FILE BINARI

Un file binario è una pura sequenza di byte, senza alcuna strutturazione particolare

- **È un'astrazione di memorizzazione *assolutamente generale*, usabile per memorizzare su file *informazioni di qualsiasi natura***
  - snapshot della memoria
  - rappresentazioni interne binarie di numeri
  - immagini, audio, musica, …
  - … volendo, *anche caratteri*

- I file di testo non sono indispensabili: sono semplicemente *comodi*

FILE BINARI

- **Un file binario è una sequenza di byte**
- Può essere usato per archiviare su memoria di massa ***qualunque tipo di informazione***
- Input e output avvengono sotto forma di una ***sequenza di byte***
- ***La fine del file* è SEMPRE rilevata in base all'esito delle operazioni di lettura**
  - ***non ci può essere EOF***, perché un file binario non è una sequenza di caratteri
  - *qualsiasi byte si scegliesse come marcatore*, potrebbe sempre capitare nella sequenza

FILE BINARI

Poiché un file binario è una sequenza di byte, sono fornite due funzioni per ***leggere e scrivere sequenze di byte***

- `fread()` **legge una sequenza di byte**
- `fwrite()` **scrive una sequenza di byte**

OUTPUT BINARIO: fwrite()

Sintassi:

```
int fwrite(addr, int dim, int n, FILE *f);
```

- **scrive sul file** `n` **elementi**, ognuno grande `dim` byte (complessivamente, scrive quindi n*dim byte)
- gli elementi da scrivere vengono **prelevati dalla memoria a partire dall'indirizzo** `addr`
- ***restituisce il numero di elementi (non di byte) effettivamente scritti***, che possono essere meno di n

INPUT BINARIO: fread()

Sintassi:

```
int fread(addr, int dim, int n, FILE *f);
```

- **legge dal file `n` elementi**, ognuno grande **`dim`** byte (complessivamente, *tenta di leggere* quindi n*dim byte)
- gli elementi da leggere vengono **scritti in memoria a partire dall'indirizzo *`addr`***
- ***restituisce il numero di elementi (non di byte) effettivamente letti***, che *possono essere meno di n* se il file finisce prima. ***Controllare il valore restituito è il SOLO MODO per sapere che cosa è stato letto,* e in particolare *per scoprire se il file è terminato.***

ESEMPIO 1

Salvare su un file binario **`numeri.dat`** il contenuto di un array di dieci interi

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

int main(void)
{
  FILE *fp;
  int vet[10] = {1,2,3,4,5,6,7,8,9,10};
  if ((fp = fopen("numeri.dat","wb"))==NULL)
    exit(1); /* Errore di apertura */
  fwrite(vet, sizeof(int), 10, fp);
  fclose(fp);
}
```

In alternativa:

```
fwrite(vet, 10*sizeof(int), 1, fp)
```

sizeof() è essenziale per la portabilità del sorgente: la dimensione di **int** non è fissa

ESEMPIO 2

Leggere da un file binario **`numeri.dat`** una sequenza di interi, scrivendoli in un array

```
#include <stdio.h>
#include <stdlib.h>
int main(void){
  FILE *fp;
  int vet[40], i, n;
  if ((fp = fopen("numeri.da    'rb"))==NULL)
    exit(1); /* Errore di apertura */
  n = fread(vet,sizeof(int),40,fp);
  for (i=0; i<n; i++) printf("%d ",vet[i]);
  fclose(fp);
}
```

fread tenta di leggere 40 interi, ne legge meno se il file finisce prima

n contiene il numero di interi effettivamente letti

ESEMPIO 3

Scrivere su un file di caratteri **`testo.txt`** una sequenza di caratteri

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

int main(void){
  FILE *fp; int n;
  char msg[] = "Ah, l'esame\nsi avvicina!";
  if ((fp = fopen("testo.txt","wb"))==NULL)
    exit(1); /* Errore di apertura */
  fwrite(msg, strlen(msg)+1, 1, fp);
  fclose(fp);
}
```

Dopo averlo creato, provare ad aprire questo file con un editor qualunque

Un carattere in C ha sempre **size=1**
Scelta: salvare anche terminatore stringa

Tabella ASCII standard

Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char
00000000	0	Null	00100000	32	Spc	01000000	64	@	01100000	96	`
00000001	1	Start of heading	00100001	33	!	01000001	65	A	01100001	97	a
00000010	2	Start of text	00100010	34	"	01000010	66	B	01100010	98	b
00000011	3	End of text	00100011	35	#	01000011	67	C	01100011	99	c
00000100	4	End of transmit	00100100	36	$	01000100	68	D	01100100	100	d
00000101	5	Enquiry	00100101	37	%	01000101	69	E	01100101	101	e
00000110	6	Acknowledge	00100110	38	&	01000110	70	F	01100110	102	f
00000111	7	Audible bell	00100111	39	'	01000111	71	G	01100111	103	g
00001000	8	Backspace	00101000	40	(	01001000	72	H	01101000	104	h
00001001	9	Horizontal tab	00101001	41	)	01001001	73	I	01101001	105	i
00001010	10	Line feed	00101010	42	*	01001010	74	J	01101010	106	j
00001011	11	Vertical tab	00101011	43	+	01001011	75	K	01101011	107	k
00001100	12	Form Feed	00101100	44	,	01001100	76	L	01101100	108	l
00001101	13	Carriage return	00101101	45	-	01001101	77	M	01101101	109	m
00001110	14	Shift out	00101110	46	.	01001110	78	N	01101110	110	n
00001111	15	Shift in	00101111	47	/	01001111	79	O	01101111	111	o
00010000	16	Data link escape	00110000	48	0	01010000	80	P	01110000	112	p
00010001	17	Device control 1	00110001	49	1	01010001	81	Q	01110001	113	q
00010010	18	Device control 2	00110010	50	2	01010010	82	R	01110010	114	r
00010011	19	Device control 3	00110011	51	3	01010011	83	S	01110011	115	s
00010100	20	Device control 4	00110100	52	4	01010100	84	T	01110100	116	t
00010101	21	Neg. acknowledge	00110101	53	5	01010101	85	U	01110101	117	u
00010110	22	Synchronous idle	00110110	54	6	01010110	86	V	01110110	118	v
00010111	23	End trans. block	00110111	55	7	01010111	87	W	01110111	119	w
00011000	24	Cancel	00111000	56	8	01011000	88	X	01111000	120	x
00011001	25	End of medium	00111001	57	9	01011001	89	Y	01111001	121	y
00011010	26	Substitution	00111010	58	:	01011010	90	Z	01111010	122	z
00011011	27	Escape	00111011	59	;	01011011	91	[	01111011	123	{
00011100	28	File separator	00111100	60	<	01011100	92	\	01111100	124	|
00011101	29	Group separator	00111101	61	=	01011101	93	]	01111101	125	}
00011110	30	Record Separator	00111110	62	>	01011110	94	^	01111110	126	~
00011111	31	Unit separator	00111111	63	?	01011111	95		01111111	127	Del

ESEMPIO 4: OUTPUT DI NUMERI

L'uso di file binari consente di rendere evidente la differenza fra la ***rappresentazione interna*** di un numero e la sua ***rappresentazione esterna*** come *stringa di caratteri in una certa base*

- Supponiamo che sia `int x = 31466;`
- Che differenza c'è fra:

```
fprintf(file,"%d", x);
fwrite(&x, sizeof(int), 1, file);
```

ESEMPIO 4: OUTPUT DI NUMERI

Se **`x`** è un intero che vale **`31466`**, internamente la sua rappresentazione è (su 16 bit): `01111010 11101010`

- **`fwrite()` emette direttamente tale sequenza,** scrivendo quindi i ***due byte*** sopra indicati
- **`fprintf()`** invece **emette la sequenza di caratteri ASCII** corrispondenti alla rappresentazione esterna del numero 31466, ossia i ***cinque byte***

`00110011 00110001 00110100 00110110 00110110`

Se **per sbaglio** si emettessero **su un file di testo** (o su video) direttamente i due byte: `01111010 11101010`

si otterrebbero *i caratteri corrispondenti al codice ASCII di quei byte:* **`êz`**

ESEMPIO 5: INPUT DI NUMERI

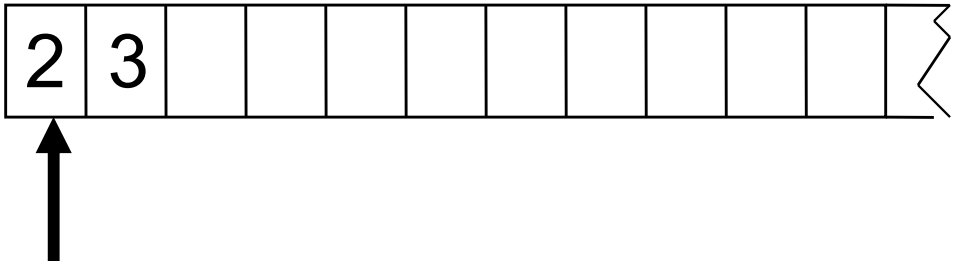
Analogamente, che differenza c'è fra

```
fscanf(file, "%d", &x);
```

e

```
fread(&x, sizeof(int), 1, file);
```

nell'ipotesi che il file (di testo) contenga la sequenza di caratteri "23"?



ESEMPIO 5: INPUT DI NUMERI

`fscanf()` preleva la **stringa di caratteri ASCII**

che costituisce la rappresentazione esterna del numero, e la **converte** nella corrispondente rappresentazione interna, ottenendo i due byte:

`00000000 00010111`

che rappresentano in binario il valore ***ventitre***

ESEMPIO 5: INPUT DI NUMERI

`fread()` invece **preleverebbe i due byte**

credendoli già la rappresentazione interna di un numero, senza fare alcuna conversione

In questo modo sarebbe inserita nella variabile x *esattamente la sequenza di byte sopra indicata*, che verrebbe quindi interpretata come il numero ***tredicimilacentosei***

ESEMPIO COMPLETO FILE BINARIO

È dato un file binario **`people.dat`** i cui record rappresentano *ciascuno i dati di una persona*, secondo il seguente formato:

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

Si noti che la ***creazione del file binario deve essere sempre fatta da programma***, mentre per i file di testo può essere fatta con un text editor (che produce sempre e solo file di testo)

CREAZIONE FILE BINARIO

È necessario scrivere un programma che lo crei strutturandolo in modo che ogni record contenga una

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

I dati di ogni persona da inserire nel file vengono richiesti all'utente tramite la funzione **`leggiel()`** che non ha parametri e restituisce come valore di ritorno la **`struct persona`** letta. Quindi il prototipo è:

```
struct persona leggiel();
```

CREAZIONE FILE BINARIO

```
struct persona leggiel(){
      struct  persona e;

      printf(”Cognome ? ");
      scanf("%s", e.cognome);
      printf("\n Nome ? ");
      scanf("%s",e.nome);
      printf("\nSesso ? ");
      scanf("%s",e.sesso);
      printf("\nAnno nascita ? ");
      scanf("%d", &e.anno);
      return e;
}
```

CREAZIONE FILE BINARIO

```
#include <stdio.h>
#include <stdlib.h>
struct persona{
 char cognome[31], nome[31], sesso[2];
 int anno;
};
struct persona leggiel();
int main(void){
FILE *f; struct persona e; int fine=0;
f=fopen("people.dat", "wb");
  while (!fine)
    { e=leggiel();
      fwrite(&e,sizeof(struct persona),1,f);
      printf("\nFine (SI=1, NO=0)?");
      scanf("%d", &fine);
    }
  fclose(f); }
```

CREAZIONE FILE BINARIO

L'esecuzione del programma precedente crea il file binario contenente i dati immessi dall'utente. Solo a questo punto il file può essere utilizzato

Il file **`people.dat`** non è visualizzabile tramite un text editor: questo sarebbe il risultato

rossi ≻▯ ÿÿ @▯ T ▯ —8▯ ▯ â3 mario▯

ESEMPIO COMPLETO FILE BINARIO

Ora si vuole scrivere un programma che

- legga record per record i dati dal file
- ponga i dati in un array di *persone*
- *… (poi svolgeremo elaborazioni su essi)*

ESEMPIO COMPLETO FILE BINARIO

1) Definire una struttura di tipo `persona`

Occorre definire una `struct` adatta a ospitare i dati elencati:

- **cognome** → array di 30+1 caratteri
- **nome** → array di 30+1 caratteri
- **sesso** → array di 1+1 caratteri
- **anno di nascita** → un intero

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

ESEMPIO COMPLETO FILE BINARIO

2) definire un array di **`struct persona`**
3) aprire il file in lettura

```
int main(void) {
 struct persona v[DIM];
 FILE* f = fopen("people.dat", "rb");
 if (f==NULL) {
  printf("Il file non esiste");
  exit(1); /* terminazione del programma */
 }
 ...
}
```

ESEMPIO COMPLETO FILE BINARIO

4) leggere i record dal file, e porre i dati di ogni persona in una cella dell'array

Come organizzare la lettura?

```
int fread(addr, int dim, int n, FILE *f);
```

- legge dal file **n** elementi, ognuno grande **dim** byte (complessivamente, legge quindi n*dim byte)
- gli elementi da leggere vengono scritti in memoria a partire dall'indirizzo ***addr***

Uso fread()

ESEMPIO COMPLETO FILE BINARIO

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

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

int main(void) {
 struct persona v[DIM]; int i=0; FILE* f;
 if ((f=fopen("people.dat", "rb"))==NULL) {
  printf("Il file non esiste!"); exit(1); }
 while(fread(&v[i],sizeof(struct persona),1,f)>0)
      i++;
}
```

ESEMPIO COMPLETO FILE BINARIO

Che cosa far leggere a `fread()`?

Se vogliamo, anche l'intero vettore di strutture: unica lettura per **`DIM`** *record (solo se sappiamo a priori che i record da leggere sono* ***esattamente*** **`DIM`***)*

```
fread(v,sizeof(struct persona),DIM,f)
```

ESEMPIO COMPLETO FILE BINARIO

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

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

int main(void) {
 struct persona v[DIM]; int i=0; FILE* f;
 if ((f=fopen("people.dat", "rb"))==NULL) {
  printf("Il file non esiste!"); exit(1); }
 fread(v,sizeof(struct persona),DIM,f);
}
```