# Files

Fondamenti di Programmazione 2021/2022

Francesco Tortorella

# File e file system

- Le unità di memoria di massa (dischi magnetici e/o SSD) forniscono il supporto fisico per la memorizzazione permanente dei dati, e presentano caratteristiche estremamente diverse a seconda della casa costruttrice e del tipo di unità.
- Il File System offre una visione logica uniforme della memorizzazione dei dati basata su un'unità di memoria logica, **il file**, definita indipendentemente dalle caratteristiche fisiche delle particolari unità.

# Il file

- Il file è un insieme di informazioni, correlate e registrate nella memoria di massa, identificato da un nome, che può essere formato da più sottoparti.
  - **nome**: si riferisce ai contenuti del file
  - **estensione**: si riferisce al tipo del file
- Dal punto di vista dell'utente, un file è la più piccola porzione (logica) di memoria di massa: i dati, cioè, possono essere scritti nella memoria di massa solo all'interno di un file.

# Contenuti di un file

- Le informazioni registrate all'interno di un file sono di due tipi:
- Dati veri e propri
  - programmi eseguibili
  - testi
  - immagini
  - dati numerici
  - …
- Informazioni (attributi) di interesse per l'utente
  - dimensione del file
  - data di creazione e/o ultima modifica
  - permessi di accesso

# Organizzazione logica dei files

- L'insieme dei file presenti in memoria di massa è organizzato secondo una struttura gerarchica ad albero, in cui i nodi intermedi costituiscono le directory (che raggruppano altri files e directory secondo un criterio di omogeneità), mentre le foglie rappresentano i file.
- All'interno di tale struttura, un particolare file è univocamente identificato dal path (o percorso) che localizza la directory in cui il file è memorizzato

# Organizzazione fisica dei files

- Da un punto di vista fisico, la registrazione del file sull'unità di memoria di massa viene realizzata dal sistema operativo disponendo il contenuto del file su un insieme di blocchi di memoria di lunghezza prefissata disponibili sull'unità.
- A seconda della tecnologia impiegata per realizzare l'unità di memoria di massa, le operazioni saranno più o meno efficienti. Attualmente, le unità di memoria a stato solido (SSD) presentano numerosi vantaggi rispetto a quelle di tipo magnetico (HDD).

# Il concetto di *stream*

- Uno **stream** è un'astrazione per indicare il flusso di dati tra un programma C e un dispositivo di I/O esterno o un file memorizzato su memoria di massa.
- Può essere sommariamente rappresentato come una sorgente o una destinazione di sequenze di dati di lunghezza indefinita.
- Esistono due tipi di stream:
  - **stream binario**: sequenza di bit
  - **stream di testo**: sequenza di caratteri organizzati in linee
- Lo stream costituisce un'interfaccia verso il file fisico e consente di realizzare da programma operazioni di I/O su di esso.

# Stream standard

- All'inizio dell'esecuzione, ogni programma C apre 3 stream di default (o standard).
- Standard input: è lo stream dal quale il programma riceve i dati in ingresso
- Standard output: è lo stream sul quale il programma invia i dati in uscita
- Standard error: è lo stream sul quale il programma invia i messaggi di errore
- I tre stream sono tipicamente associati al terminale dell'utente (tastiera e video)

# Il tipo `FILE`

- Quando il programma deve operare su un file presente su memoria di massa, deve creare un nuovo stream attraverso il quale si muove il flusso di dati con il file definito
- Per gestire uno stream in C, si usa il tipo **`FILE`** (tipo strutturato) definito nell'header file **`stdio.h`**.
- Una variabile di tipo **`FILE`** descrive uno stream.

# Apertura e chiusura di un file

- Poiché un file è un'entità del sistema operativo, per agire su esso dall'interno di un programma occorre stabilire una corrispondenza fra:
  - il file come risulta (tramite il suo nome) al sistema operativo
  - una variabile definita nel programma.
  - Questa corrispondenza abilita l'**apertura del file**
- Al termine, la corrispondenza fra nome del file e variabile usata dal programma per operare su esso dovrà essere soppressa, mediante l'operazione di **chiusura del file**.

# La funzione `fopen`

```
FILE* fopen (char filename[], char mode[]);
```

- **`filename`** è il nome del file da aprire. Può contener il percorso completo o relativo.
- **`mode`** è una stringa che specifica la modalità di apertura
  - **"w":** file da aprire in scrittura; se il file esiste i suoi contenuti si perdono, altrimenti viene creato
  - **"r":** file da aprire in lettura; il file deve essere esistente (altrimenti restituisce **NULL**)
  - **"a":** file da aprire in scrittura; se il file esiste i dati scritti si aggiungono in coda ai contenuti esistenti, altrimenti il file viene creato
  - **"w+":** file da aprire in scrittura e lettura; se il file esiste i suoi contenuti si perdono, altrimenti viene creato
  - **"r+":** file da aprire in scrittura e lettura; il file deve essere esistente (altrimenti restituisce **NULL**)
  - **"a+":** file da aprire in scrittura e lettura; tutte le operazioni di scrittura aggiungono i dati scritti in coda ai contenuti esistenti. Il file viene creato, se non esistente

# La funzione `fopen`

- La funzione restituisce un valore di tipo **`FILE*`**, un puntatore ad una struttura **`FILE`** che gestisce lo stream associato al file aperto.
- Il valore deve ovviamente essere assegnato ad una variabile di tipo **`FILE*`**

```
#include <stdio.h>
int main(){
    FILE* fp;

    fp = fopen("dati.txt","r");
    …
```

# La funzione `fopen`

- Nel caso la chiamata a **`fopen`** non vada a buon fine (es. il file non esiste), il valore restituito è uguale a **`NULL`**, una costante definita in **`stdio.h`**, che costituisce un valore speciale che indica che il puntatore non sta puntando ad alcun oggetto.
- In questo modo possiamo verificare se l'operazione di apertura è andata a buon fine:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    FILE* fp;

    fp = fopen("dati.txt","r");
    if(fp == NULL){
      printf("Errore nell'apertura del file\n");
      exit(1);
    }
    … /* resto del programma */
```

# La funzione `fclose`

- Se l'apertura del file si è realizzata correttamente, è stata creata la corrispondenza tra il file *dati.txt* e la variabile **`fp`**.
- E' quindi possibile operare sul file tramite la variabile **`fp`.**
- Per terminare il collegamento tra file e variabile si invoca la funzione

  **`int fclose (FILE* stream);`**

  che chiude il file associato a **`stream`** e termina l'associazione tra file e variabile.

# Esempio di uso di `fopen` e `fclose`

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    FILE* fp;

    fp = fopen("dati.txt","r");
    if(fp == NULL){
      printf("Errore nell'apertura del file\n");
      exit(1);
    }

    /* in queste istruzioni si accede a dati.txt
       attraverso la variabile fp */
    …

    fclose(fp);
    /* da qui in poi fp non è più connessa a dati.txt */
    /* e può essere connessa ad un altro file */

    …
}
```

# Operazioni di lettura/scrittura su file

- Abbiamo di fatto considerato un file testo (per aprire un file binario va aggiunto il carattere **`b`** alla stringa che indica la modalità di apertura, es. **`"wb"`**, **`"r+b"`**, **`"rb+"`**, …)
- Possiamo quindi realizzare sui file le stesse operazioni di I/O che realizzavamo sul terminale di utente.

# Funzioni di lettura/scrittura su file di testo

```
int fprintf(FILE* stream,
            char format[], … );
```

- Molto simile alla funzione **`printf`** che già conosciamo:

```
int printf(char format[], … );
```

- L'unica differenza è che **`fprintf`** specifica lo stream su cui si effettua l'output; per il resto, le due funzioni operano nello stesso modo.

# Funzioni di lettura/scrittura su file di testo

- Analogamente è disponibile la funzione

```
int fscanf(FILE* stream,
           char format[], ... );
```

che opera in maniera simile a **`scanf()`**

- Un aspetto da tenere presente è come verificare che l'insieme dei dati presenti nel file sia terminato.
- Nel caso la **`fscanf`** cerchi di leggere oltre il limite di un file, restituisce un valore uguale ad **`EOF`** (End Of File), una costante definita in **`stdio.h`**

# Esempio

```
#include <stdio.h>
#define LENGTH 50
int main(){
        FILE *fp;
        char s[LENGTH+1];
        if( (fp = fopen("file1.txt","r") ) == NULL){
         printf("Impossibile aprire \n");
         exit(EXIT_FAILURE);
        }
          printf("le stringhe lette dal file sono:\n");
          while  (fscanf(fp, "%s", s)!=EOF)
              printf("%s\n", s);
         fclose(fp);
         return 0;
}
```

# La funzione `feof()`

```
int feof ( FILE * stream );
```

- La funzione **`feof`** verifica se si è raggiunta la fine del file associato a **`stream`** e, in tal caso, restituisce un valore diverso da zero (true in una valutazione booleana)

# Esempio

```
#include <stdio.h>
#define LENGTH 50
int main(){
        FILE *fp; char s[LENGTH+1];
        fp = fopen("prova.txt","r");
        if (fp==NULL)
                return EXIT_FAILURE; /* Errore di apertura */
        else {
                while (!feof(fp)) {
                        fscanf(fp, "%s", s);
                        printf("%s\n",s);
                }
                fclose(fp);
        }
}
```