Gestione di file

IL CONCETTO DI FILE

- Per poter conservare dati anche dopo l'esecuzione di un programma, è necessario renderli *persistenti* ovvero *archiviarli su memoria di massa.*

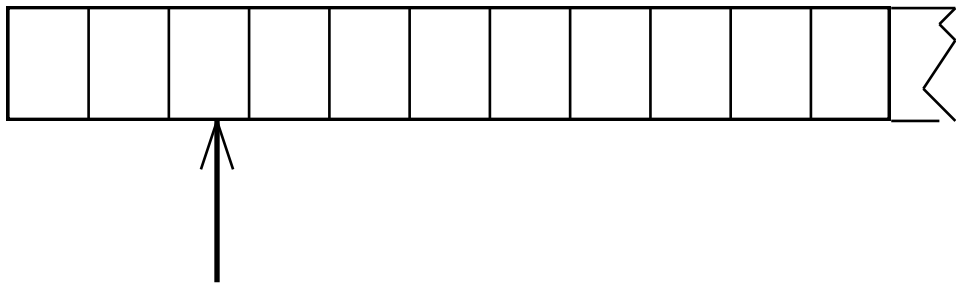
Nozione di File

Un file è una *astrazione fornita dal sistema operativo*, il cui scopo è consentire la memorizzazione di informazioni su memoria di massa.

Concettualmente, un file:

- è una *sequenza di registrazioni (record) uniformi,* cioè dello stesso tipo
- ha *dimensione potenzialmente illimitata*
- è *ad accesso sequenziale*.

Accesso ai file



- Una *testina di lettura/scrittura* indica in ogni istante la registrazione (record) corrente.
- Inizialmente, la testina si trova sulla prima posizione.
- Dopo ogni operazione di lettura/scrittura, la testina si sposta sulla registrazione successiva.
- E' illegale tentare di accedere *oltre la fine del file.*

OPERARE SUI FILE

- A livello di sistema operativo, un file è denotato univocamente dal suo **nome assoluto**, che comprende il *percorso* e il *nome relativo*. A sua volta, in certi sistemi operativi il percorso può comprendere il nome dell'unità.
- in DOS o Windows: C:\temp**prova1.c**
- in UNIX e Linux: /usr/temp/**prova1.c**

OPERARE SUI FILE (cont.)

- Poiché un file è un'entità del sistema operativo e un programma può lavorare solo su simboli definiti e noti entro il programma stesso, per agire su un file dall'interno di un programma occorre **stabilire una corrispondenza** fra:
  - *il nome del file* come risulta al sistema operativo
  - *un nome di variabile* definito nel programma.
- Questa operazione appartiene al modello di coordinazione e si chiama **apertura di un file**.
- Il programma potrà poi operare sul file *agendo sulla variabile definita al suo interno*. Il sistema operativo provvederà ad effettuare l'operazione richiesta sul file associato a tale simbolo.
- La corrispondenza può essere distrutta mediante l'operazione di **chiusura del file.**

FILE IN C

- Il modello di coordinazione della Standard I/O Library è basato sul concetto di file che vede come *stream* di byte.
  - In particolare, i tre stream:
    - o *standard input* (**stdin**)
    - o *standard output* (**stdout**)
    - o *standard error* (**stderr**)

    corrispondono a *file già aperti*

- Per gestire la corrispondenza fra *il nome del file* e *una variabile del programma* che rappresenta uno stream, la Standard I/O Library definisce il tipo **FILE**.

IL TIPO FILE

Il tipo File

Il tipo FILE è una struttura definita nella libreria `stdio.h`, che rimane trasparente all'utente e che spesso cambia da un compilatore all'altro.

- Le strutture di tipo FILE non sono mai gestite direttamente dall'utente, ma solo ed esclusivamente dalle funzioni della Standard I/O Library.
- Nei suoi programmi, l'utente dovrà solo definire, ove necessario, dei **puntatori a FILE.**

APERTURA e CHIUSURA DI UN FILE

- Il modello di coordinazione della Standard I/O Library distingue due **tipi** di file:
  - ✓ **file di testo** → sequenza di *caratteri*
  - ✓ **file binario** → sequenza di *byte*
- Per aprire un file, **stdio.h** dichiara la funzione **fopen():**

```
FILE* fopen(char fname[], char mode[])
```

apre il file di nome *fname* nel modo specificato in *mode* e restituisce un puntatore a FILE.

La stringa `mode` specifica come aprire il file:

Modalità di apertura		Eventualmente seguita da	
r	***lettura (read)***	**t**	***testo*** (default)
w	***scrittura (write)***	**b**	***binaria***
a	***aggiunta (append)***	**+**	***modifica***

- Alla fine, per chiudere un file si usa **fclose()**:

```
int fclose(FILE*)
```

ESEMPI DI APERTURE DI FILE DI TESTO

- Apertura di un file di testo (nella stessa directory) in **lettura**:

```
FILE *f;
f = fopen(”pippo.txt","r");
```

- Apertura di un file di testo in **scrittura**:

```
FILE *f;
f = fopen(”pippo.txt","w");
```

- Apertura di un file di testo in **aggiunta** (in coda):

```
FILE *f;
f = fopen(”pippo.txt","a");
```

- Apertura di un file di testo in **lettura con modifiche**:

```
FILE *f;
f = fopen(”pippo.txt","r+");
```

NOTE

- lo specificatore di modo "**t**", che è il default, solitamente si omette
- l’apertura di un file in modifica (r+,w+,a+) richiede delle operazioni di riposizionamento della testina che non vedremo!

OPERAZIONI SU FILE

- Il valore restituito da **fopen()** è un puntatore a FILE, *da usare in tutte le successive operazioni sul file.*
- Tale puntatore è `NULL` in caso di fallimento
  - **controllare sempre** il risultato di fopen()

ESEMPIO

Apertura di un file di testo in **lettura con modifiche**:

```
FILE *f;
f = fopen(”pippo.txt","r+");
```

… e successiva chiusura del file

```
fclose(f);
```

OPERAZIONI SUI FILE DI TESTO

- La libreria standard fornisce funzioni *del tutto analoghe* a quelle già viste per i canali di I/O predefiniti:

	Funzione da console	*Funzione da file*
a caratteri	`int getchar(void);`	`int fgetc(FILE* f);`
	`int putchar(int c);`	`int fputc(int c, FILE* f);`
a stringhe di caratteri	`char* gets(char* s);`	`char* fgets(char* s, int n, FILE* f);`
	`int puts(char* s);`	`int fputs(char* s, FILE* f);`
formattato	`int printf(char* fm, expr1,...exprN);`	`int fprintf(FILE* f, char*fm, expr1,...exprN);`
	`int scanf(char* fm, addr1,...addrN);`	`int fscanf(FILE* f, char* fm, addr1,...addrN);`

- A queste si aggiungono altre funzioni di utilità generale.

Esempio 1

Archiviazione di ciò che viene inserito da console (digitandolo su tastiera) su un file di testo.

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

main(){
  FILE *fp;
  int c;
  fp = fopen("prova.txt","w");
  if (fp==NULL){
  /* esce con un codice d'errore */
       fprintf(stderr, "Errore di apertura file\n");
       exit(1);
  }
  while ((c=getchar())!=EOF)
       fputc(c,fp);
  fclose(fp);
}
```

Note sull'esempio 1

- **`fopen()`** restituisce un puntatore `NULL` in caso di errore nell'apertura del file (ad esempio, se non c'è spazio su disco o il disco è protetto)
- il messaggio d'errore è emesso sul canale di errore standard stderr (di norma associato al video)
- la funzione di libreria **`exit()`**, dichiarata in stdlib.h, fa terminare il programma, restituendo al sistema operativo un codice d'errore (convenzione: 0 ➔ ok; altro numero ➔ errore)

Esempio 2

Stampare a video il contenuto di un file di testo.

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

main(){
  FILE *fp;
  int c;
  fp = fopen("prova.txt",”r");
  if (fp==NULL){
       fprintf(stderr, "Errore di apertura file\n");
       exit(1);
  }
  while ((c=fgetc(fp))!=EOF)
       putchar(c);
  fclose(fp);
}
```

NOTA **`fgetc()`** restituisce il numero di caratteri letti o EOF in caso di fine file o errore in lettura.

Esempio 3

- Archiviare in un file di testo una serie di dati relativi a persone (cognome, nome ed età), una persona per riga. Il numero di persone non è noto a priori.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
      char cognome[20], nome [20];
      int eta; } persona;
main(){
  FILE *fp;
  persona p;
  int continua;
  fp = fopen("archivio.txt","a"); /* append */
```

Esempio 3 (cont.)

```
if (fp==NULL) {
        fprintf(stderr, "Errore di apertura file \n");
        exit(1);}
do {
        printf("Immettere Cognome, Nome ed età:\t");
        scanf("%s %s %d", p.cognome, p.nome, &p.eta);
        fprintf(fp,"%s %s %d\n", p.cognome, p.nome, p.eta);
        printf("Continuare? (0-NO/1-SI)");
        scanf("%d",&continua);
  } while (continua);
fclose(fp);}
```

Esempio 4

Leggere da un file di testo e mostrare a video una serie di dati relativi a persone (cognome, nome ed età) ipotizzando che sia archiviata una persona per riga.

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

typedef struct {
       char cognome[20], nome [20];
       int eta; } persona;
main(){
  FILE *fp;
  persona p;
  fp = fopen("archivio.txt","r");
  if (fp==NULL) {
       fprintf(stderr, "Errore di apertura\n");
       exit(1);}
  while (fscanf(fp,"%s%s%d", p.cognome, p.nome, &p.eta) >0 )
       printf("Cognome: %s, Nome: %s, Età:%s\n",
                                p.cognome, p.nome, p.eta);
  fclose(fp);}
```

NOTE sull'Esempio 4

- *La funzione* `fscanf()` *restituisce il numero di elementi letti*, che è zero quando viene raggiunta la fine del file: *questo consente di usarla come condizione del while* (si continua se > 0).
- È la filosofia di fondo del C: *prima si tenta l'operazione, poi si scopre come è andata (ad es. se il file era finito).*
- Per questo *tutte le funzioni restituiscono un valore da cui si può capire se il file è finito.*

I FILE BINARI

- Un file binario è una **sequenza di byte**, e viene usato in C ***per archiviare*** **su memoria di massa** ***qualunque tipo di informazione*** *che non sia un testo.*
- In particolare, si usano file binari per archiviare:
  - valori numerici (int, float, double, …)
  - tipi definiti dall'utente (struct, vettori, …)
- Un file binario potrebbe anche essere usato per archiviare testi, ma solitamente *non conviene farlo*, perché le funzioni disponibili perr l'I/O da file di testo sono *molto più sofisticate* di quelle disponibili per l'I/O da file binario.
- Per operare sui ***file binari***, la libreria standard fornisce due funzioni:
  - **`fread()`** per leggere una *sequenza di byte*
  - **`fwrite()`** per scrivere una *sequenza di byte.*
- Le sequenze di byte lette o scritte *non sono interpretate*

OPERAZIONI SUI FILE BINARI

`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 in memoria a partire dall'indirizzo ***addr***
- restituisce il numero di *elementi* effettivamente scritti (possono essere meno di **n**).

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

- legge dal file **n** elementi, ognuno grande **dim** byte *(complessivamente, tenta quindi di leggere* **n×dim** *byte)*
- gli elementi da leggere vengono posti in memoria a partire dall'indirizzo ***addr***
- restituisce il numero di *elementi* effettivamente letti (possono essere meno di **n**, al limite anche zero in caso di fine file).

Esempio 5

Salvare su un file binario il contenuto di un vettore di interi.

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

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

NOTE sull'Esempio 5

- il nome di un vettore, `vet` rappresenta già un indirizzo (`vett=&vett[0]`)
- L'operatore `sizeof()` calcola la dimensione in byte del tipo argomento della funzione
- L'operatore `sizeof()` è essenziale perché il programma sia portabile da una piattaforma all'altra
- in alternativa, era possibile implementare un ciclo di dieci `fwrite()`

Esempio 6

Leggere da un file binario degli interi e memorizzarli in un vettore.

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

main(){
  FILE *fp;
  int vet[40], i, n;

  if ((fp = fopen(”numeri.dat","rb"))==NULL) {
       fprintf(stderr, "Errore di apertura\n");
       exit(1);}

  n = fread(vet,sizeof(int),40,fp);

  for (i=0; i<n; i++) printf("%d ",vet[i]);
  fclose(fp);}
```

sebbene `fread()` tenti di leggere 40 interi (il massimo compatibile con le dimensioni del vettore), il file potrebbe contenerne meno

➔ è importante considerare il valore restituito, n

Esempio 7

Archiviare in coda ad un file binario già esistente una serie di dati relativi a persone (ogni persona è caratterizzata da cognome, nome ed età).

Si confronti questo esempio con l'esempio 3.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
      char cognome[20], nome [20];
      int eta; } persona;
main(){
  FILE *fp;
  persona p;
  int continua;
  if ((fp = fopen("archivio.dat","ab"))==NULL) {
      fprintf(stderr, "Errore di apertura\n");exit(1);}
  do {printf("Immettere Cognome, Nome ed età:\t");
      scanf("%s%s%d", p.cognome, p.nome, &p.eta);
      fwrite(&p,sizeof(persona),1,fp);
      printf("Continuare?"); scanf("%d",&continua);}
  while (continua);
  fclose(fp);}
```

Esempio 8

Leggere da un file binario e mostrare a video i dati relativi alle persone presenti nel file (il tipo `persona` è dato).

Si confronti questo esempio con l'esempio 4.

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

typedef struct {
        char cognome[20], nome [20];
        int eta; } persona;

main(){
  FILE *fp;
  persona p;

  if ((fp = fopen("archivio.dat","rb"))==NULL) {
        fprintf(stderr, "Errore di apertura\n");
        exit(1);}

  while (fread(&p,sizeof(persona),1,fp)>0)
        printf("Cognome: %s, Nome: %s, Età:%s\n",
                                p.cognome, p.nome, p.eta);
  fclose(fp);}
```

NOTE sull'Esempio 8

- il ciclo termina quando `fread()` restituisce 0 (che corrisponde al numero di elementi letti)
- è essenziale che la `typedef` sia **identica** a quella usata quando il file è stato scritto: altrimenti, si leggeranno solo sequenze di byte *senza senso*

FILE BINARI… O FILE DI TESTO?

- Riprendiamo le due definizioni:
  - file di testo → sequenza di ***caratteri***
  - file binario → sequenza di ***byte***
- *Poiché un carattere in C è ampio esattamente un byte*, ci si può chiedere quale sia realmente la differenza fra i due tipi di file.
- In effetti, a livello di sistema *un file di testo e un file binario sono identici*.

Tale classificazione è stata introdotta per facilitare la gestione dei file:

- un testo è una sequenza di caratteri, alcuni dei quali (ad es. '\n') hanno un *significato particolare*
- leggere e scrivere testi è *talmente frequente* che non vogliamo ogni volta dover utilizzare funzioni a basso livello quali **`fread()`** e **`fwrite()`**

Esercizi

1. Scrivere un programma per la gestione di persone (si veda il tipo definito negli esempi precedenti). Il programma deve
   1. Aprire il file archivio.dat
   2. Caricare in un vettore di al più 40 elementi di tipo persona il contenuto del file
   3. Stampare il vettore
   4. Presentare all'utente il seguente menu:

      1- aggiungi una persona

      2-cancella una persona

      3- esci

      Il programma deve implementare un ciclo dove l'utente ha la possibilità di scegliere una delle opzioni fino a quando non digita l'opzione 4. Al termine di ogni operazione deve essere stampato il contenuto del vettore.
   5. Al termine il programma deve riscrivere sul file archivio.dat il contenuto aggiornato del vettore.

   Il punto 3 e tutte le opzioni del punto 4 devono essere realizzate attraverso funzioni. In particolare
   - al punto 1 del menu deve corrispondere la chiamata di una funzione che aggiorna il vettore aggiungendo una nuova persona con dati richiesti all'interno della funzione.
   - Al punto 2 del menu il main deve richiedere il nome e il cognome della persona da cancellare. Queste informazioni assieme al vettore e la sua dim. effettiva costituiscono i parametri della funzione che provvede a cancellare tutte le occorrenze delle persone con nome e cognome dato

Esercizi (cont.)

2. Scrivere un programma che scrive su un file binario una matrice 4x4 con valori inseriti da tastiera
3. Scrivere un programma che carica dal file di cui all'esercizio precedete la matrice 4x4 contenuta e ne stampa il contenuto
4. Scrivere un programma che genera il file binario ingredienti.dat contententte gli ingredienti di una ricetta. Ogni ingrediente è costituito da nome -10 caratteri, unità di misura – valori: litri e grammi, quantità – reale. Gli ingredienti vengono inseriti da tastiera dall'utente.
5. Dati come parametri formali
   - un file binario già aperto in lettura e contenente ingredienti di una ricetta (nome -10 caratteri, unità di misura – valori:litri e grammi, quantità – reale)
   - il nome di ingrediente

   scrivere una funzione che copia nel secondo file binario tutti e soli gli ingredienti del primo file binario corrispondenti al nome specificato e restituisce la quantità totale dell'ingrediente necessaria per la ricetta

   Scrivere quindi un main che utilizza la procedura appena definita per trasferire nel file binario ingrOUT.dat tutti gli ingredienti del file binario ingredienti.dat relativi a un ingrediente inserito da tastiera dall'utente