PROGRAMMAZIONE 2: SPERIMENTAZIONI

Lezione 7 – Elaborazione di file in C

Agenda

- Introduzione
- File ad accesso sequenziale e diretto (casuale)
- File e stream in C
  - Struttura FILE
  - Funzioni di elaborazione di file
    - fopen
      - modalità di apertura dei file
    - fclose
    - fprintf
    - fscanf
    - rewind
  - Vantaggi e svantaggi dei file ad accesso sequenziale
  - File ad accesso diretto (casuale)
    - Creazione di un file ad accesso diretto con fwrite
    - Raw Data
    - Lettura di dati da un file ad accesso diretto con fread
    - Scrittura e lettura di array con fwrite e fread
    - Collocare il puntatore di posizione del file con fseek
    - Esempio

Video lezioni disponibili su YouTube

Introduzione

- La memorizzazione dei dati in tutte le tipologie di variabili studiate sinora è **temporanea**: tali dati vengono **perduti** quando un programma termina.
- I **file** vengono usati per la memorizzazione persistente e a lungo termine dei dati.
- I dispositivi memorizzano file su **memorie secondarie** (di **massa**): hard-disk, ssd, sd, dvd, ecc...

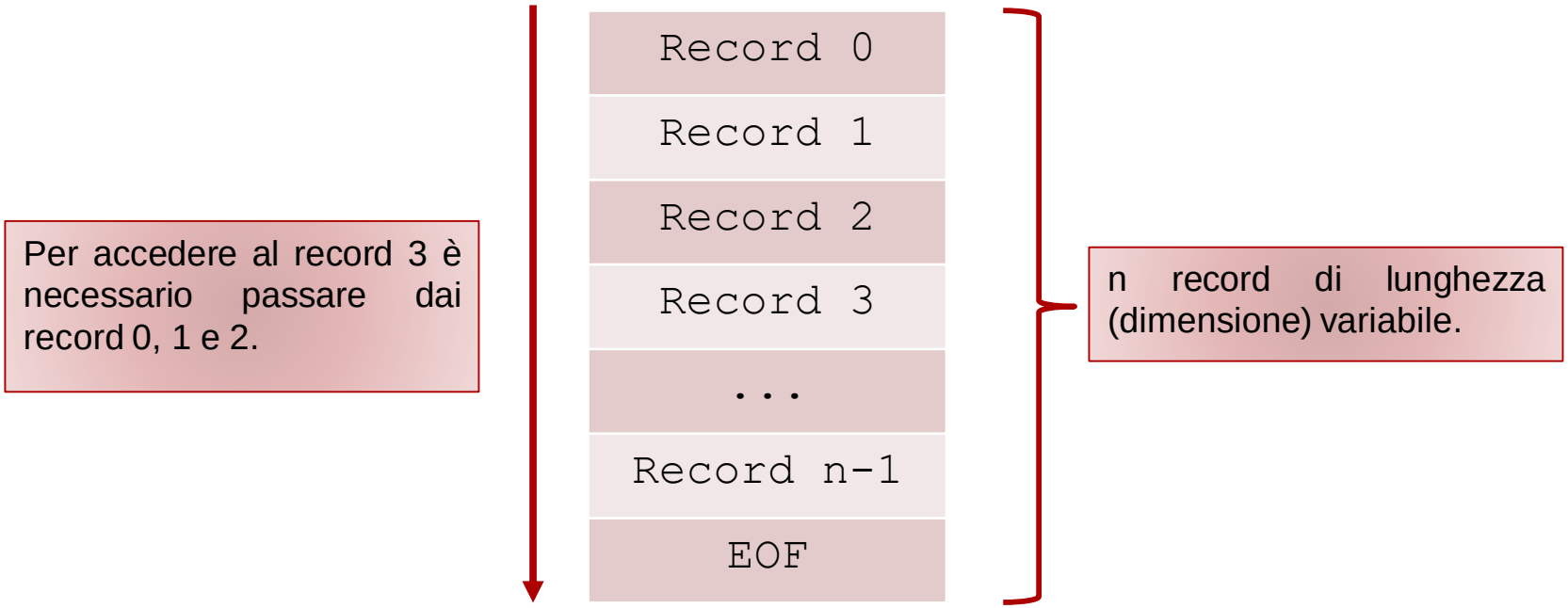
File ad accesso sequenziale e diretto (casuale)

- In questa lezione si studierà la gestione tramite C dei file ad **accesso sequenziale** e ad **accesso diretto** (o **casuale**).
- Una trattazione approfondita sui file stessi e sulla loro gestione tramite file system si vedrà nei corsi di Sistemi Operativi.

File ad accesso sequenziale e diretto (casuale)

- I file ad **accesso sequenziale** contengono record di dati a lunghezza (dimensione) variabile.
- L'accesso ad uno specifico record diventa possibile solo dopo una scansione dell'archivio, ossia leggendo tutti i record che lo precedono partendo dal primo.

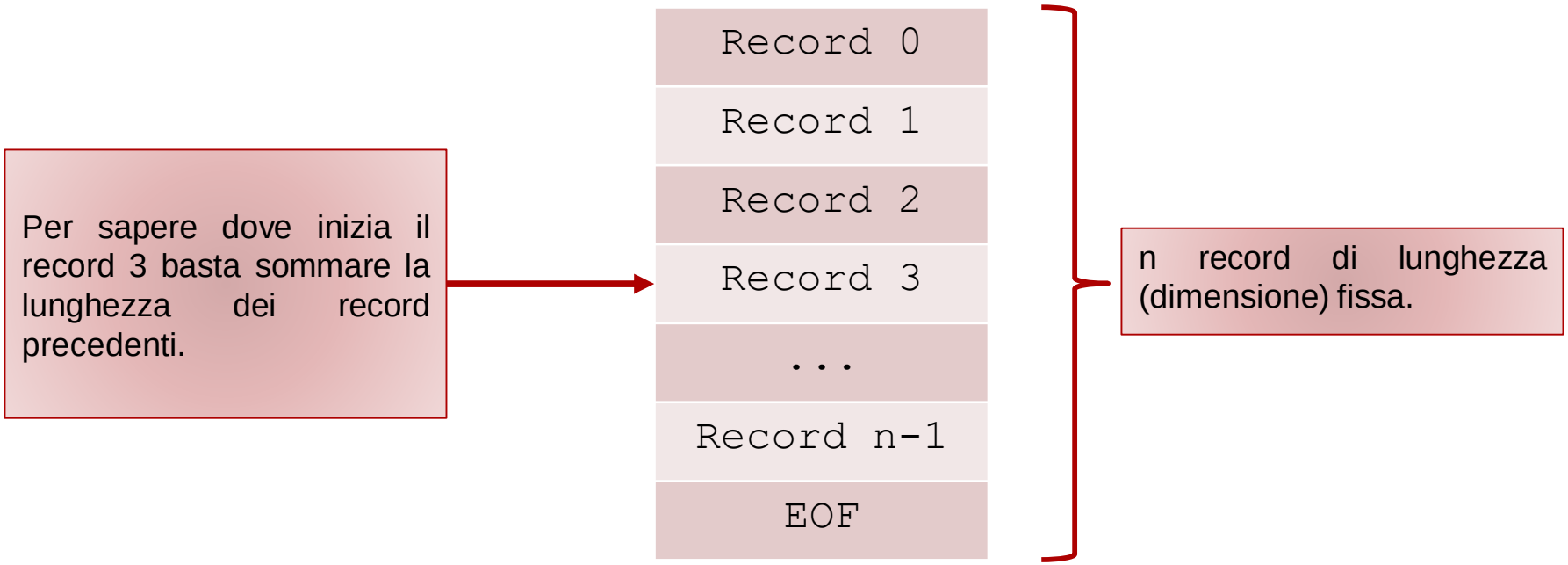
File ad accesso sequenziale e diretto (casuale)



File ad accesso sequenziale e diretto (casuale)

- I file ad **accesso diretto (casuale)** contengono record di dati a lunghezza (dimensione) fissa (costante).
- Quella ad accesso diretto (casuale) è un tipo di organizzazione in cui **ciascun record viene identificato dalla posizione occupata all'interno della sequenza di record del file**.
  - La posizione è determinata sommando le lunghezze dei record precedenti.

File ad accesso sequenziale e diretto (casuale)



File e stream

- Il linguaggio C vede ogni file come uno **stream** (**flusso**) sequenziale di byte.
- Ogni file termina con un marcatore di **end-of-file** (**eof**).

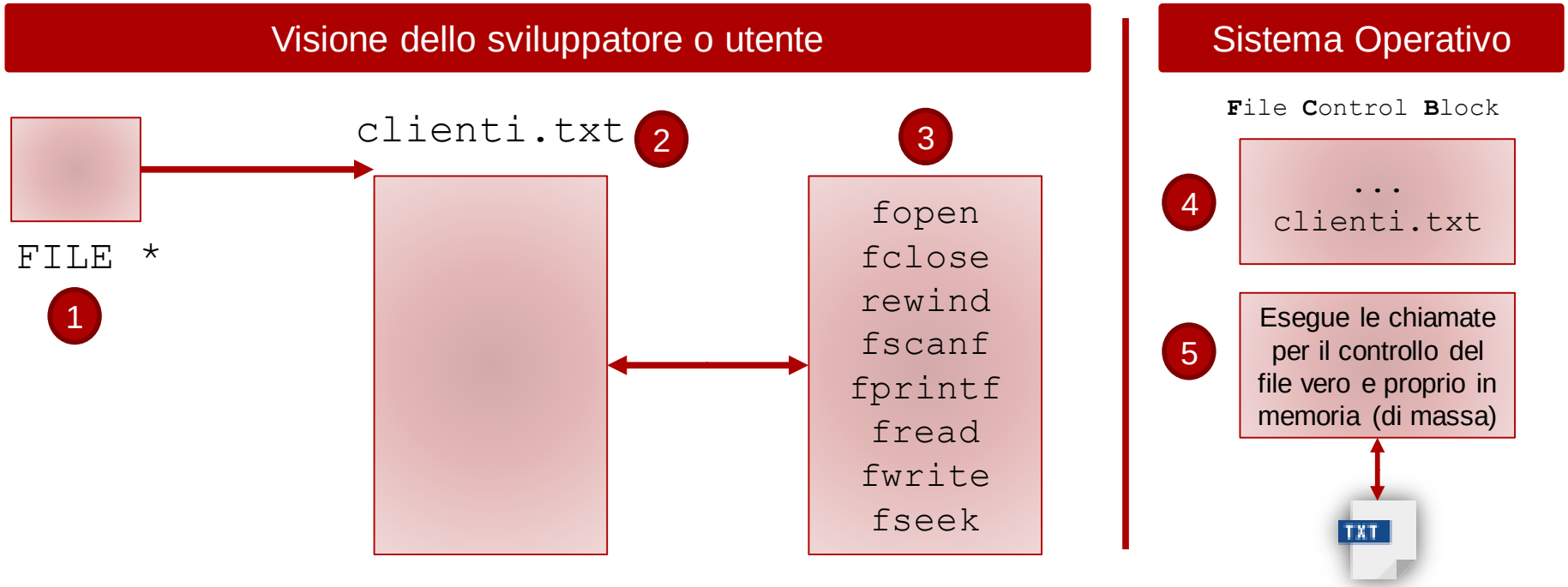
Record 0	Record 1	...	EOF
0	1	n-1	n

File e stream

- In generale, quando si avvia un programma in C, vengono aperti automaticamente **tre stream**:
  1) lo standard **input** (**stdin**);
  2) lo standard **output** (**stdout**);
  3) lo standard **error** (**stderr**) -utilizzato per i messaggi d'errore a video-.
- Gli stream forniscono canali di comunicazione tra file e programmi.
- Ad esempio, lo stream **stdin** permette ad un programma di leggere dati dalla tastiera, come lo **stdout** permette di visualizzare dati a video.

Struttura FILE

- L'apertura di un file restituisce un **puntatore a una struttura** `FILE` (definita in `<stdio.h>`), contenente informazioni usate per elaborare il file.



Struttura FILE

- Ogni programma in C amministra i file con una struttura `FILE`.
- Ogni file aperto **deve** avere un puntatore di tipo `FILE` dichiarato separatamente.

```
#include <stdio.h>

int main(void)
{
  FILE *fPtr1, *fPtr2;
  ...
  ...
}
```

Funzioni di elaborazione di file

- La libreria `<stdio.h>` fornisce varie funzioni per gestire i file. In questa lezione si vedranno:
  1) `fopen` (per accesso sequenziale e diretto);
  2) `fclose` (per accesso sequenziale e diretto);
  3) `rewind` (per accesso sequenziale e diretto);
  4) `fscanf` (per accesso sequenziale);
  5) `fprintf` (per accesso sequenziale);
  6) `fread` (per accesso diretto);
  7) `fwrite` (per accesso diretto);
  8) `fseek` (per accesso diretto).

Funzione `fopen`

- La funzione `fopen` (definita in `<stdio.h>`) stabilisce una linea di comunicazione con un file.
  - `FILE *fopen(const char *filename, const char *mode).`
- Solitamente si indica che `fopen` "apre il file".
- La `fopen` ha **due** argomenti:
  - nome del file (o percorso al file) in formato stringa;
  - modalità di apertura del file in formato stringa.
- La `fopen` restituisce un puntatore a file.
  - Il puntatore restituito può essere `NULL` **in caso di impossibilità ad aprire il file** (accesso non consentito, percorso errato).

Funzione fopen

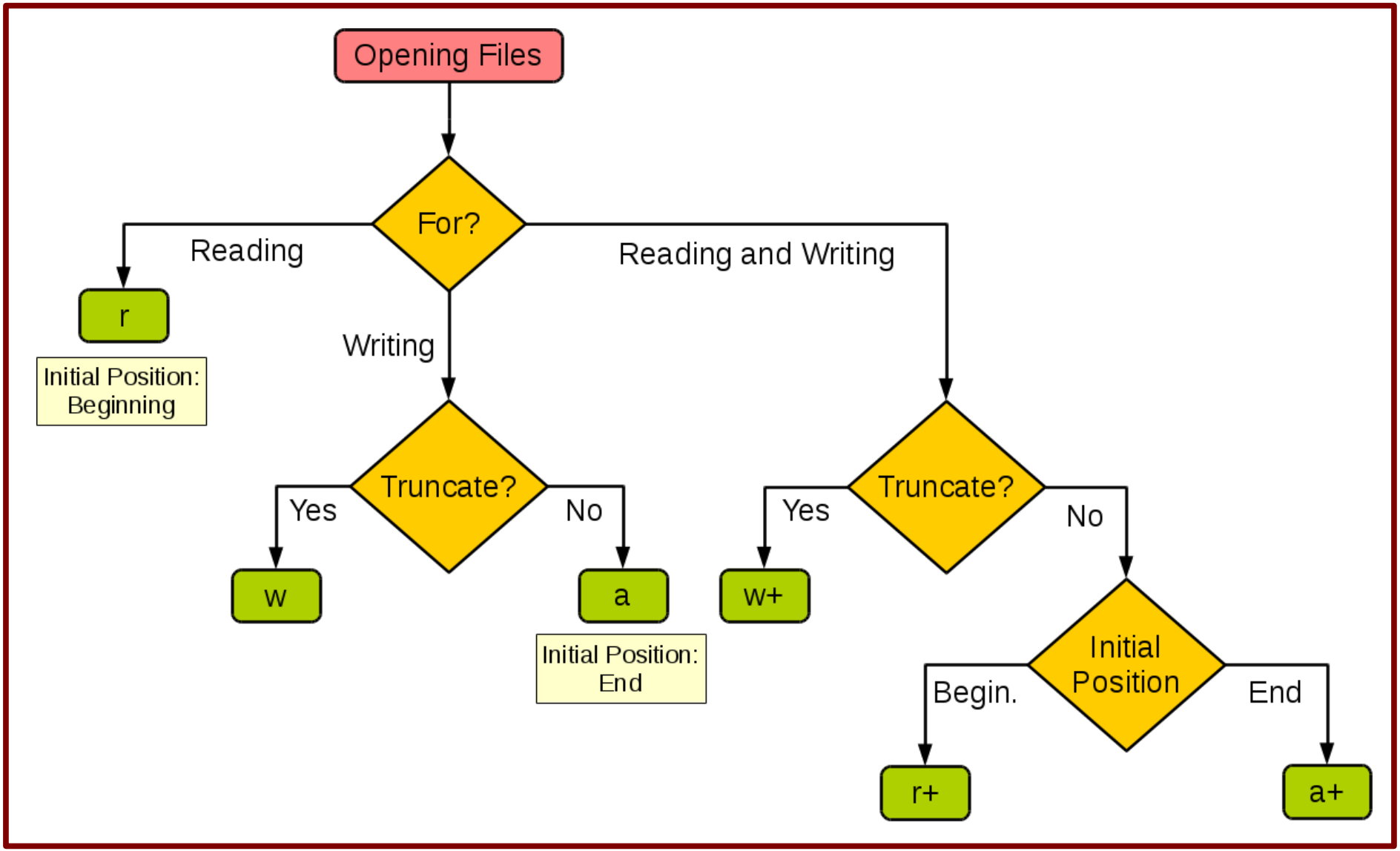
```
#include <stdio.h>

int main(void)
{
  FILE *fPtr1, *fPtr2;
  fPtr1 = fopen("test1.txt", "w");
  fPtr2 = fopen("C:\data\test1.txt", "r");
  if (fPtr1!=NULL){
        // istruzioni se il file è stato aperto correttamente
        ...
  }
  if (fPtr2!=NULL){
        // istruzioni se il file è stato aperto correttamente
        ...
  }
}
```

Modalità di apertura dei file

- I file possono essere aperti secondo diverse modalità.
- Ciascuna modalità ha una corrispondente modalità **binaria** (contenente la lettera `b`) per i file ad accesso diretto (casuale).
- I metodi più importanti:

Modalità	Descrizione
`r`	Apre un file già esistente in lettura.
`w`	Crea un file per la scrittura. *Se il file esiste già elimina i contenuti già esistenti, altrimenti lo crea vuoto.*
`a`	Apre o crea un file per scrivere dati alla fine del file (le operazioni di scrittura aggiungono dati in fondo al file).
`r+`	Apre un file in lettura e scrittura (il file deve essere già esistente).
`w+`	Apre un file in lettura e scrittura. *Se il file esiste già elimina i contenuti già esistenti, altrimenti lo crea vuoto.*
`a+`	Apre o crea un file per leggere scrivere dati alla fine del file (le operazioni di scrittura aggiungono dati in fondo al file).

Modalità di apertura dei file



Modalità di apertura dei file

- Aprire in lettura un file inesistente (o con percorso errato) è un errore che restituisce un puntatore a `NULL`.
- Aprire in lettura o scrittura un file senza che si abbiano i diritti opportuni (dipendenti dal sistema operativo), è un errore che restituisce un puntatore a `NULL`.
- Scrivere un file quando non c'è spazio disponibile sulla memoria di massa comporta un errore in fase di esecuzione.
- Aprire un file in modalità di scrittura (`"w"`) provoca l'eliminazione di tutti i contenuti già presenti nel file.
- È consigliato aprire un file solo in lettura se i suoi contenuti non devono essere modificati; ciò previene le modifiche non intenzionali e migliora le prestazioni del programma.

Funzione `fclose`

- La funzione `fclose` (definita in `<stdio.h>`) riceve come argomento il puntatore al file e *chiude* l'accesso al file da parte del programma.
  - `int fclose(FILE *stream);`
- Se la funzione `fclose` non viene esplicitamente invocata durante l'esecuzione del programma, il sistema operativo *chiuderà* l'accesso al file al termine del programma.

La chiusura di un file può liberare risorse che altri utenti o programmi stanno aspettando, quindi è sempre bene chiudere il file quando non è più necessario.

Funzione `fprintf`

- La funzione `fprintf` (definita in `<stdio.h>`) è equivalente alla `printf`, ma la `fprintf` riceve come primo argomento anche un puntatore a file in cui scrivere i dati.
  - `int fprintf(FILE *stream, const char *format, ...).`
- La funzione restituisce il numero di caratteri scritti sul file al termine della scrittura.
- Ad ogni esecuzione, dopo la scrittura il puntatore si sposta alla linea (o posizione) successiva.

```
do
    {
      printf("Inserire ID del conto (-1 per terminare): ");
      scanf("%d", &idconto);
      if (idconto!=-1)
      {
        printf("Inserire saldo: ");
        scanf("%lf", &saldo);
        fprintf(fPtr,"%d %.2f\n", idconto, saldo);
      }
    }
while(idconto!=-1);
```

Esempio 1

es1.c

```c
int main(void)
{
  // puntatore al file
  FILE *fPtr;
  // dati da inserire nel FILE
  unsigned int idconto;
  double saldo;

  // crea il file "test.txt" nella stessa cartella dove è eseguito il programma
  fPtr = fopen("es1.txt","w");

  // se ha potuto creare il file
  if (fPtr!=NULL)
  {
    do
    {
      printf("Inserire ID del conto (0 per terminare): ");
      scanf("%d", &idconto);
      if (idconto>0)
      {
        printf("Inserire saldo: ");
        scanf("%lf", &saldo);
        int n = fprintf(fPtr,"%d %.2f\n", idconto, saldo);
        printf("Caratteri scritti su file: %d\n", n);
      }
    }
    while(idconto!=0);

    // chiude il file puntato
    fclose(fPtr);
  }
  else
  {
    printf("Impossibile accedere al file");
  }

  return 1;
}
```

Funzione `fscanf`

- La funzione `fscanf` (definita in `<stdio.h>`) è equivalente alla `scanf`, ma la `fscanf` riceve come primo argomento anche un puntatore a file da cui leggere i dati.
- `int fscanf(FILE *stream, const char *format, ...).`
- La funzione restituisce il numero di elementi letti e scritti nelle variabili, 0 in caso di errore.
- Ad ogni esecuzione, la funzione legge una riga dal file, salva i dati letti nelle relative variabili secondo il formato specificato e sposta il puntatore alla riga (o posizione) successiva.

Esempio 2

es2.c

```c
int main(void)
{
  // puntatore al file
  FILE *fPtr;
  // dati da estrarre dal FILE
  unsigned int idconto;
  double saldo;

  int n;

  // apre il file in lettura nella stessa cartella dove è eseguito il programma
  fPtr = fopen("es2.txt","r");

  // se ha potuto creare il file
  if (fPtr!=NULL)
  {
    // finché non viene raggiunta la fine del file
    while (!feof(fPtr))
    {
      // estrae i dati da una riga di dati dal file e li salva in due variabili
      n = fscanf(fPtr,"%d %lf",&idconto, &saldo);
      printf("Valori letti: %d \n", n);
      printf("ID conto: %d, Saldo: %.2f \n", idconto, saldo);
    }
    // chiude il file puntato
    fclose(fPtr);
  }
  else
  {
    printf("Impossibile accedere al file");
  }

  return 1;
}
```

Funzione `rewind`

- Dopo aver attraversato tramite un puntatore tutti i record (dati) di un file, potrebbe essere necessario ricominciare a scorrere il file per una seconda funzionalità.
- In tal caso, potrebbe essere necessario riposizionare il puntatore all'inizio del file.
- La funzione `rewind()` riceve come parametro un puntatore a file e lo riposiziona all'inizio del file puntato (cioè al byte 0).
  - Il puntatore di posizione del file non è realmente un puntatore, bensì un valore intero che specifica il byte nel file oggetto della successiva lettura/scrittura. Questo valore è detto **file offset**.

Esempio 3

es3.c

```
23    if (fPtr!=NULL)
24    {
25      // finché non viene raggiunta la fine del file
26      while (!feof(fPtr))
27      {
28        // estrae i dati da una riga di dati (record) dal file e li salva in due variabili
29        n = fscanf(fPtr,"%d %lf",&idconto, &saldo);
30        // printf("Valori letti: %d \n", n);
31        printf("ID conto: %d, Saldo: %.2f \n", idconto, saldo);
32      }
33
34      // ricerca di un dato in un file sequenziale
35      printf("Inserire l'ID del conto: ");
36      scanf("%d", &idc);
37      // riporta il puntatore alla prima riga del file
38      rewind(fPtr);
39      c = 0;
40      while (!feof(fPtr))
41      {
42        // estrae i dati da una riga di dati (record) dal file e li salva in due variabili
43        fscanf(fPtr,"%d %lf",&idconto, &saldo);
44        // se l'id del conto corriposnde a quello trovato visualizza il saldo
45        if (idconto==idc)
46        {
47          printf("ID conto: %d, Saldo: %.2f \n", idconto, saldo);
48          c++;
49          break; // interrompe la ricerca
50        }
51      }
```

Vantaggi e svantaggi dei file ad accesso sequenziale

- L'organizzazione di un archivio più semplice da realizzare è **quella sequenziale**; tale organizzazione non prevede alcun meccanismo che permetta di risalire direttamente alla posizione di un record, pertanto non fornisce la possibilità di un accesso diretto ad un record specifico.
- L'accesso in lettura ad uno specifico record diventa possibile **solo dopo una scansione dell'archivio**, ossia leggendo tutti i record che lo precedono partendo dal primo.
- Per quanto riguarda l'inserimento di un nuovo record, invece, l'organizzazione sequenziale prevede che possa avvenire solo in aggiunta (append), cioè accodando il nuovo record a quelli già presenti nel file, o in riscrittura, cioè scrivendolo in prima posizione con la perdita di tutti i dati già presenti.
- L'organizzazione sequenziale si dimostra indicata essenzialmente in quei casi in cui i record dell'archivio debbano essere utilizzati frequentemente nello stesso ordine in cui sono inseriti nell'archivio stesso, ovvero quando si debba fare per l'appunto un'elaborazione sequenziale dei record.
- Può essere questo il caso, per esempio, dei file di testo che possono essere pensati come un flusso di caratteri (byte stream) in cui la fine di ciascun record, ossia una riga di testo (che è di lunghezza variabile), è segnalata al file system con un'opportuna sequenza di caratteri (per esempio `CR+LF`, nel caso dei sistemi Windows).
- L'organizzazione sequenziale, così come avviene nel caso di un file di testo, presenta il vantaggio di rendere possibile **l'uso di record a lunghezza variabile**; per contro, però, permette di disporre **solo di un metodo di accesso sequenziale ai record**.
- **Essa pertanto è svantaggiosa con le applicazioni in cui sia fondamentale rendere minimi i tempi di risposta e, quindi, per questa sua inefficienza, non è adatta per applicazioni di tipo interattivo.**

File ad accesso diretto (casuale)

- Come precedentemente affermato, i record creati in un file ad accesso sequenziale formattati con la `fprintf` **non sono necessariamente della stessa lunghezza**.
- In un'organizzazione su file ad **accesso diretto** (o **casuale**) **i record sono di lunghezza fissa** quindi vi si può accedere direttamente (e velocemente) senza effettuare ricerche in altri record.
  - Questo rende i file ad accesso diretto adatti a varie applicazioni software.
- Anche se sono presenti varie soluzioni, in questa lezione l'analisi si limiterà a file ad accesso diretto con record di dimensione (lunghezza) fissa.

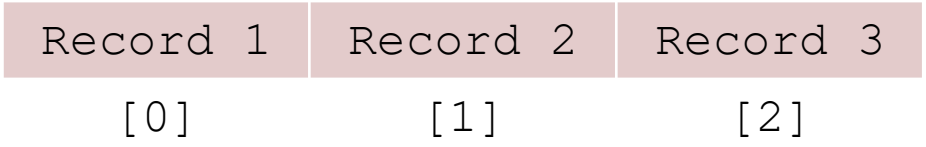
File ad accesso diretto (casuale)

- **Poiché ogni record in un file ad accesso casuale ha la stessa lunghezza** (**dimensione in byte**), la posizione esatta di un record rispetto all'inizio può essere calcolata in funzione del record.
- Si vedrà quindi che tale calcolo facilita l'accesso immediato a record anche in file di grandi dimensioni.

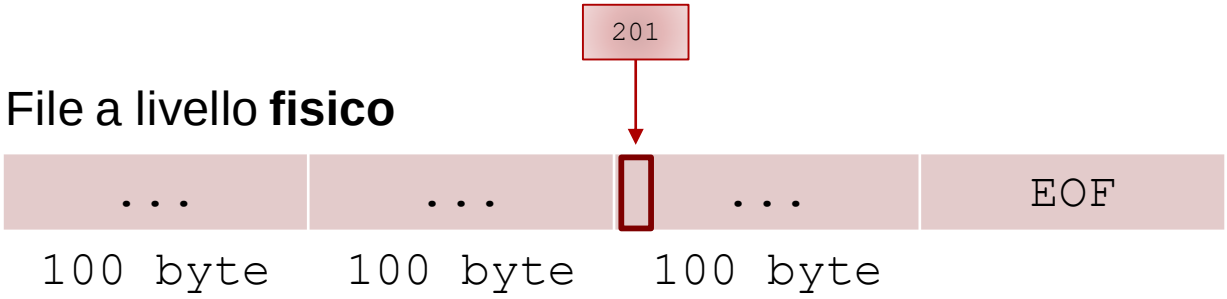
File ad accesso diretto (casuale)

- Supponendo ad esempio di memorizzare record grandi 100 byte l'uno, è possibile determinare la posizione di ognuno spostandosi di un determinato **offset**.

File a livello **logico**



File a livello **fisico**



N = numero del record a cui si vuole accedere.

L = lunghezza (dimensione) di ogni record.

Offset = (N-1)*L+1.

Es.:

Dimensione dei record: 100 byte

Record a cui si vuole accedere: 3

N = 3

L = 100

Offset = (3-1)*100+1 → **201**

File ad accesso diretto (casuale)

- I record di lunghezza (dimensione) fissa permettono quindi di inserire dati in un file ad accesso casuale senza perdere altri dati del file.
- È anche possibile aggiornare o cancellare i dati memorizzati in precedenza senza riscrivere l'intero file.
- Si vedrà quindi come:
  - creare un file ad accesso diretto;
  - inserire dati;
  - leggere i dati sia in modo sequenziale che diretto;
  - aggiornare i dati;
  - cancellare i dati non più necessari.

Creazione di un file ad accesso diretto con fwrite

- Dopo aver creato il file con la `fopen` in modalità **binaria**, è necessario scrivere i dati sul file tramite la funzione `fwrite`, definita in `<stdio.h>` come:

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream).
```

- La funzione `fwrite` trasferisce un numero specificato di byte da una variabile a un file iniziando la scrittura da una data posizione indicata dal puntatore al file.
- I parametri della `fwrite` sono:
  - l'indirizzo di memoria della variabile da scrivere;
  - la dimensione del tipo di dato da scrivere (determinato con `sizeof`);
  - il numero di elementi da scrivere (di default 1);
  - il puntatore al file su cui scrivere (nonché la posizione).

Creazione di un file ad accesso diretto con fwrite

- La funzione `fwrite` restituisce il numero di elementi scritti nel file.
  - Se il numero restituito è 0 oppure è minore del terzo parametro (ovvero quello del numero di elementi da inserire), la scrittura non è andata a buon fine.
  - **Dopo** l'esecuzione della `fwrite`, il puntatore al file viene posizionato **dopo** il record scritto.

```
if (fPtr!=NULL)
  {
    for(i=1;i <= 20; i++)
    {
      n = fwrite(&i, sizeof(int), 1, fPtr);
      printf("Dati scritti su file: %d \n", n);
    }
    fclose(fPtr);
  }
```

Creazione di un file ad accesso diretto con fwrite

es4.1.c

```
14    // apre il file ad accesso casuale (attenzione alla b)
15    fPtr = fopen(f,"wb");
16
17    // se ha potuto accedere al file
18    if (fPtr!=NULL)
19    {
20      for(i=1;i <= 20; i++)
21      {
22        // fwrite scrive il dato contenuto all'indirizzo di i
23        // essendo i intero scrive una numero di byte determinato dal sizeof(int)
24        // 1 indica di scrivere un dato solo
25        // fPtr indica file e posizione in cui scrivere
26        n = fwrite(&i, sizeof(int), 1, fPtr);
27        printf("Dati scritti su file: %d \n", n);
28      }
29      // chiude il file puntato
30      fclose(fPtr);
31    }
```

Raw Data

- Sebbene `fwrite` possa scrivere numeri, caratteri, stringhe, strutture, ecc... i dati trattati sono elaborati nel formato da computer **"raw data"** (cioè byte di dati) invece che nel formato testuale visto con fprintf e fscanf.
- **Poiché la rappresentazione "raw" ("grezza") dei dati è dipendente dal sistema, i dati grezzi potrebbero non essere leggibili su altri sistemi o da programmi prodotti con altri compilatori.**

Lettura di dati da un file ad accesso diretto con fread

- Dopo aver aperto il file con la `fopen`, è possibile leggere i dati dal file tramite la funzione `fread`, definita in `<stdio.h>` come:

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream).
```

- La funzione `fread` trasferisce un numero specificato di byte da un file ad una variabile iniziando la lettura da una data posizione indicata dal puntatore al file.
- I parametri della `fread` sono:
  - l'indirizzo di memoria della variabile in cui scrivere i dati letti;
  - la dimensione del tipo di dato da leggere (determinato con `sizeof`);
  - il numero di elementi da leggere (di default 1);
  - il puntatore al file da cui leggere (nonché la posizione).

Lettura di dati da un file ad accesso diretto con fread

- La funzione `fread` restituisce il numero di elementi letti dal file.
  - Se il numero restituito è 0 oppure è minore del terzo parametro (ovvero quello del numero di elementi da leggere), la lettura non è andata a buon fine.
- **Dopo** l'esecuzione della `fread`, il puntatore al file viene posizionato **dopo** il record letto.

```
if (fPtr!=NULL)
{
    while(!feof(fPtr))
    {
      n = fread(&x, sizeof(int), 1, fPtr);
      printf("Dati letti dal file: %d --> %d \n", n, x);
    }
    fclose(fPtr);
}
```

Lettura di dati da un file ad accesso diretto con fread

es4.2.c

```
14    // apre il file ad accesso diretto (attenzione alla b)
15    fPtr = fopen(f,"rb");
16
17    // se ha potuto accedere al file
18    if (fPtr!=NULL)
19    {
20      while(!feof(fPtr))
21      {
22        // fread legge il dato puntato da fPtr e lo salva all'indirizzo di memoria di x
23        // essendo x intero legge una numero di byte determinato dal sizeof(int)
24        // 1 indica di leggere un dato solo
25        n = fread(&x, sizeof(int), 1, fPtr);
26        printf("Dati letti dal file: %d --> %d \n", n, x);
27      }
28      // chiude il file puntato
29      fclose(fPtr);
30    }
```

Scrittura e lettura di array con fwrite e fread

- È possibile scrivere e leggere più elementi di un array di dimensione fissa fornendo alla `fwrite` e alla `fread` il puntatore al vettore e indicando il numero di elementi da scrivere / leggere.
- Dato, ad esempio, un array di interi `v` di cinque elementi, scrivendo `fwrite(&v, sizeof(int), 2, fPtr)` il programma andrà a scrivere i primi due elementi del vettore (`[0]` e `[1]`) nel file.

```
int v[]={10, 20, 30, 40, 50};
...
fPtr = fopen(f,"wb");
if (fPtr!=NULL)
{
        n = fwrite(&v, sizeof(int), 2, fPtr);
        // si può omettere la &
        // scrive su file 10 e 20
}
...
```

Scrittura e lettura di array con fwrite e fread

- È possibile scrivere e leggere più elementi di un array di dimensione fissa fornendo alla `fwrite` e alla `fread` il puntatore al vettore e indicando il numero di elementi da scrivere / leggere.
- Dato ad esempio un array di interi `v`, scrivendo `fread(&v, sizeof(int), 2, fPtr)` il programma andrà a leggere due interi dal file e salverà i relativi valori nelle prime due posizioni del vettore (`[0]` e `[1]`).

```
while (!feof(fPtr))
{
          // fread legge 2 interi e li salva nel vettore
          n = fread(&v, sizeof(int), 2, fPtr);
          // se al passaggio successivo ci sono meno di due interi legge quel che rimane (1 o 0)
          if (n!=0)
          {
            printf("Dati letti da file: %d\n", n);
            for(unsigned int i = 0; i<n; i++)
            {
               printf("Dato: %d \n", v[i]);
            }
        }
}
```

Scrittura e lettura di array con fwrite e fread

es5.1.c

```
 9    char *f = "es5.txt";
10    int v[]={10, 20, 30, 40, 50}; // [0] --> 10, [1] --> 20, ecc...
11
12    int n;
13
14    // apre il file
15    fPtr = fopen(f,"wb");
16
17    // se ha potuto accedere al file
18    if (fPtr!=NULL)
19    {
20        // la fwrite scrive 2 elementi quindi la posizione 0 e 1 dell'array
21        n = fwrite(&v, sizeof(int), 2, fPtr); // si può omettere la &
22        printf("Dati scritti su file: %d \n", n);
23        // chiude il file puntato
24        fclose(fPtr);
25    }
26    else
27    {
28      printf("Impossibile accedere al file %s", f);
29    }
```

Scrittura e lettura di array con fwrite e fread

es5.2.c

```
13    // apre il file secondo la modalità
14    fPtr = fopen(f,"r");
15
16    // se ha potuto creare il file
17    if (fPtr!=NULL)
18    {
19        while (!feof(fPtr))
20        {
21            n = fread(&x, sizeof(int), 1, fPtr);
22            if (n!=0)
23              printf("Dati letti da file: %d --> %d\n", n, x);
24        }
25        // chiude il file puntato
26        fclose(fPtr);
27    }
```

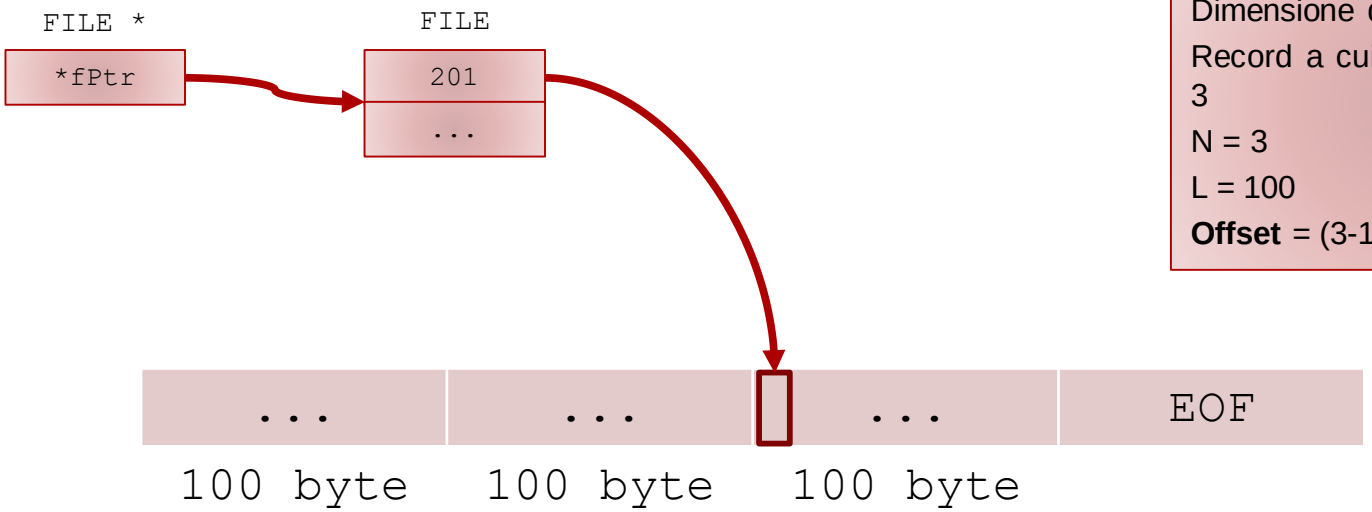
Scrittura e lettura di array con fwrite e fread

es5.3.c

```
17    // se ha potuto creare il file
18    if (fPtr!=NULL)
19    {
20        while (!feof(fPtr))
21        {
22              // fread legge 2 interi e li salva nel vettore
23              // se ci sono 3 dati la prima volta ne legge 2 la volta successiva 1
24              n = fread(&v, sizeof(int), 2, fPtr);
25              if (n!=0)
26              {
27                  printf("Dati letti da file: %d\n", n);
28                  for(unsigned int i = 0; i<n; i++)
29                  {
30                    printf("Dato: %d \n", v[i]);
31                  }
32              }
33        }
34        // chiude il file puntato
35        fclose(fPtr);
```

Collocare il puntatore di posizione del file con fseek

- La funzione `fseek` (*to seek* = *cercare*) permette di spostare il puntatore del file in una posizione arbitraria basandosi sull'**offset**, ovvero il numero di byte che intercorrono tra un record di dati e un altro.



Es.:
Dimensione dei record: 100 byte
Record a cui si vuole accedere: 3
N = 3
L = 100
Offset = (3-1)*100+1 → **201**

Collocare il puntatore di posizione del file con fseek

- La funzione `fseek` (*to seek* = *cercare*) è definita in `<stdio.h>` come segue:

```
int fseek(FILE *stream, long int offset, int whence)
```

- I parametri sono:
  - il puntatore al file;
  - l'offset, ovvero di quanti byte dovrà spostarsi il puntatore (solitamente il numero di elementi * la dimensione di un singolo elemento);
  - la posizione di riferimento da cui partire per spostarsi (ovvero per eseguire l'offset); `whence` può avere tre valori (definiti in `<stdio.h>`):
    - `SEEK_SET`: inizio del file;
    - `SEEK_CUR`: posizione corrente del puntatore;
    - `SEEK_END`: fine del file.
- La funzione restituisce 0 se lo spostamento del puntatore ha avuto successo.

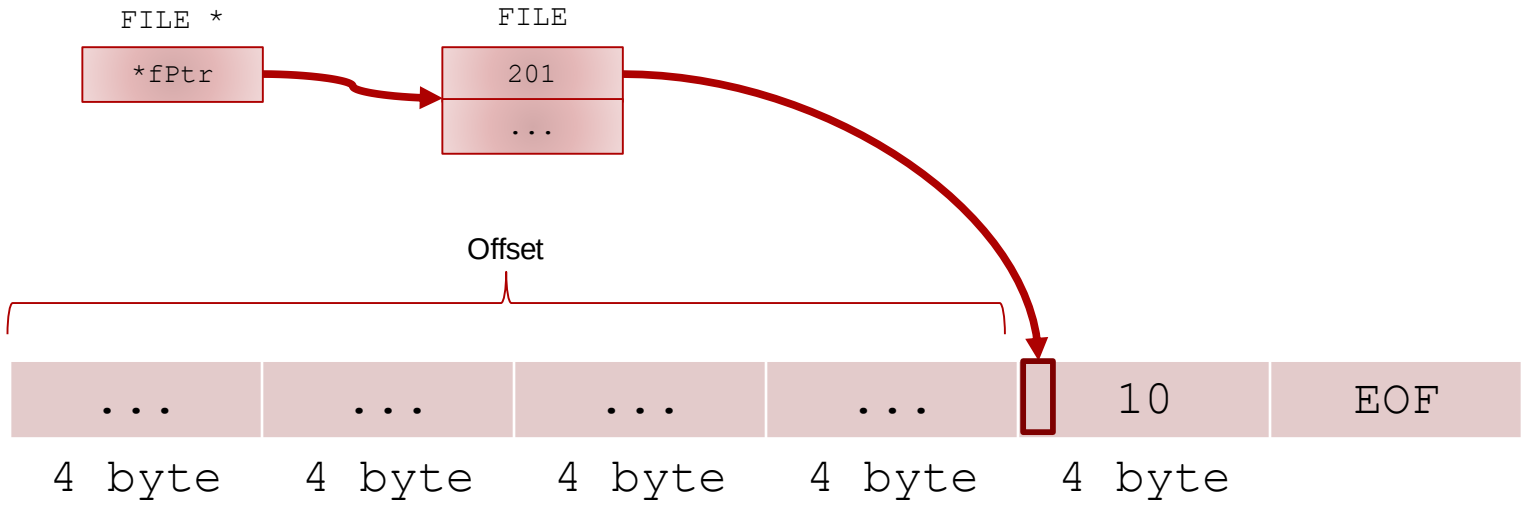
Collocare il puntatore di posizione del file con fseek

es6.1.c

```
15    // se ha potuto accedere al file
16    if (fPtr!=NULL)
17    {
18        unsigned int p = 5; // partendo da 0, indica posizione 4
19        // esegue l'offset
20        // sposta il puntatore di p-1 posizioni partendo da inizio file (SEEK_SET)
21        // ogni posizione contiene un intero
22        // quindi la dimensione di ogni posizione è sizeof(int) --> 4 byte
23        fseek(fPtr, (p-1) * sizeof(int), SEEK_SET);
24        int n = fwrite(&x, sizeof(int), 1, fPtr);
25        printf("Dati scritti su file: %d \n", n);
26        printf("Record scritto su file: %d \n", x);
27        printf("Dimensione di ogni record: %d \n", sizeof(int));
28        printf("Offset su file: %d \n", (p-1) * sizeof(int));
29        // chiude il file puntato
30        fclose(fPtr);
31    }
```

Collocare il puntatore di posizione del file con fseek

```
Prompt dei comandi

E:\UPO\Lezioni - Prog 2\Sorgenti\lezione_7_file\es6>a.exe
Dati scritti su file: 1
Record scritto su file: 10
Dimensione di ogni record: 4
Offset su file: 16
```



Esempio

- Si supponga di dover gestire in maniera semplificata un file di testo contenente i dati relativi ai conti correnti (cc) di una banca.
- Il programma dovrà avere le seguenti funzionalità:
  1) inserimento di un nuovo cc;
  2) visualizzazione di tutti i cc;
  3) modifica di un cc;
  4) eliminazione di un cc.

Esempio

- Ogni conto corrente (cc) deve contenere cognome, nome e saldo del conto.
- Ogni conto corrente (cc) ha un id numerico univoco (non possono esserci due cc con lo stesso id).
- I valori degli id partono da 1 e possono arrivare a 1000.
- Il cc con id = 1 verrà salvato nella posizione logica 0 del file, il cc con id = 2 nella posizione logica 1 e così via...

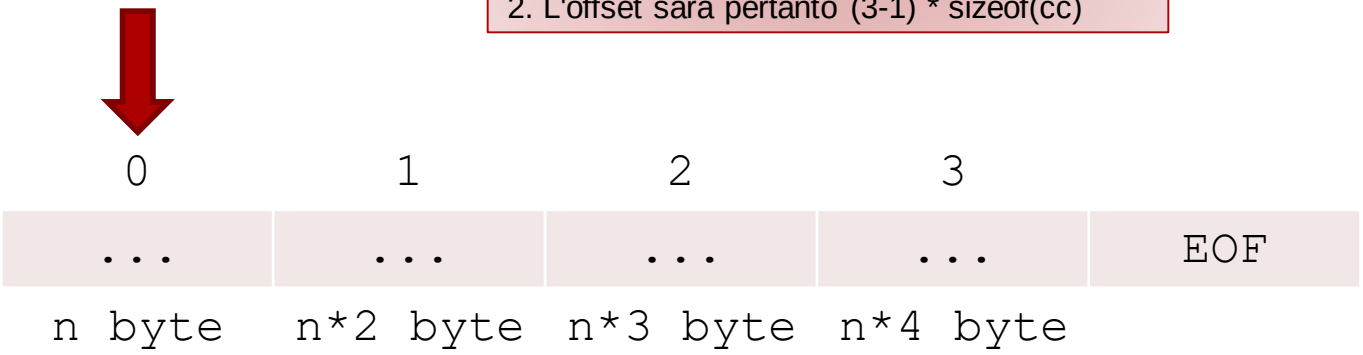
Esempio

id	1
cognome	Rossi
nome	Andrea
saldo	200.50

Il conto corrente (cc) con id=1 si troverà nella posizione logica 0 del file.

Il conto corrente con id=2 si troverà nella posizione logica 1 del file, quindi sarà necessario inserirlo dopo l'offset del primo record. L'offset sarà pertanto (2-1) * sizeof(cc).

Il cc con id=3 si troverà alla posizione logica 2. L'offset sarà pertanto (3-1) * sizeof(cc)



Esempio

es7.c

```c
#include <stdio.h>
#define SIZE 32
#define MINCC 1
#define MAXCC 1000

struct cc{
  unsigned int id;
  char cognome[SIZE];
  char nome[SIZE];
  double saldo;
};

typedef struct cc ContoCorrente;

// prototipi
void ccInserisci(FILE *fPtr);
void ccVisualizza(FILE *fPtr);
void ccAggiorna(FILE *fPtr);
void ccCancella(FILE *fPtr);
```

Esempio

es7.c

```
23  int main(void)
24  {
25    FILE *fPtr;
26    char *nomeFile = "cc.txt";
27    unsigned int menu;
28    char invio;
29
30    fPtr = fopen(nomeFile, "rb+"); // rb+ lettura e scrittura
31
32    if (fPtr==NULL)
33    {
34      printf("Impossibile accedere al file dei cc: %s \n", nomeFile);
35    }
```

- Siccome il file è ad accesso diretto e deve essere costantemente letto o aggiornato, è necessario creare il file vuoto prima di eseguire il programma la prima volta.
- Si noti che il file è aperto in modalità `rb+`, a indicare un file binario (ad accesso diretto) da aprire in modalità lettura e scrittura (linea 30).

Esempio

es7.c

```
102  void ccInserisci(FILE *fPtr)
103  {
104    unsigned int idcc;
105    ContoCorrente ccTemp;
106    unsigned int n;
107    char invio;
108
109    printf("\n*** 2 - Inserimento CC *** \n\n");
110    do{
111        printf("Inserire l'ID del CC da inserire (tra %d e %d): ", MINCC, MAXCC);
112        scanf("%d", &idcc);
113        invio = getchar();
114    }
115    while (idcc < MINCC || idcc > MAXCC);
116
117    // sposta il puntatore del file alla posizione corretta
118    fseek(fPtr, (idcc-1)*sizeof(ContoCorrente), SEEK_SET);
119
120    // verifica che il cc non esista già
121    // esegue una lettura
122    n = fread(&ccTemp, sizeof(ContoCorrente), 1, fPtr);
123
124    // se ha trovato un record valido
125    if (n>0 && ccTemp.id!=0)
126    {
127      printf("Conto corrente con ID %d gia esistente!\n\n", idcc);
128    }
129    else
130    {
131      ccTemp.id = idcc;
132      // 31 caratteri perché un posto deve essere riservato a '\0
133      printf("Cognome (max %d caratteri): ", SIZE-1);
134      scanf("%31s", &ccTemp.cognome);
135      printf("Nome (max %d caratteri): ", SIZE-1);
136      scanf("%31s", &ccTemp.nome);
137      printf("Saldo: ");
138      scanf("%lf", &ccTemp.saldo);
139      // sposta il puntatore del file alla posizione corretta
140      // in quanto dopo la fread è stato spostato
141      fseek(fPtr, (idcc-1)*sizeof(ContoCorrente), SEEK_SET);
142      n = fwrite(&ccTemp, sizeof(ContoCorrente), 1, fPtr);
143      if (n > 0)
144        printf("Conto corrente con ID %d salvato correttamente.\n", ccTemp.id);
145    }
146    printf("\n*** 2 - Fine inserimento CC *** \n\n");
147  }
```

- Quando si desidera inserire un nuovo record, il programma sposta il puntatore di un offset legato all'id con `fseek` ed effettua una lettura con `fread`.
- Se alla posizione dell'id non è già presente un record con id diverso da 0, è possibile inserire il record.
  - Si noti come tramite la `fseek` sia possibile ottenere un record molto velocemente senza scorrere tutto il file.
- Si noti che, prima di eseguire l'inserimento con la `fwrite`, il programma riposiziona il puntatore alla posizione corretta con `fseek` (linea 141).

Esempio

es7.c

```
78  void ccVisualizza(FILE *fPtr)
79  {
80    ContoCorrente ccTemp;
81    unsigned int n;
82    int i = 0; // conta i CC
83    rewind(fPtr); // porta il puntatore a inizio file per scorrerlo tutto
84    printf("\n*** 1 - Stampa i CC *** \n\n");
85    while (!feof(fPtr))
86    {
87        n = fread(&ccTemp, sizeof(ContoCorrente), 1, fPtr);
88        if (n>0 && ccTemp.id!=0)
89        {
90          i++;
91          printf("ID: %d \t Cognome: %s \t Nome: %s \t Saldo: %10.2f \n", ccTemp.id, ccTemp.cognome, ccTemp.nome, ccTemp.saldo);
92          printf("----------------------------------------------------------------\n\n");
93        }
94    }
95    if (i==0)
96    {
97      printf("Nessun conto corrente presente in memoria!\n");
98    }
99    printf("\n*** 1 - Fine Stampa i CC *** \n\n");
100 }
```

- Si noti che, prima di scorrere tutto il file per visualizzare i dati, è necessario posizionare il puntatore a inizio file (linea 83) con `rewind()`, in quanto non è possibile sapere in che posizione si trovi il puntatore al file.
- Si noti che `rewind(fPtr)` corrisponde a `fseek(fPtr,0, SEEK_SET)`.

Esempio

es7.c

```c
void ccAggiorna(FILE *fPtr)
{
  unsigned int idcc;
  ContoCorrente ccTemp;
  unsigned int n;
  double transazione;
  char invio;

  printf("\n*** 3 - Aggiorna CC esistente *** \n\n");
  do{
      printf("Inserire l'ID del CC da aggiornare (tra %d e %d): ", MINCC, MAXCC);
      scanf("%d", &idcc);
      invio = getchar();
  }
  while (idcc<MINCC || idcc >MAXCC);

  // sposta il puntatore del file alla posizione corretta
  fseek(fPtr, (idcc-1)*sizeof(ContoCorrente), SEEK_SET);

  // verifica che il CC esista eseguendo una lettura
  n = fread(&ccTemp, sizeof(ContoCorrente), 1, fPtr);

  if (n==0 || ccTemp.id==0)
  {
    printf("Conto corrente con ID %d non esistente!\n", idcc);
  }
  else
  {
    // riporta il puntatore nella posizione corretta (dopo la fread si è spostato)
    fseek(fPtr, (idcc-1)*sizeof(ContoCorrente), SEEK_SET);
    printf("Valore transazione: ");
    scanf("%lf", &transazione);
    ccTemp.saldo = ccTemp.saldo + transazione;
    n = fwrite(&ccTemp, sizeof(ContoCorrente), 1, fPtr);
    if (n > 0)
      printf("Conto corrente con ID %d aggiornato correttamente (nuovo saldo: %.2f)\n",ccTemp.id, ccTemp.saldo);
  }
  printf("\n*** 3 - Fine Aggiorna CC *** \n\n");
}
```

- L'aggiornamento accede al record di un determinato conto tramite la `fseek`.
- Se il conto esiste, si procede alla modifica del saldo.
- Si noti come, dopo la verifica dell'esistenza del conto, sia necessario riposizionare il puntatore alla posizione del record appena letto, in quando la `fread` dopo la prima lettura sposta il puntatore al record successivo.

Esempio

es7.c

```c
void ccCancella(FILE *fPtr)
{
  unsigned int idcc;
  ContoCorrente ccTemp;
  ContoCorrente ccTempVuoto = {0, "", "", 0};
  unsigned int n;
  double transazione;
  char invio;

  printf("\n*** 4 - Cancella CC esistente *** \n\n");
  do{
      printf("Inserire l'ID del CC da cancellare (tra %d e %d): ", MINCC, MAXCC);
      scanf("%d", &idcc);
      invio = getchar();
  }
  while (idcc<MINCC || idcc >MAXCC);

  // sposta il puntatore del file alla posizione corretta
  fseek(fPtr, (idcc-1)*sizeof(ContoCorrente), SEEK_SET);

  // verifica che il CC esista eseguendo una lettura
  n = fread(&ccTemp, sizeof(ContoCorrente), 1, fPtr);

  if (n==0)
  {
    printf("Conto corrente con ID %d non esistente!\n", idcc);
  }
  else
  {
    // riporta il puntatore nella posizione corretta (dopo la fread si è spostato)
    fseek(fPtr, (idcc-1)*sizeof(ContoCorrente), SEEK_SET);
    // la cancellazione inserisce un record "vuoto"
    n = fwrite(&ccTempVuoto, sizeof(ContoCorrente), 1, fPtr);
    if (n > 0)
      printf("Conto corrente con ID %d cancellato \n", idcc);
  }
  printf("\n*** 4 - Fine Cancella CC *** \n\n");
}
```

- La cancellazione accede al record di un determinato conto tramite la `fseek`.
- Se il conto esiste, si procede alla cancellazione del conto.
- Il conto non viene cancellato, ma impostato a tutti valori con 0.
- Si noti come, dopo la verifica dell'esistenza del conto, sia necessario riposizionare il puntatore alla posizione del record appena letto, in quando la `fread` dopo la prima lettura sposta il puntatore al record successivo.

FINE PRESENTAZIONE