Chuleta esencial de C

Sintaxis, punteros, memoria y librería estándar en dos páginas

Compila con `gcc -std=c17 -Wall -Wextra -O2`

Sintaxis mínima

```c
#include <stdio.h>
int main(void) {
  printf("hola %d\n", 42);
  return 0;
}
```

Reglas clave

- Declaración = tipo + identificador + `;`.
- `#include <...>` trae prototipos de funciones.
- `main` devuelve `int`; `return 0` = OK.

Tipos básicos (C17)

Tipo	Notas
`char`	1 byte (signado o no, depende de la impl.)
`short`, `int`, `long`, `long long`	Enteros; tamaño dependiente de la plataforma
`float`, `double`, `long double`	Coma flotante
`_Bool`	Booleano (`#include <stdbool.h>`)
`size_t`	Tamaño/`sizeof`; sin signo

sizeof

`sizeof(expr)` devuelve bytes. `sizeof array / sizeof *array` = nº elementos.

Operadores y precedencia

Nivel	Operadores (izq→der)
Alto	`() [] . → ++ --`
Unario	`+ - ! * & (tipo) sizeof`
Multiplic.	`* / %`
Suma	`+ -`
Shift	`« »`
Relac.	`< ⩽ > ⩾`
Igualdad	`== ≠`
Bit a bit	`& ^ |`
Lógico	`&& ||`
Condic.	`?:`
Asignación	`= += -= *= /= %= «= »= &= =^|=`
Coma	`,`

Control de flujo

`if / else`, `switch`, `for`, `while`, `do-while`, `break`, `continue`.

```c
for (size_t i = 0; i < n; ++i) { /* ... */ }
switch (op) { case '+': /*...*/ break; default: /*...*/ }
```

Funciones

Prototipos en cabecera; evita `implicit int`.

```c
int suma(int a, int b);
static inline int sqr(int x){ return x*x; }
```

Paso por valor; para "paso por referencia" usa punteros.

Punteros & memoria

> **Concepto clave**: un puntero almacena una dirección. `*p` desreferencia, `&x` obtiene dirección.

```c
int x = 7; int *p = &x; *p = 9; // x == 9
```

Aritmética: suma/resta en unidades de `sizeof(*p)`.
NULL puntero nulo; **no** desreferenciar.

Memoria dinámica

```c
#include <stdlib.h>
int *v = malloc(n * sizeof *v);
if (!v) { /* manejar error */ }
/* ... */
free(v); v = NULL;
```

Reglas: inicializa, comprueba `malloc`, libera una vez, anula puntero.

Arrays y cadenas

```c
int a[4] = {1,2,3,4};
char s[] = "hola"; // incluye '\0'
```

`<string.h>`: `memcpy`, `memset`, `strlen`, `strcpy`, `strncpy`, `strcmp`.
OJO: evitar `gets` (no existe en C11+). Usa `fgets`.

struct/union/enum

```c
typedef struct { int x, y; } Punto;
Punto p = {.x=1,.y=2};
```

`enum` crea constantes enteras; `union` comparte memoria entre campos.

E/S estándar

```c
#include <stdio.h>
printf("%d %f %s", i, d, s);
scanf("%d", &i); // validar retorno
FILE *f = fopen("out.txt","w");
fprintf(f, "hola\n"); fclose(f);
```

Formato: `%d %u %ld %zu %f %lf %p %s`. Cuidado con tamaños.

Errores comunes

- Desbordes de buffer; usa longitudes y `fgets`.
- Usar punteros colgantes o dobles `free`.
- Mezclar `signed/unsigned` en comparaciones.
- Olvidar `return` o prototipos.

Compilación

```
# GCC
gcc -std=c17 -Wall -Wextra -Wpedantic -O2 prog.c -o prog
# Sanitizers (debug)
```

```
clang -g -fsanitize=address,undefined -fno-omit-frame-
    pointer prog.c
```

Preprocesador

```
#define MAX 1024
#define SQ(x) ((x)*(x))
#ifdef DEBUG
  #define LOG(...) fprintf(stderr, __VA_ARGS__)
#else
  #define LOG(...)
#endif
```

`#include`, `#define`, `#if`, `#ifdef`, `#pragma once` (en cabeceras, no estándar C pero común).

Patrones útiles

Iterar con índice y puntero

```
for (int *it=a, *end=a+n; it!=end; ++it) { /*...*/ }
```

Inicialización segura

```
typedef struct { int ok; double v; } Res;
Res r = { .ok = 0, .v = 0.0 };
```

Guía de formato & estilo

- Una declaración por línea; inicializa al declarar.
- `const` por defecto donde aplique; funciones pequeñas `static`.
- Evita macros complejas; prefiere funciones inline.

Límites & tamaños

```
#include <limits.h>
#include <float.h>
// INT_MAX, SIZE_MAX, DBL_MIN, etc.
```

Algoritmos mini (plantillas)

Búsqueda lineal

```
int find(const int *a, size_t n, int key){
  for (size_t i=0;i<n;++i) if (a[i]==key) return (int)i;
  return -1;
}
```

Ordenación (qsort)

```
int cmp_int(const void *a, const void *b){
  int x=*(const int*)a, y=*(const int*)b;
  return (x>y)-(x<y);
}
qsort(a, n, sizeof *a, cmp_int);
```

Cadenas seguras

```
size_t safe_copy(char *dst, size_t cap, const char *src){
  size_t n=0; while(n+1<cap && src[n]){ dst[n]=src[n]; ++n;
        }
  dst[n]='\0'; return n;
}
```

Punteros avanzados

Regla de oro aliasing: dos punteros de tipos incompatibles no deben apuntar al mismo objeto (strict aliasing). Usa `unsigned char*` para acceso byte a byte.

```
int x=0x12345678; unsigned char *b=(unsigned char*)&x; //
    bytes
```

I/O de bajo nivel

```
#include <unistd.h>
ssize_t n = write(1, "hola\n", 5); // POSIX
```

Errores & diagnósticos

```
#include <errno.h>
#include <string.h>
if (f==NULL){ fprintf(stderr,"err: %s\n", strerror(errno));
     }
```

Bit a bit

```
unsigned m = 0b1010; // C23 (o usa 10u)
unsigned set(unsigned x, unsigned bit){ return x | (1u<<bit
    ); }
unsigned clr(unsigned x, unsigned bit){ return x & ~(1u<<
    bit); }
unsigned tog(unsigned x, unsigned bit){ return x ^ (1u<<bit
    ); }
int test(unsigned x, unsigned bit){ return (x>>bit)&1u; }
```

Macros de utilidad

```
#define COUNT_OF(a) (sizeof(a)/sizeof *(a))
#define CLAMP(x,lo,hi) ((x)<(lo)?(lo):((x)>(hi)?(hi):(x)))
#define UNUSED(x) (void)(x)
```

Checklist de revisión

- ¿Prototipos visibles? ¿`-Wall -Wextra` limpios?
- ¿`malloc`/`free` balanceados? ¿sin fugas (valgrind)?
- ¿Índices y tamaños en `size_t`? ¿conversiones seguras?
- ¿Entradas validadas? ¿errores propagados correctamente?

Atajos mentales

- **Puntero = dirección**; `*` lee/escribe, `&` obtiene.
- **Array decae** a `T*` salvo en `sizeof` y `&`.
- **_Bool** es 0/1; cualquier no-cero es verdadero.
- **UB** (comportamiento indef.) ⇒ resultados impredecibles.

Enlaces rápidos

- Especificación: https://port70.net/~nsz/c/c11/n1570.html
- Referencia libc: https://man7.org/linux/man-pages/
- Página web: https://rigle.dev/

Autor: Rodrigo Iglesias (Rigle.dev).