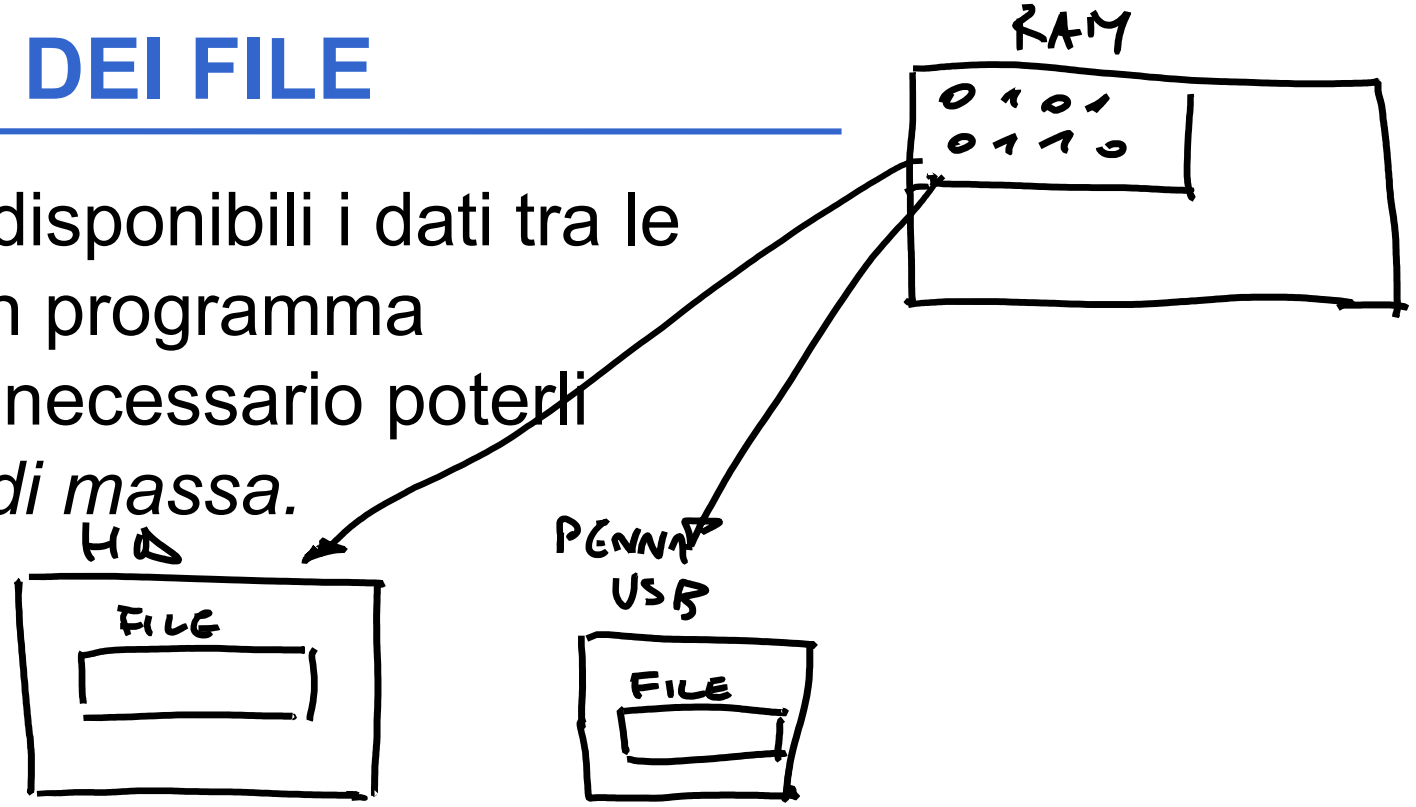
File
File di testo

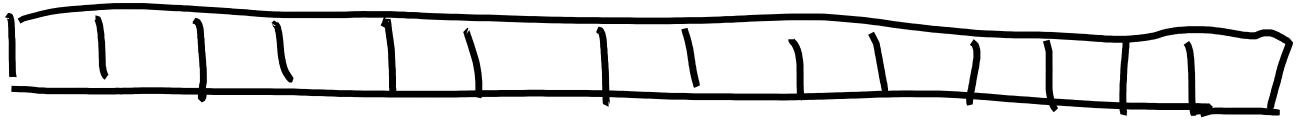
GESTIONE DEI FILE



- Per poter mantenere disponibili i dati tra le diverse esecuzioni di un programma (*persistenza* dei dati) è necessario poterli *archiviare su memoria di massa.*
  - dischi
  - nastri
  - cd
  - …
- I file possono essere manipolati (aperti, letti, scritti…) all'interno di programmi C

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*.

OPERARE SUI FILE

- A livello di sistema operativo un file è denotato univocamente dal suo ***nome assoluto***, che comprende il **percorso** e il **nome relativo**.
- In certi sistemi operativi il percorso può comprendere anche il ***nome dell'unità.***
  - *in DOS o Windows:*

UNITÀ → `C:\temp\prova1.c`

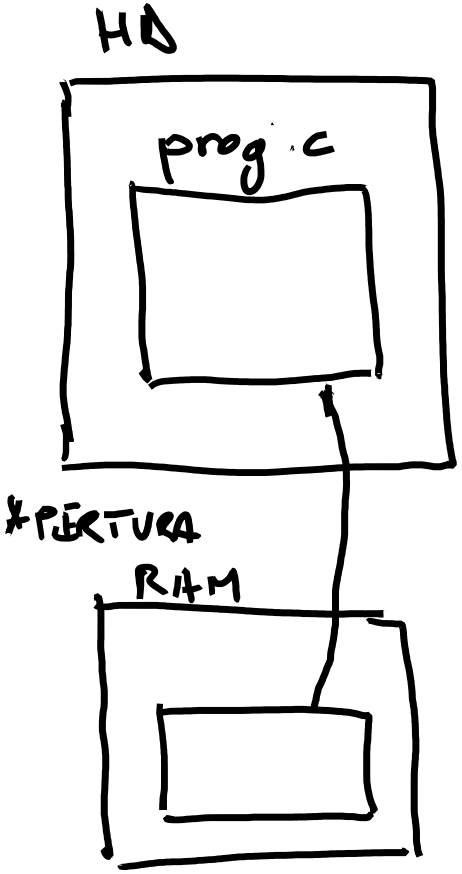
  - *in UNIX e Linux:*

`/usr/temp/prova1.c`

radice

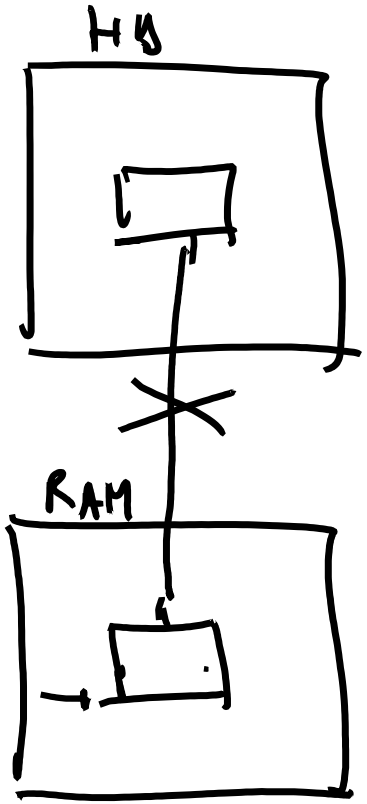
APERTURA DI FILE

- 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*
- Durante la fase di apertura si stabilisce anche la *modalità* di apertura del file
  - apertura in lettura
  - apertura in scrittura
  - ...



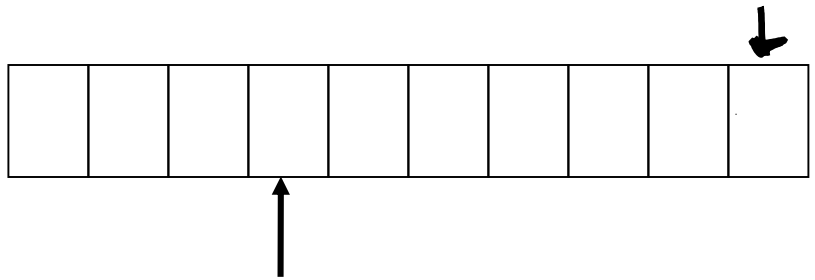
APERTURA E CHIUSURA DI FILE

- 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*.



ASTRAZIONE: testina di lettura/scrittura

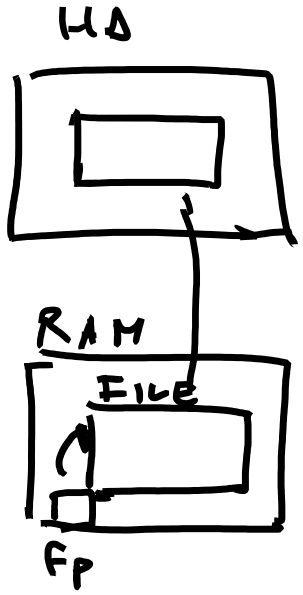
- Una testina di lettura/scrittura (concettuale) indica in ogni istante il record corrente:
  - inizialmente, la testina si trova per ipotesi sulla prima posizione
  - dopo ogni operazione di lettura / scrittura, essa si sposta sulla registrazione successiva.-> **accesso sequenziale al file**



- 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`**.



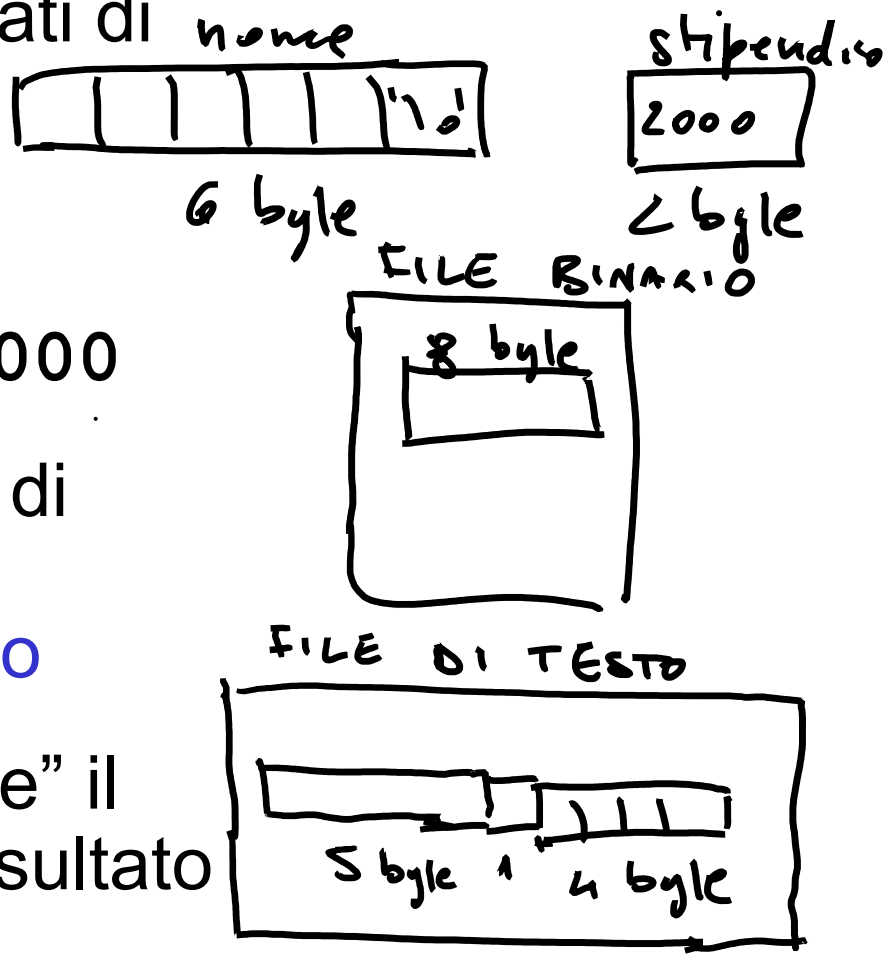

```
FILE *fp
// apertura
fp = ...
funzione (fp, ...)
chiudi(fp);
```

Come rappresentiamo i dati?

Ad esempio, vogliamo scrivere su un file i dati di una persona

```
char Nome[]="Luigi"

unsigned short int stipendio = 2000
```



- possiamo immaginare di ricopiare le celle di memoria che rappresentano le variabili direttamente sul file (6+2 byte): File Binario
- oppure possiamo immaginare di “stampare” il contenuto delle celle e scrivere sul file il risultato (6+4byte): File di Testo

Come rappresentiamo i dati?

Rappresentazione interna

- Più sintetica
- Non c'è bisogno di effettuare conversioni ad ogni lettura/scrittura
- Si può vedere il contenuto del file solo con un programma che conosce l'organizzazione dei dati

FORMATO

FILE BINARI

Rappresentazione esterna

- Meno sintetica
- Necessità di conversione ad ogni lettura/scrittura
- Si può verificare il contenuto del file con un semplice editor di testo

Ascii

FILE di TESTO

FILE IN C: APERTURA

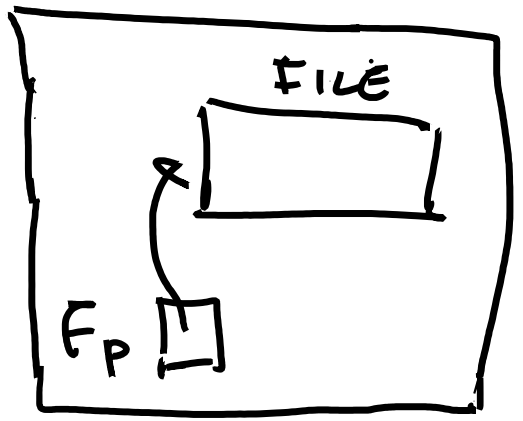
FILE * fp ;

fp = fopen("nome", ...);

Per aprire un file si usa la funzione:

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

Questa funzione apre il file di nome `fname` nel `modo` specificato, e restituisce un puntatore a `FILE` (che punta a una nuova struttura **`FILE`** appositamente creata).



- NB: il nome del file (in particolare il path) è indicato in maniera diversa nei diversi sistemi operativi (\ nei percorsi oppure /, presenza o assenza di unità, etc).
  In C per indicare il carattere `'\'` si usa la notazione `'\\'`

"testi\\nome.txt"

FILE IN C: APERTURA

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

Per aprire un file si usa la funzione:

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

`modo` specifica *come* aprire il file:

- `r` apertura in lettura (read). Se il file non esiste → fallimento.
- `w` apertura di un file vuoto in scrittura (write). Se il file esiste il suo contenuto viene cancellato.
- `a` apertura in aggiunta (append). Crea il file se non esiste.

- seguito opzionalmente da:
  - `t` apertura in modalità testo (default)
  - `b` apertura in modalità binaria
- ed eventualmente da
  - `+` apertura con possibilità di *lettura e scrittura.*

"r+b"

FILE IN C: APERTURA

Modi:

- `r+` apertura in lettura e scrittura. Se il file non esiste → fallimento.
- `w+` apertura un file vuoto in lettura e scrittura. Se il file esiste il suo contenuto viene distrutto.
- `a+` apertura in lettura e aggiunta. Se il file non esiste viene creato.

Nota: non si può passare da lettura a scrittura e viceversa se non si fa una operazione di `fflush`, `fseek` o `rewind` (V. prossimi lucidi)

FILE IN C: APERTURA

- Il *puntatore a* **`FILE`** restituito da **`fopen()`** si deve usare in tutte le successive operazioni sul file.
  - esso assume il valore **`NULL`** in caso l'apertura sia fallita
  - controllarlo *è il solo modo per sapere se il file è stato davvero aperto: non dimenticarlo!*
  - se non è stato aperto, il programma *non può proseguire* → **procedura `exit()`**
    (`#include <stdlib.h>`)

```
… FILE *fp;
  fp = fopen("esempio.txt","rt");
  if (fp==NULL)
  {   printf("file esempio.txt non trovato");
      exit(-1);
  }
```

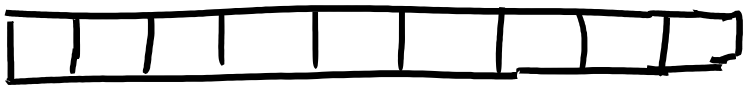
FILE IN C: CHIUSURA

Per chiudere un file si usa la funzione:

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

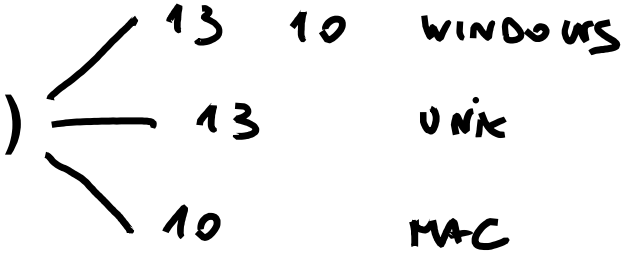
```
if ( fclose(fp) != 0)
    printf("errore\n");
```

- 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 DI TESTO

- Un **file di testo** è un file che contiene ***sequenze di caratteri***
- È un caso ***estremamente frequente***, con ***caratteristiche proprie:***
  - esiste un concetto di ***riga*** e di ***fine riga*** (`'\n'`)
  - certi caratteri sono ***stampabili a video*** (quelli di codice ASCII ≥ 32), altri no



FILE DI TESTO (segue)

Funzione da console	Funzione da file
`int getchar(void);`	`int fgetc(FILE* f);`
`int putchar(int c);`	`int fputc(int c, FILE* f);`
`char* gets(char* s);`	`char* fgets(char* s, int n, FILE* f);`
`int puts(char* s);`	`int fputs(char* s, FILE* f);`
`int printf( ... );`	`int fprintf(FILE* f, ... );`
`int scanf( ... );`	`int fscanf(FILE* f, ... );`

- tutte le funzioni da file acquistano una "**f**" davanti nel nome (qualcuna però cambia leggermente nome)
- tutte le funzioni da file hanno un *parametro in più*, che è appunto il puntatore al **FILE** aperto
- per riferimento: da terminale, **man 3 <nome funzione>**

```
FILE *fp = fopen(..., "w");
fprintf(fp, "ferrara");
int a;
fscanf(fp, "%d", &a);
if (fscanf(fp, "%d %c", ...) == 2)
```

ESERCIZIO

- Si scriva su un file di testo di nome **`prova.txt`** quello che l'utente inserisce da tastiera parola per parola, finché non inserisce la parola **`"FINE"`**.

INPUT
ferrara ↵
bologna ↵
FINE ↵

prova.txt
ferrara bologna

printf ("ferrara");
≡
fprintf(stdout, "ferrara");

FILE DI TESTO E CONSOLE

- In realtà, anche per leggere da tastiera e scrivere su video, il C usa le procedure per i file.
- Ci sono 3 file, detti canali di I/O standard, che sono già aperti:
  - **stdin** è un file di testo aperto in lettura, di norma agganciato alla tastiera
  - **stdout** è un file di testo aperto in scrittura, di norma agganciato al video
  - **stderr** è un altro file di testo aperto in scrittura, di norma agganciato al video
- Le funzioni di I/O disponibili per i file di testo sono una generalizzazione di quelle già note per i canali di I/O standard.

scanf(...);
≡
fscanf(stdin, ...);

FUNZIONE feof()

END OF 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 precedente.
- Restituisce il valore
  - 0 (falso logico) se non è stata raggiunta la fine del file,
  - un valore diverso da zero (vero logico), se è stata raggiunta la fine del file

```
while (feof (fp)==0)
{

}
```

ESEMPIO

- Si mostri a video il contenuto di un file di testo il cui nome viene inserito da tastiera

INPUT
infinito.txt

```
Inserisci il nome di un file: infinito.txt

Sempre caro mi fu quest'ermo colle,
E questa siepe, che da tanta parte
De l'ultimo orizzonte il guardo esclude.
Ma sedendo e mirando, interminati
Spazi di la` da quella, e sovrumani
Silenzi, e profondissima quiete
Io nel pensier mi fingo, ove per poco
Il cor non si spaura. E come il vento
Odo stormir tra queste piante, io quello
Infinito silenzio a questa voce
Vo comparando: e mi sovvien l'eterno,
E le morte stagioni, e la presente
E viva, e 'l suon di lei. Così tra questa
Immensita` s'annega il pensier mio:
E 'l naufragar m'è dolce in questo mare.
```

OUTPUT

Luigi via_machiavelli_30 _555...
mahca

ESERCIZIO

Un file di testo **`rubrica.txt`** contiene una rubrica del telefono, in cui per ogni persona sono memorizzati di seguito

- nome (stringa di max 20 caratteri senza spazi, incluso terminatore)
- indirizzo (stringa di max 30 caratteri senza spazi, incluso '\0')
- numero (stringa di max 15 caratteri incluso '\0')

Si scriva un programma C che legge da tastiera un nome, cerca la persona corrispondente nel file **`rubrica.txt`** e visualizza sullo schermo i dati della persona (se trovata)

INPUT

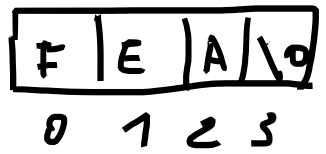
luigi

OUTPUT

luigi via_mach... 555...

fscanf(.., "%s", ..)

```
char prov[4];
fgets(prov, 4, fp);
```

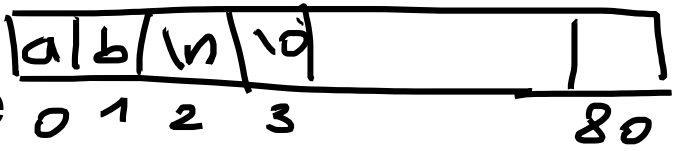


LETTURA DI STRINGHE

char s[81] FILE *fp

fgets(s, 81, fp);

```
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.
- Il carattere newline, se letto, e' mantenuto nella stringa s.
- Restituisce la stringa letta in caso di corretta terminazione; NULL in caso di errore o fine del file.

INPUT

a b ↵ ...

a b ␣ c d ↵ ...

```
char prov[3];
fscanf(fp, "%s", prov);
```

FE

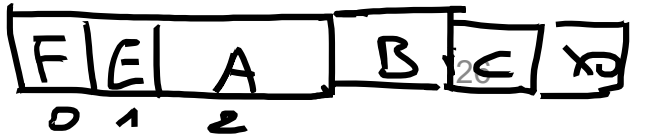
INPUT

FE ↵

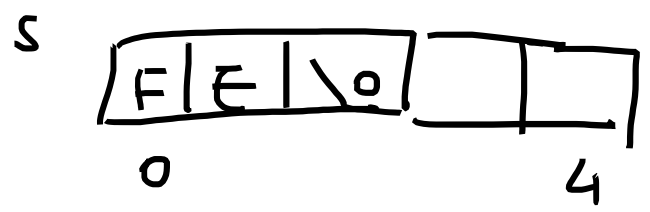
FE ABC ↵

prov

PROBLEMA



SCRITTURA DI STRINGHE




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

fputs(s, fp);

fp

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

- Restituisce un numero non negativo in caso di terminazione corretta; EOF altrimenti.

ESEMPIO COMPLETO FILE TESTO

È dato un file di testo **`people.txt`** le cui righe rappresentano *ciascuna i dati di una persona*, secondo il seguente formato:

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

Si vuole scrivere un programma che

- legga riga per riga i dati dal file
- e ponga i dati in un array di *persone*

VARIANTE

E se usassimo *un singolo carattere* per rappresentare il sesso?

Non più:

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

Ma:

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

VARIANTE

Cosa cambierebbe?

- `fscanf` elimina *automaticamente* gli spazi prima di leggere una stringa o un numero (intero o reale)... ***ma non prima di leggere un singolo carattere***, perché se lo facesse non riuscirebbe a leggere il carattere spazio.
- **Ma noi non sappiamo quanti spazi ci sono fra nome e sesso!** La specifica non lo dice!
- Quindi, *non possiamo sapere a priori dov'è il carattere che ci interessa!*

VARIANTE

Due possibilità:

- **scelta "furba"**: ***introdurre comunque una stringa di due caratteri*** e usarla per far leggere il carattere relativo al sesso a **`fscanf`**
  Poi, copiare il primo carattere al suo posto.
- **scelta "fai da te":** costruirsi un ciclo che *salti tutti gli spazi* fino al primo carattere non-spazio, poi recuperare quest'ultimo
  → non consente più di usare una singola **`fscanf`** per gestire tutta la fase di lettura.

VARIANTE - VERSIONE "FURBA"

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

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

main() {
 persona v[DIM]; int k=0; FILE* f; char s[2];
 if ((f=fopen("people.txt", "r"))==NULL) {
  perror("Il file non esiste!"); exit(1); }
 while(fscanf(f,"%s%s%s%d",  v[k].cognome,
       v[k].nome, s, &v[k].anno ) == 4){
     v[k].sesso = s[0]; k++; }
}
```

Un singolo carattere

Stringa ausiliaria

Copiatura carattere

VARIANTE - VERSIONE "FAI DA TE"

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

main() {
 persona v[DIM]; int k=0; FILE* f; char ch;
 if ((f=fopen("people.txt", "r"))==NULL) {
  perror("Il file non esiste!"); exit(1); }
 while(fscanf(f,"%s%s",   v[k].cognome,
              v[k].nome) == 2){
  while((ch=fgetc(f))==' ');
     v[k].sesso = ch;
  fscanf(f,"%d",&v[k].anno);  k...
}
```

Un singolo carattere

Carattere ausiliario

Legge solo cognome e nome

Salta spazi

Copia il carattere

Legge l'anno

VARIANTE - VERSIONE "FAI DA TE"

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

main()
 persona
 if ((f
  perror(
 while(fscanf(f,
           v[k].nome) == 2){
  do fscanf(f,"%c", &ch); while (ch==' ');
     v[k].sesso = ch
  fscanf(f,"%
}
```

Alternativa: anziché **fgetc**, si può usare **fscanf** per leggere il singolo carattere
→ occorre un ciclo do/while (prima si legge, poi si verifica cosa si è letto)

Ricorda: il singolo carattere richiede l'estrazione esplicita dall'indirizzo

ESEMPIO FORMATO POSIZIONALE

È dato un file di testo **`elenco.txt`** le cui righe rappresentano *ciascuna i dati di una persona*, secondo il seguente formato:

- **cognome** (esattamente 10 caratteri)
- **nome** (esattamente 10 caratteri)
- **sesso** (esattamente un carattere)
- **anno di nascita**

I primi due possono contenere spazi al loro interno.

NB: non sono previsti spazi espliciti di separazione

ESEMPIO FORMATO POSIZIONALE

Cosa cambia rispetto a prima?

- sappiamo esattamente dove iniziano e dove finiscono i singoli campi
- non possiamo sfruttare gli spazi per separare cognome e nome

Un possibile file **`elenco.txt`**:

```
Rossi      Mario      M1947
Ferretti   Paola      F1982
De Paoli   Gian MarcoM1988
Bolognesi Anna Rita F1976
...
```

I vari campi possono essere "attaccati": tanto, sappiamo a priori dove inizia l'uno e finisce l'altro

ESEMPIO FORMATO POSIZIONALE

Come fare le letture?

- **non possiamo usare `fscanf(f,"%s",…)`**
  - **si fermerebbe al primo spazio**
  - **potrebbe leggere più caratteri del necessario (si pensi a `Gian MarcoM1988`)**
- **però possiamo usare `fscanf` in un'altra modalità, *specificando quanti caratteri leggere*.** Ad esempio, per leggerne dieci:

```
fscanf(f,"%10c",…)
```

Così legge esattamente 10 caratteri, spazi inclusi

ESEMPIO FORMATO POSIZIONALE

Come fare le letture?

- **<u>non</u> possiamo usare `fscanf(f,"%s",…)`**
  - **si fermerebbe al primo spazio**
  - **potrebbe leggere più caratteri del necessario (si pensi a `Gian MarcoM1988`)**
- **però possiamo usare `fscanf` nell'altra modalità, <u>*specificando quanti caratteri leggere*</u>.** Ad esempio, per leggerne dieci:

```
fscanf(f,"%10c",…)
```

Così legge <u>esattamente 10 caratteri</u>, <u>spazi inclusi</u>

ATTENZIONE: viene riempito un array di caratteri, <u>*senza inserire alcun terminatore*</u>. Occorre <u>aggiungerlo a parte</u>.

ESEMPIO 4 - PROGRAMMA COMPLETO

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

typedef struct {
 char cognome[11], nome[11], sesso; int anno;
} persona;

main() {
 persona v[DIM]; int k=0; FILE*
 if ((f=fopen("elenco.txt",  r"))==
  perror("Il file non esiste!");
 while(fscanf(f,"%10c%10c%c%d\n", v[k].cognome,
    v[k].nome, &v[k].sesso, &v[k].anno ) != EOF){
     v[k].cognome[10]=v[k].nome[10]='\0'; k++;}
}
```

Sappiamo esattamente la dimensione: 10 +1

Legge esattamente 10 caratteri (spazi inclusi)

Legge 1 carattere e un intero (ricordare &)

Ricordare il terminatore!