Il concetto di file

..... è una parte logicamente contigua del contenuto di un dispositivo

⇨ Deve essere capace di memorizzare grandi quantità di dati

⇨ I dati memorizzati devono sopravvivere alla fine del programma che li usa

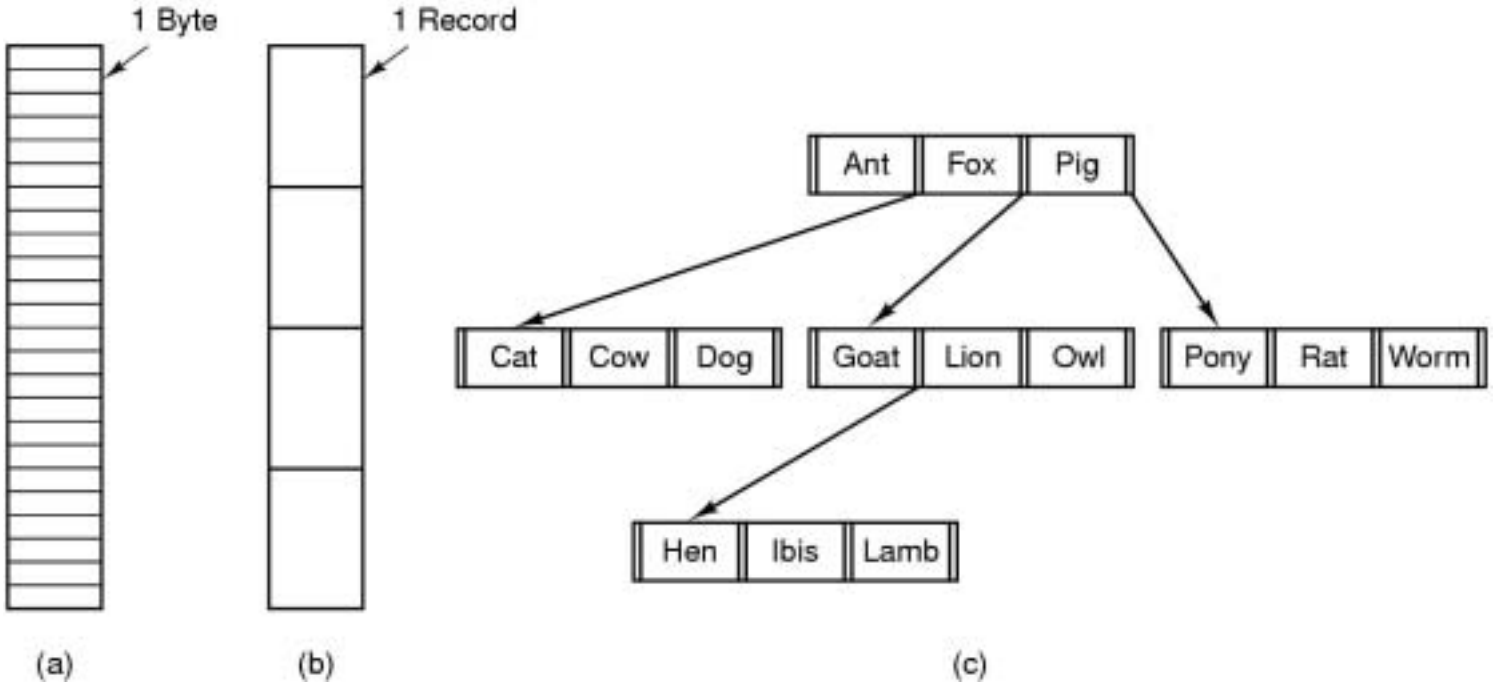
⇨ Più programmi devono poter accedere ai dati in esso contenuti

Il contenuto dei file

- Dati
  - numerici
  - caratteri
  - binari
- Programmi

Strutture di file

- Nessuna – sequenza di parole, byte (fig. a)
- Struttura a record semplici
  - Di linee
  - Di lunghezza fissa (fig. b)
  - Di lunghezza variabile
- Strutture complesse (fig. c)
  - Documenti con formato
  - Programmi in formato rilocabile e caricabile

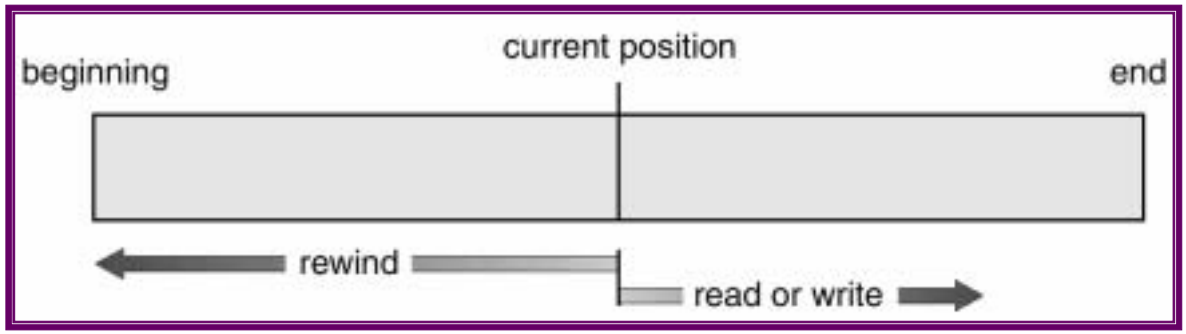


Metodi di accesso ai file

- ✓ **Accesso sequenziale**

  read next
  write next
  reset

  no *read after last write (rewrite)*



- ✓ **Accesso diretto**

  read n
  write n
  position to n
  read next
  write next
  rewrite n

  n = relative block number

Le operazioni sui file in C

I file sono una risorsa di un sistema di calcolo e sono pertanto gestiti dal Sistema Operativo.

Per Input/Output (I/O) si intende l'insieme delle operazioni di ingresso ed uscita, cioè di scambio di informazioni tra il programma e le unità periferiche del calcolatore (video, tastiera, dischi, etc.), che sono trattate, a livello software, come file.

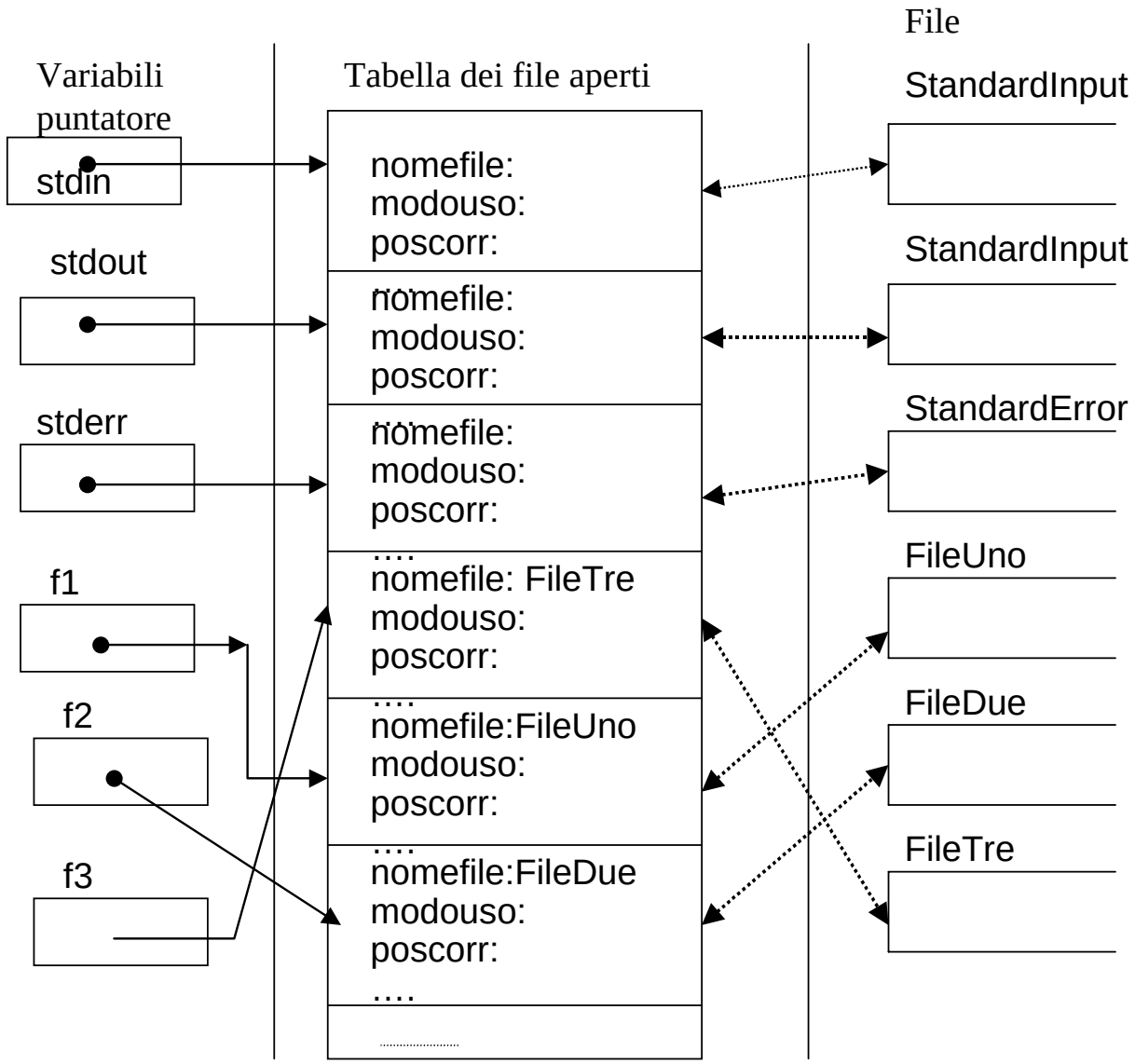
L'I/O è interamente implementato in C mediante funzioni di libreria, in coerenza con la filosofia che sta alla base del C stesso, cioè di **linguaggio svincolato dall'ambiente** in cui il programma deve operare, e pertanto portabile.

Il linguaggio C consente di sfruttare tale impostazione, mediante il concetto di **stream**, cioè di flusso di byte da o verso una periferica: leggere dati da un file non è diverso da leggerli dalla tastiera e scrivere in un file è del tutto analogo a scrivere sul video. Associando ad ogni periferica uno stream, tutte le periferiche sono gestite, ad alto livello, nello stesso modo.

Tecnicamente, uno stream è un meccanismo software in grado di semplificare l'interazione a basso livello tra programma e periferica associata, in modo che il programma possa ignorare del tutto la natura della periferica.

Lo stream rappresenta, per il programmatore, una interfaccia per la lettura e l'invio di dati tra il programma e la periferica: **l'apertura di un flusso di comunicazione indica al Sistema Operativo (SO) l'intenzione di aprire un file esistente o la creazione e l'apertura di un nuovo file**.

Gli stream in C



Gli stream in C

Un programma C può aprire un flusso di comunicazione:

- **binario** (sequenza di byte)
- **di tipo testo** (sequenza di caratteri generalmente suddivisa in linee terminanti con un carattere di *newline*).

Un programma C può servirsi degli **stream standard** senza alcuna operazione preliminare: è sufficiente che nel sorgente compaia la direttiva

```
#include <stdio.h>
```

Di fatto, molte funzioni standard di libreria che gestiscono l'I/O li utilizzano in modo del tutto trasparente: ad esempio `printf()` non scrive a video, ma sullo stream **stdout**. L'output di `printf()` compare perciò a video.

La funzione `scanf()`, invece, legge un carattere dallo standard input, cioè dallo stream **stdin**: di norma la tastiera.

La segnalazione di eventuali errori da parte delle funzioni standard di libreria viene effettuata sullo stream **stdout**, che è normalmente associato al video.

Con un'operazione di **redirezione** è possibile forzare le funzioni di libreria a leggere o scrivere da/su un altro stream, cioè da/su altre periferiche.

La struttura di dati di tipo `FILE`

Il C non limita l'uso degli stream a quelli standard; al contrario, **via stream può essere gestito qualunque file**.
Per accedere ad un file, un programma C deve aprire (**open**) un flusso di comunicazione verso il file, dichiarando una variabile di tipo puntatore che punta all'interno della Tabella dei file aperti del SO ad una struttura di dati di tipo `FILE`, che contiene varie informazioni relative a ciascun file:

- il campo *modalità di utilizzo* del file (lettura, scrittura, lettura e scrittura);
- il campo *posizione corrente* sul file (indicante l'indirizzo del prossimo byte da leggere/scrivere);
- il campo *indicatore di errore* che indica la presenza di un errore di lettura/scrittura;
- il campo *indicatore di end-of-file* (`eof`) che indica se è stata raggiunta la fine del file.

La dichiarazione di uno `stream` file

Ogni variabile che punti alla struttura di dati relativa ad un file deve essere dichiarata:

```
FILE  *f1, *f2, *f3;
```

ove **`f1, f2 e f3`** sono 3 diverse variabili pointer a 3 diversi file.

Associazione di uno stream ad un file

L'**associazione dello stream al file** avviene mediante la funzione di libreria **fopen()**, che riceve quali parametri due stringhe, contenenti, rispettivamente, il nome del file (eventualmente completo di path) e l'indicazione della modalità di apertura del medesimo.

```
FILE   *fopen(nome-file, modo-apertura);
```

Aprire un file significa rendere disponibile un "canale" di accesso al medesimo, attraverso il quale leggere e scrivere i dati; il nome del file deve essere valido secondo le regole del sistema operativo, mentre le modalità possibili di apertura sono le seguenti:

Modalità di apertura di un file con fopen()

MODO	Significato
"r"	sul file sono possibili solo operazioni di lettura; il file deve esistere.
"w"	sul file sono possibili solo operazioni di scrittura; il file, se non esistente, viene creato; se esiste la sua lunghezza è troncata a 0 byte.
"a"	sul file sono possibili solo operazioni di scrittura, ma a partire dalla fine del file (append mode); in pratica il file può essere solo "allungato", ma non sovrascritto. Il file, se non esistente, viene creato.
"r+"	sul file sono possibili operazioni di lettura e di scrittura. Il file deve esistere.
"w+"	sul file sono possibili operazioni di lettura e di scrittura. Il file, se non esistente, viene creato; se esiste la sua lunghezza è troncata a 0 byte.
"a+"	sul file sono possibili operazioni di lettura e di scrittura, queste ultime a partire dalla fine del file (append mode); in pratica il file può essere solo "allungato", ma non sovrascritto. Il file, se non esistente, viene creato.

Utilità dell'apertura di un file

La `fopen()` restituisce un valore che deve essere assegnato allo stream, perché questo possa essere in seguito utilizzato per le desiderate operazioni sul file; in caso di errore viene restituito `NULL`.

Si possono pertanto interpretare correttamente le righe di codice

```
if(!(outStream = fopen("C:\PROVE\PIPPO","wt")))
    fprintf(stderr,"Errore nell'apertura del file.\n");
```

Con la chiamata a `fopen()` viene aperto il file `PIPPO` (se non esiste viene creato), per operazioni di sola scrittura. Il file aperto è associato allo stream `outStream`; in caso di errore (se `fopen()` restituisce `NULL`) viene visualizzato un opportuno messaggio (scritto sullo standard error).

Chiusura di un file

Al termine delle operazioni sul file è opportuno "chiuderlo", cioè rilasciare le risorse di sistema che il sistema operativo dedica alla sua gestione. La funzione **`fclose()`**, inoltre, rilascia anche lo stream precedentemente allocato da `fopen()`, che non può più essere utilizzato, salvo, naturalmente, il caso in cui gli sia assegnato un nuovo valore restituito da un'altra chiamata alla `fopen()`.

La `fclose()` richiede che le sia passato come parametro lo stream da chiudere e non restituisce alcun valore.

Per chiudere tutti gli stream aperti dal programma è disponibile la **`fcloseall()`**, che non richiede alcun parametro.

Operazioni di lettura e scrittura su file

Le operazioni di lettura e scrittura su file possono essere effettuate in 4 modi diversi:

- ✓ precisando il formato dei dati (**I/O formattato**)
- ✓ accedendo ai dati carattere per carattere (**I/O per caratteri**)
- ✓ accedendo ai dati linea (stringa di caratteri) per linea (**I/O per linee**)
- ✓ accedendo ai dati blocco di dati (testuali o binari) per blocco di dati (**I/O per blocchi**)

I/O formattato

Funzioni di libreria

`printf(`*stringa di controllo*, *elementi***`)`**	consente di scrivere gli *elementi* indicati con format sullo standard output (stdout), il video
`scanf(`*stringa di controllo*, *indirizzo elementi***`)`**	consente di leggere gli *elementi* indicati con format dallo standard input (stdin), la tastiera
`fprintf(``FILE *outStream,` *stringa di controllo*, *elementi***`)`**	consente di scrivere gli *elementi* indicati con format sul file precisato dall'utente tramite il puntatore `outStream`
`fscanf(``FILE *inStream,` *stringa di controllo*, *indirizzo elementi***`)`**	consente di leggere gli *elementi* indicati con format dal file precisato dall'utente tramite il puntatore `inStream`

I/O formattato

La scrittura nel file è effettuata da `fprintf()`, in modo del tutto analogo a quello già sperimentato con `stdout` e `stderr`; la sola differenza è che questa volta lo stream si chiama `outStream`. La `fprintf()`scrive sullo stream secondo le specifiche di una stringa di formato e restituisce il numero di caratteri scritti; la restituzione del valore associato alla costante manifesta `EOF` (definita in `STDIO.H`) significa che si è verificato un errore.

Come **`fprintf()`**, anche **`fscanf()`**opera su uno **stream di input formattato** e richiede che i primi due parametri siano, rispettivamente, lo stream e una stringa di formato ed accetta un numero variabile di parametri. Tuttavia i parametri di `fscanf()` che seguono la stringa di formato sono puntatori alle variabili che dovranno contenere i dati letti dallo stream. Anche `fscanf()` restituisce il numero di campi ai quali ha assegnato un valore.

L'uso dei puntatori è indispensabile, perché `fscanf()` deve restituire alla funzione chiamante un certo numero di valori, cioè modificare il contenuto di un certo numero di variabili: dal momento che in C le funzioni possono restituire un solo valore e, comunque, il passaggio dei parametri avviene mediante una copia del dato originario, l'unico metodo possibile per modificare effettivamente quelle variabili è utilizzare puntatori che le indirizzino. La stringa di formato descrive l'aspetto di ciò che la funzione legge dallo stream. In particolare, per ogni carattere diverso da spazio, tabulazione, a capo (`"\n"`) e percentuale, `fscanf()` si aspetta in arrivo dallo stream un carattere; in corrispondenza di uno spazio, tabulazione o ritorno a capo la funzione continua a leggere dallo stream in attesa del primo carattere diverso da uno dei tre e trascura tutti gli spazi, tabulazioni e ritorni a capo; il carattere "%"introduce una specifica di formato che indica a `fscanf()` come convertire i dati provenienti dallo stream.

E' evidente che deve esserci una corrispondenza tra le direttive di formato e i puntatori passati alla funzione: ad esempio, ad una direttiva "%d", che indica un intero, deve corrispondere un puntatore ad intero. Il carattere `"*"` posto tra il carattere "%" e quello che indica il tipo di conversione indica a `fscanf()` di ignorare quel campo.

I/O per caratteri

Funzioni di libreria

`getchar(void)`	legge il prossimo carattere dallo standard input (stdin), la tastiera, restituendolo come intero
`putchar(int c)`	scrive il prossimo carattere sullo standard output (stdout), il video
`getc(FILE *inStream)`	legge il prossimo carattere dal file precisato dall'utente tramite il puntatore `inStream`
`putc(int c,FILE *outStream)`	scrive il prossimo carattere sul file precisato dall'utente tramite il puntatore `outStream`
`fgetc(FILE *inStream)`	legge il prossimo carattere dal file precisato dall'utente tramite il puntatore `inStream`
`fputc(int c,FILE *outStream)`	scrive il prossimo carattere sul file precisato dall'utente tramite il puntatore `outStream`

`getc()` e `put()` sono realizzate in maniera da risultare più veloci in esecuzione, ma presentano il rischio di effetti collaterali.

I/O per linee

Funzioni di libreria

`*gets(char *s)`	legge caratteri dallo stdin fino a raggiungere un carattere di `newline` o di end-of-file (`EOF`). `s` punta al primo elemento del vettore contenente i dati letti, che ha, come ultimo elemento non `newline` ma `0`. In caso di errore restituisce `NULL`.
`puts(char *s)`	scrive sullo stdout il contenuto della stringa puntata da `s`, seguita da un carattere di `newline`. Restituisce il valore `0` se l'operazione viene eseguita correttamente, $\neq$`0` altrimenti.
`*fgets(char *s, int n, FILE *inStream)`	legge caratteri dal file puntato da `inStream` finchè non ha letto `n-1` elementi o un carattere di `newline` un `EOF`. Per il resto agisce come `*gets()`.
`puts(char *s, FILE *outStream)`	scrive sul file puntato da `outStream` il contenuto della stringa puntata da `s`, senza ulteriori caratteri. Per il resto agisce come `puts()`.

I/O per blocchi (dati testuali o binari)

Funzioni di libreria

`fread(void *ptr, size-of, num-of, FILE *inStream)`

Legge un blocco di dati testuali o binari dal file puntato da `inStream` e li memorizza nel vettore puntato da `ptr`. La funzione termina correttamente se legge il numero di byte richiesti (`size-of` * `num-of`). Termina anche se incontra un EOF o se si verifica un errore. Restituisce il numero di elementi effettivamente letti.

`fwrite(void *ptr, size-of, num-of, FILE *outStream)`

scrive un blocco di dati testuali o binari sul file puntato da `outStream` prelevandoli dal vettore puntato da `ptr`. La funzione termina correttamente se scrive il numero di byte richiesti (`size-of` * `num-of`). Termina anche se incontra un EOF o se si verifica un errore. Restituisce il numero di elementi effettivamente scritti.