

# Il linguaggio C

## I file

### mylib.c

```
#include "mylib.h"

int leggi(int a[], int dim) {
    int i=0;
    while( scanf("%d", a+i) ) {
        i++;
        if( i>=dim ) break;
    }
    return i;
}

void scambia(int *a, int *b) {
    int t;
    t = *a; *a = *b; *b = t;
}
```

### mylib.h

```
#include <stdio.h>
#define N 25

int leggi(int*, int);
void scambia(int*, int*);
void ordina(int*, int);
void stampa(int*, int);
```

### contribuenti.txt

```
Paolucci, Enrico, 24000, 23.0
Pasquini, Laura, 21000, 21.0
Pierantoni, Marco, 37000, 25.0
Poli, Valerio, 15000, 19.0
Ramponi, Stefano, 800, 0.0
Regoli, Fabio, 1000, 0.0
Ricci, Adriana, 56000, 39.0
Ricciardi, Daniele, 22000, 23.0
```

### fornitori.dat

```
Fontana      Maria Le#Qdd3 0 a4
Pasquini     Laura !8e*e 1 af
Martini      Enrico ,37% 1 bf
Poli         Valerio *FC9s 0 b0
Lombardo     Esteban 88(Dd 1 13
Cervellini   Chiara 0,9s& 1 3r
Cellai       Alessio &&8s0 0 c4
```

### mylib.obj

```
dd327h23d3832D323974DN29
D$(*w$w(M(8W$FDF(FK$98W
$Fk$W$
$Fjw948798$w4f4wf8$Wffw
$8*84W$WQ#)$ q3qqw8 c(#
$$Q#((8$#Q$C$
$)CC84397jdsadj#Q#(8j##
Q8jq3od(Q*4k!` `978ED-
A{]Apdc(*08 4$%&\DCA
\CAS08
```

### ese1.exe

```
*84W$WQ#)$ q3qqw8 (#$ $
Q#((8$ #Q$C$ $)CC 84397j
dsadj#Qdd327h23d3832D32
3974DN29D$(*w$w(M(8W
$FDF(FK$98W$Fk$W$
$#(8j##Q8jq3od(Q*4k!
` `978EDA{]Apdc(Fjw948798
$w4f4wf8$Wffw$8
```

# I File

Il file è l'unità logica di memorizzazione dei dati su memoria di massa. Consente una *memorizzazione persistente* dei dati, *non limitata* dalle dimensioni della memoria centrale.

Ogni programma vede il file come *una sequenza di componenti (record logici)*, terminata da una "marca" di fine file (End Of File, EOF)

I file sono gestiti dal Sistema Operativo. L'accesso ai file da parte di programmi C può avvenire grazie a funzioni standard definite nella libreria di I/O (`<stdio.h>`) che tengono conto delle funzionalità del S.O ospite.

In C i file vengono distinti in due categorie:

- *file di testo*, trattati come sequenze di caratteri. organizzati in linee (ciascuna terminata da '`\n`')
- *file binari*, visti come sequenze di bit

# File di testo

Sono file di caratteri, organizzati in linee.

Ogni linea è terminata da una marca di fine linea (*newline*, carattere ' \n ' ).

→ Il *record logico* può essere il singolo carattere, la parola, oppure la linea.

```
Egregio Sig. Rossi,
con la presente
le rendo noto di aver provveduto
...
```

# Esempi di file di testo

- File sorgenti di programmi C
- File che contengono dei dati in formato “leggibile”

```
#include <stdio.h>
#define N 25

int leggi(int*, int);
void scambia(int*, int*);
void ordina(int*, int);
void stampa(int*, int);
```

## contribuenti.txt

```
Paolucci, Enrico, 24000, 23.0
Pasquini, Laura, 21000, 21.0
Pierantoni, Marco, 37000, 25.0
Poli, Valerio, 15000, 19.0
Ramponi, Stefano, 800, 0.0
Regoli, Fabio, 1000, 0.0
Ricci, Adriana, 56000, 39.0
Ricciardi, Daniele, 22000, 23.0
```

*record logico = linea*

*record logico = parola*

*record logico = carattere*

# Gestione di file in C

I file hanno una struttura *sequenziale*

I record logici sono organizzati in una sequenza; per accedere ad un particolare record logico, è necessario “scorrere” tutti quelli che lo precedono:



Per accedere ad un file da un programma C, è necessario predisporre una variabile che lo rappresenti (*puntatore a file*)

**Puntatore a file:**

è una variabile che viene utilizzata per riferirsi a un file nelle operazioni di accesso (lettura e scrittura). Implicitamente essa indica:

- il *file*
- l'*elemento corrente* all'interno della sequenza

Ad esempio:

```
FILE *fp;
```

→ il tipo **FILE** è un tipo non primitivo dichiarato nel file `stdio.h`.

# Gestione di file in C

## Apertura di un file:

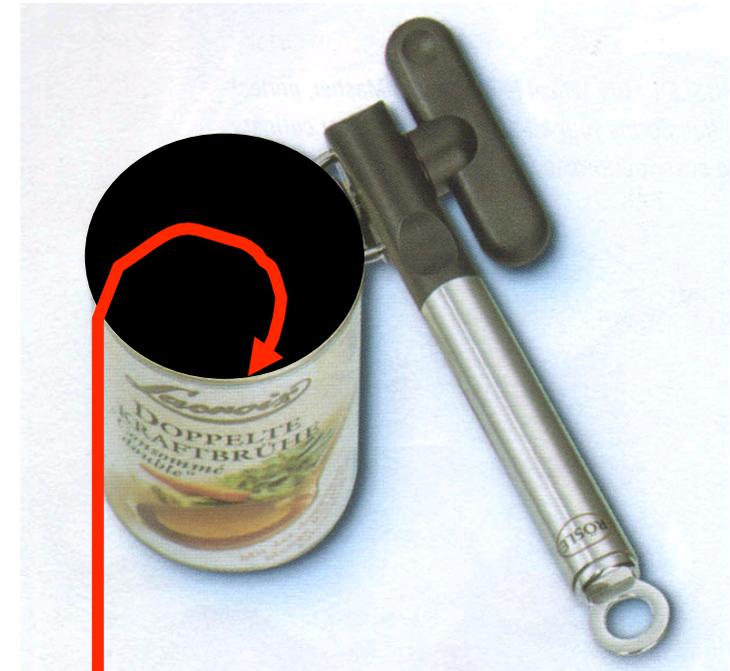
Prima di accedere ad un file è necessario *aprirlo*: l'operazione di apertura compie le azioni preliminari necessarie affinché si possa accedere al file possa ("in lettura" o "in scrittura"). L'operazione di apertura inizializza il puntatore al file.

## Accesso ad un file:

Una volta aperto il file, è possibile leggere/scrivere il file, tramite il puntatore a file.

## Chiusura di un file:

Alla fine di una sessione di accesso (lettura o scrittura) ad un file è necessario chiudere il file per memorizzare permanentemente il suo contenuto in memoria di massa



FILE \*fp

# Apertura di un File

```
FILE *fopen( char *name, char *mode );
```

dove:

- **name** è un vettore di caratteri che rappresenta il nome (assoluto o relativo) del file nel file system;
- **mode** esprime la modalità di accesso scelta.
  - "r", in lettura (read)
  - "w", in scrittura (write)
  - "a", scrittura, aggiunta in fondo (append)
  - "b", a fianco ad una delle precedenti, indica che il file è binario (se non specificato, il file è di testo)

Se eseguita con successo, l'operazione di apertura ritorna come risultato un *puntatore* al file aperto (**FILE \***):

Se l'apertura fallisce, `fopen` restituisce il valore **NULL**.

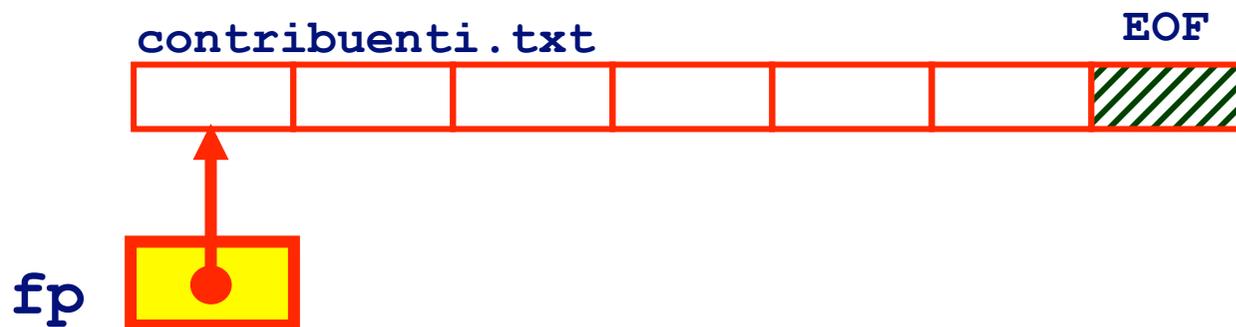
Tipici motivi di fallimento:

- Il file non esiste
- Il file viene aperto in una modalità incompatibile con le sue proprietà (ad esempio: apro in scrittura un file a sola lettura, *read only*), etc.

# Apertura in lettura ("r")

```
FILE *fp;  
fp = fopen("contribuenti.txt", "r");
```

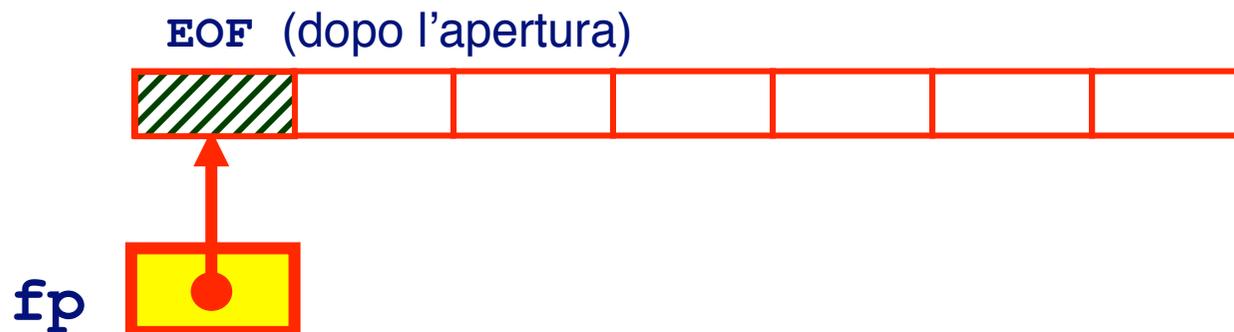
Se il file non è vuoto:



# Apertura in scrittura ("w")

```
FILE *fp;  
fp = fopen("contribuenti.txt", "w");
```

Anche se il file non è vuoto:

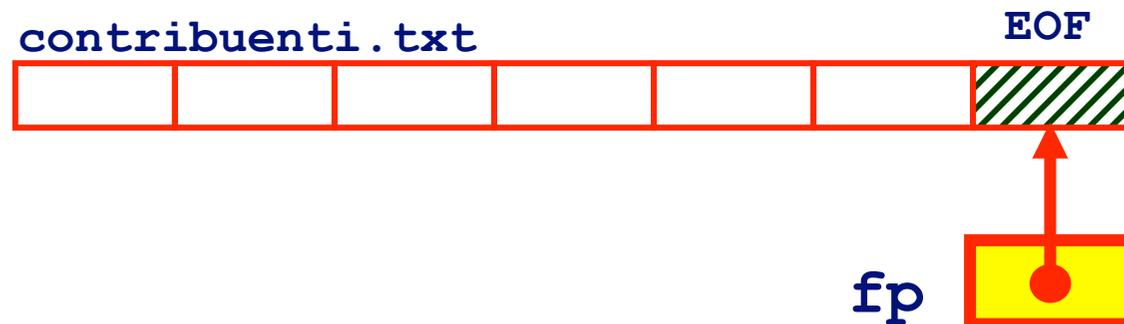


Se il file esisteva già, il suo contenuto viene perso: si scriveranno i nuovi record logici, sopra i pre-esistenti, a partire dal primo.

# Apertura in *append* ("a")

```
FILE *fp;
```

```
fp = fopen("contribuenti.txt", "a");
```



Il puntatore al file si posiziona sull'elemento successivo all'ultimo elemento significativo del file. Se il file esisteva già, il suo contenuto non viene perso.

# Esempio

```
FILE *fp;  
fp=fopen("c:\temp\contribuenti.txt", "r");  
<uso del file>
```

**fp** rappresenta, dall'apertura in poi, il *riferimento* da utilizzare nelle operazioni di accesso al file (lettura) **contribuenti.txt** posto nel direttorio **c:\temp\**.

**fp** individua due cose:

- il file stesso
- l'elemento corrente all'interno del file

# Chiusura di un File

Al termine di una sessione di accesso al file, esso deve essere chiuso.

L'operazione di chiusura si realizza con la funzione

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

dove `fp` rappresenta il puntatore al file da chiudere.

`fclose` ritorna come risultato un intero:

- se l'operazione di chiusura è eseguita *correttamente* restituisce il valore 0
- se la chiusura *non è andata a buon fine*, ritorna la costante `EOF`.

# Esempio

```
#include <stdio.h>
main()
{
    FILE *fp;
    fp = fopen("prova.doc", "w")
    <scrittura di prova.doc>
    fclose(fp);
    return 0;
}
```

# File standard di I/O

Esistono tre file testo che sono **aperti automaticamente** all'inizio di ogni esecuzione:

- *stdin*, standard input (tastiera), aperto in lettura
  - *stdout*, standard output (video), aperto in scrittura
  - *stderr*, standard error (video), aperto in scrittura
- *stdin*, *stdout*, *stderr* sono variabili di tipo *puntatore a file*
- esse sono automaticamente (ed implicitamente) definite (non vanno (ri-)definite).

# se fine file

- Durante la fase di accesso ad un file è possibile verificare la presenza della marca di fine file con la funzione di libreria:

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

- **feof(fp)** controlla se è stata raggiunta la fine del file **fp** nella operazione di lettura o scrittura precedente:
  - Restituisce il valore 0 se non è stata raggiunta la fine del file,
  - altrimenti restituisce un valore diverso da zero.

# Letture e scrittura di file

Una volta aperto un file, su di esso si può accedere in lettura e/o scrittura, compatibilmente con quanto specificato in fase di apertura.

Per file di testo sono disponibili funzioni di:

- Lettura/scrittura **con formato**
- Lettura/scrittura **di caratteri**
- Lettura/scrittura **di stringhe di caratteri**

Per file binari si utilizzano funzioni di:

- Lettura/scrittura **di blocchi**: fread, fwrite

# Accesso a file di testo: Lettura con formato

Esistono funzioni simili a `scanf` e `printf`, ma con un parametro aggiuntivo rappresentante il puntatore al file di testo sul quale si vuole leggere o scrivere

## Lettura con formato:

Si usa la funzione `fscanf`:

```
int fscanf (FILE *fp, str-ctrl, ind-el);
```

dove:

- `fp` è il puntatore al file
- `str-ctrl` indica il formato dei dati da leggere
- `ind-el` è la lista degli indirizzi delle variabili a cui assegnare i valori letti.

# Esempio

```
main() {  
    FILE *fp;  
    int A; char B; float C;  
    fp = fopen( "dati.txt", "r" );  
    fscanf( fp, "%d%c%f", &A, &B, &C );  
    ...  
    fclose( fp );  
}
```

come per la scanf

**fscanf** restituisce il numero di elementi letti, oppure un valore negativo in caso di errore.

# Scrittura con formato

Si usa la funzione `fprintf`:

```
int fprintf ( FILE *fp, str-ctrl, elem );
```

dove:

- `fp` è il puntatore al file
- `str-ctrl` indica il formato dei dati da scrivere
- `elem` è la lista dei valori (espressioni) da scrivere.

Restituisce il numero di elementi scritti, oppure un valore negativo in caso di errore.

# Esempio

```
FILE *fp;  
float C=0.27;  
fp=fopen( "risultati.txt", "w" );  
fprintf( fp, "Risultato: %f", c*3.14 );  
...  
fclose( fp );
```

# Esempio

Visualizzazione del contenuto di un file di testo:

```
#include <stdio.h>
main() {
    char c;
    FILE *fp;

    fp=fopen( "testo.txt", "r" );
    while ( fscanf( fp, "%c", &c )>0 )
        printf( "%c", c );
    fclose( fp );
}
```

# Lettura/scrittura di stringhe

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

Trasferisce nella stringa **s** i caratteri letti dal file puntato da **fp**, fino a quando ha letto **n-1** caratteri, oppure ha incontrato un newline, oppure la fine del file.

La **fgets** mantiene il newline nella stringa **s**. Restituisce:

- la stringa letta in caso di corretta terminazione;
- **NULL** in caso di errore o fine del file.

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

Trasferisce la stringa **s** (terminata da '**\0**') nel file puntato da **fp**. Non copia il carattere terminatore '**\0**' né aggiunge un newline finale.

Restituisce:

- l'ultimo carattere scritto in caso di terminazione corretta;
- EOF altrimenti.

# Accesso a file binari

- Si può leggere o scrivere da un file binario un intero blocco di dati (binari).
- Un file binario memorizza dati *di qualunque tipo*, in particolare dati che non sono necessariamente caratteri (interi, reali, vettori o strutture).
- Per la lettura/scrittura a blocchi è necessario che il file sia stato aperto in modo *binario* (modo "b").

# Lettura/scrittura di blocchi

## Lettura

```
int fread(void *vet, int size, int n, FILE *fp);
```

Legge (al più)  $n$  oggetti di dimensione  $size$  dal file puntato da  $fp$ , e li colloca nel vettore  $vet$ . Restituisce un intero che rappresenta il numero di oggetti effettivamente letti.

## Scrittura

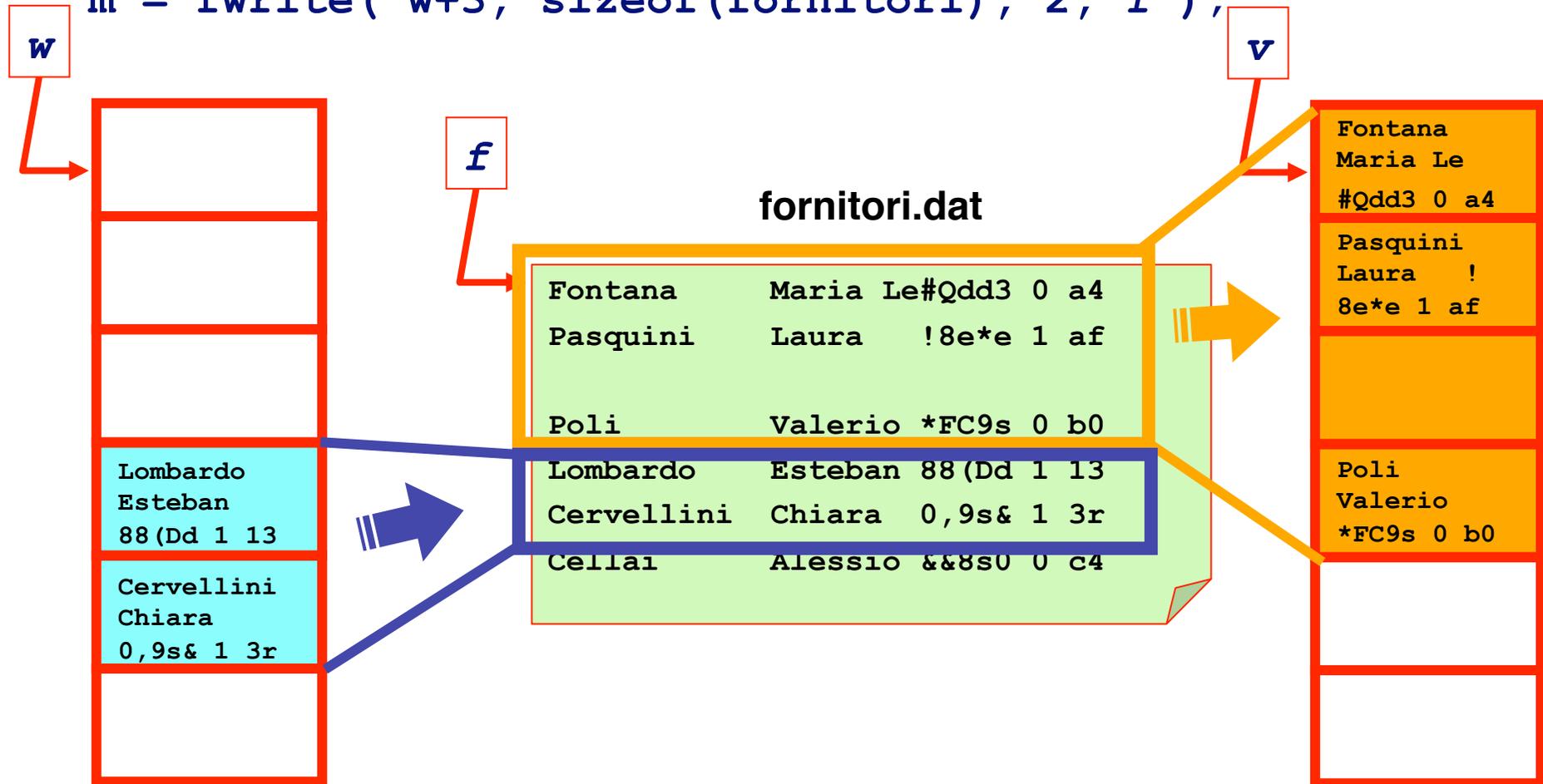
```
int fwrite(void *vet, int size, int n, FILE *fp);
```

Copia dal vettore  $vet$   $n$  oggetti di dimensione  $size$  e li scrive sul file puntato da  $fp$ . Restituisce un intero che rappresenta il numero di oggetti effettivamente scritti (inferiore ad  $n$  solo in caso di errore, o fine del file).

# Letture/scrittura di blocchi

```
n = fread( v, sizeof(fornitori), 4, f );
```

```
m = fwrite( w+3, sizeof(fornitori), 2, f );
```



# Esempio: scrittura di una sequenza di record (dati da input) in un file binario

```
#include <stdio.h>
typedef struct{
    char nome[20], cognome[20];
    int reddito; }persona;
main()
{ FILE *fp;
  persona p;
  int fine=0;
  fp=fopen("archivio.dat","wb");
  do {
    printf("Dati persona?");
    scanf("%s%s%d",&p.nome,&p.cognome,&p.reddito);
    fwrite(&p,sizeof(persona),1,fp);
    printf("Fine (si=1,no=0)?");
    scanf("%d",&fine);
  } while(!fine);
  fclose(fp);
}
```

# Esempio: lettura e stampa del contenuto di un file binario

```
#include<stdio.h>
typedef struct{
    char nome[20], cognome[20];
    int reddito; }persona;

main()
{ FILE *fp;
  persona p;
  fp=fopen("archivio.dat","rb");
  while( fread(&p, sizeof(persona),1, fp)>0 )
    printf("%s%s%d", p.nome, p.cognome,
                                                    p.reddito) ;
  fclose(fp) ;
}
```

# Esempio: dato il nome di un file, scrivere nel file un vettore di interi

```
#include <stdio.h>
main()
{FILE *file;
char nome[30];
int i,n,tab[12]={3, 6, -12, 5, -76, 3,
                32, 12, 65, 1, 0, -9};

gets(nome);
if ( (file=fopen(nome,"wb"))==NULL ) {
    printf("Impossibile aprire file d'uscita\n");
    return 1;
}
fwrite(tab, sizeof( int ), 12, file );
fclose(file);
exit(0);
}
```

# Esempio: copiare il contenuto di un file binario di interi ( $\leq 40$ ) in un vettore

```
#include <stdio.h>
#define MAX 40
main()
{FILE *file;
 char nome[30];
 int i, n, tab[MAX];
 gets(nome);
 if ( (file=fopen(nome, "rb"))==NULL ) {
   printf("Impossibile aprire file d'ingresso\n");
   return 1;
 }
 n=fread(tab, sizeof(int), MAX, file);
 fclose(file);
 for(i=0;i<n;i++)
   printf("%d%c", tab[i], (i==n-1) ? '\n' : '\t');
 return 0;
}
```

# Esempio: scrivere il contenuto di un vettore di record in un file binario

```
#include <stdio.h>
#define DIM 5

int crea_vettore(Persona V[], int dim);

typedef struct {
    char nome[15], cognome[15], via[10];
    int eta; } Persona;
Persona P[DIM];

int crea_vettore(Persona P[], int dim) {
    int i=0, char s[80];
    while (!feof(stdin) && i<dim) {
        scanf("%s%s%s%d", P[i].nome, P[i].cognome,
            P[i].via, &(P[i].eta));
        i++; }
    return i;
}
```

# Esempio: scrivere il contenuto di un vettore di record in un file binario

```
main() {
    int n;
    FILE *file;
    char nome[30];

    gets(nome);
    n=crea_vettore(P,DIM);
    if ( (file=fopen(nome, "wb"))==NULL ) {
        printf("Impossibile aprire file\n");
        return 1;
    }
    fwrite(P,sizeof(Persona),n,file);
    fclose(file);
}
```

# File ad accesso diretto

Il C consente di gestire i file non solo come sequenziali, ma anche come file ad accesso diretto.

## Posizionamento in un file:

La funzione *fseek* della Standard Library consente il posizionamento del puntatore al file su un qualunque byte.

```
int fseek (FILE *f, long offset, int origin)
```

si sposta di *offset* byte a partire dalla posizione *origin* (i valori significativi di *origin* sono: 0, 1 o 2).

Restituisce:

- 0 se ha spostato la posizione sul file
- un valore diverso da 0, altrimenti.

Origine dello spostamento:

<b>SEEK_SET</b>	0	inizio file
<b>SEEK_CUR</b>	1	posizione attuale nel file
<b>SEEK_END</b>	2	fine file

# Posizionamento all'inizio di un file

Per posizionarsi all'inizio di un file già aperto è possibile utilizzare anche la funzione `rewind`:

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

```
file=fopen(argv[1], "r");  
<lettura di vari record logici>;  
rewind(file);
```

equivale a:

```
fseek (f, 0, SEEK_SET);
```

# Posizione corrente nel file

La funzione `ftell` restituisce la posizione del byte sul quale si è posizionati nel file al momento della chiamata della funzione:

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

`ftell` restituisce -1 in caso di errore.

Il valore restituito da `ftell` può essere utilizzato in una chiamata ad `fseek`.

# Esercizio

- Realizzare una funzione

```
long flength(FILE *)
```

che restituisca la lunghezza in byte di un dato file

- senza modificare la posizione del file pointer...

# Esercizio (per casa)

- Realizzare una funzione

```
int intersect_ord(FILE *, int*, int*)
```

che, dati due vettori ordinati di interi, stampi su un file di testo gli interi presenti in entrambi i vettori

- Realizzare una funzione

```
int intersect(FILE *, int*, int*)
```

che, dati due vettori (non necessariamente ordinati) di interi, stampi su un file binario gli interi presenti in entrambi i vettori