }
if (ferror(f)) { } else if (feof(f)) { }
```

Never loop on `while (!feof(f))` — test the read. `stdout` is fully buffered when redirected; `stderr` is not. `_Exit`/`abort` do not flush.

STRINGS, MEMORY & CONVERT

```
strlen strcmp strncmp strchr strrchr
strstr strspn strcspn strdup(C23)
memcpy(no overlap!) memmove memset memcmp
memchr memset_explicit(C23)
isalpha isdigit isspace toupper  // ctype

errno = 0;  char *end;
long v = strtol(text, &end, 10);
if (end == text || *end || errno == ERANGE)
    /* reject */;
strtoul strtoll strtod strtof imaxabs
```

`ctype` needs `(unsigned char)`. `strncpy` may not terminate. `strtok` mutates and holds static state. Never `ato*`.

UTF-8 is just bytes: `strlen` counts bytes; split only where `(b & 0xC0) != 0x80`.

MATH, BITS & TIME

```
fabs sqrt pow hypot fma log1p exp
floor ceil trunc round nearbyint
isnan isinf isfinite fpclassify signbit
abs labs div ldiv  // link with -lm

// <stdbit.h> C23
stdc_count_ones stdc_bit_width
stdc_leading_zeros stdc_trailing_zeros
stdc_bit_ceil stdc_has_single_bit
__STDC_ENDIAN_NATIVE__

time_t now = time(nullptr);
struct timespec ts; timespec_get(&ts, TIME_UTC);
struct tm tm; gmtime_r(&now, &tm);  // C23
strftime(b, sizeof b, "%Y-%m-%dT%H:%M:%SZ", &tm);
mktime(&tm)/local timegm(&tm)/UTC
difftime(a,b)
```

`tm_year` is since 1900, `tm_mon` is 0-based; set `tm_isdst = -1`. `NAN != NAN`: use `isnan`. `rand` is not secure.

PREPROCESSOR

```
#include <stdio.h>   #include "local.h"
#ifndef PROJECT_H
#define PROJECT_H
#endif  /* include guard */

#define SQUARE(x) ((x) * (x))   // parens!
#define STMT(a) do { f(a); } while (0)
#define STR(x) #x       // stringify
#define CAT(a,b) a##b    // paste
#define LOG(f, ...) \
  printf(f "\n" __VA_OPT__(,) __VA_ARGS__)

#if defined(X) && __STDC_VERSION__ >= 202311L
#elifdef Y      // C23
#elifndef Z     // C23
#endif
#if __has_include(<stdbit.h>)
#if __has_c_attribute(nodiscard)
#error "unsupported"   #warning "note"
__FILE__ __LINE__ __func__
__STDC_VERSION__
```

Prefer `enum`/`constexpr`/`static inline` to macros. Never use a parameter twice.

_GENERIC, TYPEOF, AUTO

```
#define ABS(x) _Generic((x), \
    int: abs_i, long: abs_l, \
    double: fabs, default: abs_i)(x)

typeof(expr) copy = expr;       // C23
typeof_unqual(cexpr) writable;  // strips const
auto n = 42;                    // C23: int
#include <tgmath.h>   // sqrt over all types
```

`_Generic`'s controlling expression is not evaluated; arrays decay, so match `char *`. All branches must parse.

THREADS `<THREADS.H>`

```
int worker(void *arg) { return 0; }

thrd_t t;
thrd_create(&t, worker, arg);  // thrd_success
thrd_join(t, &result);         // or thrd_detach
thrd_sleep(&ts, nullptr); thrd_yield();

mtx_t m; mtx_init(&m, mtx_plain);
mtx_lock(&m); /* critical */
mtx_unlock(&m);
mtx_trylock mtx_timedlock mtx_destroy

cnd_t c; cnd_init(&c);
while (!predicate)            // while, not if!
    cnd_wait(&c, &m);
cnd_signal(&c);  cnd_broadcast(&c);

static once_flag f = ONCE_FLAG_INIT;
call_once(&f, init);
thread_local int per_thread;
tss_create(&key, destructor);
```

Unsynchronized shared access = data race = UB. One lock order everywhere. Optional: check `__STDC_NO_THREADS__`; link `-pthread`.

ATOMICS `<STDATOMIC.H>`

```
atomic_int n = 0;   _Atomic long m;
atomic_store(&n, 1);  atomic_load(&n);
atomic_fetch_add(&n, 1);   // _sub _or _and
atomic_exchange(&n, 5);

int expected = atomic_load(&n), desired;
do { desired = f(expected); }
while (!atomic_compare_exchange_weak(
        &n, &expected, desired));

atomic_flag lk = ATOMIC_FLAG_INIT;
while (atomic_flag_test_and_set(&lk)) ;
atomic_flag_clear(&lk);
```

Orders: `seq_cst` (default) · `release` store pairs with `acquire` load (publishes data) · `relaxed` for independent counters only.

CAS overwrites `expected` on failure. Never publish data with `relaxed`.

ERROR HANDLING & UB

```
enum Status { OK = 0, E_INVAL, E_NOMEM };
enum Status f(const char *in, char **out)
{
    *out = nullptr;          // safe default
    if (!in) return E_INVAL;
    ...
    goto cleanup;            // one exit
}
errno = 0; ... if (errno == ERANGE) { }
perror("fopen"); strerror(errno);
assert(invariant);       // NOT input checks
static_assert(CHAR_BIT == 8);
```

UB roll-call: signed overflow · /0 · shift ≥ width · out-of-bounds · null deref · use-after-free · double free · uninitialized read · strict aliasing · misaligned access · data race · two unsequenced writes.

Reinterpret bytes with `memcpy` or a `union`, never `*(float*)&i`. `assert` vanishes under `-DNDEBUG`.

C23 AT A GLANCE

`bool true false`	keywords; `<stdbool.h>` deprecated
`static_assert`	keyword, message optional
`thread_local`	keyword
`nullptr`	typed null, `nullptr_t`
`constexpr`	typed constant objects **GCC 15 / Clang 19**
`typeof / auto`	type of expr / inference
`_BitInt(N)`	exact-width ints, `wb` suffix
`0b1010 / 1'000`	binary + digit separators
`{ }`	empty initializer, zeroes all
`[[attributes]]`	nodiscard, fallthrough, …
`__VA_OPT__`	zero-arg variadic macros
`#embed`	binary file inclusion **GCC 15 / Clang 19**
`enum E : T`	fixed underlying type
`<stdckdint.h>`	`ckd_add/sub/mul` checked
`<stdbit.h>`	popcount, bit width, endianness
`removed`	trigraphs, K&R parameter lists

`__STDC_VERSION__ >= 202311L`. `-std=c23` needs GCC 14+ / Clang 18+ (older: `-std=c2x`).

TOOLCHAIN

```
-std=c23 -Wall -Wextra -Werror -Wpedantic
-Wshadow -Wconversion -Wvla -Wformat=2
-Wstrict-prototypes -Wwrite-strings
-Og -g3              // debug
-O2 -flto -DNDEBUG -D_FORTIFY_SOURCE=3
-fsanitize=address,undefined  // ASan+UBSan
-fsanitize=thread  // races (not with ASan)
-fanalyzer         // GCC static analysis
make · cmake -S . -B build && ctest
clang-format -i · clang-tidy -p build
cppcheck --enable=all · valgrind
gdb ./app · lldb ./app · rr replay
perf stat/record · doxygen Doxyfile
```

Test frameworks: Unity, CMocka, Check, Criterion. Coverage `--coverage`+`lcov`; fuzz `-fsanitize=fuzzer`.