RIFERIMENTO RAPIDO C

Sintassi, puntatori, gestione memoria, essenziali della libreria standard

Basi

Hello World

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

Compilazione ed Esecuzione

```c
gcc -o app main.c          # compila
gcc -Wall -Wextra -std=c17 main.c  # rigoroso
./app                      # esegui
```

Commenti

```c
// commento su una riga (C99+)
/* commento
   multiriga */
```

Tipi di Dato

Tipi Primitivi

`char`	1 byte, carattere o intero piccolo
`short`	Almeno 16 bit
`int`	Almeno 16 bit (tipicamente 32)
`long`	Almeno 32 bit
`long long`	Almeno 64 bit (C99+)
`float`	IEEE-754 32 bit
`double`	IEEE-754 64 bit
`_Bool / bool`	`0` o `1` (usare `<stdbool.h>` per `bool`)

Tipi a Larghezza Fissa (stdint.h)

`int8_t, uint8_t`	Esatto 8 bit con/senza segno
`int16_t, uint16_t`	Esatto 16 bit
`int32_t, uint32_t`	Esatto 32 bit
`int64_t, uint64_t`	Esatto 64 bit
`size_t`	Senza segno, risultato di `sizeof`

Cast di Tipo

```c
int i = (int)3.14;          // cast esplicito
double d = (double)5 / 2;   // 2.5, non 2
char c = (char)65;          // 'A'
```

Controllo del Flusso

If / Else

```c
if (x > 0) { printf("positivo\n"); }
else if (x == 0) { printf("zero\n"); }
else { printf("negativo\n"); }
```

Switch

```c
switch (choice) {
    case 1:  printf("uno\n"); break;
    case 2:  printf("due\n"); break;
    default: printf("altro\n");
}
```

Cicli

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

Istruzioni di Salto

`break`	Esce dal ciclo o switch più interno
`continue`	Salta alla prossima iterazione
`return`	Esce dalla funzione con valore opzionale
`goto label`	Salta all'etichetta (usare raramente)

Funzioni

Dichiarazione e Definizione

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

Puntatori a Funzione

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

Funzioni Static

```c
// visibile solo in questa unità di traduzione
static int helper(int x) {
    return x * 2;
}
```

Puntatori

Basi dei Puntatori

```c
int x = 42;
int *p = &x;        // p punta a x
printf("%d\n", *p); // dereferenziamento: 42
*p = 100;           // x è ora 100
```

Aritmetica dei Puntatori

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

Pattern Comuni con Puntatori

`int *p = NULL`	Puntatore null (inizializzare sempre)
`void *`	Puntatore generico (deve essere castato per usarlo)
`const int *p`	Puntatore a costante (non può modificare il valore)
`int *const p`	Puntatore costante (non può essere riassegnato)
`int **pp`	Puntatore a puntatore (doppia indirezione)

Array e Stringhe

Array

```c
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]);
```

Funzioni Stringa (string.h)

`strlen(s)`	Lunghezza (escluso terminatore null)
`strcpy(dst, src)`	Copia stringa (non sicuro, preferire `strncpy`)
`strncpy(dst, src, n)`	Copia al massimo n caratteri
`strcat(dst, src)`	Concatena stringhe
`strcmp(a, b)`	Confronta: 0 se uguale, <0 o >0 altrimenti
`strchr(s, c)`	Trova prima occorrenza del carattere
`strstr(haystack, needle)`	Trova sottostringa

Letterali Stringa

```c
char greeting[] = "hello";     // array modificabile
const char *msg = "world";     // puntatore al letterale
char buf[64];
snprintf(buf, sizeof(buf), "%s %s", greeting, msg);
```

Struct

Definizione e Utilizzo

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

Typedef

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

Puntatori a Struct

```c
void set_age(Person *p, int age) {
    p->age = age;    // operatore freccia
}
```

Enum e Union

```c
enum Color { RED, GREEN, BLUE };
union Data { int i; float f; char c; };
// i membri dell'union condividono la stessa memoria
```

Gestione della Memoria

Allocazione Dinamica (stdlib.h)

```c
int *arr = malloc(10 * sizeof(int));
if (arr == NULL) { /* gestisci errore */ }
arr = realloc(arr, 20 * sizeof(int));
free(arr);
arr = NULL;   // evita puntatore dangling
```

Funzioni di Allocazione

`malloc(size)`	Alloca memoria non inizializzata
`calloc(count, size)`	Alloca e inizializza a zero
`realloc(ptr, size)`	Ridimensiona blocco allocato in precedenza
`free(ptr)`	Libera la memoria allocata

Errori Comuni

`Memory leak`	Dimenticare di chiamare `free()` sulla memoria allocata
`Double free`	Chiamare `free()` due volte sullo stesso puntatore
`Dangling pointer`	Usare il puntatore dopo `free()` — impostare a NULL
`Buffer overflow`	Scrivere oltre i limiti allocati

I/O su File

Lettura di File

```c
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);
```

Scrittura su File

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

Modalità File

`"r"`	Lettura (il file deve esistere)
`"w"`	Scrittura (tronca o crea)
`"a"`	Aggiunta (crea se necessario)
`"rb", "wb"`	Lettura / scrittura binaria
`"r+"`	Lettura e scrittura (il file deve esistere)

Preprocessore

Direttive

```c
#include <stdio.h>      // header di sistema
#include "myheader.h"   // header locale
#define PI 3.14159
#define MAX(a, b) ((a) > (b) ? (a) : (b))
```

Compilazione Condizionale

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

Macro Comuni

`__FILE__`	Nome del file sorgente corrente
`__LINE__`	Numero di riga corrente
`__func__`	Nome della funzione corrente (C99+)
`__DATE__`	Stringa data di compilazione
`NULL`	Costante puntatore null
`sizeof(x)`	Dimensione in byte del tipo o variabile