Gestione dei dati tramite FILE

- i programmi nella loro esecuzione usano dati in input (stdin) per inizializzare variabili e generano dati come output (stdout) delle istruzioni eseguite, tramite la linea di comando dell'utente (es. printf e scanf) → per ovviare a questo problema e salvare sia variabili in input che i risultati output si possono usare degli archivi di memoria permanente su cui vengono caricati i dati in modo automatico per un riutilizzo futuro, detti file

- i **file**: sono archivi di memoria che consentono il salvataggio persistente di dati su un suopporto fisico ed il caricamento di dati da supporto fisico (es. hard disk, usb, dvd)

- i file sono gestiti dal sistema operativ**o t**ramite un **file system** e funzioni dedicate a cui si può accedere tramite la libreria <stdio.h>

- i file sono **strutture dati sequenziali**, sono letti e scritti some uno **stream di byte**: file binario → sequenza di byte non interpretata e file di testo → sequenza di caratteri interpretata regolata da caratteri separatori come il newline '\n';

- in C tutte le periferiche (qualsiasi dispositivo di input output) sono viste come file: stdin=tastiera (scanf), stdout=monitor(printf), → quindi si può leggere e scrivere da/su ogni periferica con le stesse modalità, quelle dei file.
- i **tre stream** principali che si aprono in automatico quando inizia un programma sono:
  1) **standard input** (tastiera), 2) **standard output** (stampa a video), 3) **standard error** (per i messaggi di errore)

- per essere utilizzato il file deve essere aperto e chiuso, ovvero aprire e chiudere la comunicazione tra programma e file, la variabile che consente di inizializzare la comunicazione tra proramma e file è : **`FILE*`** (puntatore a una struttura di tipo file)
  1) dichiaro la variabile `FILE*`
  2) apro il file con `fopen`;
  3) ...
  4) chiudo il file `fclose(nome_file)`

- il **descrittore** è una struttura dati di tipo `struct FILE`, un indice intero in un array del sistema operativo chiamato **tabella dei file aperti** in cui il sistema operativo associa ai programmi in esecuzione un **blocco di controllo del file** (**FCB**) in cui sono memorizzate tutte le informazioni usate dal sistema operativo per amministrare un particolare file.
  Il descrittore contiene:
  a) la modalità d'uso: `'w'`=write, `'r'`=read, `'a'`=append,
  b) l'indicatore della posizione della testina nel file,
  c) la dimensione del file e la posizione di fine documento (variabile **`eof`**, end of file),
  d) eventuali errori

  → l'apertura restituisce un puntatore a un descrittore di tipo `FILE*` che contiene le informazioni sopracitate

- Funzioni del puntatore `FILE *fp`:

  ★ <u>apertura</u> o creazione nuovo file: `FILE* fopen(char* "nomefile", char* "modalità");`
  →crea un puntatore al descrittore `FILE *fp`,
  i parametri sono una stringa recante il nome del file che può includere informazioni sul percorso/path per localizzare il file nel file system, una stringa recante la modalità di apertura del file (`'w', 'r', 'a'`);
  se c'è un errore fopen restituisce `null`, la funzione va sempre verificata

  ★ <u>chiusura</u>: `int fclose(FILE *fp)` → restituisce un intero che dice 0 = buon fine, EOF = errore

- ☆ errore: `int ferror(FILE *fp)` → gestisce gli errori verificatisi dopo l'invocazione della variabile restituendo un intero: 0 = no errori, altrimenti restituisce il codice specifico dell'errore;

- ☆ fine file: `int feof(FILE *fp)` → restituisce un intero che indica la posizione della testina nel file, se è alla fine restituisce 0 (falso);

- modalità di apertura del file di testo:
  `read 'r'` → il file è aperto in modalità lettura senza modifica, la testina è posizionata all'inizio del file
  `write 'w'` → il file viene aperto in modalità scrittura che permette la modifica o il file viene creato, nb! se il file è già esistente cancella il contenuto precedente e la testitna si posiziona ll'inizio del file per scriverne uno nuovo;
  `append 'a'` → il file è aperto in modalità scrittura in appendice o viene creato, se il file è già esistente esso non viene sovrascritto ma la testina si posiziona alla fine del contenuto già presente per aggiungerne di nuovo.

- file binari: le modalità di apertura devono essere seguite da una b che specifica il tipo di file binario `'rb'`, `'wb'`, `'ab'`;

- modalità miste: `'r+'/'r+b'` lettura e scrittura, `'w+'/'w+b'` crea un file in lettura e scrittura, `'a+'/'a+b'` crea o apre un file in lettura e scrittura appendice (non le vedremo);

- file di testo: ci sono tre modi di lettura e scrittura per i file di testo
  - ☆ `fscanf()/ fprintf()` per scrivere un **formato** alla volta (analoghe a scanf() e printf();
  - ☆ `fgetc()/ fputc()` un **carattere** alla volta (analoghe a getchar() e putchar());
  - ☆ `fgets()/ fputs()` una **linea di testo** alla volta (analoghe a gets() e puts());

Funzioni di fibreria per FILE

- funzioni di lettura e scrittura per formati

  - ☆ `int fprintf(FILE *fp; str_di_formato, espressioni);`
  - ☆ `int fscanf(FILE *fp; str_di_formato, indirizzi di variabili);`

  → entrambe restituiscono un intero pari al numero di elementi letto o scritti, 0 = errore, -1 = sono oltre la eof, imposrtante è la stringa di formato che consente la corretta lettura
  (es. `fprintf(stdout, "%d %s %.2f\n", account, name, balance);`
  `fscanf (fp, "%d%29s%1f", &account, name, &balance);`)

- funzioni di lettura e scrittura per un singolo carattere
  da stdin/stdout
  - ☆ `int getchar (void);` // legge un singolo carattere da standard input e lo restituisce come un intero
  - ☆ `int putchar (int c);` //scrive un carattere su sandard output

  da file
  - ☆ `int fgetc (FILE *fp);` // legge un carattere dal file e lo restituisce come intero
  - ☆ `int fputc(int c, FILE *fp);` //scrive un carattere sul file

- funzioni di lettura e scrittura di linee di testo
  da stdin/stdout
  - ☆ `char *gets(char *s);` // s è l'array in cui copiare la stringa letta da stdin, risulterà terminata da un '\0', non si può limitare la dimensione di dati in input quindi restituisce s se è stata eseguita con successo, null se è errore (non crontrolla che la stringa s sia abbastanza grande)
  - ☆ `int puts(char *s);` //scrive la stringa s escluso il '\0' a cui sostituisce un '\n', restituisce n >= 0 se ok, eof se errore

da file

- ☆ `char *fgets(char *s, int n, FILE *fp);` //legge da file al più n-1 caratteri o fino a '\n' o eof, se incocntra '\n' lo inserisce tra gli n-1 caratteri e mette alla fine il terminatore '\0', restituisce s o NULL se errore;
- ☆ `int fputs(char *s, FILE *fp);` //si limita a scrivere una stringa omettendo il carattere '\0' e senza aggiungere '\n', restituisce 0 se ok e eof se errore

- funzioni di lettura e scrittura per blocchi di byte: permettono di leggere o scrivere un intero blocco di dati testuali o binari, per scrivere su file una struct o un array senza cicli:

  - ☆ `size_t fread (void *buffer, size_t size, size_t count, FILE *stream);` //legge un blocco di dati da file,
  - ☆ `size_t fwrite (const void *buffer, size_t size, size_t count, FILE *stream));` //scrive un blocco di dati su file

  → size_t = array di piccole dimensioni, *buffer = puntatore al tipo di struttura dati da cui voglio leggere, size_t size = dimensione in byte del singolo elemento da leggere, size_t count = numero di elementi,
  FILE *stream = file da cui voglio leggere o scrivere. Entrambe restituiscono un intero pari al numero di elementi letti o scritti

  (es.
  ```
  int a[10];
  int n = fwrite(a, sizeof(a[0]), dim_a, fp);
  int n = fread(a, sizeof(a[0]), dim_a, fp))
  ```

- funzioni per accesso diretto a byte: a ogni stream è essociata una posizione, che all'apertura viene impostata all'inizio del file e spostata automaticamente con ogni operazione di lettura e scrittura. Si può tuttavia accedere ad uno specifico byte con la funzioni:

  - ☆ `int fseek (FILE *fp, long offset, int refpoint);` //imposta la posizione corrente ad un valore pari ad uno spostamento positivo o negativo in byte (offset), calcolato rispetto ad un punto di partenza che puo essere:
    - ☆ `refpoint = SEEK_SET`, se inizio file, (costanti di stdio.h)
    - ☆ `refpoint = SEEK_CUR`, se posizione corrente,
    - ☆ `refpoint = SEEK_END`, fine del file, restituisce 0 se lo spostamento fallisce;

  - ☆ `long int ftell(FILE *fp );` // restituisce la posizione della testina rispetto all'inizio, nei file binari è il numero di byte dall'inizio e nei file testuali dipende da come sono codificati i char (2 byte su windows, 1 byte su macos e linus), long int è un tipo di dato che indica un intero di grandi dimensioni;

  - ☆ `void rewind(FILE *fp);` // riavvolge il file e la testina dalla posizione corrente torna all'inizio, (equivale a `fseek (fp, 0, SEEK_SET)`);

Esempi:

(es fscanf/fprintf: scrivere in c un programma che aperto il file “ris.txt” copi in una variabile di tipo t_parteA il suo contenuto)

```c
#define LUN_STRINGA (50+1)
#define NUM_STUDENTI (300)
typedef struct {
            char nome[LUN_STRINGA];         // nome
            char cognome[LUN_STRINGA];      // cognome
            int matricola;          // numero di matricola
            int domande[8];         // array per 8 risposte
} t_risultati;
typedef t_risultati t_parteA[NUM_STUDENTI];    // tipo array con
NUM_STUDENTI celle

int main() {
    t_parteA parteA; // tabella risultati
    FILE* fp; // puntatore a file
    fp = fopen("ris.txt","r");
    if (fp == NULL) { // controllo errore di apertura
        printf("\n Errore durante l'apertura del file: ris.txt \n");
        return -1;
    }
    int count = 0; // indice per scorrere l'array parteA (e per
    contare le righe del file)
    while( feof(fp) == 0 && count < NUM_STUDENTI ) { // lettura del
    file riga per riga
        fscanf(fp, "%s", parteA[count].nome);
        fscanf(fp, "%s", parteA[count].cognome);
        fscanf(fp, "%d", &parteA[count].matricola);
        for(int i = 0; i < 8; i += 1)
                fscanf(fp, "%d", &parteA[count].domande[i]);
        count++; // incremento contatore
    }
    fclose(fp); // chiusura file
    // qui possono essere elaborati i dati letti dal file
    // il contatore count contiene il numero di Parti A consegnate
    return 0;
}
```

(es. fgetc: visualizzazione file di testo)

```c
#include <stdio.h>

int main () {

  FILE *file = NULL;          // dichiarazione variabile per file
  int carattere;              // carattere da leggere
  // apertura file in modalita' lettura
  file = fopen("/Users/nomeutente/Desktop/FileDaLeggere.txt", "r");

  if (!file) {        // controllo se il file non può essere aperto
    printf("Errore apertura file.\n");
    return 1;
  }

  carattere = fgetc(file);    // lettura primo carattere
  while (carattere != EOF) {  // finche' non si arriva alla fine
    printf("%c", carattere);          // stampa carattere
    carattere = fgetc(file);          // lettura prossimo carattere
  }

  fclose(file);                       // chiusura file
  return 0;
}
```

(es. fgets: lettura di stringa da uno stream)

```c
#include <stdio.h>

int main () {

  FILE *file = NULL;          // dichiarazione variabile per file
  int carattere;              // carattere da leggere
  // apertura file in modalita' lettura
  file = fopen("/Users/nomeutente/Desktop/FileDaLeggere.txt", "r");

  if (!file) {        // controllo se il file non può essere aperto
    printf("Errore apertura file.\n");
    return 1;
  }

  carattere = fgetc(file);    // lettura primo carattere
  while (carattere != EOF) {  // finche' non si arriva alla fine
    printf("%c", carattere);          // stampa carattere
    carattere = fgetc(file);          // lettura prossimo carattere
  }

  fclose(file);                       // chiusura file
  return 0;
}
```