Referência Rápida C

Sintaxe, ponteiros, gerenciamento de memória, biblioteca padrão essencial

Básico

Hello World

```
#include <stdio.h>
int main(void) {
    printf("Hello, World!\n");
    return 0;
}
```

Compilar e Executar

```
gcc -o app main.c          # compilar
gcc -Wall -Wextra -std=c17 main.c  # strict
./app                       # executar
```

Comentários

```
// single-line comment (C99+)
/* multi-line
   comment */
```

Tipos de Dados

Tipos Primitivos

`char`	1 byte, caractere ou inteiro pequeno
`short`	Pelo menos 16 bits
`int`	Pelo menos 16 bits (tipicamente 32)
`long`	Pelo menos 32 bits
`long long`	Pelo menos 64 bits (C99+)
`float`	IEEE-754 de 32 bits
`double`	IEEE-754 de 64 bits
`_Bool / bool`	**`0`** ou **`1`** (use **`<stdbool.h>`** para **`bool`**)

Tipos de Largura Fixa (stdint.h)

`int8_t, uint8_t`	Exato 8 bits com sinal / sem sinal
`int16_t, uint16_t`	Exato 16 bits
`int32_t, uint32_t`	Exato 32 bits
`int64_t, uint64_t`	Exato 64 bits
`size_t`	Sem sinal, resultado de **`sizeof`**

Conversão de Tipo

```
int i = (int)3.14;          // cast explícito
double d = (double)5 / 2;   // 2.5, not 2
char c = (char)65;          // 'A'
```

Controle de Fluxo

If / Else

```
if (x > 0) { printf("positive\n"); }
else if (x == 0) { printf("zero\n"); }
else { printf("negative\n"); }
```

Switch

```
switch (choice) {
    case 1:  printf("one\n"); break;
    case 2:  printf("two\n"); break;
    default: printf("other\n");
}
```

Laços

```
for (int i = 0; i < 10; i++) { }
while (condition) { }
do { } while (condition);
```

Instruções de Desvio

`break`	Sair do laço interno ou switch
`continue`	Pular para próxima iteração
`return`	Sair da função com valor opcional
`goto label`	Saltar para label (usar com parcimônia)

Funções

Declaração e Definição

```
int add(int a, int b);         // prototype
int add(int a, int b) {
    return a + b;
}
```

Ponteiros para Funções

```
int (*op)(int, int) = add;
int result = op(3, 4);       // calls add(3, 4)
typedef int (*MathFn)(int, int);
MathFn fn = add;
```

Funções Estáticas

```
// visible only within this translation unit
static int helper(int x) {
    return x * 2;
}
```

Ponteiros

Ponteiros Básico

```
int x = 42;
int *p = &x;        // p aponta para x
printf("%d\n", *p); // desreferência: 42
*p = 100;           // x é agora 100
```

Aritmética de Ponteiros

```
int arr[] = {10, 20, 30};
int *p = arr;
printf("%d\n", *(p + 1));  // 20
printf("%d\n", p[2]);      // 30 (same as *(p+2))
```

Padrões Comuns de Ponteiros

`int *p = NULL`	Ponteiro nulo (sempre inicializar)
`void *`	Ponteiro genérico (deve fazer cast para usar)
`const int *p`	Ponteiro para constante (não pode modificar valor)
`int *const p`	Ponteiro constante (não pode reatribuir ponteiro)
`int **pp`	Ponteiro para ponteiro (indireção dupla)

Arrays e Strings

Arrays

```
int nums[5] = {1, 2, 3, 4, 5};
int matrix[2][3] = {{1,2,3}, {4,5,6}};
int len = sizeof(nums) / sizeof(nums[0]);
```

Funções de String (string.h)

`strlen(s)`	Comprimento (excluindo terminador nulo)
`strcpy(dst, src)`	Copiar string (não seguro, prefira **`strncpy`**)
`strncpy(dst, src, n)`	Copiar no máximo n caracteres
`strcat(dst, src)`	Concatenar strings
`strcmp(a, b)`	Comparar: 0 se igual, <0 ou >0 caso contrário
`strchr(s, c)`	Encontrar primeira ocorrência do caractere
`strstr(haystack, needle)`	Encontrar substring

Literais de String

```
char greeting[] = "hello";     // array mutável
const char *msg = "world";     // ponteiro para literal
char buf[64];
snprintf(buf, sizeof(buf), "%s %s", greeting, msg);
```

Structs

Definição e Uso

```
struct Point { double x; double y; };
struct Point p = {1.0, 2.0};
printf("(%g, %g)\n", p.x, p.y);
```

Typedef

```
typedef struct {
    char name[50];
    int age;
} Person;
Person p = {"Alice", 30};
```

Ponteiros para Struct

```
void set_age(Person *p, int age) {
    p->age = age;    // operador seta
}
```

Enums e Unions

```
enum Color { RED, GREEN, BLUE };
union Data { int i; float f; char c; };
// union members share the same memory
```

Gerenciamento de Memória

Alocação Dinâmica (stdlib.h)

```
int *arr = malloc(10 * sizeof(int));
if (arr == NULL) { /* handle error */ }
arr = realloc(arr, 20 * sizeof(int));
free(arr);
arr = NULL;   // evitar ponteiro dangling
```

Funções de Alocação

`malloc(size)`	Alocar memória não inicializada
`calloc(count, size)`	Alocar e inicializar com zero
`realloc(ptr, size)`	Redimensionar bloco alocado anteriormente
`free(ptr)`	Liberar memória alocada

Armadilhas Comuns

`Memory leak`	Esquecer de chamar **`free()`** na memória alocada
`Double free`	Chamar **`free()`** duas vezes no mesmo ponteiro
`Dangling pointer`	Usar ponteiro após **`free()`** — definir como NULL
`Buffer overflow`	Escrever além dos limites alocados

I/O de Arquivo

Lendo Arquivos

```
FILE *f = fopen("data.txt", "r");
if (!f) { perror("open"); return 1; }
char line[256];
while (fgets(line, sizeof(line), f)) printf("%s", line);
fclose(f);
```

Escrevendo em Arquivos

```
FILE *f = fopen("out.txt", "w");
fprintf(f, "value: %d\n", 42);
fputs("hello\n", f);
fclose(f);
```

Referência Rápida C

Modos de Arquivo

`"r"`	Leitura (arquivo deve existir)
`"w"`	Escrita (trunca ou cria)
`"a"`	Acrescentar (cria se necessário)
`"rb", "wb"`	Leitura / escrita binária
`"r+"`	Leitura e escrita (arquivo deve existir)

Pré-processador

Diretivas

```
#include <stdio.h>       // system header
#include "myheader.h"    // local header
#define PI 3.14159
#define MAX(a, b) ((a) > (b) ? (a) : (b))
```

Compilação Condicional

```
#ifdef DEBUG
    printf("debug: x = %d\n", x);
#endif
#ifndef HEADER_H   /* include guard */
#define HEADER_H   /* ... */ #endif
```

Macros Comuns

`__FILE__`	Nome do arquivo de código atual
`__LINE__`	Número da linha atual
`__func__`	Nome da função atual (C99+)
`__DATE__`	String de data de compilação
`NULL`	Constante de ponteiro nulo
`sizeof(x)`	Tamanho do tipo ou variável em bytes