

# **Corso di Sistemi Operativi**

## **APPUNTI SUL LINGUAGGIO C**



**Memoria secondaria: File**

# Gestione della memoria secondaria

La gestione della memoria secondaria in un linguaggio di programmazione consente la memorizzazione permanente dei dati e la loro condivisione tra programmi diversi.

Questo è utile quando:

- ✓ la mole dei dati non consente la gestione in memoria principale
- ✓ i dati hanno un'esistenza indipendente dalle esecuzioni dei programmi che li utilizzano.

In memoria secondaria i dati sono organizzati in **file**.



## File

Un file è un contenitore di informazione permanente.

- La “vita” di un file è indipendente dalla vita del programma che ne fa uso.
- Un file continua ad esistere anche dopo il termine dell'esecuzione del programma.

# File


**File:** a **livello logico** è una sequenza (insieme) di record.

Data

ora	minuti	secondi
-----	--------	---------



**Record**

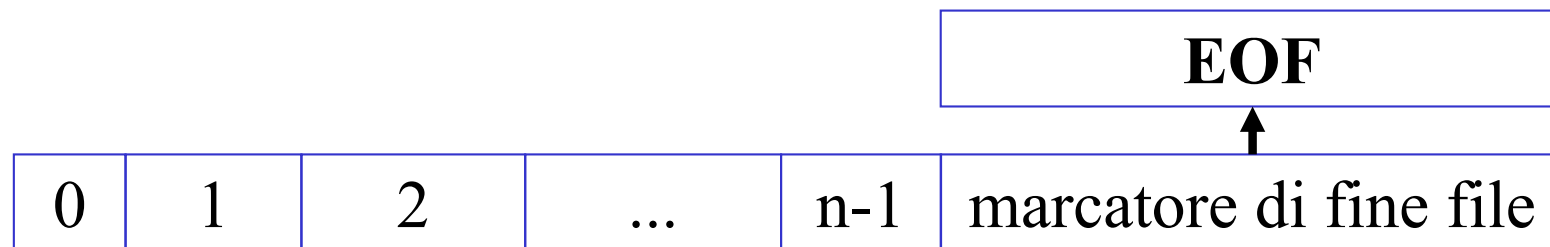



**File**

## File

File: a **livello logico** è una sequenza (insieme) di record.

Il **metodo di accesso** a un file è la tecnica usata dal programma per accedere ai record del file.



*Il C vede in questo modo un file: una sequenza di byte con un marcatore alla fine per indicare la fine del file.*

## **File**

Un programma C prima di poter utilizzare un file deve aprire un flusso di comunicazione. Al termine dell'utilizzo di un file il flusso di comunicazione viene chiuso.

- ❑ **Flusso di Caratteri**

- ❑ **Flusso Binario**

## Definizione File

Dichiarazione di file in C

**FILE \*fp;**

Definisce una variabile di tipo "puntatore a file".

## Gestione File C

Nel linguaggio C:

- un file è semplicemente una sequenza di byte,
- ogni dispositivo fisico viene visto come un file.

Funzioni per la gestione dei file in ANSI C

- **fopen, fclose;**
- **fgetc, fputc;**
- **fgets, fputs;**
- **fprintf, fscanf;**
- **fread, fwrite;**
- **fseek, fsetpos;**
- **ftell, fgetpos;**

Contenute nella libreria: **stdio.h**



## Apertura di un file

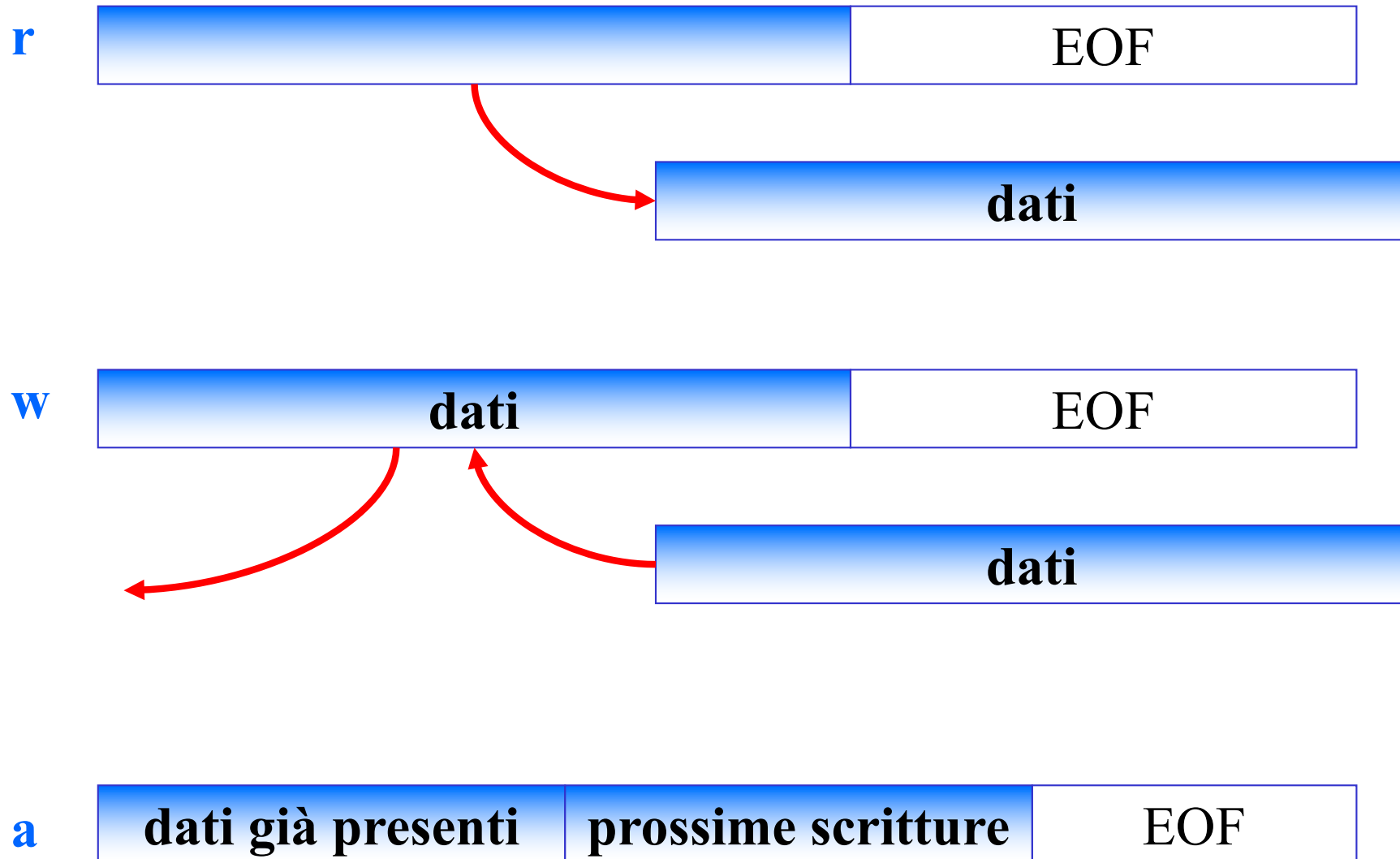
La funzione **fopen** "apre" un file ovvero lo predispone ad una certa elaborazione e inizializza una area di memoria (detta buffer o canale) attraverso la quale vengono scambiati i dati tra memoria principale e secondaria. E' una operazione che deve essere effettuata sempre prima di ogni tipo di elaborazione sul file. Se l'apertura del file non ha successo la funzione restituisce **NULL**.

## Apertura di un file

È possibile specificare la modalità di apertura di un file attraverso uno dei seguenti caratteri:

- **r** [read] il file deve già esistere; viene aperto in lettura e viene restituito il puntatore all'inizio del file;
- **w** [write] il file viene creato se non esiste o viene sovrascritto se esiste; viene aperto in scrittura e viene restituito il puntatore all'inizio del file;
- **a** [append] il file viene creato se non esiste; viene aperto in scrittura con inserimenti solo in fondo al file e viene restituito il puntatore alla fine del file;
- **+** [update] si usa in combinazione con una delle prime tre modalità; permette l'aggiornamento del file;
- **b** [binary] si usa in combinazione con una delle prime tre modalità; permette la gestione di dati in formato binario;

## Apertura di un file



## Apertura di un file

`FILE * fopen(nomefile, modalità)`

- ❑ nomefile - stringa che indica il nome del file
- ❑ modalità - modalità di apertura:
  - o “r” - Lettura (testo)
  - o “w” - Scrittura (testo)
  - o “a” - Scrittura a fine file (testo)
  - o “rb” - Lettura (binario)
  - o “wb” - Scrittura (binario)
  - o “ab” - Scrittura a fine file (binario)
  - o “r+”, “w+”, “a+” – Lettura e Scrittura (testo)
  - o “rb+”, “wb+”, “ab+” – Lettura e Scrittura (binario)

## Chiusura di un file

La funzione **fclose** "chiude" un file ovvero lo rilascia chiudendo il relativo canale di comunicazione. Se la chiusura del file non ha successo la funzione restituisce **NULL**. E' una operazione che deve essere effettuata sempre alla fine delle elaborazione sul file.

## Chiusura di un file

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

□ fp puntatore al file da chiudere

## Chiusura di un file

Esempio:

```
FILE *fp;
```

```
....
```

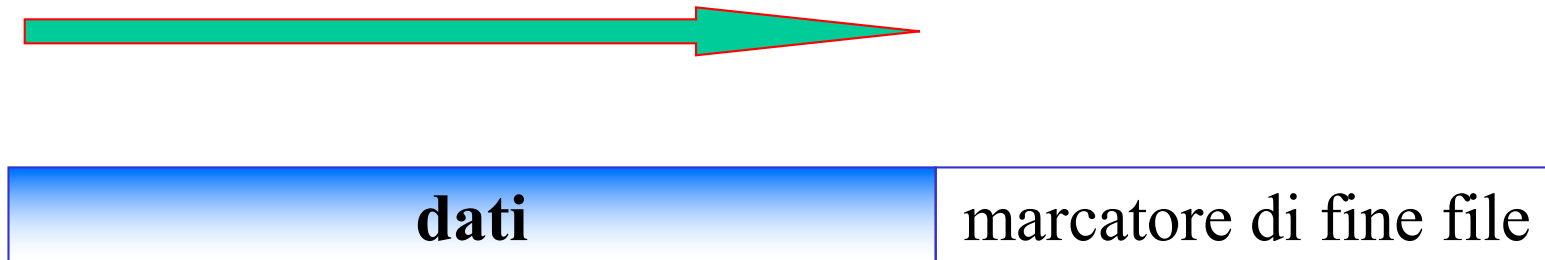
```
fp = fopen("dati.txt","r+");
```

```
....
```

```
fclose(fp);
```

## Accesso Sequenziale File C

Un file è una sequenza di dati, e tutte le operazioni su questi partiranno dall'inizio e si muoveranno in sequenza fino alla fine del file



- **Operazioni di input/output;**
- **Operazioni di lettura e scrittura.**



## Lettura e scrittura di un file

La funzione **fputc** scrive un singolo carattere in un file. Restituisce il carattere stesso se l'operazione è andata a buon fine, altrimenti restituisce **EOF** (costante speciale il cui valore non è significativo).

```
int fputc(int c, FILE *pf)
```

c – carattere da scrivere sul file

pf – puntatore al file

valore di ritorno:

- o EOF – in caso di errore
- o c – operazione OK

# Letture e scrittura di un file

**Esempio:**

```
/* scrive in un file i caratteri in input */  
#include<stdio.h>  
  
main() {  
    FILE *fp;  
    int c;  
    fp=fopen("dati.txt", "w");  
    if (fp==NULL) printf("Errore di I/O");  
    else  
        while((c = getchar()) != '\n')  
            fputc(c, fp);  
        fclose(fp);  
}
```

## Lettura e scrittura di un file

La funzione **fgetc** legge un carattere dal file specificato. Restituisce il carattere letto (come intero) se l'operazione è andata a buon fine, altrimenti restituisce **EOF** se si è raggiunta la fine del file.

```
int fgetc (FILE * pf)
```

pf – puntatore al file

valore di ritorno – carattere letto

# Lettura e scrittura di un file

**Esempio:**

```
/*copia il contenuto di un file in un altro*/  
  
#include<stdio.h>  
  
main() {  
  
    FILE *ifp, *ofp;  
  
    int c;  
  
    ifp=fopen("dati.txt", "r");  
    ofp=fopen("copia.txt", "w");  
  
    while((c = fgetc(ifp)) != EOF) fputc(c, ofp);  
  
    fclose(ofp); fclose(ifp);  
  
}
```

## Letture e scrittura di stringhe

La funzione **fgets** legge un numero specificato di caratteri da un file e li memorizza in una stringa aggiungendo il carattere di fine stringa. Restituisce la stringa se l'operazione è andata a buon fine, altrimenti restituisce **NULL**.

```
char * fgets (char *s, int n, FILE *fp)
```

s – puntatore al vettore su cui vengono inseriti i caratteri letti

n – vengono letti n-1 caratteri, o fino a newline o fino alla fine del file

fp – puntatore al file su cui leggere

valore di ritorno

s – operazione a buon fine

NULL - altrimenti

# Letture e scrittura di stringhe

**Esempio:**

```
/* stampa i primi 10 caratteri di un file */  
#include <stdio.h>  
  
main() {  
    FILE *fp;  
    char str[11];  
    if ((fp=fopen("dati.txt", "r")) == NULL)  
        printf("Errore di I/O");  
    else  
        if(fgets(str, sizeof(str), fp) != NULL)  
            printf("%s", str);  
        fclose(fp);  
}
```

## Letture e scrittura di stringhe

La funzione **fputs** scrive una stringa nel file specificato senza aggiungere il carattere di fine stringa. Restituisce **0** se l'operazione è andata a buon fine, altrimenti restituisce **EOF**.

```
int fputs (char *s, FILE *fp)
```

s – puntatore al vettore degli elementi che saranno inseriti

fp – puntatore al file su cui inserire gli elementi.

valore di ritorno

0 – operazione OK

EOF - altrimenti

## Letture e scrittura di stringhe

**Esempio:**

```
/* appende una stringa in un file */  
#include <stdio.h>  
  
main() {  
    FILE *fp;  
    if ( (fp=fopen ("dati.txt", "a") ) ==NULL)  
        printf ("Errore di I/O");  
    else  
        if (fputs ("Fine", fp) ==EOF)  
            printf ("Errore in scrittura");  
        fclose (fp);  
}
```



## Operazioni di I/O formattate su file

La funzione **fscanf** legge caratteri da un file secondo il formato specificato e assegna i valori letti ai suoi successivi argomenti. Restituisce il numero di caratteri letti se l'operazione è andata a buon fine, altrimenti restituisce **EOF**.

```
int fscanf(FILE *pf, str_cont, elementi)
```

## Operazioni di I/O formattate su file

La funzione **fprintf** scrive caratteri in un file secondo il formato specificato. Restituisce il numero di caratteri scritti se l'operazione è andata a buon fine, altrimenti restituisce un valore negativo.

```
int fprintf(FILE *pf, str_cont, elementi)
```

```
#include <stdio.h>

main() {
    FILE * pf;
    char v[30];
    pf=fopen("pippo.txt", "w");
    fprintf(pf, "%s", "ciao");
    fclose(pf);
    pf=fopen("pippo.txt", "r");
    fscanf(pf, "%s", v);
    fclose(pf);
    printf("%s", v);
}
```

Esempio:

```
/* memorizza una sequenza di interi in un file, scrivendoli uno per
   riga */
#include<stdio.h>
main() {
    FILE *fp;
    int dato, n, i;
    if ((fp=fopen("dati.txt", "w"))==NULL)
        printf("Errore di I/O");
    else {
        printf("N.dati da inserire:");
        scanf("%d", &n);
        for(i=0; i<n; i++) {
            printf("Dato n. %d:", i+1);
            scanf("%d", &dato);
            fprintf(fp, "%d\n", dato);
        }
        fclose(fp);
    }
}
```

## Funzioni di gestione di errori su file

Quando si rileva un errore in una operazione di accesso a file, viene aggiornato degli "indicatori di stato" (fine file ed errore su file). Inoltre, la variabile intera **errno** (in **<errno.h>**) registra un numero che dà indicazioni sul tipo di errore.

La funzione **fEOF** restituisce un valore diverso da **0** se è stata raggiunta la fine del file, altrimenti restituisce **0**.

La funzione **ferror** restituisce un valore diverso da **0** se si è verificato un errore sul file, altrimenti restituisce **0**.

La funzione  **perror** stampa una stringa specificata seguita da un messaggio che indica il tipo di errore commesso.

Esempio:

```
#include<stdio.h>
main() {
FILE *ifp, *ofp;
int c;
ifp=fopen("dati.txt","r");
if (ferror(ifp)) {
perror("Errore di I/O:"); return;
}
ofp=fopen("copia.txt","w");
if (ferror(ofp)) {
perror("Errore di I/O:"); return;
}
c = fgetc(ifp);
while(!feof(ifp)) {
fputc(c, ofp);
c = fgetc(ifp);
}
fclose(ofp); fclose(ifp);
}
```

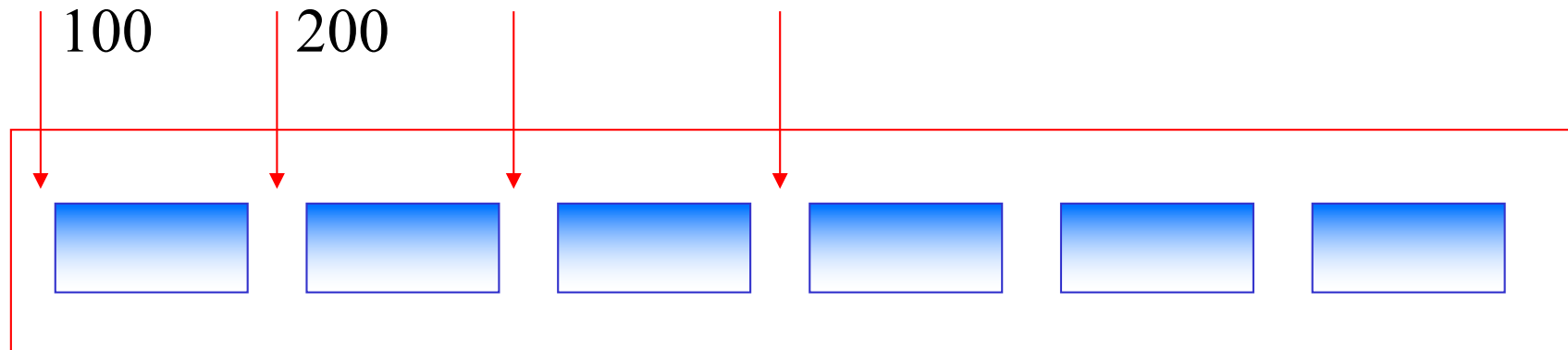
## File speciali

- ❑ **Stdin** : standard di input (tastiera)
  - ❑ **Stdout** : standard di output (monitor)
  - ❑ **Stderr** : standard di errore (spesso sempre il monitor)
- Sono predefiniti e non è necessario ne dichiararli ne ridefinirli.

## Accesso casuale

I record di un file creati con l'output formattato della funzione `fprintf` non avranno necessariamente la stessa lunghezza: la modalità di accesso poi a questi file è di tipo sequenziale (dall'inizio alla fine).

Un file ad **accesso casuale** invece presenta record di lunghezza fissa e vi si potrà accedere in modo diretto (e quindi velocemente), senza passare attraverso altri record.





## Funzioni di I/O diretto su file: Accesso casuale

La funzione **fread** legge da un file un numero specificato di oggetti di una certa ampiezza e li memorizza in un vettore. Restituisce il numero di oggetti letti.

```
int fread (void *buf, int size, int count, FILE *fp);
```

- o **\*buf** è il puntatore alla regione di memoria dove andranno archiviati i dati da leggere;
- o **size** è la lunghezza del singolo dato;
- o **count** è il numero dei dati;
- o **\*fp** è il file sorgente

## Funzioni di I/O diretto su file: Accesso casuale

La funzione **fwrite** scrive su un file un numero specificato di oggetti di una certa ampiezza, prelevandoli da un vettore. Restituisce il numero di oggetti scritti.

```
int fwrite (void *buf, int size, int count, FILE *fp);
```

- o **\*buf** è il puntatore alla regione di memoria sorgente;
- o **size** è la lunghezza del singolo dato;
- o **count** è il numero dei dati;
- o **\*fp** è il file su cui scrivere

## Funzioni di posizionamento su file

La funzione **fseek** determina una posizione su un file. Le successive istruzioni di I/O sul file operano a partire da questa posizione. Si specifica uno spiazzamento **s** e un'origine **o**: la posizione è a **s** byte a partire da **o**. La funzione restituisce un valore nullo in caso di errore.

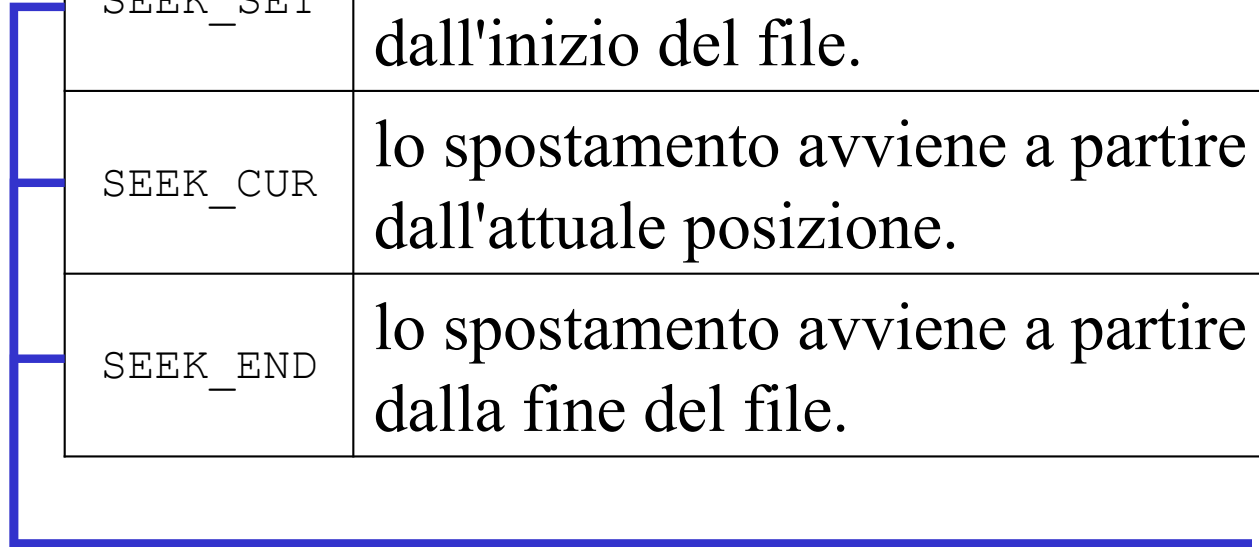
```
int fseek (FILE *fp, long s, int o);
```

La funzione **ftell** restituisce la posizione corrente su un file (0 se all'inizio), oppure, in caso di errore, un numero negativo.

```
long ftell (FILE *fp);
```

## MODALITA' OPERATIVE DI `fseek()`

Costante	Significato
<code>SEEK_SET</code>	lo spostamento avviene a partire dall'inizio del file.
<code>SEEK_CUR</code>	lo spostamento avviene a partire dall'attuale posizione.
<code>SEEK_END</code>	lo spostamento avviene a partire dalla fine del file.



```
int fseek (FILE *fp, long s, int o);
```

La **fseek()** restituisce 0 se l'operazione riesce;  
in caso di errore è restituito un valore diverso da 0.

## Funzioni di posizionamento su file

La funzione **fsetpos** posiziona il file nella posizione memorizzata in una variabile di memoria. Restituisce un valore nullo in caso di errore.

```
int fsetpos(FILE *fp, fpos_t *current pos)
```

La funzione **fgetpos** memorizza in una variabile di memoria la posizione corrente su un file. Restituisce un valore nullo in caso di errore.

```
int fgetpos(FILE *fp, fpos_t *current pos)
```

**Dato un file di testo mesi.txt, si supponga che sia costituito da righe ciascuna contenente una stringa (nome del mese) ed un intero (numero di giorni). Si stampino a video i nomi dei mesi che hanno 31 giorni.**

**mesi.txt**

Gennaio 31

Marzo 31

Febbraio 28

Dicembre 31

Novembre 30

**mesi.txt**

Gennaio 31  
Marzo 31  
Febbraio 28  
Dicembre 31  
Novembre 30



<stringa> <int>  
<stringa> <int>  
<stringa> <int>  
<stringa> <int>  
...



*Accesso sequenziale*

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

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

```
main() {
```

```
    char nome[20];
```

```
    int giorni;
```

```
    FILE* f;
```

```
    f=fopen("mesi.txt", "r");
```

```
    if (f==NULL) {
```

```
        printf("Il file non esiste!"); exit(1); }
```

```
    while (fscanf(f, "%s %d\n", nome, &giorni) != EOF)
```

```
        if (giorni == 31)
```

```
            printf ("%s\n", nome);
```

```
            fclose(f);
```

```
    }
```



**Dato un file binario mesi.dat, si supponga che contenga strutture così configurate: una stringa (nome del mese) ed un intero (numero di giorni); si memorizzi il contenuto del file in un vettore di strutture e si stampino a video i nomi dei mesi che hanno 31 giorni.**

**mesi.dat**

Gennaio 31

Marzo 31

Febbraio 28

Dicembre 31

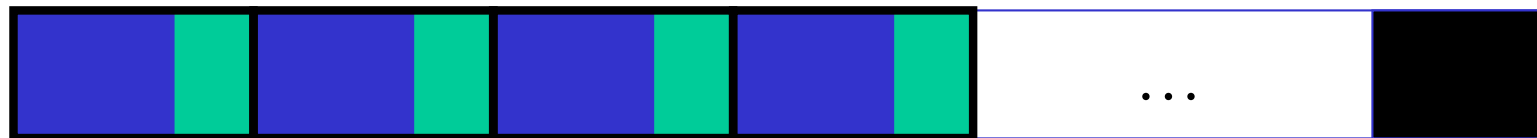
Novembre 30

**mesi.dat**

Gennaio 31  
Marzo 31  
Febbraio 28  
Dicembre 31  
Novembre 30



<stringa>	<int>
<stringa>	<int>
<stringa>	<int>
<stringa>	<int>
...	



*Accesso casuale*

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

typedef struct {int giorni; char nome [20];} mese;

main() {
    int i;
    mese v[12]; /* al massimo saranno 12*/
    FILE* f=fopen("mesi.dat","rb");

    if (f==NULL) {
        printf("Il file non esiste!"); exit(1); }
    while (fread(&v[i],sizeof(mese),1,f) >0) {
        if (v[i].giorni == 31) printf("%s\n",v[i].nome);
        i++;
    }
    fclose(f);
}
```

**Dato un file di testo alunni.txt, si supponga che sia costituito da righe ciascuna contenente una stringa (matricola dell'alunno) ed un intero (numero esami superati). Si dia da input una matricola: settare il numero di esami dell'alunn con quella matricola incrementandolo di uno.**

**alunni.txt**



```
MRTD 8
YTRG 7
HHYU 6
JJII 10
```

YTRG  
→

**alunni.txt**



```
MRTD 8
YTRG 8
HHYU 6
JJII 10
```

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

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

```
#include <string.h>
```

```
main() {
```

```
    char matr[20];
```

```
    char matr2[20];
```

```
    int esami; int test = 0;
```

```
    long pos;
```

```
    scanf("%s\n", matr);
```

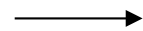
```
    FILE* f;
```

```
    f=fopen("alunni.txt", "r+");
```

```
    if (f==NULL) { printf("Il file non esiste!"); exit(1); }
```

matr = YTRG

**alunni.txt**



MRTD 8

YTRG **7**

HHYU 6

JJII 10

```
while ((!test)&&(!feof)) {  
    pos = ftell(f);  
    fscanf(f,"%s %d\n", matr2, &esami);  
    if (strcmp(matr,matr2)) {  
        fseek(f,pos,SEEK_SET); esami++;  
        fprintf(f,"%s %d\n", matr2, esami); test=1; }  
    }  
    fclose(f);  
}
```

## alunni.txt

pos



MRTD 8

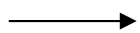
YTRG 7

HHYU 6

JJII 10

matr = YTRG

pos



MRTD 8

YTRG 7

HHYU 6

JJII 10

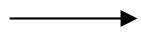
matr = YTRG

matr2 = MRTD



## alunni.txt

pos



MRTD 8

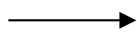
YTRG 7

HHYU 6

JJII 10

matr = YTRG

pos



MRTD 8

YTRG 7

HHYU 6

JJII 10

matr = YTRG

matr2 = YTRG

## alunni.txt

pos



MRTD 8

YTRG 7

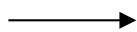
HHYU 6

JJII 10

matr = YTRG

matr2 = YTRG

pos



MRTD 8

YTRG 8

HHYU 6

JJII 10