



Laboratorio 2 A

Gestione file e directory in C (libreria), file binari e posizionamento su file.

Prof. Patrizio Dazzi • a.y. 2025/26

<https://pages.di.unipi.it/dazzi/>
patrizio.dazzi@unipi.it

C è come una sega a catena: se sai usarla bene puoi costruire una casa; se non la sai usare, puoi segarti una gamba.

Premessa



Obiettivi della lezione

- Aprire/chiudere file con fopen/fclose
- Leggere/scrivere testo: fgetc, fgets, fputc, fputs, fprintf
- Leggere/scrivere binario: fread/fwrite
- Posizionarsi nel file: fseek, ftell, rewind
- Buffering, fflush, setvbuf
- Gestione errori: feof, perror, perror
- Pattern robusti e pitfall comuni

Nota: focus su libreria standard C11 (no syscall POSIX: open/read/write).

Panorama: I/O file in C11

- Header: #include <stdio.h> (<errno.h> per i codici di errore)
- Tipo opaco: FILE * (dipendente dall'implementazione)
- Flussi standard: stdin, stdout, stderr
- Modalità: *testo* e *binario* ("b")
- Buffering: line-buffered, full-buffered, unbuffered (dipendente dall'implementazione)

Apertura e lettura



Aprire e chiudere file

- FILE *fopen(const char *path, const char *mode);
- Modalità: "r", "w", "a", con varianti "+" e "b"
- Controllare sempre il valore di ritorno
- int fclose(FILE *stream);
(ritorna EOF su errore)

```
1 FILE *fp =
2     fopen("input.dat", "rb");
3
4 if (!fp) {
5     perror("fopen"); return 1;
6 }
7
8 /* ... usa fp ... */
9
10 if (fclose(fp) == EOF) {
11     perror("fclose");
12 }
```

I/O testuale a carattere e a riga

- Lettura: fgetc, fgets
- Scrittura: fputc, fputs
- Uso tipico: parsing a linee, log

```
1 char buf[256];
2 while (fgets(buf, sizeof buf, fp))
3 {
4     /* processa la riga */
5 }
```

Formattato: fprintf/fscanf e parsing robusto

- fprintf: output formattato
- fscanf: input formattato (attenzione a spazi/limiti)
- Spesso più robusto: fgets + sscanf

```
1 /* fragile */
2 fscanf(fp, "%d %63s", &id, name);
3
4 /* robusto */
5 if (fgets(buf, sizeof buf, fp)) {
6     if (sscanf(buf, "%d %63s", &id, name)==2){
7         /* ok */
8     }
9 }
```

I/O binario: fread/fwrite

- `size_t fread(void *ptr,
size_t size, size_t nmemb,
FILE *)`
- `size_t fwrite(const void
*ptr, size_t size, size_t
nmemb, FILE *)`
- Gestire short read/write; verificare
`ferror`

```
1 typedef struct {  
2     int id; double v;  
3 } rec;  
4  
5 rec r[128];  
6 size_t n =  
7     fread(r, sizeof *r, 128, fp);  
8  
9 if (n < 128 && ferror(fp)) {  
10    perror("fread");  
11 }
```

Posizionamento: fseek, ftell, rewind

- int fseek(FILE*, long off, int origin)
- long ftell(FILE*) (limiti su file molto grandi in LP32)
- void rewind(FILE*)

```
1 long pos = ftell(fp);
2 fseek(fp, 0, SEEK_END);
3 long size = ftell(fp);
4 fseek(fp, pos, SEEK_SET)
;
```

Posizionamento, buffering,
errori, rinomina, rimozione



Buffering e sincronizzazione

- `fflush(FILE*)`: forza lo scarico del buffer in uscita
- `setvbuf(FILE*, char *buf, int mode, size_t size)`: controlla il buffering
- `stderr`: spesso non bufferizzato (utile per diagnosi)

```
1 fprintf(fp, "record %d\n", i);
2 fflush(fp); /* visibilita' immediata quando serve */
```

Stato del flusso ed errori

- int feof(FILE*), int ferror(FILE*)
- Reset errori: clearerr(FILE*)
- Diagnostica: perror("msg"), strerror(errno)
- Pattern: testare *subito* i ritorni di chiamata

```
1 if (ferror(fp)) {  
2     perror("I/O error");  
3     clearerr(fp);  
4 }  
5 if (feof(fp)) {  
6     /* fine file */  
7 }
```

Rinominare, rimuovere, temporanei

- int rename(const char *old, const char *new);
- int remove(const char *path);
- FILE *tmpfile(void); crea file temporaneo anonimo (eliminato alla chiusura)

```
1 if (rename("out.tmp", "out.dat") != 0) {  
2 perror("rename");  
3 }
```

Wrap-up



Esempio: lettura CSV minimale

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(void) {
5     FILE *fp = fopen("data.csv", "r");
6     if (!fp) { perror("data.csv"); return 1; }
7     char line[512];
8     while (fgets(line, sizeof line, fp)) {
9         /* rimuovi newline */
10        line[strcspn(line, "\r\n")] = '\0';
11        /* split semplice (no quote) */
12        char *p = strtok(line, ",");
13        while (p) { printf("[%s] ", p); p = strtok(NULL, ","); }
14        putchar('\n');
15    }
16    fclose(fp);
17 }
```

Pitfall comuni

- Dimenticare b su Windows → corruzione di binari
- Non controllare i valori di ritorno (`fgets`, `fread`, `fwrite`)
- Usare `feof` come condizione di ciclo: testare la *lettura* invece
- Buffer troppo piccoli / overflow: usare limiti (%Ns), tipi `size_t`
- Confondere `fseek`/`ftell` su file grandi (preferire API estese se disponibili)
- Mescolare I/O formattato e binario senza `fflush`/riposizionamento

Linee guida di stile

- Errori leggibili: perror + contesto
- Dimensioni dei buffer costanti e documentate (no “magic numbers”)

Micro-esercizi

1. Implementare `size_t file_size(const char* path)` con `fseek/ftell`.
2. Filtro: leggere da `stdin` e scrivere su `stdout` le sole righe che contengono una sottostringa.
3. Serializzare un array di struct su file binario e ricaricarlo verificando il conteggio.

Riferimenti

- Man page POSIX: `fopen(3)`, `fread(3)`, `fwrite(3)`, `fseek(3)`

Q & A

*Thank you for your attention!
Any Questions?*

patrizio.dazzi@unipi.it