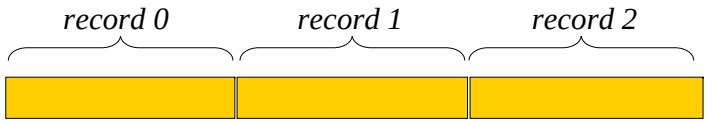
File ad accesso diretto

Ver. 2.4

© 2010 - Claudio Fornaro - Corso di programmazione in C

Struttura a record

- Il file è organizzato in ***record*** aventi tutti la stessa struttura e quindi dimensione



- Ogni record viene letto/scritto in un'unica operazione (tutti i suoi byte insieme)
- L'accesso al record *N*-esimo è molto veloce in quanto per determinare dove esso inizia (il suo offset) basta calcolare: *N* * *L* (dove *L* è la lunghezza dei record)
- *offset:* distanza (in byte) dall'inizio del file

Modalità di apertura

- Nella `fopen` si indica che si usano *file binari* (dati grezzi) specificando i modi: `"rb"`, `"wb"`, `"ab"`, `"r+b"`, `"w+b"` e `"a+b"`
- La differenza tra `"r+b"` e `"w+b"` è solo nell'apertura:
  `"r+b"` → se il file non c'è, dà errore,
  `"w+b"` → anche se il file c'è, lo ricrea vuoto
- Nelle modalità `"ab"` e `"a+b"` il file position pointer è pronto per aggiungere dati in fondo al file

Modalità di apertura

- Il modo `"a+b"` è simile a `"w+b"`, salvo che permette di scrivere solo in fondo al file; anche se con `fseek` si può spostare l'offset indietro, la scrittura avviene sempre al fondo del file e i dati precedenti non possono essere modificati
- La chiusura dei file binari si ottiene sempre con la funzione `fclose`
- I file binari non sono portabili in quanto i dati possono essere memorizzati in modi diversi su architetture diverse (es. *big-endian* e *low-endian*)

Lettura

- La lettura di un blocco di byte si ottiene mediante la funzione
  `fread(`*p, dim_oggetto, num_oggetti, fp*`);`
  - legge dal file *fp* un numero di oggetti pari a *num_oggetti* ciascuno di dimensione pari a *dim_oggetto* byte e li colloca nel **vettore** di oggetti puntato da *p*
  - restituisce il numero di *oggetti* (non di byte) effettivamente letti (può essere inferiore a *num_oggetti* in caso di errore o fine file)
  - fa avanzare il file position pointer del numero di *byte* effettivamente letti
  - si devono usare le funzioni `feof` e `ferror` per distinguere gli errori dalla fine del file

Lettura

- Esempi
```
struct X var1, var2;
fread(&var1, sizeof(var1), 1, fp);
fread(&var2,sizeof(struct X),1,fp);
```
ogni `fread` legge **un** blocco di byte (oggetto, record) della dimensione di una `struct X` e lo mette nella variabile di cui viene fornito il puntatore (rispettivamente `var1` e `var2`)
- ```
  struct X vet[100];
  fread(vet,sizeof(struct X),100,fp);
  ```
  vengono letti **100** record corrispondenti a 100 `struct X` e memorizzati ordinatamente nel vettore `vet`

Scrittura

- La scrittura di un blocco di byte si ottiene mediante la funzione
  `fwrite(`*p, dim_oggetto, num_oggetti, fp*`);`
  - scrive nel file *fp* un numero di oggetti pari a *num_oggetti* ciascuno di dimensione pari a *dim_oggetto* byte prelevandoli dal **vettore** di oggetti puntati da *p*
  - restituisce il numero di *oggetti* (non di byte) effettivamente scritti (può essere inferiore a *num_oggetti* in caso di errore)
  - fa avanzare il file position pointer del numero di *byte* effettivamente scritti

Scrittura

- ```
  struct X var;
  fwrite(&var, sizeof var, 1, fp);
  fwrite(&var,sizeof(struct X),1,fp);
  ```
  ogni `fwrite` scrive un blocco di byte (oggetto, record) della dimensione di una `struct X` prelevandolo dalla variabile `var` di cui viene fornito il puntatore
- ```
  struct X vet[10];
  fwrite(vet,sizeof(struct X),10,fp);
  ```
  vengono scritti 10 record della dimensione di una `struct X` prelevandoli ordinatamente dal vettore `vett`
  ```

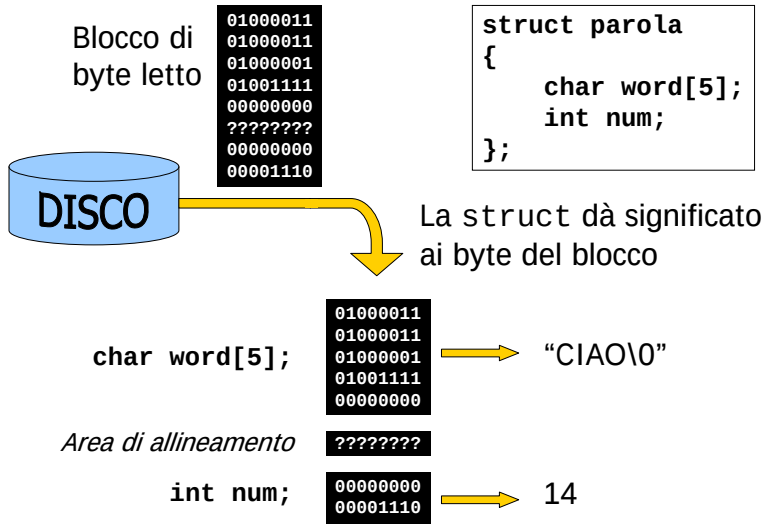
Scrittura

- *Attenzione!*
  Per scrivere in due record successivi la stessa `var` E' SBAGLIATO scrivere:
  `fwrite(&var, sizeof var, 2, fp);`
  In questo modo la `fwrite` scrive sul disco `2*sizeof(var)` byte a partire dal primo byte di `var`, sforando la dimensione di `var`, trattandolo come se fosse un vettore di 2 elementi
- Bisogna invece scrivere 2 `fwrite`:
  `fwrite(&var, sizeof var, 1, fp);`
  `fwrite(&var, sizeof var, 1, fp);`

Lettura e scrittura di blocchi

- Le funzioni `fread` e `fwrite` leggono e scrivono un *blocco di byte* di dimensione pari a *dim_oggetto* * *num_oggetti*
  Le funzioni ricevono un puntatore e non hanno nessuna informazione sul contenuto di questo blocco che potrebbe essere una variabile o un vettore di qualsiasi tipo: sta al programmatore usare il tipo di oggetto appropriato
- Solo quando questo blocco di byte viene letto dalla `fread` e inserito in una variabile o vettore allora il contenuto diventa significativo

Lettura e scrittura di blocchi



Inizializzazione

- In alcuni casi è utile prevedere una inizializzazione del file con strutture vuote, ad esempio per rendere più semplice verificare se un record è già stato scritto

```
struct X vett[MAX], vuot={"",0};
fp = fopen(file, "wb");
for (i=0; i<MAX; i++)
  fwrite(&vuot, sizeof vuot,1,fp);
fclose(fp);
```

Come già detto, non si può sostituire al ciclo `for` una `fwrite` con *num_oggetti* pari a MAX

Posizionamento

- Le funzioni `fread` e `fwrite` posizionano automaticamente il file position pointer (offset) all'inizio del record successivo
- Il posizionamento generico si ottiene con la funzione `fseek`

  `fseek(` *fp*`,` *offset*`,` *origine*`);`

  dove:
  - *offset* (di tipo `long`) indica su quale carattere, a partire da *origine*, spostare il file position pointer
  - *origine* indica da dove calcolare l'*offset*:
    - `SEEK_SET` → da inizio file
    - `SEEK_CURR` → dalla posizione corrente
    - `SEEK_END` → da fine file

Posizionamento

- Esempio

  `fseek(fp, 20, SEEK_SET);`

  sposta il file position pointer al 21º carattere del file (il primo ha posizione 0)
- La `fseek` può indicare un offset oltre la fine del file, questo "allunga" il file (i byte non scritti hanno valore indeterminato)
- Per conoscere la posizione attuale del file position pointer (quindi l'offset) si può usare la funzione `ftell`

  `posiz = ftell(`*fp*`)`

  `posiz` è una variabile di tipo `long`

  `ftell` restituisce `-1L` in caso di errore

Posizionamento

- Per tornare all'inizio del file si può usare la funzione `fseek`:

  `fseek(`*fp*`, 0, SEEK_SET);`
- Ma la soluzione migliore è:

  `rewind(`*fp*`);`

  che oltre a eseguire la `fseek` precedente esegue anche la `clearerr(`*fp*`)` che azzera l'indicazione di errori e del raggiungimento della fine del file

Posizionamento

- `fgetpos(`*fp*`,` *ptr*`)`

  memorizza in *ptr* l'offset corrente del file *fp*, dà un valore non nullo in caso di errore
- `fsetpos(`*fp*`,` *ptr*`)`

  posiziona l'offset corrente del file *fp* con il valore indicato in *ptr*, dà un valore non nullo in caso di errore
- *ptr* deve essere definito con:

  `fps_t *`*ptr*`;`

  il tipo `fps_t` è un tipo appropriato per contenere il valore di un qualsiasi offset, mentre `fseek/ftell` possono usare solo valori `long` (potrebbe essere una limitazione)

Troncamento

- La libreria standard del C non fornisce alcuna funzione in grado di accorciare un file
- Possono esistere funzioni proprietarie quali `truncate`, `ftruncate` e altre
- Non è altresì possibile mediate funzioni standard eliminare dati dalla testa di un file
- La soluzione standard per entrambi i casi consiste nel creare un nuovo file copiandovi i dati da tenere ed escludendo quelli da rimuovere

Esercizi

1. Si scriva un programma per gestire un'agenda elettronica. I dati relativi alle persone sono: nome, cognome, indirizzo, telefono, nota, *possono contenere spazi* e sono organizzati in una `struct persona`. Si definisca una dimensione massima per il numero di nominativi. Un file binario ad accesso diretto denominato `database.dat` contiene i record di tutti i nominativi.
(continua)

Esercizi

(*Continuazione*)

Il programma deve fornire un'interfaccia a menu per inserire, cancellare, cercare i dati relativi ad una persona (in base al cognome), visualizzare alfabeticamente tutti i cognomi/nomi senza dettagli. I dati completi siano mantenuti *esclusivamente sul file*, in memoria si tenga invece un *indice alfabetico* dei nomi (vettore di strutture contenenti solo cognome, nome e numero del record sul file) per accedere al nominativo.