UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

Corso di Laurea in Informatica e Tecnologie per la Produzione del Software *(Track B)* - A.A. 2018/2019

Laboratorio di Informatica

Elaborazione di File

docente: Veronica Rossano

Veronica.rossano@uniba.it

Slides ispirate ai contenuti proposti dal prof. Corrado Mencar. Grazie.

Input/Output

- **Input e Output sono due concetti fondamentali** della Programmazione
- **Un algoritmo** è una sequenza finita di passi che acquisisce un input e produce in output un risultato

Input/Output

- **Input e Output sono due concetti fondamentali** della Programmazione
- **Un algoritmo** è una sequenza finita di passi che acquisisce un input e produce in output un risultato

- Finora abbiamo utilizzato soltanto i cosiddetti **'standard input'** (tastiera) e **'standard output'** (lo schermo)

Input/Output – i File - Esempio

- **Esempio**
  - Scrivere un programma che acquisisca in input l'elenco degli studenti che hanno sostenuto l'esame con il relativo voto, e stampi in output le matricole degli studenti che hanno superato l'esame

Come siamo abituati a immaginare questo programma?

Input/Output – i File - Esempio

- **Esempio**
  - Scrivere un programma che acquisisca in input l'elenco degli studenti che hanno sostenuto l'esame con il relativo voto, e stampi in output le matricole degli studenti che hanno superato l'esame


```
printf(«Inserisci matricola e voto:»);
scanf(«%d %d», &matricola, &voto)
…
```


Input/Output – i File - Esempio

- **Esempio**
  - Scrivere un programma che acquisisca in input l'elenco degli studenti che hanno sostenuto l'esame con il relativo voto, e stampi in output le matricole degli studenti che hanno superato l'esame


```
printf(«Inserisci matricola e voto:»);
scanf(«%d %d», &matricola, &voto)
…
```

```
printf(«%d\n», &matricola)
```


Input/Output – i File - Esempio

- **Esempio**
  - Scrivere un programma che acquisisca in input l'elenco degli studenti che hanno sostenuto l'esame con il relativo voto, e stampi in output le matricole degli studenti che hanno superato l'esame


```
printf(«Inserisci matricola e voto:»);
scanf(«%d %d», &matricola, &voto)
…
```

```
printf(«%d\n», &matricola)
```

Esistono modalità alternative per gestire input e output?

Input/Output – i File

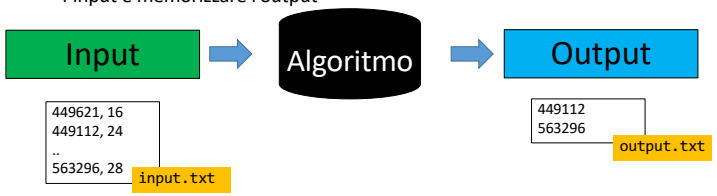
- **I File svolgono un ruolo fondamentale nell'ambito della programmazione**

Input/Output – i File

- **I File svolgono un ruolo fondamentale nell'ambito della programmazione**
  - Possono essere utilizzati per **acquisire in automatico l'input**, senza doverlo digitare da tastiera
  - Possono essere utilizzati per **memorizzare in modo persistente l'output** del programma
    - *Normalmente l'output del programma viene perso al termine dell'esecuzione del programma*
- **Il Linguaggio C fornisce degli strumenti per accedere, creare ed elaborare i file**

Input/Output – i File - Esempio

- **Il Linguaggio C fornisce degli strumenti per accedere, creare ed elaborare i file**
  - La stessa implementazione dell'algoritmo può utilizzare i file per acquisire l'input e memorizzare l'output

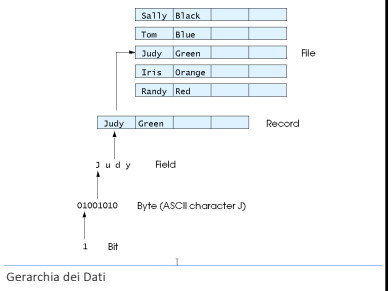


File

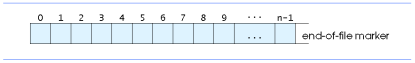
- **Nel Linguaggio C i file vengono gestiti utilizzando il concetto di `stream (flusso)`**
- Uno stream **è una sequenza di dati**
  - Dati: delle stringhe, dei valori interi, delle **`struct`**, etc.
  - Tipicamente, i file si utilizzano per memorizzare a lungo termine delle **sequenze di record** (es. le **`struct`**), ma possono essere utilizzati per memorizzare in modo persistente **ogni tipologia di dato.**

File

- **Nel Linguaggio C i file vengono gestiti utilizzando il concetto di `stream (flusso)`**
- Uno stream **è una sequenza di dati**
  - Dati: delle stringhe, dei valori interi, delle **`struct`**, etc.
  - Tipicamente, i file si utilizzano per memorizzare a lungo termine delle **sequenze di record** (es. le **`struct`**), ma possono essere utilizzati per memorizzare in modo persistente **ogni tipologia di dato.**

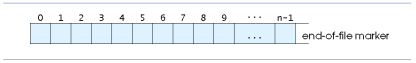


File



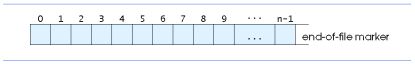
- **Lo stream è un concetto astratto**
  - Anche la comunicazione su Internet avviene sfruttando lo stesso concetto (flussi di dati vengono scambiati tra un client e un web server, attraverso un browser)
- **Un input stream** è un flusso di dati che può essere **letto**
  - Apertura di uno stream
  - Lettura di un dato
  - Avanzamento al dato successivo
  - Verifica di fine stream
  - Chiusura di uno stream

File



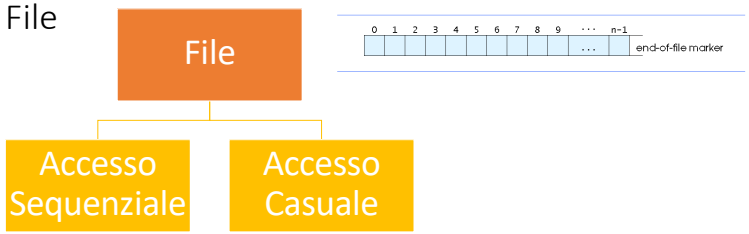
- **Lo stream è un concetto astratto**
  - Anche la comunicazione su Internet avviene sfruttando lo stesso concetto (flussi di dati vengono scambiati tra un client e un web server, attraverso un browser)
- **Un input stream** è un flusso di dati che può essere **letto**
  - Apertura di uno stream
  - Lettura di un dato
  - Avanzamento al dato successivo
  - Verifica di fine stream
  - Chiusura di uno stream
- **Un output stream** è un flusso di dati che può essere **scritto**
  - Apertura di uno stream
  - Scrittura di un dato
  - Accodamento di un dato
  - Chiusura di uno stream

File

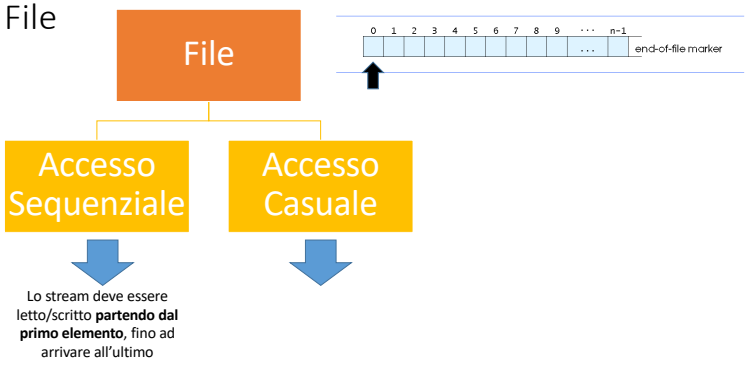


- **Lo stream è un concetto astratto**
  - Anche la comunicazione su Internet avviene sfruttando lo stesso concetto (flussi di dati vengono scambiati tra un client e un web server, attraverso un browser)
- **Abbiamo già incontrato il concetto di stream di dati**
  - **L'input da tastiera** è uno stream di dati **`(stdin)`**
  - **L'output sullo schermo** è uno stream di dati **`(stdout)`**
  - I metodi per operare sui file sono delle «varianti» delle classiche **`printf()`** **e** **`scanf()`** usate finora.

File

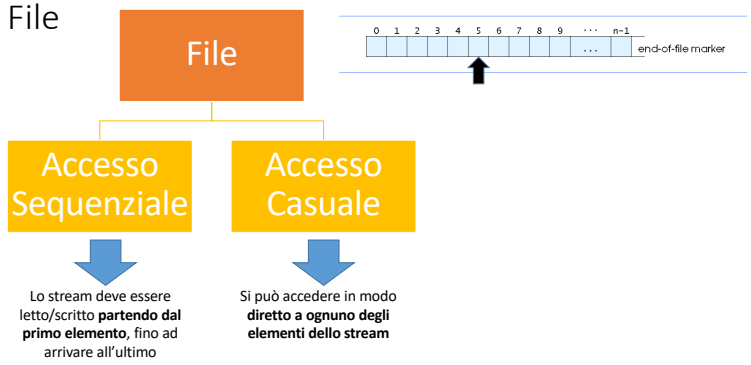


File



Lo stream deve essere letto/scritto **partendo dal primo elemento**, fino ad arrivare all'ultimo

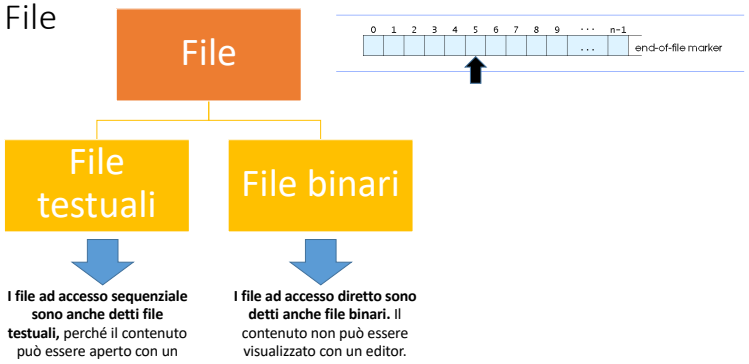
File



Lo stream deve essere letto/scritto **partendo dal primo elemento**, fino ad arrivare all'ultimo

Si può accedere in modo **diretto a ognuno degli elementi dello stream**

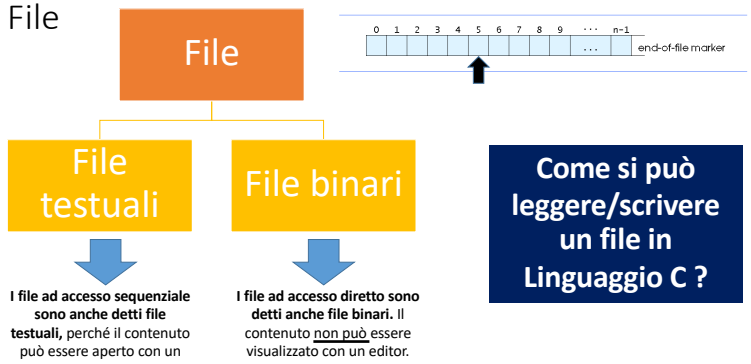
File



I file ad accesso sequenziale sono anche detti file testuali, perché il contenuto può essere aperto con un editor di testo e visualizzato

I file ad accesso diretto sono detti anche file binari. Il contenuto non può essere visualizzato con un editor.

File



I file ad accesso sequenziale sono anche detti file testuali, perché il contenuto può essere aperto con un editor di testo e visualizzato

I file ad accesso diretto sono detti anche file binari. Il contenuto non può essere visualizzato con un editor.

Come si può leggere/scrivere un file in Linguaggio C ?

File

- **Dichiarare una variabile di tipo file**
  - **`FILE *fileName`**
  - E' un puntatore. **A cosa?**

File

- **Dichiarare una variabile di tipo file**
  - **`FILE *fileName`**
  - E' un puntatore. **A cosa?**
    - A una **`struct`** di tipo **FILE** , definita in **`<stdio.h>`**
    - **Per utilizzare i file bisogna includere la libreria** (che già includiamo per le classiche operazioni di input/output)

File

- **Dichiarare una variabile di tipo file**
  - **`FILE *fileName`**
  - E' un puntatore. **A cosa?**
    - A una **`struct`** di tipo **FILE** , definita in **`<stdio.h>`**
    - Cosa contiene questa **`struct`**?
      - **Informazioni di sistema**, come **l'indice del file nella tabella dei File Aperti** del Sistema Operativo, che serve a recuperare il **File Control Block del file**

File

- **Dichiarare una variabile di tipo file**
  - **`FILE *fileName`**
  - E' un puntatore. **A cosa?**
    - A una **`struct`** di tipo **FILE** , definita in **`<stdio.h>`**
    - Cosa contiene questa **`struct`**?
      - **Informazioni di sistema**, come **l'indice del file nella tabella dei File Aperti** del Sistema Operativo, che serve a recuperare il **File Control Block del file**
      - Contiene informazioni sui **permessi del file** (lettura, scrittura, etc.), data di creazione/modifica, **la locazione fisica dei blocchi di memoria**

File

```
typedef struct {
        char *fpos; /* Current position of file pointer (absolute address) */
        void *base; /* Pointer to the base of the file */
        unsigned short handle; /* File handle */
        short flags; /* Flags (see FileFlags) */
        short unget; /* 1-byte buffer for ungetc (b15=1 if non-empty) */
        unsigned long alloc; /* Number of currently allocated bytes for the file */
        unsigned short buffincrement; /* Number of bytes allocated at once */
} FILE;
```


File

- **Quali operazioni possiamo fare sui file?**
  - Apertura File
  - Lettura Dati
  - Scrittura Dati
  - Chiusura File
  - «Riavvolgimento» dello Stream
  - Verifica di fine Stream

File

- **Quali operazioni possiamo fare sui file?**
  - Apertura File
  - Lettura Dati
  - Scrittura Dati
  - Chiusura File
  - «Riavvolgimento» dello Stream
  - Verifica di fine Stream
  - Collocare il puntatore su un punto preciso del file (utile soprattutto per i file binari)

Apertura File

- **`FILE* fopen(const char* filename, const char* mode);`**
  - `filename` → nome del file da aprire
  - `mode` → modalità d'apertura
- **Esempio**
  - **`FILE* file = fopen("content.txt", "w");`**

Apertura File

- `FILE* fopen(const char* filename, const char* mode);`
  - `filename` → nome del file da aprire
  - `mode` → modalità d'apertura

- **Esempio**
  - `FILE* file = fopen("content.txt", "w");` (nome del file, modalità)
  - Se il file esiste, restituisce **un puntatore al file**. Altrimenti restituisce **NULL.**

Apertura File - Modalità

Modalità	Cosa fa
r	open a text file for **reading**
w	truncate to zero length or **create a text file for writing**
a	append; open or create text file for **writing at end-of-file**
r+	**open text file for update (reading and writing)**
w+	truncate to zero length or create a text file for update
a+	append; open or create text file for update
rb	open a binary file for **reading**
wb	truncate to zero length or **create a binary file for writing**
ab	append; open or create binary file for **writing at end-of-file**
rb+	**open binary file for update (reading and writing)**
wb+	truncate to zero length or create a binary file for update
ab+	append; open or create binary file for update

Apertura File - Modalità

File **Testuali**

File **Binari**

Modalità	Cosa fa
r	open a text file for **reading**
w	truncate to zero length or **create a text file for writing**
a	append; open or create text file for **writing at end-of-file**
r+	**open text file for update (reading and writing)**
w+	truncate to zero length or create a text file for update
a+	append; open or create text file for update
rb	open a binary file for **reading**
wb	truncate to zero length or **create a binary file for writing**
ab	append; open or create binary file for **writing at end-of-file**
rb+	**open binary file for update (reading and writing)**
wb+	truncate to zero length or create a binary file for update
ab+	append; open or create binary file for update

Chiusura File

- `int fclose(const char* filename);`
  - `filename` → nome del file da chiudere

- Un file aperto ha, di norma, un buffer associato
  - **buffer:** area di memoria di appoggio, utilizzata per velocizzare le operazioni di I/O
- La chiusura di un file assicura che il contenuto del buffer **sia traferito nello stream**
- La chiusura disassocia il descrittore FILE dallo stream e libera risorse

- **Esempio**
  - `fclose(file);`

Apertura/Chiusura File (Esempio)

```c
#include <stdio.h>

int main() {
  FILE *file;

  if((file = fopen("test.txt","r")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```


Apertura/Chiusura File (Esempio)

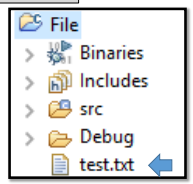
```c
#include <stdio.h>

int main() {
  FILE *file;

  if((file = fopen("test.txt","r")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

Attenzione alle parentesi! **L'intera condizione** deve essere diversa da NULL

Apertura/Chiusura File (Esempio)

```c
#include <stdio.h>

int main() {
  FILE *file;

  if((file = fopen("test.txt","r")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

IMPORTANTE: il file deve trovarsi nella cartella principale del progetto!



Apertura/Chiusura File (Esempio)

```c
#include <stdio.h>

int main() {
  FILE *file;

  if((file = fopen("test.txt","r")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

Se il file esiste (e quindi il puntatore è diverso da NULL) stampa un messaggio

Apertura/Chiusura File (Esempio)

```c
#include <stdio.h>

int main() {
  FILE *file;

  if((file = fopen("test.txt","r")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

fclose(file)==0 se il file viene chiuso correttamente. **Lo zero equivale alla negazione!**

Lettura da File

- **int fscanf(FILE* stream, const char* format, ...);**
  - stream → nome del file da cui leggere i dati
  - format → specificatore del formato dei dati

- **Esempio**
  - int value=0; FILE *file;
  - **fscanf(file, ”%d”, &value);**

- Segue lo stesso formato della **scanf( )** che abbiamo già utilizzato
- Bisogna semplicemente **aggiungere il puntatore al file**

Lettura Dati da File (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file
    printf("Valore Letto: %d\n", value);
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```


Lettura Dati da File (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file
    printf("Valore Letto: %d\n", value);
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

Variabile intera dove memorizzare il valore

Lettura Dati da File (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file
    printf("Valore Letto: %d\n", value);
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

Leggo valore intero dal file aperto e le memorizzo nella variabile **value,** e lo stampo.

Lettura Dati da File (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file
    printf("Valore Letto: %d\n", value);
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

```
gcc version 4.6.3
>
File Aperto
Valore Letto: 10
File Chiuso
>
```

Leggo valore intero dal file aperto e le memorizzo nella variabile **value,** e lo stampo.

Scrittura su File

- **`int fprintf(FILE* stream, const char* format, ...);`**
  - `stream` → nome del file da cui leggere i dati
  - `format` → specificatore del formato dei dati

- **Esempio**
  - `int value=0; FILE *file;`
  - **`fprintf(file, "%d", value);`**

- Segue lo stesso formato della **`printf()`** che abbiamo già utilizzato
- Bisogna semplicemente **aggiungere il puntatore al file**

Scrittura Dati su File (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

Scrittura Dati su File (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

Modifichiamo la modalità di apertura, perché dobbiamo fare **sia lettura che scrittura.**

Cosa succede se utilizziamo «r» invece di «r+» ?

Scrittura Dati su File (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

Il programma stampa «Valore Scritto» ma non scrive nulla.
Come risolvere?

Scrittura Dati su File (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

Il programma stampa «Valore Scritto» ma non scrive nulla.
Come risolvere?

Si può aggiungere un controllo sull'istruzione `fprintf( )`

`fprintf` restituisce un intero pari al numero di caratteri scritti. **Se non ha scritto caratteri o c'è stato un errore, il valore restituito è negativo.**

Modificando l'istruzione al rigo 16 in:
`if( fprintf(file, "%d", 123) > 0)`
Aggiungiamo un controllo che aumenta la solidità del programma

Scrittura Dati su File (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

Scrive 123 nel file. **Lo accoda a valle dei dati presenti nel file (accesso sequenziale!)**

Assumendo che nel file '**`test.txt`**' sia contenuto in partenza il valore '**10**', quale output avremo **dopo due esecuzioni** del programma?

Scrittura Dati su File (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

```
gcc version 4.6.3
>
File Aperto
Valore Letto: 10123123
Valore Scritto!
File Chiuso
>
```

Output dopo due esecuzioni. Ad ogni esecuzione aggiunge 123 in coda al valore iniziale letto dal file.

Accesso sequenziale: i nuovi dati vengono accodati in base alla posizione del puntatore

Nota importante

- Le funzioni **`fscanf()`** e **`fprintf()`** prendono in input **un generico stream.**
- Conosciamo altri stream? **Si**
  - **`stdin`** → standard **input**
  - **`stdout`** → standard **output**
- Le stesse funzioni possono anche essere **redirezionate verso gli standard input/output**, leggendo da tastiera e scrivendo sullo schermo come siamo abituati a fare!

Nota importante

- Le funzioni **`fscanf()`** e **`fprintf()`** prendono in input **un generico stream.**
- Conosciamo altri stream? **Si**
  - **`stdin`** → standard **input**
  - **`stdout`** → standard **output**
- Le stesse funzioni possono anche essere **redirezionate verso gli standard input/output**, leggendo da tastiera e scrivendo sullo schermo come siamo abituati a fare!

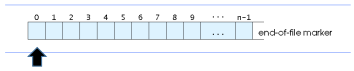
- Dunque
  - **`fprintf(stdout, «%d», 10) == printf(«%d», 10)`**
  - **`fscanf(stdin, «%d», &value) == scanf(«%d», &value)`**

Riavvolgimento dello Stream

- **`void rewind(FILE* stream);`**
  - `stream` → nome dello stream da «riavvolgere»

- **Esempio**
  - `int value=0; FILE *file;`
  - **`fprintf(file, ”%d”, &value);`**
  - **`rewind(file)`**

- **Risultato**
  - Riporta il **puntatore all'inizio del file**



Riavvolgimento dello Stream (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

```


Riavvolgimento dello Stream (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

```

Riavvolgo lo stream e poi scrivo **su file un nuovo valore.**

Riavvolgimento dello Stream (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

```

Supponendo che nel file sia memorizzato in partenza il valore 10, qual è l'output di questo programma?

Riavvolgimento dello Stream (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

```

Supponendo che nel file sia memorizzato in partenza il valore 10, qual è l'output di questo programma?

12323

Perché?

Riavvolgimento dello Stream (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

```

Supponendo che nel file sia memorizzato in partenza il valore 10, qual è l'output di questo programma?

12323

Perché?

Perché il file è ad **accesso sequenziale**

Ricostruiamo la situazione

Riavvolgimento dello Stream (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

```

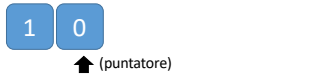
Supponendo che nel file sia memorizzato in partenza il valore 10, qual è l'output di questo programma?

12323

Perché?

Perché il file è ad **accesso sequenziale**

Ricostruiamo la situazione



Riavvolgimento dello Stream (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

```

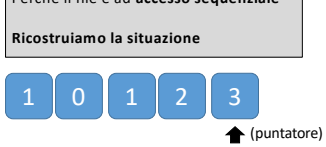
Supponendo che nel file sia memorizzato in partenza il valore 10, qual è l'output di questo programma?

12323

Perché?

Perché il file è ad **accesso sequenziale**

Ricostruiamo la situazione



Riavvolgimento dello Stream (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

```

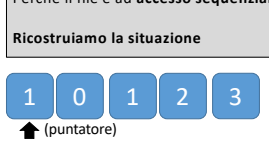
Supponendo che nel file sia memorizzato in partenza il valore 10, qual è l'output di questo programma?

12323

Perché?

Perché il file è ad **accesso sequenziale**

Ricostruiamo la situazione



Riavvolgimento dello Stream (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

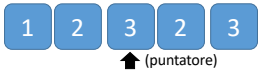
Supponendo che nel file sia memorizzato in partenza il valore 10, qual è l'output di questo programma?

12323

Perché?

Perché il file è ad **accesso sequenziale**

Ricostruiamo la situazione

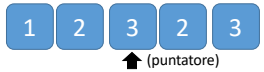


Riavvolgimento dello Stream (Esempio)

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

Nei file ad accesso sequenziale il contenuto viene letto elemento per elemento, quindi può capitare di sovrascrivere (anche non volendo) contenuti presenti nel file.

I file ad accesso casuale risolvono questo problema!



Verifica di Fine Stream

- `int feof(FILE* stream);`
  - `stream` → nome dello stream da verificare
  - Restituisce **`true`** se il file è terminato
- **Esempio**

```c
FILE file;
while(!feof(file)) {
   //cose
}
```

- **A cosa serve?**
  - A implementare dei cicli che scorrano tra i contenuti di un file

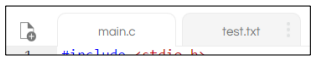
Esercizio 10.1

- Implementare un programma che acquisisca da tastiera **nome (o matricola) e voto d'esame per cinque individui**.
- I valori acquisiti devono essere memorizzati su file (una coppia di valori per ogni riga).
- Il programma deve poi leggere il file dall'inizio e **stampare a schermo i nomi degli studenti che hanno superato l'esame.**

- Scelte progettuali
  - I dati di input possono essere memorizzati in variabili singole o in **`struct`**
  - I vari passaggi devono essere implementati **seguendo i principi della programmazione modulare** (è sufficiente un unico modulo, ma bisogna definire delle procedure o funzioni)

Esercizio 10.1 -Note

- Per utilizzare i file su **Repl.it** è necessario seguire la consueta procedura di creazione di un nuovo file, già utilizzata per la programmazione modulare
  - Chiaramente, rinominare il file in modo opportuno (es. `test.txt`)
- Su **Eclipse** creare un nuovo file (es. su Windows: Nuovo → Documento di Testo), rinominarlo opportunamente e inserirlo nel Workspace.



Esercizio 10.1 - Soluzione

```c
#include <stdio.h>
#define PASSED 18

// prototipi di funzione
void inputData(FILE *input);
void printPassed(FILE *input);

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    inputData(file); // acquisisce input
    rewind(file);
    printPassed(file); // stampa promossi
  }

  fclose(file);
}

```


Esercizio 10.1 - Soluzione

```c
#include <stdio.h>
#define PASSED 18

// prototipi di funzione
void inputData(FILE *input);
void printPassed(FILE *input);

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    inputData(file); // acquisisce input
    rewind(file);
    printPassed(file); // stampa promossi
  }

  fclose(file);
}

```

Dichiaro i prototipi di funzione per le due funzionalità richieste. **Sono delle procedure, perché non producono nessun dato.**

Esercizio 10.1 – Soluzione (cont.)

```c
void inputData(FILE *input) {
  char name[10];
  int vote;

  for(int i=0; i<5; i++) {
    printf("Inserisci nome e voto (separati da spazio): ");
    scanf("%9s%d", name, &vote); // leggo valore da tastiera

    fprintf(input, "%s\t%d\n", name, vote); // scrivo su file
  }
}

```

La funzione legge i valori in input (da tastiera) e utilizza la funzione **`fprintf()`** per scrivere su file

Esercizio 10.1 – Soluzione (cont.)

```
23  void inputData(FILE *input) {
24    char name[10];
25    int vote;
26
27    for(int i=0; i<5; i++) {
28      printf("Inserisci nome e voto (separati da spazio): ");
29      scanf("%9s%d", name, &vote); // leggo valore da tastiera
30
31      fprintf(input, "%s\t%d\n", name, vote); // scrivo su file
32    }
33  }
34
```

Preferibile usare una costante ☺

La funzione legge i valori in input (da tastiera) e utilizza la funzione **`fprintf()`** per scrivere su file

Esercizio 10.1 – Soluzione (cont.)

```
23  void inputData(FILE *input) {
24    char name[10];
25    int vote;
26
27    for(int i=0; i<5; i++) {
28      printf("Inserisci nome e voto (separati da spazio): ");
29      scanf("%9s%d", name, &vote); // leggo valore da tastiera
30
31      fprintf(input, "%s\t%d\n", name, vote); // scrivo su file
32    }
33  }
34
```

Preferibile aggiungere un controllo sul voto!

La funzione legge i valori in input (da tastiera) e utilizza la funzione **`fprintf()`** per scrivere su file

Esercizio 10.1 – Soluzione (cont.)

```
35  void printPassed(FILE *input) {
36    char name[10];
37    int vote;
38
39    puts("-----\nStudenti Promossi:\n-----");
40
41    while(!feof(input)) { // leggo dal file
42      int read = fscanf(input, "%9s%d", name, &vote);
43
44      if(read>0 && vote>PASSED)
45        printf("%s\t%d\n", name, vote); // scrivo su file
46    }
47  }
```

Finchè l'input non è terminato, legge dati dal file. Se il voto è maggiore di 18, stampa a schermo.

Esercizio 10.1 – Soluzione (cont.)

```
35  void printPassed(FILE *input) {
36    char name[10];
37    int vote;
38
39    puts("-----\nStudenti Promossi:\n-----");
40
41    while(!feof(input)) { // leggo dal file
42      int read = fscanf(input, "%9s%d", name, &vote);
43
44      if(read>0 && vote>PASSED)
45        printf("%s\t%d\n", name, vote); // scrivo su file
46    }
47  }
```

Finchè l'input non è terminato, legge dati dal file. Se il voto è maggiore di 18, stampa a schermo.

A cosa serve? Gestione dei casi limite!

Esercizio 10.1 – Soluzione (cont.)

```
35 void printPassed(FILE *input) {
36     char name[10];
37     int vote;
38
39     puts("-----\nStudenti Promossi:\n-----");
40
41     while(!feof(input)) { // leggo dal file
42       int read = fscanf(input, "%9s%d", name, &vote);
43
44       if(read>0 && vote>PASSED)
45         printf("%s\t%d\n", name, vote); // scrivo su file
46     }
47 }
```

Finchè l'input non è terminato, legge dati dal file. Se il voto è maggiore di 18, stampa a schermo.

A cosa serve? Gestione dei casi limite!
Può accadere che il file non sia ancora terminato, ma l'input non sia quello che cerchiamo quindi la `fscanf()` non va a buon fine

`fscanf()` restituisce un valore minore di 0 se non ha letto ciò che abbiamo chiesto

Esercizio 10.1 – Soluzione (cont.)

```
35 void printPassed(FILE *input) {
36     char name[10];
37     int vote;
38
39     puts("-----\nStudenti Promossi:\n-----");
40
41     while(!feof(input)) { // leggo dal file
42       int read = fscanf(input, "%9s%d", name, &vote);
43
44       if(read>0 && vote>PASSED)
45         printf("%s\t%d\n", name, vote); // scrivo su file
46     }
47 }
```

Ci vengono in mente **altri possibili prototipi** per questa procedura?

Esercizio 10.1 – Soluzione (cont.)

```
35 void printPassed(FILE *input) {
36     char name[10];
37     int vote;
38
39     puts("-----\nStudenti Promossi:\n-----");
40
41     while(!feof(input)) { // leggo dal file
42       int read = fscanf(input, "%9s%d", name, &vote);
43
44       if(read>0 && vote>PASSED)
45         printf("%s\t%d\n", name, vote); // scrivo su file
46     }
47 }
```

Ci vengono in mente **altri possibili prototipi** per questa procedura?

1) **La funzione poteva acquisire in input anche il valore PASSED come secondo parametro**
2) **La funzione poteva restituire un vettore di nomi invece che stamparli DENTRO la funzione (separazione delle competenze più corretta)**

Esercizio 10.1 – Soluzione (cont.)

```
35 void printPassed(FILE *input) {
36     char name[10];
37     int vote;
38
39     puts("-----\nStudenti Promossi:\n-----");
40
41     while(!feof(input)) { // leggo dal file
42       int read = fscanf(input, "%9s%d", name, &vote);
43
44       if(read>0 && vote>PASSED)
45         printf("%s\t%d\n", name, vote); // scrivo su file
46     }
47 }
```

Ci vengono in mente **altri possibili prototipi** per questa procedura?

1) **La funzione poteva acquisire in input anche il valore PASSED come secondo parametro**
2) **La funzione poteva restituire un vettore di nomi invece che stamparli DENTRO la funzione (separazione delle competenze più corretta)**

```
char** printPassed(FILE *input, int PASSED)
```

(prototipo alternativo! Provate a implementarlo!)

Fermare l'input in maniera elegante

```
int
main ()
{
        FILE *primo_file;
        char insegnamento[60];
        int voto;
        primo_file=fopen("Esami.txt", "a+"); // appende le informazioni alla fine del file
         printf("Il programma memorizza in un file Esami.txt le votazioni riportate \n da un singolo s
        if (primo_file==NULL)
                printf("\n\n\n********** Impossibile aprire il file********** \n\n");
        else
        {
          printf("Inserire gli insegnamenti e le rispettive votazioni riportate (Es. Programmazione 30
          scanf("%s %d", insegnamento, &voto);
           while (!feof (stdin))
           {
                fprintf(primo_file, "%s %d\n", insegnamento, voto);
                scanf("%s %d", insegnamento, &voto);
           }
           printf("File creato\n\n");
          fclose(primo_file);
        }

        system("pause");
        return(0);

}
```

```
While (!feof (stdin))
{
//corpo
}
```

Consente di terminare l'inserimento con CRTL+Z (win) CRTL+D (mac e linux)

Esercizio 10.1 - Estensione

- Quanto sarebbe complicato estendere il programma permettendo all'utente di aprire il file e modificare il voto di uno studente?

Esercizio 10.1 - Estensione

- Quanto sarebbe complicato estendere il programma permettendo all'utente di aprire il file e modificare il voto di uno studente?

Cataldo	30
Anna	15
Luigi	20
Gabriella	27
Teresa	18
Francesco	21
Maria	28

Cataldo	30
Anna	15
Luigi	20
Gabriella	27
Teresa	18
Francesco	**24**
Maria	28

Esercizio 10.1 - Estensione

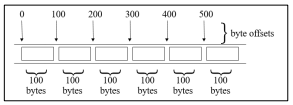
- Quanto sarebbe complicato estendere il programma permettendo all'utente di aprire il file e modificare il voto di uno studente?

Cataldo	30
Anna	15
Luigi	20
Gabriella	27
Teresa	18
Francesco	21
Maria	28

Cataldo	30
Anna	15
Luigi	20
Gabriella	27
Teresa	18
Francesco	**24**
Maria	28

Sarebbe molto complicato, perché i dati sono **memorizzati in modo sequenziale. Tipicamente si procede riscrivendo da zero il file, copiando i vecchi valori e modificando quello da aggiornare.**
Troppo oneroso per file di enormi dimensioni!

Soluzione: File Binari

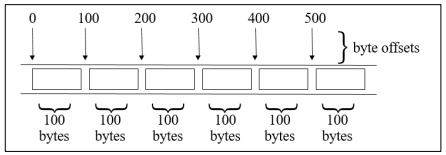


- **I file binari** risolvono il problema della sovrascrittura di contenuti che può avvenire nei file testuali
- Sono detti anche **file ad accesso casuale** perché il puntatore **non scorre sequenzialmente i contenuti**
  - L'accesso avviene puntando una **specifica locazione di memoria**
  - **Garantiscono maggiore flessibilità: i contenuti possono essere aggiornati e modificati** senza sovrascrivere quello che prima era memorizzato

File Binari (ad accesso casuale)

Pro e Contro

- Dati in un file ad accesso casuale
  - **Non formattati** (memorizzati come "raw bytes")
    - Tutti i dati delle stesso tipo (`int`, per esempio) utilizzano la stessa quantità di memoria
    - Tutti i record dello stesso tipo hanno una lunghezza fissa
    - I dati non sono human readable



Apertura File – Modalità (Recap)

	Modalità	Cosa fa
File **Testuali**	r	open a text file for **reading**
	w	truncate to zero length or **create a text file for writing**
	a	append; open or create text file for **writing at end-of-file**
	r+	**open text file for update (reading and writing)**
	w+	truncate to zero length or create a text file for update
	a+	append; open or create text file for update
File **Binari**	rb	open a binary file for **reading**
	wb	truncate to zero length or **create a binary file for writing**
	ab	append; open or create binary file for **writing at end-of-file**
	rb+	**open binary file for update (reading and writing)**
	wb+	truncate to zero length or create a binary file for update
	ab+	append; open or create binary file for update

Apertura File – Modalità (Recap)

	Modalità	Cosa fa
File **Testuali**	r	open a text file for **reading**
	w	**for writing**
	a	**t end-of-file**
	r+	**vriting)**
	w+	for update
	a+	update
File **Binari**	rb	open a binary file for **reading**
	wb	truncate to zero length or **create a binary file for writing**
	ab	append; open or create binary file for **writing at end-of-file**
	rb+	**open binary file for update (reading and writing)**
	wb+	truncate to zero length or create a binary file for update
	ab+	append; open or create binary file for update

Per lavorare sui file binari si utilizzano gli stessi metodi (astrazione!) dei file binari. **Bisogna modificare soltanto la modalità d'accesso per indicare che si sta lavorando su un file binario.**

Quali operazioni possiamo fare? (Reminder)

- **Quali operazioni possiamo fare sui file?**
  - Apertura File
  - Lettura Dati
  - Scrittura Dati
  - Chiusura File
  - «Riavvolgimento» dello Stream
  - Verifica di fine Stream
  - Collocare il puntatore su un punto preciso del file (utile soprattutto per i file binari)

Scrittura su File Binari

- `size_t fwrite(const void* ptr, size_t size, size_t nmemb, FILE* stream);`
  - `ptr` → puntatore alla variabile da «copiare» nel blocco dati
  - `size` → dimensione del blocco dati da scrivere
  - `nmemb` → numero dei blocchi di memoria da scrivere
  - `stream` → puntatore al file su cui scrivere i dati

Scrittura su File Binari

?

- `size_t fwrite(const void* ptr, size_t size, size_t nmemb, FILE* stream);`
  - `ptr` → puntatore alla variabile da «copiare» nel blocco dati
  - `size` → dimensione del blocco dati da scrivere
  - `nmemb` → numero dei blocchi di memoria da scrivere
  - `stream` → puntatore al file su cui scrivere i dati

Scrittura su File Binari

Tipo di dati «intero» senza segno, a 16 bit. Viene tipicamente utilizzato per memorizzare **le dimensioni delle variabili** o **i contatori negli cicli**.

- `size_t fwrite(const void* ptr, size_t size, size_t nmemb, FILE* stream);`
  - `ptr` → puntatore alla variabile da «copiare» nel blocco dati
  - `size` → dimensione del blocco dati da scrivere
  - `nmemb` → numero dei blocchi di memoria da scrivere
  - `stream` → puntatore al file su cui scrivere i dati

Scrittura su File Binari

- `size_t fwrite(const void* ptr, size_t size, size_t nmemb, FILE* stream);`
  - `ptr` → puntatore alla variabile da «copiare» nel blocco dati
  - `size` → dimensione del blocco dati da scrivere
  - `nmemb` → numero dei blocchi di memoria da scrivere
  - `stream` → puntatore al file su cui scrivere i dati
- **Cosa fa?**
  - Scrive nello `stream` un numero di elementi pari a `nmemb`, ognuno di dimensione `size`, attualmente memorizzati in `ptr`
  - Chiamata più complessa. **Analizziamola passo passo.**

Scrittura su File Binari

- `size_t fwrite(const void* ptr, size_t size, size_t nmemb, FILE* stream);`
  - `ptr` → puntatore alla variabile da «copiare» nel blocco dati
  - `size` → dimensione del blocco dati da scrivere
  - `nmemb` → numero dei blocchi di memoria da scrivere
  - `stream` → puntatore al file su cui scrivere i dati
- **Cosa cambia rispetto ai file sequenziali?** Cosa c'è di diverso?

Scrittura su File Binari

- `size_t fwrite(const void* ptr, size_t size, size_t nmemb, FILE* stream);`
  - `ptr` → puntatore alla variabile da «copiare» nel blocco dati
  - `size` → dimensione del blocco dati da scrivere
  - `nmemb` → numero dei blocchi di memoria da scrivere
  - `stream` → puntatore al file su cui scrivere i dati
- **Cosa cambia rispetto ai file sequenziali?** Cosa c'è di diverso?
  - `size` → dimensione del blocco dati da scrivere
  - `nmemb` → numero dei blocchi di memoria (elementi) da scrivere

Scrittura su File Binari

- `size_t fwrite(const void* ptr, size_t size, size_t nmemb, FILE* stream);`
  - `ptr` → puntatore alla variabile da «copiare» nel blocco dati
  - `size` → dimensione del blocco dati da scrivere
  - `nmemb` → numero dei blocchi di memoria da scrivere
  - `stream` → puntatore al file su cui scrivere i dati
- **Cosa cambia rispetto ai file sequenziali?** Cosa c'è di diverso?
  - `size` → dimensione del blocco dati da scrivere
  - `nmemb` → numero dei blocchi di memoria da scrivere
  - Abbiamo quindi bisogno di sapere **quanto sono «grandi» i dati** che vogliamo scrivere su file.

Scrittura su File Binari

- **Come facciamo a capire quanto sono grandi i dati?**

Scrittura su File Binari

- **Come facciamo a capire quanto sono grandi i dati?**
  - **Operatore `sizeof( )` → restituisce la dimensione di una variabile (in byte)**

```
int i = 0;
sizeof(i) == 4
```

  - Nel nostro caso, per ogni variabile da scrivere, bisogna utilizzare **`sizeof( )`** sulla variabile per spiegare al compilatore di quanta memoria abbiamo bisogno
  - Il numero dei blocchi che ci servono è invece pari al numero di elementi che vogliamo memorizzare (1 se è una variabile singola, N se è un vettore di dimensione N).

Scrittura su File Binari

- **Come facciamo a capire quanto sono grandi i dati?**
  - **Operatore `sizeof( )` → restituisce la dimensione di una variabile (in byte)**

```
int i = 0;
sizeof(i) == 4
```

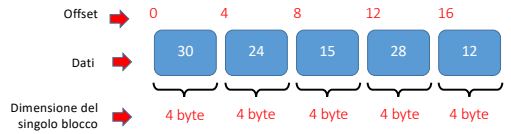
  - Nel nostro caso, per ogni variabile da scrivere, bisogna utilizzare **`sizeof( )`** sulla variabile per spiegare al compilatore di quanta memoria abbiamo bisogno
  - Il numero dei blocchi che ci servono è invece pari al numero di elementi che vogliamo memorizzare (1 se è una variabile singola, N se è un vettore di dimensione N).
- Supponendo di memorizzare in un file binario un voto d'esame, la chiamata diventa quindi
  - **`FILE* file; int vote = 30;`**
  - **`fwrite(&vote, sizeof(vote), 1, file);`**

Scrittura su File Binari

- Supponendo di memorizzare in un file binario un voto d'esame, la chiamata diventa

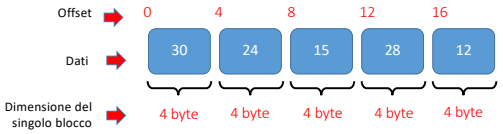
```
FILE* file; int vote = 30;
fwrite(&vote, sizeof(vote), 1, file);
```

- In questo modo ad ogni elemento che **memorizziamo viene associata una dimensione predefinita**. Supponendo di memorizzare una sequenza di cinque voti, la situazione in memoria sarà la seguente



Scrittura su File Binari

- Supponendo di memorizzare in un file binario un voto d'esame, la chiamata diventa
```
FILE* file; int vote = 30;
fwrite(&vote, sizeof(vote), 1, file);
```
- In questo modo ad ogni elemento che **memorizziamo viene associata una dimensione predefinita**. Supponendo di memorizzare una sequenza di cinque voti, la situazione in memoria sarà la seguente



Nota: i dati vengono accodati. L'ultimo elemento viene sempre memorizzato **in coda al file**

Lettura da File Binari

- `size_t fread(const void* ptr, size_t size, size_t nmemb, FILE* stream);`
  - `ptr` → puntatore alla variabile da «copiare» nel blocco dati
  - `size` → dimensione del blocco dati da scrivere
  - `nmemb` → numero dei blocchi di memoria da scrivere
  - `stream` → puntatore al file su cui scrivere i dati
- **Funzionamento** analogo a `fwrite`

Lettura da File Binari

- `size_t fread(const void* ptr, size_t size, size_t nmemb, FILE* stream);`
  - `ptr` → puntatore alla variabile che conterrà il valore
  - `size` → dimensione del blocco dati da leggere
  - `nmemb` → numero dei blocchi di memoria da leggere
  - `stream` → puntatore al file su cui leggere i dati
- **Funzionamento** analogo a `fwrite`
  - Supponiamo di voler leggere una **variabile intera da file**

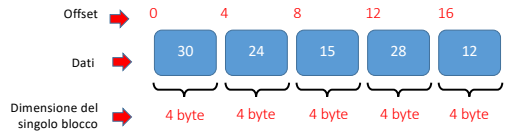
```
int value=0; FILE* file;
fread(&value, sizeof(value), 1, file)
```


Lettura da File Binari

- **Funzionamento** analogo a `fwrite`
  - Supponiamo di voler leggere una **variabile intera da file**

```
int value=0; FILE* file;
fread(&value, sizeof(value), 1, file)
```

Risultato?

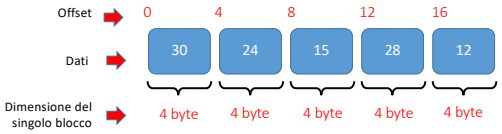


Lettura da File Binari

- **Funzionamento** analogo a **`fwrite`**
  - Supponiamo di voler leggere una **variabile intera da file**

```
int value=0; FILE* file;
fread(&value, sizeof(value), 1, file)
```

Risultato?
30



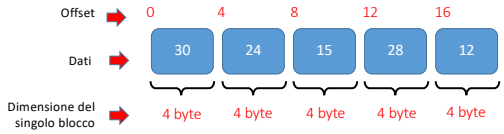
Lettura da File Binari

- **Funzionamento** analogo a **`fwrite`**
  - Supponiamo di voler leggere una **variabile intera da file**

```
int value=0; FILE* file;
fread(&value, sizeof(value), 1, file)
```

Risultato?
30

Non manca qualcosa?

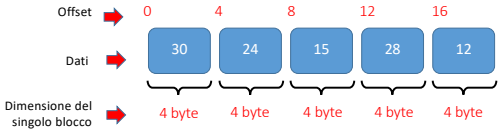


Lettura da File Binari

- **Funzionamento** analogo a **`fwrite`**
  - Supponiamo di voler leggere una **variabile intera da file**

```
int value=0; FILE* file;
fread(&value, sizeof(value), 1, file)
```

Uno dei vantaggi dei file binari è di sapere quanto spazio è allocato per ciascun dato, in modo da poter puntare direttamente a un elemento. Come facciamo?



Posizionare il puntatore

- `int fseek(FILE* stream, log int offset, int whence);`
  - `stream` → puntatore al file su cui scrivere i dati
  - `offset` → spostamento all'interno del file
  - `whence` → posizione iniziale del puntatore
    - `SEEK_SET:` dall'inizio del file
    - `SEEK_END:` dalla fine del file
    - `SEEK_CUR:` rispetto alla posizione corrente

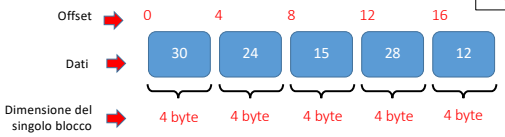
- **`fseek`** sposta il puntatore di **`offset`** byte rispetto alla posizione **`whence`** iniziale
  - I possibili valori di **`whence`** sono le tre costanti **`SEEK_SET`**, **`SEEK_END`**, **`SEEK_CUR`**

Posizionare il puntatore - Esempio

- **int fseek(FILE* stream, log int offset, int whence);**
  - stream → puntatore al file su cui scrivere i dati
  - offset → spostamento all'interno del file
  - whence → posizione iniziale del puntatore
    - SEEK_SET: dall'inizio del file
    - SEEK_END: dalla fine del file
    - SEEK_CUR: rispetto alla posizione corrente

Come faccio a puntare **il quarto elemento nel file?**

offset=?
whence=?

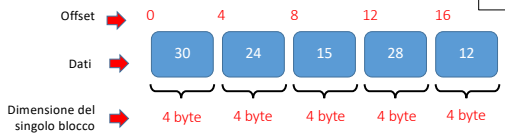


Posizionare il puntatore - Esempio

- **int fseek(FILE* stream, log int offset, int whence);**
  - stream → puntatore al file su cui scrivere i dati
  - offset → spostamento all'interno del file
  - whence → posizione iniziale del puntatore
    - SEEK_SET: dall'inizio del file
    - SEEK_END: dalla fine del file
    - SEEK_CUR: rispetto alla posizione corrente

Come faccio a puntare **il quarto elemento nel file?**

offset=12
whence=SEEK_SET

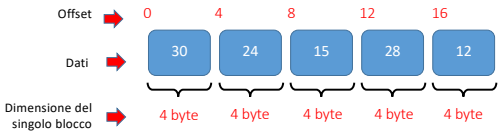


Posizionare il puntatore - Esempio

- **int fseek(FILE* stream, log int offset, int whence);**
  - stream → puntatore al file su cui scrivere i dati
  - offset → spostamento all'interno del file
  - whence → posizione iniziale del puntatore
    - SEEK_SET: dall'inizio del file
    - SEEK_END: dalla fine del file
    - SEEK_CUR: rispetto alla posizione corrente

Come faccio a puntare **il k-esimo elemento nel file?**

offset=k*(sizeof(var))



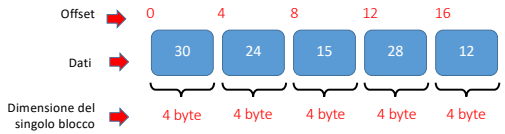
Posizionare il puntatore - Esempio

- **int fseek(FILE* stream, log int offset, int whence);**
  - stream → puntatore al file su cui scrivere i dati
  - offset → spostamento all'interno del file
  - whence → posizione iniziale del puntatore
    - SEEK_SET: dall'inizio del file
    - SEEK_END: dalla fine del file
    - SEEK_CUR: rispetto alla posizione corrente

Come faccio a puntare **il k-esimo elemento nel file?**

offset=k*(sizeof(var))

Dimensione **delle variabili allocate**



Lettura e Scrittura su File Binari (esempio)

```
1  #include <stdio.h>
2
3  int main() {
4      FILE* file;
5
6      if((file = fopen("test.dat","rb+")) == NULL) {
7          puts("Errore nell'apertura"); // errore in apertura
8      }
9      else {
10       for(int i=1; i<=5; i++) {
11         int value = i*10; //assegno valore
12
13         fwrite(&value, sizeof(i), 1, file); //scrivo
14         printf("Write:%d\n",value);
15       }
16
17       puts(""); //formattazione
```

Lavoro su **file binari**

Lettura e Scrittura su File Binari (esempio)

```
1  #include <stdio.h>
2
3  int main() {
4      FILE* file;
5
6      if((file = fopen("test.dat","rb+")) == NULL) {
7          puts("Errore nell'apertura"); // errore in apertura
8      }
9      else {
10       for(int i=1; i<=5; i++) {
11         int value = i*10; //assegno valore
12
13         fwrite(&value, sizeof(i), 1, file); //scrivo
14         printf("Write:%d\n",value);
15       }
16
17       puts(""); //formattazione
```

Definisco variabile intera e scrivo su file un blocco di 4 byte.

Operazione ripetuta cinque volte

Lettura e Scrittura su File Binari (esempio)

```
19       for(int i=0; i<5; i++) {
20         int value = 0;
21
22         fseek(file, i*sizeof(i), SEEK_SET); //posiziono puntatore
23         fread(&value, sizeof(value), 1, file); //leggo valore
24
25         printf("Read:%d\n",value); //stampo valore letto
26       }
27     }
28
29 }
```

Posiziono il **puntatore sull'i-esimo elemento** e ne leggo il valore

Lettura e Scrittura su File Binari (esempio)

```
19       for(int i=0; i<5; i++) {
20         int value = 0;
21
22         fseek(file, i*sizeof(i), SEEK_SET); //posiziono puntatore
23         fread(&value, sizeof(value), 1, file); //leggo valore
24
25         printf("Read:%d\n",value); //stampo valore letto
26       }
27     }
28
29 }
```

Posiziono il **puntatore sull'i-esimo elemento** e ne leggo il valore

Primo Ciclo → offset = 0
Secondo Ciclo → offset = 4
Terzo Ciclo → offset = 8

Importante: in esempi più complessi (ad esempio, supponiamo di memorizzare una intera `struct` in un file) , il risultato restituito dall'operatore `sizeof` può avere dimensioni molto diverse!

Appendice: elaborazione di file

- Lettura di Singoli Caratteri

```
int getc(FILE* stream);
int fgetc(FILE* stream);
int getchar(void);
```

- **getc/ fgetc** leggono il successivo carattere da uno stream (convertito in int) e avanza la posizione di un byte.
  - La funzione restituisce il carattere letto oppure la costante **EOF**, indicatrice della fine dello stream.
- **getchar = getc(stdin)**

Appendice: elaborazione di file

- Scrittura di Singoli Caratteri

```
int fputc(int c, FILE* stream);
int putc(int c, FILE* stream);
int putchar(int c);
```

- **fputc** / **putc** putscrive un carattere nello stream specificato
- **putchar = putc(stdout)**
- Restituisce il carattere scritto oppure **EOF**

Appendice: elaborazione di file

- Lettura/Scrittura di Stringhe

```
char* fgets(char* s, int n, FILE* stream);
char* gets(char* s);
int fputs(const char* s, FILE* stream);
int puts(const char* s);
```

- **fgets** legge al massimo n-1 caratteri dallo stream, memorizzandoli a partire da s. Aggiunge '\0' al termine. L'eventuale newline è inclusa. Restituisce s in caso di successo, altrimenti NULL
- **gets** legge da **stdin** fino a una newline o la fine del file. La newline viene sostituita da '\0'
- **fputs** scrive la stringa **s** nello stream. Restituisce >0 oppure EOF in caso di errore
- **puts** scrive su **stdout** e aggiunge una newline