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***
  - **"fotografie" della memoria**
  - **rappresentazioni interne binarie di numeri**
  - **immagini, canzoni campionate, ….**
  - **..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 rilevata in base *all'esito delle operazioni di lettura***
  - ***non c'è EOF,* perché un file binario non è una sequenza di caratteri**
  - **la lunghezza del file è registrata dal sistema op.**
  - ***qualsiasi byte si scegliesse come marcatore*, potrebbe sempre capitare nella sequenza!**

FILE BINARI

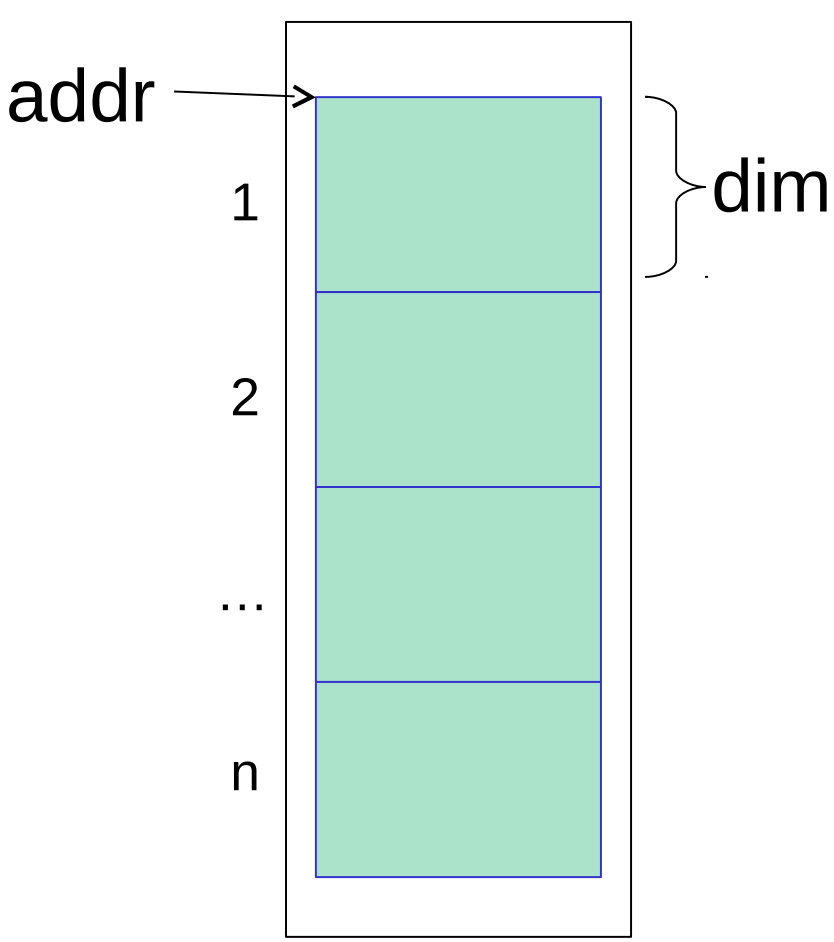
- **Poiché un file binario è una sequenza di byte, sono fornite due funzioni per *leggere* e *scrivere* sequenze di byte**
  - **`fread()` legge una sequenza di byte**
  - **`fwrite()` scrive una sequenza di byte**
- **Essendo pure sequenze di byte, esse *non sono interpretate*: l'interpretazione è "negli occhi di chi guarda".**
- **Quindi, possono rappresentare *qualunque informazione*** (testi, numeri, immagini...)

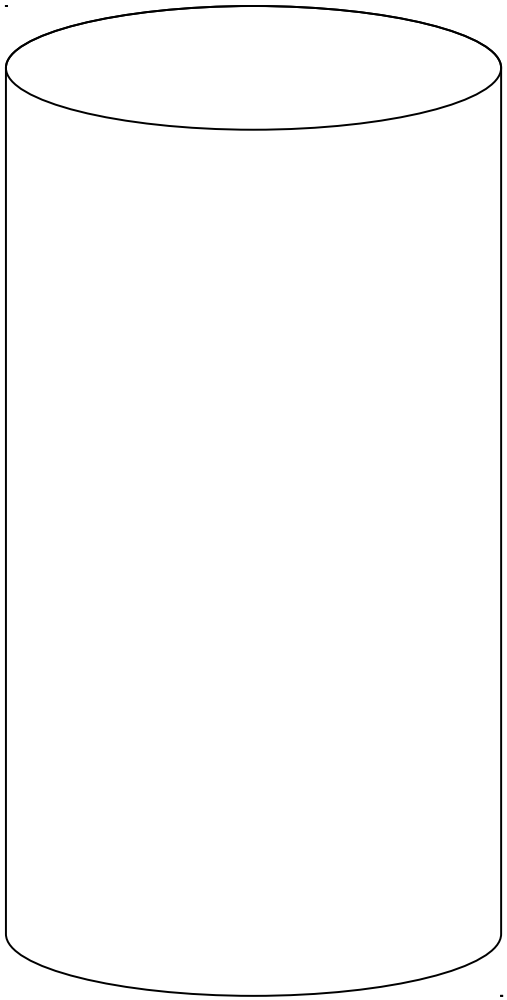
OUTPUT BINARIO: `fwrite()`

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`.**


addr
1
2
...
n
dim



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 un modo per sapere cosa è stato letto* e, in particolare, *per scoprire se il file è finito.***

ESEMPIO 1

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

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

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

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

L'operatore **`sizeof`** è essenziale per la portabilità: 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>
main()
{ FILE *fp;
  int vet[40], i, n;
  fp = fopen("numeri.dat","rb");
  if (fp==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);
}
```

n contiene il numero di interi effettivamente letti

fread tenta di leggere 40 interi, ma ne legge meno se il file finisce prima (come qui)

ESEMPIO 3

Leggere da un file di caratteri **`testo.txt`** una sequenza di caratteri, ponendoli in una stringa.

```
#include <stdio.h>
#include <stdlib.h>
main()
{ FILE *fp;
  char msg[80], n;
  fp = fopen("testo.txt","rb");
  if (fp==NULL)
    exit(1); /* Errore di apertura */
  n = fread(msg,1,80,fp);
  printf("%s",msg);
  fclose(fp);
}
```

Esperimento: provare a leggere un file (corto) creato con un editor qualunque

n contiene il numero di **char** effettivamente letti (*che non ci interessa, perché tanto c'è il terminatore ...)*

ESEMPIO 4

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

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

main()
{ FILE *fp;
  char msg[] = "Ah, l'esame\nsi avvicina!";
  fp = fopen("testo.txt","wb");
  if (fp==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 (es. blocco note).
(e il terminatore?..)

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

ESEMPIO 5: 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);
```

e

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

?

ESEMPIO 5: 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`

ESEMPIO 5: OUTPUT DI NUMERI

- 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:* **`zÛ`**

- **Niente di anche solo vagamente correlato al numero 31466 di partenza!!**

ESEMPIO 6: INPUT DI NUMERI

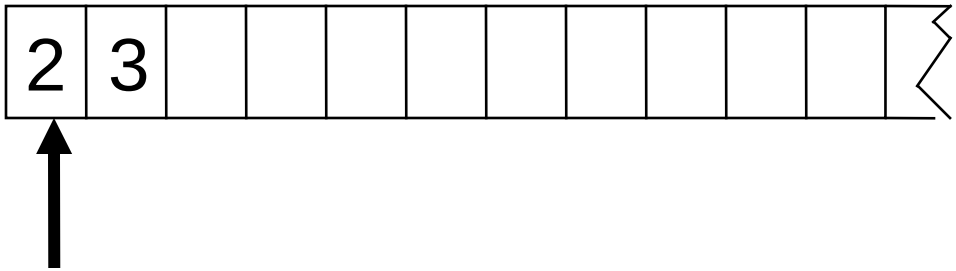
- Analogamente, che differenza c'è fra

```
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 6: 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 6: INPUT DI NUMERI

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

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

- Tale modo di agire porterebbe a inserire nella variabile **x** esattamente la sequenza di byte sopra indicata, che verrebbe quindi interpretata come il numero *dodicimilaottocentocinquantuno!*

ESEMPIO 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 fatta da programma, mentre per i file di testo può essere fatta con un text editor.

CREAZIONE FILE BINARIO

Per creare un file binario è necessario scrivere un programma che lo crei strutturandolo in modo che ogni record contenga una **`struct persona`**

```
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

Mentre la definizione è:

```
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();
main()
{  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 è il risultato

```
rossi >  ÿÿ  @ T      —8          â3  mario  ôÜ _ ôÜ Aw
 O  F  _         DÝ M                 nuinH2ô1 ô1 ô1
```

ESEMPIO COMPLETO FILE BINARIO

Ora si vuole scrivere un programma che

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

ESEMPIO COMPLETO FILE BINARIO

Come organizzarsi?

1) Definire una struttura **`persona`**

Poi, nel main:

2) Definire un array di strutture **`persona`**
3) Aprire il file in lettura
4) Leggere un record per volta, e porre i dati di quella persona in una cella dell'array

→ Servirà un indice per indicare la prossima cella libera nell'array.

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

ricordarsi lo spazio per il terminatore

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

ESEMPIO COMPLETO FILE BINARIO

Poi, nel main:

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

```
main()
{struct persona v[DIM];
 FILE* f = fopen("people.dat", "rb");
 if (f==NULL)
 {   …/* controllo che il file sia
        effettivamente aperto */
 }
 ...
}
```

Hp: massimo DIM persone

ESEMPIO COMPLETO FILE BINARIO

Poi, nel main:

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

```
main()
{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

Poi, nel main:

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

Poi, nel main:

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

---

Cosa far leggere a `fread`?

- *L'intero vettore di strutture: unica lettura per* **`DIM`** *record*

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

- *Un record alla volta all'interno di un ciclo*

```
i=0
while(fread(&v[i],sizeof(struct persona),1,f)>0)
      i++;
```

ESEMPIO COMPLETO FILE BINARIO

Poi, nel main:

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

---

Dove mettere quello che si legge?

- Abbiamo definito un array di **`struct persona`**, **`v`**
- L'indice **`k`** indica la prima cella libera → **`v[k]`**
- Tale cella è una struttura fatta di *cognome*, *nome*, *sesso*, *anno* → ciò che si estrae da un record va direttamente nella struttura **`v[k]`**

ESEMPIO COMPLETO FILE BINARIO

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

Dichiara la procedura **exit()**

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

main()
{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

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

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

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

Dichiara la procedura **exit()**

ACCESSO DIRETTO

Il C consente di gestire i file anche ad accesso diretto utilizzando una serie di funzioni della libreria standard.

La funzione **`fseek`** consente di spostare la testina di lettura su un qualunque byte del file

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

Sposta la testina di *offset* byte a partire dalla posizione *origin* (che vale 0, 1 o 2).

Se lo spostamento ha successo fornisce 0 altrimenti un numero diverso da 0

ACCESSO DIRETTO

Origine dello spostamento:

		costanti definite in **`stdio.h`**
0	inizio file	**`SEEK_SET`**
1	posizione attuale nel file	**`SEEK_CUR`**
2	fine file	**`SEEK_END`**

rewind

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

Posiziona la testina all'inizio del file

NOTA: quando si apre un file in lettura e scrittura (modalità "r+", "w+", "a+") per passare da lettura a scrittura e viceversa bisogna sempre eseguire una istruzione **`fflush`**, **`fseek`** o **`rewind`**.

ftell

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

Restituisce la posizione del byte su cui è posizionata la testina al momento della chiamata, restituisce –1 in caso di errore.

Il valore restituito dalla **ftell** può essere utilizzato in una chiamata della **fseek**

Esercizio

- Un file binario STIPENDI.DAT contiene le seguenti informazioni sui dipendenti di una ditta:
  - nome (stringa di 10 caratteri)
  - stipendio (intero)
- L'azienda ha deciso di aumentare del 10% lo stipendio dei dipendenti che prendono meno di 1000€/mese
- Si scriva un programma C che effettua tale aggiornamento

fseek (2)

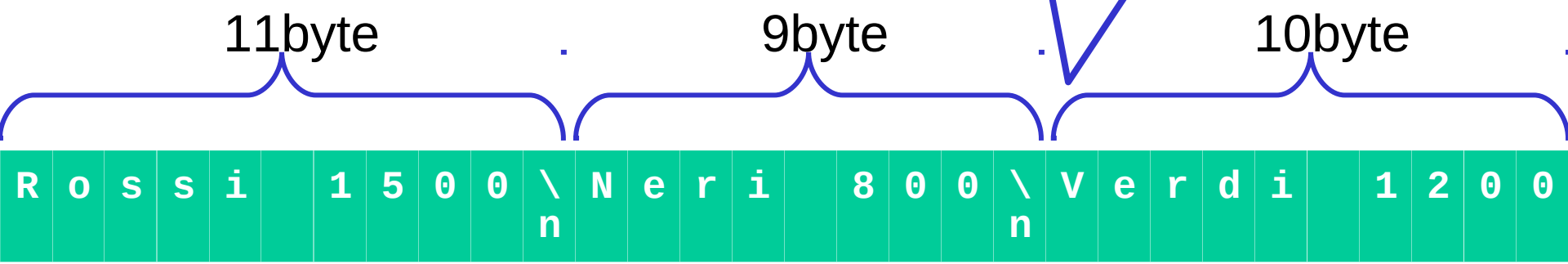
Attenzione: per file aperti in modalità testo, fseek ha un uso limitato, perché non c'è una corrispondenza tra i caratteri del file e i caratteri del testo (un "a capo" possono essere due caratteri) e quindi quando chiamiamo la fseek con un dato offset possiamo non ottenere la posizione che ci aspetteremmo

Dimensione fissa o variabile

- Nei file di testo ogni registrazione potrebbe usare un numero di byte diverso

```
Rossi 1500
Neri 800
Verdi 1200
```

Come faccio a calcolare dove comincia il terzo dato?

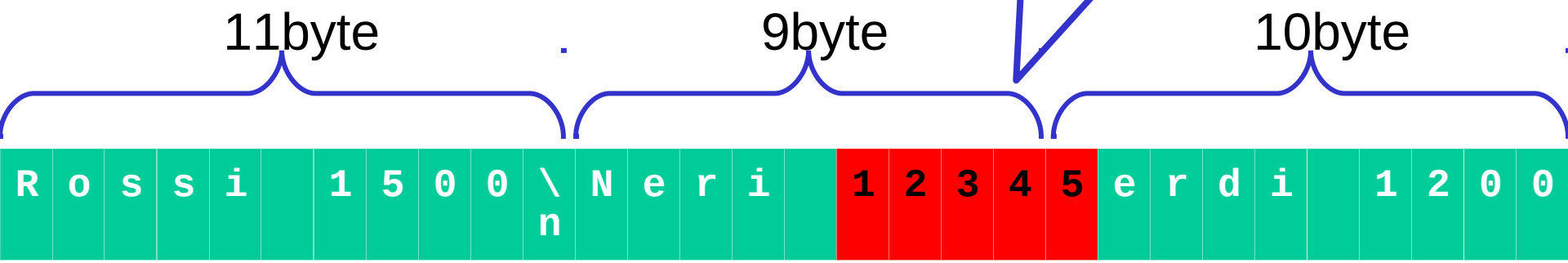


Dimensione fissa o variabile

- Nei file di testo ogni registrazione potrebbe usare un numero di byte diverso

```
Rossi 1500
Neri 12345erdi 1200
```

Se modifico la cifra di Neri potrei sovrascrivere i dati successivi!



ESEMPIO

Programma che sostituisce tutte le minuscole in maiuscole in un file testo fornito come (unico) dato di ingresso.

Specifica di I livello:

Aprire il file in modalità aggiornamento

Leggere uno alla volta i caratteri:

- se il carattere è minuscolo
  - spostare la testina indietro di una posizione
  - scrivere il carattere convertito in maiuscolo

ESEMPIO

```
#include <stdio.h>
#include <stdlib.h>
main()
{  FILE *fp;
   char nomefile[50], ch;
   printf("Nome file?"); scanf("%s",nomefile);
   if ((fp=fopen(nomefile, "r+"))==NULL)
      exit(-1);
   while(!feof(fp))
   { fscanf(fp,"%c",&ch);
     if ((ch<='z') && (ch>='a'))
     {  fseek(fp, ftell(fp)-1, SEEK_SET);
        fprintf(fp,"%c",ch+('A'-'a'));
        fseek(fp, 0, SEEK_CUR);
     }
   }
   fclose(fp);
}
```

ESEMPIO

- Il file è aperto con modalità “r+” (aggiornamento, posizione all’inizio del file)
- L’apertura di un file in aggiornamento “+” (abbinato ad uno qualunque di “r’, “w”, “a”) richiede esplicitamente che, dopo una sequenza di letture, prima di effettuare una qualunque scrittura venga utilizzata una delle funzioni di posizionamento su file (e analogamente per scritture seguite da letture).

ESEMPIO

- La fseek è utilizzata per riposizionarsi sul carattere appena letto se questo è minuscolo

```
fseek(file,ftell(file)-
      1,SEEK_SET);
```

- E’ inoltre obbligatoria per poter alternare letture e scritture su file

```
fseek(file,0,SEEK_CUR);
```

ALTRE FUNZIONI RELATIVE AI FILE

```
int ferror(FILE *fp)
```

- Controlla se è stato commesso un errore nella operazione di lettura o scrittura precedente. Restituisce il valore 0 se non c'è stato errore altrimenti un valore diverso da 0

```
void clearerr(FILE *fp)
```

- Riporta al valore di default lo stato delle informazioni (eof ed error) di fine file ed errore associate al file

Ricerca binaria su file

Esercizio:

- Un file binario INDIRIZZI.IND contiene
  - *nome* (stringa di 20 caratteri)
  - *via* (stringa di 20 caratteri)
  - *numero civico* (intero)

  per un insieme di persone.
- Il file è ordinato per il campo *nome*
- Si scriva una funzione ricorsiva che effettua la ricerca binaria sul file
- Suggerimento: Si utilizzi la fseek per posizionarsi sull'elemento corretto nel file

ESERCIZIO

- Un file binario PILOTI.DAT contiene informazioni sull'ordine di arrivo dei piloti nelle gare di Formula 1. Il file contiene:
  - Il numero di piloti **np** (int)
  - il numero di gare **ng** (int)
  - una matrice **M** 10x10 di interi
- L'elemento della matrice **M[i][j]** rappresenta la posizione in cui il pilota **i** e` arrivato nella gara **j**
- Nella matrice, solo i primi **np** x **ng** elementi sono significativi
- Si ha inoltre un file di testo PUNTEGGI.TXT, che contiene una sequenza di **k** interi. Il primo elemento e` il punteggio ottenuto quando un pilota arriva primo, il **k**-esimo il punteggio di chi arriva **k**-esimo. Se un pilota arriva oltre la **k**-esima posizione non ottiene punti.
- Si scriva un programma in linguaggio C che
  - legge i due file
  - costruisce un array che contiene, per ogni pilota, il punteggio totale alla fine del campionato, tramite una procedura o funzione.