11 - File

I file sono strutture dati persistenti, utilizzate per memorizzare informazioni che devono sopravvivere alla terminazione di un programma.
Essi sono solitamente memorizzati su dischi e possono essere letti o scritti da programmi, la loro funzione principale è garantire la **persistenza dei dati**, cioè mantenere il contenuto delle variabili anche dopo la fine dell'esecuzione del programma.

Tipologie di File: Testo e Binari

I file sono strutture dati **sequenziali**, il che significa che i dati vengono letti e scritti in sequenza. Esistono due tipi principali di file:

- **File di testo**: contengono caratteri leggibili (ad esempio lettere, numeri e simboli) e sono interpretati. Il contenuto può essere facilmente letto dagli utenti e alcuni caratteri, come il newline (`\n`), hanno significati specifici.
- **File binari**: sono sequenze di byte non interpretate, contengono dati generici che non sono direttamente leggibili dagli utenti senza un'applicazione specifica.

Dal punto di vista del computer, non c'è differenza tra file di testo e binari, poiché entrambi sono sequenze di byte. La distinzione sta solo nell'interpretazione dei dati.

Gestione dei File e il Sistema Operativo

I file sono gestiti dal sistema operativo (S.O.), che li rende visibili ai programmi attraverso funzioni di libreria. Prima di essere utilizzato un file deve essere **aperto** e successivamente deve essere **chiuso** per liberare risorse.
In C, anche le periferiche (come terminali e stampanti) sono trattate come file, consentendo operazioni di lettura e scrittura uniformi come dei file di testo con funzioni come `stdin` o `stdout`

Rappresentazione Interna dei File

Quando un file viene aperto, il sistema operativo assegna un **descrittore di file**, che contiene informazioni come:

- Modalità d'uso (lettura o scrittura).
- Posizione corrente all'interno del file.
- Indicatore di errore
- Indicatore di fine file (EOF).

Il descrittore è un puntatore a una struttura che memorizza questi dati.

Dichiarazione e Apertura di un File

In C, un file viene gestito tramite un puntatore a un descrittore (`FILE * fp`). Per aprire un file, si utilizza la funzione

```
FILE * fopen (char * nomefile, char * modalità)
```

che accetta due argomenti: il nome del file (che da il percorso del file o viene interpretato nella cartella dell'eseguibile) e la modalità di apertura (o creazione se inesistente) . Le modalità principali sono:

- `"r"` : lettura (testo).
- `"w"` : scrittura (cancella il contenuto esistente).
- `"a"` : aggiunta (scrive alla fine del file).
- `"rb"` , `"wb"` , `"ab"` : analoghe alle precedenti, ma per file binari.

Se l'apertura fallisce, `fopen` restituisce `NULL` .

Chiusura dei File

Ogni volta che un file viene aperto, il sistema crea un descrittore dove ci sono i dettagli e un buffer in memoria dove viene portata una porzione di file.
Per evitare sprechi di memoria, è essenziale chiudere i file con `fclose` , questa funzione rilascia il descrittore di tipo FILE e restituisce `0` in caso di successo o `EOF` in caso di errore.

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

Gestione della Fine del File

Per verificare se si è raggiunta la fine di un file, si utilizza la funzione `feof` , che restituisce `0` se non si è alla fine del file.

```
int feof (FILE *fp)
```

Lettura e Scrittura nei File

Esistono diversi modi per leggere e scrivere nei file:

1. **File di testo**:
   - Formattato (con `fprintf` e `fscanf`).
   - Un carattere alla volta.
   - Per linee di testo (fino a `\n`).
2. **File binari**:
   - Per blocchi di byte, ad esempio con le funzioni `fread` e `fwrite` .

Le funzioni `fprintf` e `fscanf` sono analoghe a `printf` e `scanf`, ma operano su file specifici.

```
int fprintf (FILE * fp, str_di_controllo, elementi)
```

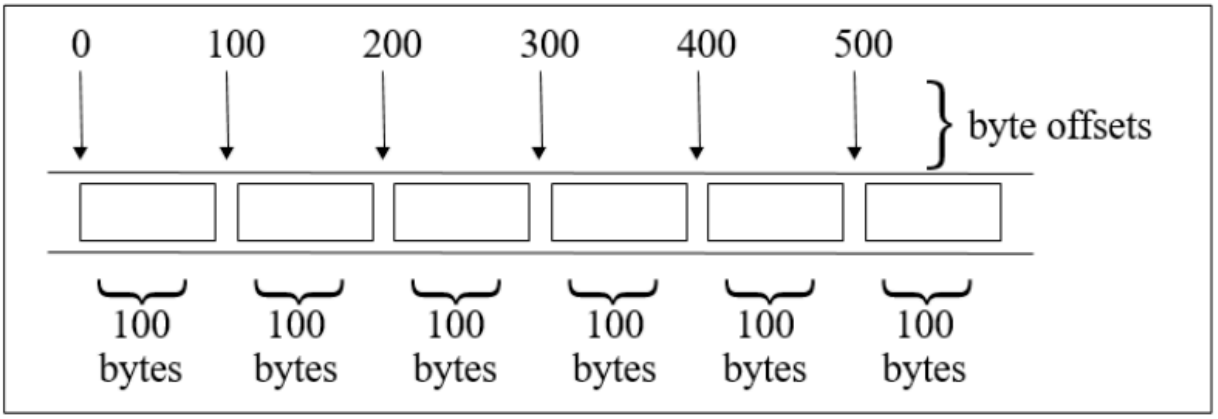
```
int fscanf (FILE * fp, str_di_controllo, indirizzo_elementi)
```

Queste funzioni restituiscono il numero di elementi effettivament letti/scritti, o zero se errore

Modalità di Apertura dei File

Modalità	Descrizione
`r`	Lettura (testo).
`w`	Scrittura (cancella contenuto esistente).
`a`	Aggiunta alla fine del file.
`r+`	Lettura e scrittura (testo).
`w+`	Scrittura e lettura (cancella contenuto esistente).
`a+`	Aggiunta e lettura.
`rb`	Lettura (binario).
`wb`	Scrittura (binario).
`ab`	Aggiunta (binario).
`rb+`	Lettura e scrittura (binario).
`wb+`	Scrittura e lettura (binario).
`ab+`	Aggiunta e lettura (binario).

I dati di un file binario ad accesso casuale non sono formattati (memorizzati come raw bytes), usando quindi tutti la stessa quantità di memoria e lunghezza, oltre a non essere leggibili da un umano



Lettura e Scrittura per Blocchi di Byte

La lettura e la scrittura di file binari avvengono tramite blocchi di byte. Questo approccio è particolarmente utile quando si devono manipolare dati complessi, come strutture o array, poiché consente di operare su un'intera porzione di memoria in un'unica operazione.
Le funzioni principali per la lettura e scrittura di blocchi di byte sono:

- `fread` : legge un numero specifico di byte da un file e li memorizza in un'area di memoria.

```
int fread(void *punt, size_t dim_blocco, size_t num_blocchi, FILE *fp);
```

- `punt` : puntatore all'area di memoria dove memorizzare i dati.
- `dim_blocco` : dimensione di ogni blocco in byte.
- `num_blocchi` : numero di blocchi da leggere.
- `fp` : puntatore al file.

- `fwrite` : scrive un numero specifico di byte da un'area di memoria a un file.

```
 int fwrite(const void *punt, size_t dim_blocco, size_t num_blocchi, FILE
*fp);
```

- `punt` : puntatore all'area di memoria contenente i dati da scrivere.
- `dim_blocco` : dimensione di ogni blocco in byte.
- `num_blocchi` : numero di blocchi da scrivere.
- `fp` : puntatore al file.

Entrambe le funzioni restituiscono il numero effettivo di blocchi letti o scritti. Se il valore restituito è inferiore a `num_blocchi` , potrebbe essersi verificato un errore o essere stato raggiunto l'end-of-file (EOF) durante la lettura.

Accesso Diretto ai File

L'accesso diretto consente di manipolare il puntatore del file per accedere a specifiche posizioni, come se il file fosse un array di blocchi di byte. Questo approccio è reso possibile da funzioni come `fseek` e `rewind` .

La funzione `fseek` permette di spostare il puntatore del file a una posizione specifica, calcolata rispetto a un punto di riferimento. La sua sintassi è:

```
int fseek(FILE *fp, long offset, int refpoint);
```

- `fp` : puntatore al file.
- `offset` : spostamento (positivo o negativo) in byte.

- `refpoint`: punto di riferimento per il calcolo della posizione:
  - `SEEK_SET`: inizio del file.
  - `SEEK_CUR`: posizione corrente.
  - `SEEK_END`: fine del file.

  Ad esempio:
- `fseek(fp, 0L, SEEK_SET)` posiziona il puntatore all'inizio del file.
- `fseek(fp, 0L, SEEK_END)` posiziona il puntatore alla fine del file.

La funzione restituisce `0` se l'operazione ha successo, o un valore diverso in caso di errore.

La funzione `rewind` è un'operazione semplificata che riporta il puntatore all'inizio del file e azzera l'indicatore di errore:

```
void rewind(FILE *fp);
```

È equivalente a chiamare `fseek(fp, 0, SEEK_SET)`.