

*Fondamenti di Informatica e Laboratorio T-AB*  
*Ingegneria Elettronica e Telecomunicazioni*

---

# Lab 10

## Gestione file di testo

# Esercizio 1

---

- Realizzare un programma che, aperto un file di testo di nome “Prova.txt” in modalità “scrittura”, provveda a leggere da input delle parole separate da spazi (stringhe di al più 63 caratteri) e le scriva nel file di testo.
- Il programma termina quando l’utente inserisce la parola “fine”. Si abbia cura di chiudere il file prima di terminare definitivamente il programma.
- Si controlli il corretto funzionamento del programma accedendo direttamente al file di testo con un editor (ad es. Notepad).

# Esercizio 1 - Soluzione

---

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

int main(void) {
    FILE * fp;
    char s[64];

    if ((fp=fopen("prova.txt", "w")) == NULL)
        exit(1);

    do {
        scanf("%s", s);
        if (strcmp("fine", s) != 0)
            fprintf(fp, "%s ", s);
    } while (strcmp("fine", s) != 0);

    fclose (fp);

    return (0);
}
```

## Esercizio 2

---

- Realizzare un modulo `tempi.h/tempi.c` per la gestione del tempo di giro su un circuito di MotoGP.
- Il modulo deve definire (utilizzando la `typedef`) la struttura “Tempo”, con i tre campi “minuti”, “secondi”, “millesimi”.
- Il modulo deve definire la funzione:

```
int differenza(Tempo t1, Tempo t2);
```

Che restituisce (in millesimi) la differenza tra `t1` e `t2`.

## Esercizio 2

---

- Il modulo deve poi definire la funzione:

```
int leggiTempi(FILE* file, Tempo partenza[],  
              Tempo arrivo[]);
```

- Che legge da un file di testo il tempo di partenza e di arrivo di una sequenza di giri e li inserisce nei due array “partenza” e “arrivo”. La funzione deve restituire il numero di elementi letti. Si supponga che il file di testo contenga una riga per ogni coppia partenza-arrivo, e che minuti, secondi e millesimi siano separati da spazi o tabulazioni. Si supponga che non vi possano essere più di 100 coppie partenza-arrivo. Si supponga che ogni linea contenga sempre 6 campi (min/sec/mill) per le due misure.

## Esercizio 2

---

- Si scriva un programma che legge dal file di testo “tempi.txt” una serie di tempi di partenza ed arrivo (quindi, di tempi di giro), mediante la funzione “leggiTempi”.
- Il programma stampi poi a video il numero di giri effettuati in un tempo inferiore a quello impiegato per il primo giro.

## Esercizio 3

---

Sia dato il file di testo "dati.txt" contenente i dati relativi agli studenti immatricolati al primo anno della Facoltà di Ingegneria

In particolare, le informazioni sono memorizzate nel file "dati.txt" come segue: ognuna delle linee del file contiene i dati relativi ad un nuovo studente; in particolare:

- 1 Matricola: un intero che indica il numero di matricola dello studente
- 2 CdL: un intero che indica il corso di laurea (CdL) dello studente (es. 2145)

# Esercizio 3

---

Sia dato un secondo file di testo, “indirizzi.txt” che contiene, invece, l’indirizzo di ogni studente, e in particolare, in ogni linea, separati da uno spazio:

- **Matricola**: il numero di matricola dello studente (un intero)
- **Nome**: il nome dello studente, al più 20 caratteri senza spazi
- **Cognome**: il cognome dello studente, al più 30 caratteri senza spazi
- **Città**: una stringa che riporta la città di residenza dello studente, al più 30 caratteri senza spazi



## Esercizio 3

---

Si sviluppi il modulo “studenti.h/studenti.c”, che definisca (mediante typedef) la strutture:

- DatiCorso, per contenere le informazioni di “dati.txt”
- DatiIndirizzo, per quelle di “indirizzi.txt”
- DatiStudiante, per contenere tutte le informazioni su uno studente.

1. Il modulo deve definire le funzioni:

```
int leggiCorsi(FILE* file, DatiCorso corsi[]);
```

```
int leggiIndirizzi(FILE* file, DatiIndirizzo indirizzi[]);
```

Per riempire i due vettori “dati” ed “indirizzi” da due file di testo (tipo “dati.txt” ed “indirizzi.txt”). Le funzioni restituiscono il numero di elementi letti.

## Esercizio 3

---

Si scriva un programma in linguaggio C che:

1. A partire dai file "dati.txt" e "indirizzi.txt" costruisca un array "T" di strutture DatiStudente. Si assuma che non vi siano mai più di 1000 studenti.
2. A partire dall'array "T", e dato da input un intero C che rappresenta un CdL, **stampi la percentuale di studenti** (rispetto al numero totale delle matricole) **iscritti al corso C** [Ad esempio, se il numero totale delle matricole è 1000, e quello degli studenti iscritti a C è 200, il programma stamperà "20%"]
3. Scriva su un terzo file di testo "bologna.txt", nome, cognome e numero di matricola di tutti gli studenti che abitano a Bologna

# Movimenti Sullo Stream

---

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

```
int fseek(FILE *stream, Long offset, int whence);  
void rewind(FILE *stream);
```

La funzione **fseek()** imposta l'indicatore di posizione del file. La prossima operazione di I/O su **stream** verrà eseguita dalla nuova posizione impostata.

La posizione è calcolata aggiungendo **offset** (che può assumere anche valori negativi) a **whence**.

**whence** può valere **SEEK\_SET**, **SEEK\_CUR** o **SEEK\_END** per specificare rispettivamente il riferimento dall'inizio file, dalla posizione corrente o dalla fine file. In caso di successo, **fseek()** ritorna 0 e viene cancellato l'indicatore di fine file. In caso di fallimento ritorna -1.

La funzione **rewind()** imposta l'indicatore di posizione del file puntato da **stream** all'inizio file.

È equivalente a: `(void)fseek(stream, 0L, SEEK_SET)` eccetto al fatto che viene resettato anche l'indicatore di errore.

# Esercizio 4

---

Sono dati due file di testo `cineprogramma.txt` e `sale.txt` che contengono, rispettivamente, il programma settimanale dei film in proiezione e le descrizioni delle sale in città. Più precisamente, ogni riga di `cineprogramma.txt` contiene, nell'ordine:

- **titolo del film** (non più di 30 caratteri senza spazi), uno e un solo spazio di separazione
- **nome della sala** (non più di 20 caratteri senza spazi), uno e un solo spazio di separazione
- **3 orari** di inizio proiezione (3 numeri interi separati da caratteri '-'), terminatore di riga

mentre ogni riga di `sale.txt` contiene, nell'ordine:

- **nome della sala** (non più di 20 caratteri senza spazi), uno e un solo spazio di separazione
- **costo del biglietto** (numero reale), terminatore di riga

# Esercizio 4

---

## **cinoprogramma.txt:**

TheKingdom Nosadella 18-20-22  
Dogville Fellini 17-20-22  
OttoEMezzo Capitol 17-20-23  
BreakingWaves Modernissimo 15-19-23

## **sale.txt:**

Capitol 6.00  
Fellini 5.50  
Modernissimo 6.00  
Nosadella 6.50

## Esercizio 4

---

- 1) Si scriva una procedura `load(...)` che riceva come parametri di ingresso **due puntatori** a file di testo e “restituisca” come parametri di uscita **un vettore `y` contenente strutture `film`** (titolo film, costo biglietto) e **il numero degli elementi `N` inseriti in `y`.**

Per semplicità si supponga che tutte le sale contenute nel primo file siano presenti anche nel secondo, e una sola volta. Si supponga che non vi possano essere più di 100 film e 10 sale.

Suggerimento: si definiscano strutture e si riempiano array temporanei per raggiungere l'obiettivo.

## Esercizio 4

---

2) Si scriva un programma C che, utilizzando la procedura `load(...)` precedentemente definita, **riempia un vettore prezzi** (supposto di dimensione massima `DIM=100`) **di strutture film di cui sopra**, derivanti dai file “`cinoprogramma.txt`” e “`sale.txt`”

Il programma deve poi stampare a terminale tutti gli elementi di **prezzi** il cui costo del biglietto è **inferiore alla media di tutti i costi caricati nel vettore**

# Esercizio 5

---

Sono dati due file di testo `anagrafe.txt` e `fatture.txt` che contengono, rispettivamente, i dati anagrafici di alcuni clienti e l'elenco delle fatture

Più precisamente, ogni riga di `anagrafe.txt` contiene, nell'ordine:

- **Codice Cliente** (numero intero) , uno e un solo spazio di separazione
- **Nome del cliente** (non più di 30 caratteri senza spazi), uno e un solo spazio di separazione
- **Città** (non più di 20 caratteri senza spazi), uno e un solo spazio di separazione

Ogni cliente compare nel file di `anagrafe` una ed una sola volta

Ogni riga di `fatture.txt` contiene, nell'ordine:

- **Codice Cliente** (numero intero), uno e un solo spazio di separazione
- **Numero della fattura** (numero intero), uno e un solo spazio di separazione
- **Importo della fattura** (numero reale), uno e un solo spazio di separazione
- **Un carattere** ( 'p' se la fattura è stata pagata, 'n' altrimenti), terminatore di riga



# Esercizio 5

---

**anagrafe.txt:**

```
1 Chesani Bologna
2 Bellavista Bologna
3 Mello Bologna
```

**fatture.txt:**

```
1 23 54.00 p
1 24 102.00 n
3 25 27.00 p
1 26 88.00 n
```

# Esercizio 5

---

1) Si scrivano due funzioni `leggiClienti(...)` e `leggiFatture(...)` che ricevano ciascuna come parametro di ingresso **un puntatore** a file di testo e che abbiano due parametri di uscita: a) **un vettore y di strutture atte a memorizzare (rispettivamente) clienti e fatture** lette dal file di testo; b) il numero di elementi letti.

## Esercizio 5

---

2) Si scriva una funzione `contaDebiti(...)` che riceva come parametri di ingresso **due array** (rispettivamente di strutture `Cliente` e `Fattura`), con le relative dimensioni logiche. La funzione deve avere come parametri di uscita **un vettore `y` contenente strutture `Debito`** (nome cliente, importo) e il **numero degli elementi `N`** inseriti in `y`: questo vettore deve contenere solo i dati relativi a **fatture non pagate**.

## Esercizio 5

---

3) Si scriva un programma C che, utilizzando le funzioni precedentemente definite, riempia un vettore **debitori** di strutture **debito** di cui sopra, derivanti dai file **anagrafe.txt** e **fatture.txt**. Si supponga non vi siano più di 100 clienti e 1000 fatture.

Il programma stampi poi a video il valore del debito per tutti clienti, quindi chieda all'utente il nome di un cliente e stampi il numero di fatture (non pagate) intestate a tale cliente.