

IL CONCETTO DI FILE

- Un file è una *astrazione fornita dal sistema operativo*, il cui scopo è consentire la memorizzazione di informazioni su memoria di massa.
- Concettualmente, un file è una *sequenza di registrazioni (record) uniformi*, cioè dello stesso tipo.
- Un file è un'astrazione di memorizzazione di *dimensione potenzialmente illimitata* (ma non infinita), *ad accesso sequenziale*.

APERTURA DI FILE

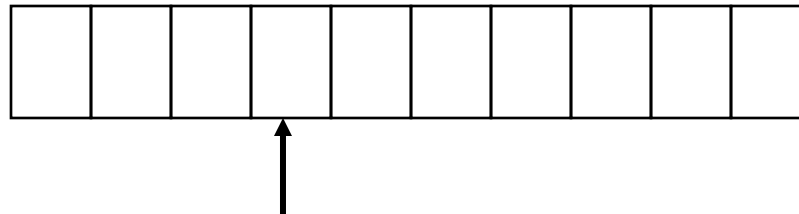
- Poiché un file è un'entità del sistema operativo, per agire su esso dall'interno di un programma occorre *stabilire una corrispondenza* fra:
 - il nome del file come risulta al sistema operativo
 - un nome di variabile definita nel programma.
- Questa operazione si chiama *apertura del file*
- Esistono varie modalità di *apertura del file*
 - apertura in lettura
 - apertura in scrittura
 - ...

APERTURA E CHIUSURA DI FILE

- Una volta aperto il file, il programma può operare su esso *operando formalmente sulla variabile definita al suo interno*
 - il sistema operativo provvederà a effettuare realmente l'operazione richiesta sul file associato a tale simbolo.
- Al termine, la corrispondenza fra *nome del file* e *variabile usata dal programma per operare su esso* dovrà essere soppressa, mediante l'operazione di *chiusura del file*.

LETTURA DI FILE

- Una testina di lettura/scrittura (concettuale) indica in ogni istante il record corrente:
 - inizialmente, la testina si trova per ipotesi sulla prima posizione
 - dopo ogni operazione di lettura / scrittura, essa si sposta sulla registrazione successiva.



- È illecito operare oltre la fine del file.

FILE IN C

- Per gestire i file, il C definisce il tipo **FILE**.
- **FILE** è una struttura definita nello header standard **stdio.h**, che l'utente non ha necessità di conoscere nei dettagli – e che spesso cambia da un compilatore all'altro!
- Le strutture **FILE** non sono *mai* gestite direttamente dall'utente, ma solo dalle funzioni della libreria standard **stdio**.
- L'utente definisce e usa, nei suoi programmi, solo *puntatori a FILE*.

FILE IN C

- Libreria standard `stdio`
`#include <stdio.h>`
- l'input avviene da un canale di input associato a un file *aperto in lettura*
- l'output avviene su un canale di output associato a un file *aperto in scrittura*
- Due tipi di file: *file binari* e *file di testo*
 - basterebbero i file binari, ma fare tutto con essi sarebbe scomodo
 - i file di testo, *pur non indispensabili*, rispondono a un'esigenza pratica molto sentita.

FILE IN C: APERTURA

- Per aprire un file si usa la funzione:

```
FILE* fopen(char fname[], char modo[])
```

Apre il file di nome **fname** nel **modo** specificato, e restituisce un puntatore a **FILE**

modo specifica *come* aprire il file:

- **r** apertura in lettura (read)
 - **w** apertura in scrittura (write)
 - **a** apertura in aggiunta (append)
- seguita opzionalmente da:
 - **t** apertura in modalità testo (default)
 - **b** apertura in modalità binaria

FILE IN C: APERTURA

- Il *puntatore a FILE* restituito da `fopen()` si deve usare in tutte le successive operazioni sul file.
 - esso è NULL in caso l'apertura sia fallita
 - controllarlo è *il solo modo per sapere se il file si sia davvero aperto: non dimenticarlo!*
- I tre canali predefiniti standard (`stdin`, `stdout`, `stderr`) sono *in tutto e per tutto dei file aperti automaticamente all'inizio dell'esecuzione di ogni programma*: quindi, il loro tipo è **FILE***.

FILE IN C: CHIUSURA

Per chiudere un file si usa la funzione:

```
int fclose(FILE*)
```

- Il valore restituito da `fclose()` è un intero
 - 0 se tutto è andato bene
 - EOF in caso di errore.
- Prima della chiusura, tutti i buffer vengono svuotati.

FILE BINARI

- Un file binario è una sequenza di byte: come tale, può essere usato per archiviare su memoria di massa *qualunque tipo di informazione*
- input e output avvengono sotto forma di una sequenza di byte
- la lunghezza del file è registrata dal sistema operativo

FILE DI TESTO COME FILE BINARI

- È un caso particolare di file binario, che coinvolge una *sequenza di caratteri*
- Ha senso trattarlo come caso a parte perché i caratteri sono un caso *estremamente frequente, con caratteristiche proprie*:
 - esiste un concetto di *linea* e di *fine linea* ('\n')
 - certi caratteri sono *stampabili a video* (quelli di codice ≥ 32), altri no
 - la sequenza di caratteri è chiusa dal carattere speciale EOF

FILE BINARI: LETTURA/SCRITTURA

- Poiché un file binario è una sequenza di byte, sono fornite due funzioni per *leggere* e *scrivere* sequenze di byte
 - **fread()** legge una sequenza di byte
 - **fwrite()** scrive una sequenza di byte
- Essendo pure sequenze di byte, possono rappresentare *qualunque informazione* (testi, numeri, immagini...)

fwrite()

Sintassi:

```
int fwrite(addr, int dim, int n, FILE *f);
```

- scrive sul file **n** elementi, ognuno grande **dim** byte (complessivamente, scrive quindi $n \times \text{dim}$ byte)
- gli elementi da scrivere vengono prelevati dalla memoria a partire dall'indirizzo **addr**
- restituisce il numero di elementi (non di byte!) effettivamente scritti, che possono essere meno di n.

fread()

Sintassi:

```
int fread(addr, int dim, int n, FILE *f);
```

- legge dal file **n** elementi, ognuno grande **dim** byte (complessivamente, legge quindi $n \times \text{dim}$ byte)
- gli elementi da leggere vengono scritti in memoria a partire dall'indirizzo **addr**
- restituisce il numero di elementi (non di byte!) effettivamente letti, che possono essere meno di n se il file finisce prima: *al limite anche zero*. *Controllare il valore restituito è il solo modo per sapere se il file è finito.*

ESEMPIO

Salvare su un file binario `numeri.dat` il contenuto di un array di dieci interi.

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

int main() {
    FILE *fp;
    int vet[10] = {1,2,3,4,5,6,7,8,9,10};
    if ((fp = fopen("numeri.dat", "wb")) == NULL)
        exit(1); /* Errore di apertura */
    fwrite(vet, sizeof(int), 10, fp);
    fclose(fp);
    return 0;}

```

La funzione `exit()` fa terminare il programma anticipatamente.

L'operatore `sizeof` è essenziale per la portabilità

ESEMPIO

Leggere da un file binario `numeri.dat` una sequenza di interi, scrivendoli in un array.

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    FILE *fp;
    int vet[40], i, n;
    if ((fp = fopen("numeri.dat", "rb")) == NULL)
        exit(1); /* Errore di apertura */
    n = fread(vet, sizeof(int), 40, fp);
    for (i=0; i<n; i++) printf("%d ", vet[i]);
    fclose(fp);
    return 0;}
```

`fread` tenta di leggere 40 interi, ma ne legge meno se il file finisce prima (come in questo caso)

`n` contiene il numero di interi effettivamente letti

ESEMPIO COMPLETO FILE BINARIO

È dato un file di binario `people.dat` i cui record rappresentano *ciascuno i dati di una persona*, secondo il seguente formato:

- **cognome** (al più 30 caratteri)
- **nome** (al più 30 caratteri)
- **sesso** (un singolo carattere, 'M' o 'F')
- **anno di nascita**

Si noti che la creazione del file binario deve essere fatta da programma, mentre per i file di testo può essere fatta con un text editor.

CREAZIONE FILE BINARIO

Per creare un file binario e' necessario scrivere un programma che lo crei strutturandolo modo che ogni record contenga una **struct persona**

```
struct persona{  
    char cognome[31], nome[31], sesso[2];  
    int anno;  
};
```

I dati di ogni persona da inserire nel file vengono richiesti all'utente tramite la funzione `leggie1()` che non ha parametri e restituisce come valore di ritorno la **struct persona** letta. Quindi il prototipo e':

```
struct persona leggiel();
```

CREAZIONE FILE BINARIO

Mentre la definizione e':

```
struct persona leggiel() {
    struct  persona e;

    printf("Cognome ? ");
    scanf("%s", e.cognome);
    printf("\n Nome ? ");
    scanf("%s", e.nome);
    printf("\nSesso ? ");
    scanf("%s", e.sesso);
    printf("\nAnno nascita ? ");
    scanf("%d", &e.anno);
    return e;
}
```

CREAZIONE FILE BINARIO

```
#include <stdio.h>
#include <stdlib.h>
struct persona{
    char cognome[31], nome[31], sesso[2];
    int anno;
};
struct persona leggiel();
int main(){
    FILE *f; struct persona e; int fine=0;
    f=fopen("people.dat", "wb");
    while (!fine)
        { e=leggiel();
          fwrite(&e,sizeof(struct persona),1,f);
          printf("\nFine (SI=1, NO=0) ? ");
          scanf("%d", &fine);
        }
    fclose(f);
    return 0;}
```

CREAZIONE FILE BINARIO

L'esecuzione del programma precedente crea il file binario contenente i dati immessi dall'utente. Solo a questo punto il file può essere utilizzato.

Il file `people.dat` non è visualizzabile tramite un text editor: questo è il risultato

```
rossi > ÿÿ @T  —8
  â3 mario_ÄAw O F _ DÝ
M
```

ESEMPIO COMPLETO FILE BINARIO

Ora si vuole scrivere un programma che

- legga record per record i dati dal file
 - e ponga i dati in un array di persone
 - *(poi svolgeremo elaborazioni su essi)*
-

ESEMPIO COMPLETO FILE BINARIO

Come organizzarsi?

- 1) Definire una struttura **persona**

Poi, nel main:

- 2) Definire un array di strutture **persona**
- 3) Aprire il file in lettura
- 4) Leggere un record per volta, e porre i dati di quella persona in una cella dell'array
 - Servirà un indice per indicare la prossima cella libera nell'array.

ESEMPIO COMPLETO FILE BINARIO

1) Definire una struttura di tipo **persona**

Occorre definire una struct adatta a ospitare i dati elencati:

- **cognome** → array di 30+1 caratteri
- **nome** → array di 30+1 caratteri
- **sex** → array di 1+1 caratteri
- **anno di nascita** → un intero

ricordarsi lo spazio per il terminatore

```
struct persona{  
    char cognome[31], nome[31], sesso[2];  
    int anno;  
};
```


ESEMPIO COMPLETO FILE BINARIO

Poi, nel main:

- 2) definire un array di `struct persona`
- 3) aprire il file in lettura

```
int main() {  
    struct persona v[DIM];  
    FILE* f = fopen("people.dat", "r");  
    if (f==NULL) {  
        .../* controllo che il file sia  
            effettivamente aperto */  
    }  
    ...  
    return 0; }
```

Hp: massimo DIM
persone

ESEMPIO COMPLETO FILE BINARIO

Poi, nel main:

- 2) definire un array di `struct persona`
 - 3) aprire il file in lettura
-

```
int main() {  
    struct persona v[DIM];  
    FILE* f = fopen("people.dat", "r");  
    if (f==NULL) {  
        printf("Il file non esiste");  
        exit(1); /* terminazione del programma */  
    }  
    ...  
    return 0; }
```

ESEMPIO COMPLETO FILE BINARIO

Poi, nel main:

- 4) leggere i record dal file, e porre i dati di ogni persona in una cella dell'array

Come organizzare la lettura?

```
int fread(addr, int dim, int n, FILE *f) ;
```

- legge dal file **n** elementi, ognuno grande **dim** byte (complessivamente, legge quindi $n \times \text{dim}$ byte)
- gli elementi da leggere vengono scritti in memoria a partire dall'indirizzo **addr**

Uso fread

ESEMPIO COMPLETO FILE BINARIO

Poi, nel main:

- 4) leggere i record dal file, e porre i dati di ogni persona in una cella dell'array

Cosa far leggere a **fread**?

- *L'intero vettore di strutture: unica lettura per DIM record*

`fread(v, sizeof(struct persona), DIM, f)`

- *Un record alla volta all'interno di un ciclo*

```
i=0
while(!feof(f)) {
    fread(&v[i], sizeof(struct persona), 1, f);
    i++
}
```

ESEMPIO COMPLETO FILE BINARIO

Poi, nel main:

- 4) leggere i record dal file, e porre i dati di ogni persona in una cella dell'array

Dove mettere quello che si legge?

- Abbiamo definito un array di `struct persona`, `v`
- L'indice `k` indica la prima cella libera → `v[k]`
- Tale cella è una struttura fatta di *cognome*, *nome*, *sex*, *anno* → ciò che si estrae da un record va direttamente nella struttura `v[k]`

ESEMPIO COMPLETO FILE BINARIO

```
#define DIM 30
#include <stdio.h>
#include <stdlib.h>
```

Dichiara la procedura `exit()`

```
struct persona{
    char cognome[31], nome[31], sesso[2];
    int anno;
};
```

```
int main() {
    struct persona v[DIM]; int i=0; FILE* f;
    if ((f=fopen("people.dat", "r"))==NULL) {
        printf("Il file non esiste!"); exit(1); }
    while (fread(&v[i], sizeof(struct persona), 1, f) > 0) {
        i++;
    }
    fclose(f);
    return 0;}
}
```

ESEMPIO COMPLETO FILE BINARIO

```
#define DIM 30
#include <stdio.h>
#include <stdlib.h>
```

Dichiara la procedura `exit()`

```
struct persona{
    char cognome[31], nome[31], sesso[2];
    int anno;
};
```

```
int main() {
    struct persona v[DIM]; int i=0; FILE* f;
    if ((f=fopen("people.dat", "r"))==NULL) {
        printf("Il file non esiste!"); exit(1); }
    fread(v,sizeof(struct persona),DIM,f);
    fclose(f);
    return 0;}
}
```