

## Files



1

## File

- **Settori**: organizzazione hw

- **Record**: insieme di campi; equivalente dei settori per il sw



2

## Tipi di file

- Il significato dei byte memorizzati all'interno dei file è noto, solitamente, solamente al programma che lo ha creato
- Si parla di **tipi di file**, che sono solitamente indicati dall'**estensione** (ultime lettere del nome del file)
  - File prodotto da Microsoft Powerpoint → . ppt
  - File prodotto da Microsoft Word → . doc
  - File prodotto da Openoffice.Org Writer → . od't
  - ...

3

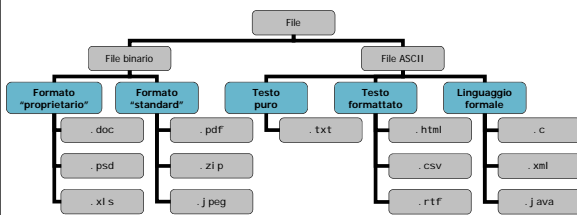
3

## Conseguenza

- Non è possibile lavorare con file gestiti da altri programmi, a meno di non conoscere il formato del file
- **Eccezione**: se il formato del file è pubblicamente documentato, allora sarà possibile leggerlo e scriverlo interpretando correttamente i byte
  - Esempio: file di testo (ASCII)
    - La codifica ASCII è utilizzata in molti campi: testi (.txt), programmi in C (.c), pagine HTML (.html), ...
  - Esempio: file Acrobat (.pdf)
    - Struttura molto complessa, ma documentata

4

## Vista d'insieme dei formati di file



5

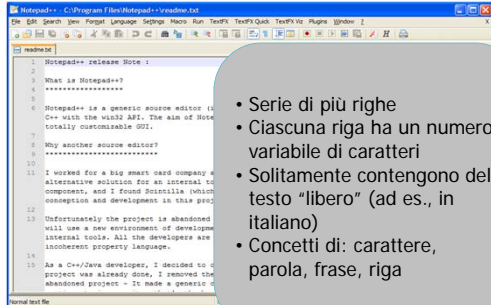
## File di "testo puro"

Normal text file

```
1 Notepad++ release Note :
2
3 What is Notepad++?
4 *****
5
6 Notepad++ is a generic source editor (it tries to be anyway) and Notepad replacement written in
7 C++ with the Visual API. The aim of Notepad++ is to offer a slim and efficient binary with a
8 totally customizable GUI.
9
10 Why another source editor?
11 *****
12
13 I worked for a big smart card company as engineer developer, and I took charge of looking for an
14 alternative solution for an internal tool coded in Java. The internal tool needed an edit
15 component, and I found Scintilla (which allows me to develop in C++) via Internet. I began my
16 conception and development in this project.
17
18 Unfortunately the project is abandoned after 3 months. The reason is political : our company
19 will use a new environment of development, as well a proprietary language, to re-develop all
20 internal tools. All the developers are forced to use this unstable and uncomfortable IDE and the
21 looser property language.
22
23 As a C++/Java developer, I decided to continue the project with my spare time. The prototype of
24 project was already done, I removed the components which depend on the specification of
25 abandoned project - It made a generic code editor. Then I made it available on Sourceforge and I
```

6

## File di "testo puro"

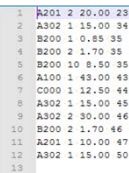


```
1 Notepad++ release Note :  
2  
3 What is Notepad++?  
4 =====  
5  
6 Notepad++ is a generic source editor (i.e.  
7 C++ with the Visual API). The aim of Notepad++  
8 is to provide a totally customizable GUI.  
9  
10 Why another source editor?  
11 =====  
12  
13 I worked for a big smart card company &  
14 Alternative Solutions for an internal tool  
15 components, and I found Scintilla (which  
16 conception and development in this project  
17  
18 Unfortunately the project is abandoned  
19 will use a new environment of development  
20 internal tools. All the developers are  
21 inconsistent property language.  
22  
23 As a C++/Java developer, I decided to  
24 project was already done, I removed the  
25 abandoned project - It made a generic C++
```

- Serie di più righe
- Ciascuna riga ha un numero variabile di caratteri
- Solitamente contengono del testo "libero" (ad es., in italiano)
- Concetti di: carattere, parola, frase, riga

7

## File di "testo formattato" (custom)

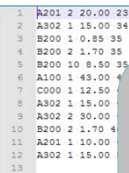


```
1 A201 2 20.00 23  
2 A302 1 15.00 34  
3 B200 1 0.85 35  
4 B200 2 1.70 35  
5 B200 10 8.50 35  
6 A100 1 49.00 43  
7 C000 1 12.50 44  
8 A302 1 15.00 45  
9 A302 2 30.00 46  
10 B200 2 1.70 46  
11 A201 1 10.00 47  
12 A302 1 15.00 50  
13
```

8

8

## File di "testo formattato" (custom)



```
1 A201 2 20.00 23  
2 A302 1 15.00 34  
3 B200 1 0.85 35  
4 B200 2 1.70 35  
5 B200 10 8.50 35  
6 A100 1 49.00 43  
7 C000 1 12.50 44  
8 A302 1 15.00 45  
9 A302 2 30.00 46  
10 B200 2 1.70 46  
11 A201 1 10.00 47  
12 A302 1 15.00 50  
13
```

- Formato "inventato" ad hoc per ciascun programma specifico
- Versione semplificata del CSV, dove il separatore è lo spazio e non vi sono virgolette delimitatrici
- È il più semplice da gestire

9

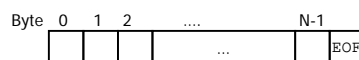
## File sequenziali

- Il modo più comune per realizzare I/O da file consiste nell'utilizzo del cosiddetto accesso *bufferizzato*
  - Informazioni prelevate dal file attraverso una memoria interna al sistema (detta *buffer*)
- Vantaggi:
  - Livello di astrazione più elevato
  - Possibilità di I/O formattato
- I/O non bufferizzato:
  - Accesso diretto a livello binario un carattere per volta

10

## File sequenziali (Cont.)

- Il C vede i file come un *flusso (stream) sequenziale di byte*
  - **Nessuna struttura particolare:**  
La strutturazione del contenuto è a carico del programmatore
  - Carattere terminatore alla fine del file: EOF



- NOTA: L'accesso sequenziale implica l'impossibilità di:
  - Leggere all'indietro
  - Saltare ad uno specifico punto del file

11

## Stream associato ad un file

- In un programma C, esiste un tipo di dato specifico per rappresentare le informazioni relative ad un file aperto
  - Denominato: **file stream** (flusso associato ad un file)
  - Tipo di dato: **FILE \*** (definito in <stdio.h>)
- "Aprire" un file significa quindi creare un nuovo stream ed associarlo ad uno specifico file sul disco

12

12

### Significato di stream

- Una volta che il file è aperto, il suo stream rappresenta
  - Un "collegamento" mediante il quale poter compiere delle operazioni sul contenuto del file
  - Le modalità di accesso scelte (testo/binario, lettura/scrittura/...)
  - La posizione attuale a cui si è arrivati nello scrivere o nel leggere il file
- Ogni operazione sul file avviene chiamando una funzione che riceve lo stream come parametro

13

13

### File sequenziali (Cont.)

- Accesso tramite una variabile di tipo `FILE*`
- Definita in `stdio.h`
- Dichiarazione:  

```
FILE* <file>;
```

`FILE*` contiene un insieme di variabili che permettono l'accesso per "tipi"
- Al momento dell'attivazione di un programma vengono automaticamente attivati tre file:
  - `stdin`
  - `stdout`
  - `stderr`

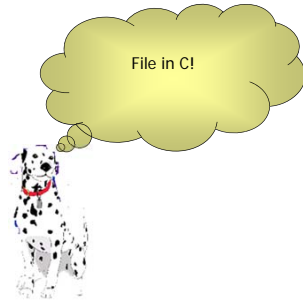
14

### File sequenziali (Cont.)

- `stdin` è automaticamente associato allo standard input (tastiera)
- `stdout` e `stderr` sono automaticamente associati allo standard output (video)
- `stdin`, `stdout`, `stderr` sono direttamente utilizzabili nelle istruzioni per l'accesso a file
  - In altre parole, sono delle variabili predefinite di tipo `FILE*`

15

## File



16

---

---

---

---

---

---

---

---

## File di testo in C

- Accesso ai file
- Funzioni `fopen/fclose`
- Funzioni `fgetc/fputc` (\*può essere `c` o `s` per caratteri o stringhe)
- Funzioni `fprintf/fscanf`
- Condizione `feof`

17

17

---

---

---

---

---

---

---

---

## File: Operazioni

- L'uso di un file passa attraverso tre fasi fondamentali
  - **Apertura del file**
  - **Accesso al file**
  - **Chiusura del file**
- Prima di aprire un file occorre dichiararlo!

18

---

---

---

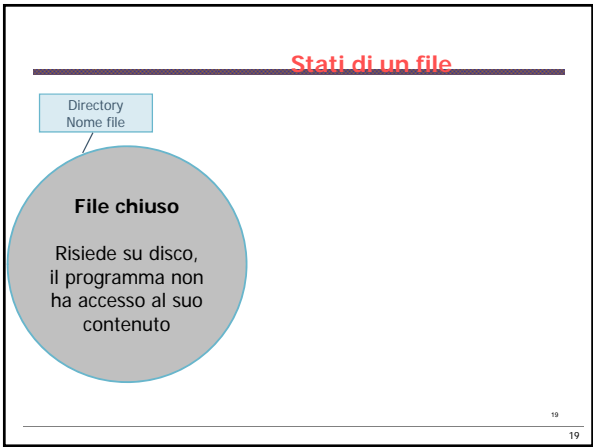
---

---

---

---

---



---

---

---

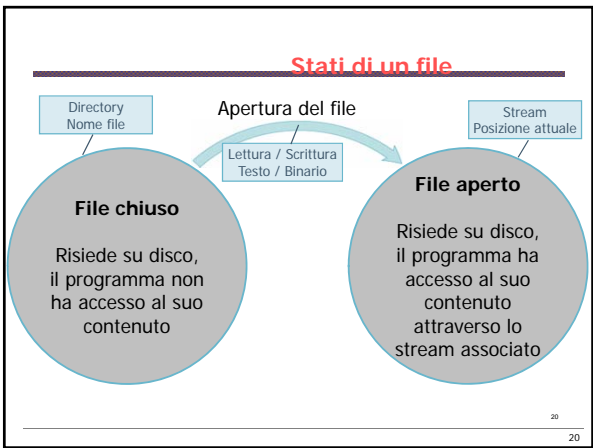
---

---

---

---

---



---

---

---

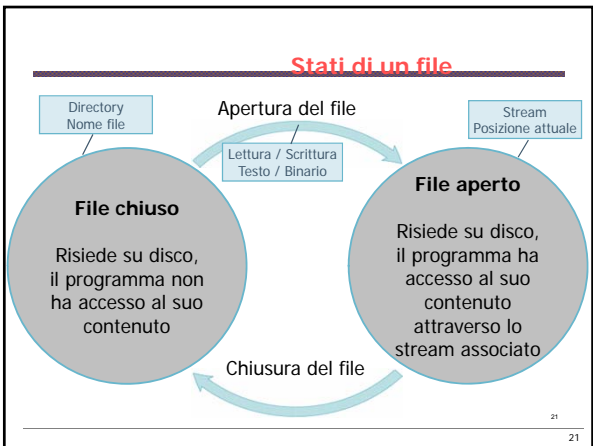
---

---

---

---

---



---

---

---

---

---

---

---

---

## Apertura di un file

- Per accedere ad un file è necessario aprirlo:
  - Apertura:  
Connessione di un file fisico (su disco) ad un file logico (interno al programma)
- Funzione:  

```
FILE* fopen(char* <nomefile>, char* <modo>);
```

<nomefile>: Nome del file fisico

22

## fopen: sintassi

```
FILE * f ;
```

Variabile stream  
di tipo FILE \*

```
...
```

```
f = fopen( "nomefile", "modo" );
```

Stringa  
contenente il  
nome del file

Modalità di  
apertura  
(stringa)

23

## Nome del file

```
f = fopen( "nomefile", "modo" );
```

```
f = fopen( "dati.txt", "modo" );
```

24



### Nome del file

```
f = fopen( "nomefile", "modo" );
```

```
f = fopen( "dati.txt", "modo" );
```

```
f = fopen( "c:\\prog\\dati.txt",  
          "modo" );
```

25

25

### Apertura di un file (Cont.)

- **<modo>**: Tipo di accesso al file
  - "r": sola lettura
  - "w": sola scrittura (cancella il file se esiste)
  - "a": *append* (aggiunge in coda ad un file)
  - "r+": lettura/scrittura su file esistente
  - "w+": lettura/scrittura su nuovo file
  - "a+": lettura/scrittura in coda o su nuovo file
- Ritorna:
  - Il puntatore al file in caso di successo
  - NULL in caso di errore

26

### Controllo dell'errore

```
FILE * f ;  
...  
f = fopen( "nomefile", "r" );  
if( f == NULL )  
{  
    printf("Impossibile aprire file\n");  
    exit(1);  
}
```

27

27

## Chiusura di un file

- Quando l'utilizzo del file fisico è terminato, è consigliabile chiudere il file:
  - Chiusura:  
Cancellazione della connessione di un file fisico (su disco) ad un file logico (interno al programma)

- Funzione:

```
int fclose(FILE* <file>);
```

<file>: File aperto in precedenza con fopen()

- Ritorna:

- 0 se l'operazione si chiude correttamente
- EOF in caso di errore

28

## fclose: sintassi

```
FILE * f ;  
...  
f = fopen( "nomefile", "modo" ) ;  
.../* accesso al file */  
fclose(f) ;
```

Variabile stream

29

29

## Avvertenze

- La funzione fclose può essere chiamata solamente su stream correttamente aperti
  - Mai chiamare fclose se f==NULL
- Dopo la chiusura del file, non è più possibile accedere allo stream
  - Eventualmente, ri-aprirlo nuovamente

30

30

### Controllo dell'errore

- La funzione `fclose` ritorna un valore di tipo intero:

```
int ris ;
...
ris = fclose(f) ;
if(ris!=0)
{
    printf("Impossibile chiudere\n") ;
    exit(1) ;
}
```

31

31

### Apertura e chiusura di un file: Esempio

```
FILE *fp; /* variabile di tipo file */
...

/* apro file 'testo.dat' in lettura */
fp = fopen("testo.dat", "r");

if (fp == NULL)
    printf("Errore nell'apertura\n");
else
{
    /* qui posso accedere a 'testo.dat' usando fp */
}
...
fclose(fp);
```

32

### Lettura di un file

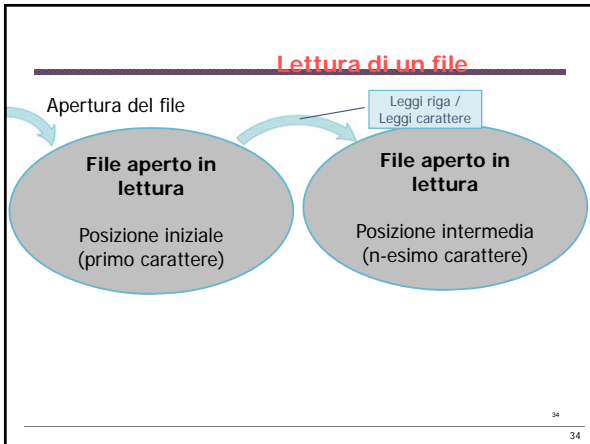
Apertura del file

**File aperto in lettura**

Posizione iniziale  
(primo carattere)

33

33



---

---

---

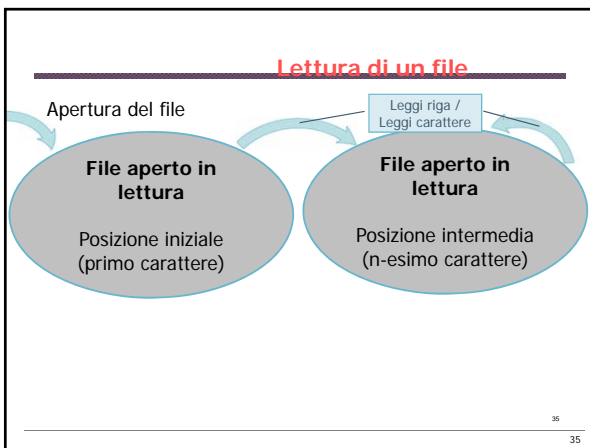
---

---

---

---

---



---

---

---

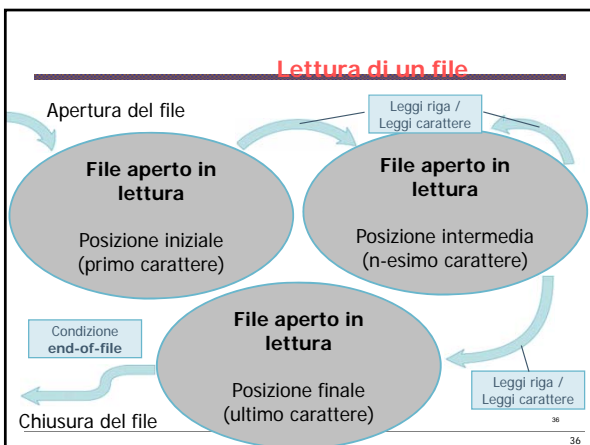
---

---

---

---

---



---

---

---

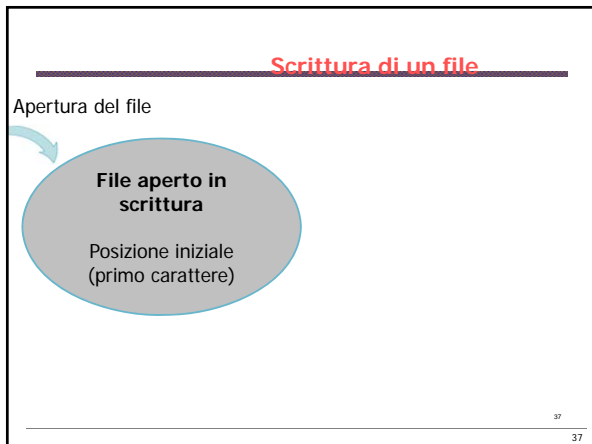
---

---

---

---

---



---

---

---

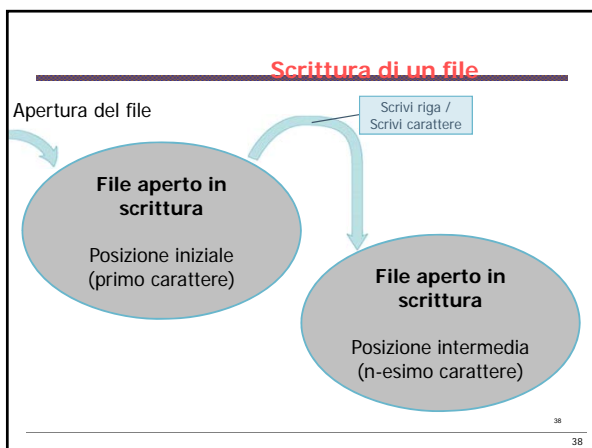
---

---

---

---

---



---

---

---

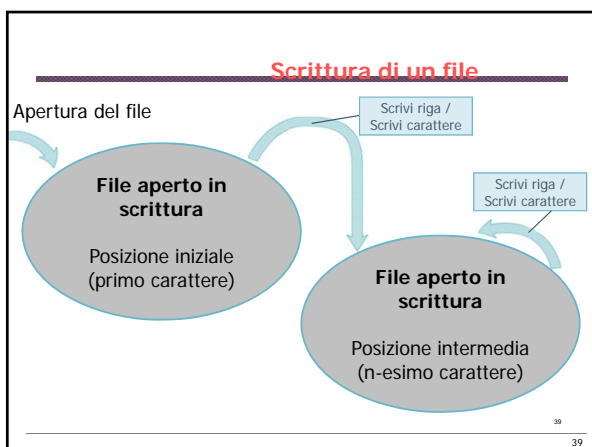
---

---

---

---

---



---

---

---

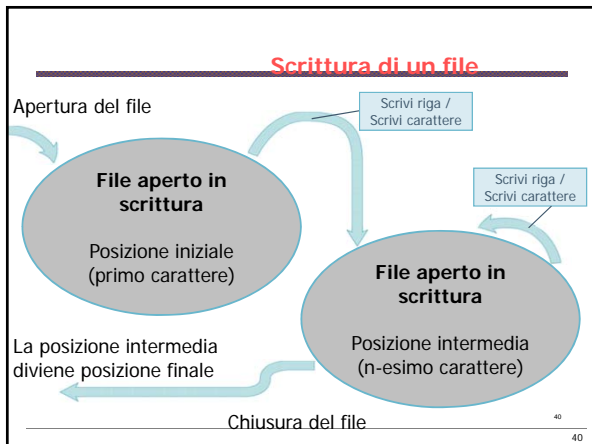
---

---

---

---

---




---

---

---

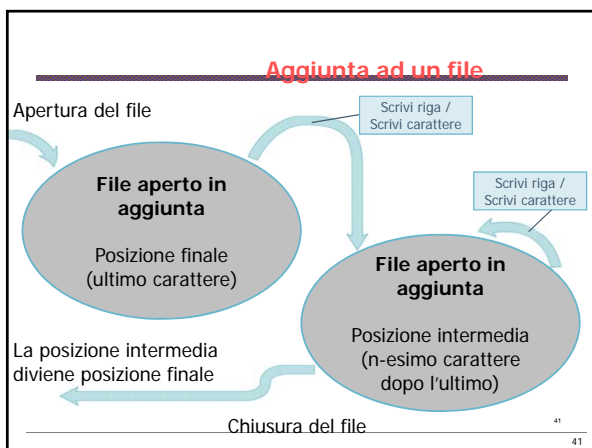
---

---

---

---

---




---

---

---

---

---

---

---

---

### Lettura a caratteri

- Lettura:
  - `int getc (FILE* <file>);`
  - `int fgetc (FILE* <file>);`
    - Legge un carattere alla volta dal file
    - Restituisce il carattere letto o EOF in caso di fine file o errore
- NOTA: `getchar()` equivale a `getc(stdin)`

42

---

---

---

---

---

---

---

---

## Scrittura a caratteri

### • Scrittura:

- `int putc (int c, FILE* <file>);`
- `int fputc (int c, FILE* <file>);`
  - Scrive un carattere alla volta nel file
  - Restituisce il carattere scritto o EOF in caso di errore

• NOTA: `putc(...)` equivale a `putc(..., stdout)`

43

## fgetc: sintassi

```
int ch ;  
ch = fgetc(f) ;
```

Stream aperto  
in lettura

Prossimo carattere del file;  
EOF se il file è finito

44

## fputc: sintassi

```
int ch ;  
fputc(ch, f) ;
```

Stream aperto in  
scrittura o in aggiunta

Carattere da  
aggiungere al file

45

## Lettura a righe

- Lettura:

```
char* fgets(char* <s>, int <n>, FILE* <file>);
```

- Legge una stringa dal file fermandosi al più dopo n-1 caratteri
- L'eventuale '\n' NON viene eliminato (diverso da gets !)
- Restituisce il puntatore alla stringa letta o NULL in caso di fine file o errore

- NOTA: gets(...) "equivale" a fgets(..., stdin)

46

## Scrittura a righe

- Scrittura:

```
int fputs(char* <s>, FILE* <file>);
```

- Scrive la stringa <s> nel senza aggiungere '\n' (diverso da puts !)
- Restituisce l'ultimo carattere scritto, oppure EOF in caso di errore

- NOTA: puts(...) "equivale" a fputs(..., stdout)

47

## fgets: sintassi

```
char str[80] ;  
fgets(str, 79, f) ;
```

Max numero di  
caratteri letti

Stringa nella quale viene  
letta la prossima riga del file  
(fino al '\n' compreso)

Stream aperto  
in lettura

48



### Fine del file

- La funzione `fgets` restituisce un valore `NULL` se ha tentato di leggere oltre la fine del file

```
char str[80] ;  
While( fgets(str, 79, f) != NULL )  
{  
    /* elabora str */  
}
```

49

### fputs: sintassi

```
char str[80] ;  
fputs(str, f) ;
```

Stringa da aggiungere al file  
(solitamente termina con `\n`)

Stream aperto in  
scrittura o in aggiunta

50

### Lettura formattata

- Lettura:  

```
int fscanf(FILE* <file>, char* <formato>, ...);
```

  - Come `scanf()`, con un parametro aggiuntivo che rappresenta un file
  - Restituisce il numero di campi convertiti, oppure `EOF` in caso di fine file
- NOTA: `scanf(...)` "equivale" a `fscanf(stdin,...)`

51

## Scrittura formattata

- Scrittura:

```
int fprintf(FILE* <file>, char* <formato>, ...);
```

- Come `printf()`, con un parametro aggiuntivo che rappresenta un file
- Restituisce il numero di byte scritti, oppure EOF in caso di errore

- NOTA: `printf(...)` "equivale" a `fprintf(stdout,...)`

52

## fscanf: sintassi

```
FILE * f ;  
fscanf(f, "formato", &variabili) ;
```

Stream aperto  
in lettura

Puntatori alle  
variabili da leggere

Formato dei dati da leggere,  
usando gli stessi specificatori  
validi per `scanf`

53

53

## Input formattato

- Qualora sia necessario leggere file con più campi nella stessa riga
  - È scomodo ricorrere alla funzione `fgetc`
  - Il risultato della funzione `fgetc` deve successivamente essere analizzato
- È possibile utilizzare una variante della funzione `scanf`, operante su uno stream aperto in lettura
  - `fscanf(f, "formato", &x, &y, &z) ;`

54

54

### Lettura di un file: fgets + sscanf

- Lettura di un file formattato in cui ogni riga abbia un dato numero di campi di tipo noto (esempio un intero, ed una stringa)
  - Uso di fgets per leggere la riga, e di sscanf per leggere i campi

```
while ((s = fgets(s,80,fp))!= NULL)
{
    sscanf( s, "%d %s", &intero, stringa );
}
```

55

---

---

---

---

---

---

---

### Scrittura formattata

- Scrittura:

```
int fprintf(FILE* <file>,char* <formato>,...);
```

  - Come printf(), con un parametro aggiuntivo che rappresenta un file
  - Restituisce il numero di byte scritti, oppure EOF in caso di errore
- NOTA: printf(...) "equivale" a fprintf(stdout,...)

56

---

---

---

---

---

---

---

### Output formattato

- Qualora sia necessario creare file con più campi nella stessa riga, è scomodo ricorrere alle funzioni fputc/fputs
- È possibile utilizzare una variante della funzione printf, operante su uno stream aperto in scrittura
  - fprintf(f, "formato", x, y, z) ;

- Esempi o:

```
nome cognome voto
s      s      d
```

57

57

---

---

---

---

---

---

---

### fprintf: sintassi

```
FILE * f ;  
fprintf(f, "formato", variabili ) ;
```

Stream aperto in scrittura o in aggiunta

Elenco delle variabili da scrivere

Formato dei dati da stampare, usando gli stessi specificatori validi per printf

58

### Altre funzioni

- `FILE* freopen(char* <nomefile>, char* <modo>);`
  - Come `fopen`, ma si applica ad un file già esistente
  - Restituisce il puntatore al file oppure `NULL`
- `int fflush(FILE* <file>);`
  - "Cancella" il contenuto di un file
  - Restituisce 0 se termina correttamente oppure `EOF`
- `int feof(FILE* <file>);`
  - Restituisce falso (0) se il puntatore NON è posizionato alla fine del file
  - Restituisce vero (10) se il puntatore è posizionato alla fine del file

59

### Schema generale di lettura da file

```
leggi un dato dal file;  
finchè (non è finito il file)  
{  
    elabora il dato;  
    leggi un dato dal file;  
}
```

- La condizione "non è finito il file" può essere realizzata in vari modi:
  - Usando i valori restituiti dalle funzioni di input (consigliato)
  - Usando la funzione `feof()`

60

### Esempio 1

- Lettura di un file formattato (esempio: Un intero per riga)
  - Uso dei valori restituiti dalle funzioni di input (`fscanf`)

```
res = fscanf (fp, "%d", &val);
while (res != EOF)
{
    elabora val;
    res = fscanf (fp, "%d", &val);
}
```

61

---

---

---

---

---

---

---

---

### Esempio 1 (Cont.)

- Versione "compatta" senza memorizzare il risultato di `fscanf()`
  - Usiamo `fscanf()` direttamente nella condizione di fine input

```
while (fscanf (fp, "%d", &val) != EOF)
{
    elabora val;
}
```

62

---

---

---

---

---

---

---

---

### Esempio 2

- Lettura di un file formattato (esempio: Un intero per riga)
  - Uso di `feof()`

```
fscanf (fp, "%d", &val);
while (!feof(fp))
{
    elabora val;
    fscanf (fp, "%d", &val);
}
```

63

---

---

---

---

---

---

---

---

### Esempio 3

- Lettura di un file non formattato
  - Uso dei valori restituiti dalle funzioni di input (`getc`)

```
c = getc(fp);
while (c != EOF)
{
    elabora c;
    c = getc(fp);
}
```

*Versione 1*

```
while ((c=getc(fp))!= EOF)
{
    elabora c;
}
```

*Versione 2*

64

---

---

---

---

---

---

---

---

### Esempio 4

- Lettura di un file non formattato
  - Uso dei valori restituiti dalle funzioni di input (`fgets`)

```
s = fgets(s,80,fp);
while (s != NULL)
{
    elabora s;
    s = fgets(s,80,fp);
}
```

*Versione 1*

```
while ((s = fgets(s,80,fp))!= NULL)
{
    elabora s;
}
```

*Versione 2*

65

---

---

---

---

---


---

---

---

### Esercizio

- Leggere un file "estremi.dat" contenente coppie di numeri interi (x,y), una per riga e scrivere un secondo file "diff.dat" che contenga le differenze x-y, una per riga
- Esempio:

File 1				File 2	
23	32			-9	
2	11			-9	
19	6			13	
23	5			18	
3	2			1	
...	...			...	

66

---

---

---

---

---

---

---

---

### Esercizio: Soluzione

```
#include <stdio.h>

main() {
    FILE *fpin, *fpout;
    int x,y;
    /* apertura del primo file */
    if ((fpin = fopen("estremi.dat","r")) == NULL)
    {
        fprintf(stderr,"Errore nell'apertura\n");
        return 1;
    }
}
```

67

---

---

---

---

---

---

---

---

### Esercizio: Soluzione (Cont.)

```
/* apertura del secondo file */
if ((fpout = fopen("diff.dat","w")) == NULL)
{
    fprintf(stderr,"Errore nell'apertura\n");
    return 1;
}
/* input */
while (fscanf(fpin,"%d %d",&x,&y) != EOF)
{
    /* ora ho a disposizione x e y */
    fprintf(fpout,"%d\n",x-y);
}
fclose (fpin);
fclose (fpout);
}
```

68

---

---

---

---

---

---

---

---

### Avvertenza

- In generale, è errato tentare di memorizzare il contenuto di un file in un vettore
  - La dimensione (numero di righe o di dati) di un file non è quasi mai nota a priori
  - Se la dimensione è nota, tipicamente è molto grande!

69

---

---

---

---

---

---

---

---

### Esercizio:

- Si legga da tastiera una sequenza di coppie (squadra, punti) da memorizzare in un file di testo
- Si scriva un programma in C che acquisisca da tale file la successione di coppie e individui la squadra con il maggior punteggio



70

---

---

---

---

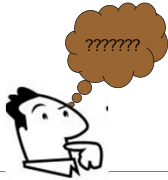
---

---

---

### Esercizio:

- Sia dato un file testo contenente i valori di una matrice  $N \times M$ .
- Si scriva un programma in C che acquisisca da tale file la matrice e calcoli di tutte le possibili sottomatrici  $3 \times 3$  quella che possiede massima la somma dei propri elementi



71

---

---

---

---

---

---

---