I File in C

1. <stdio.h>

Il linguaggio C non contiene alcuna istruzione che permetta di compiere operazioni di input (ingresso o lettura) e output (uscita o scrittura) dei dati. Operazioni di questo tipo vengono svolte mediante chiamate a funzioni definite nella libreria standard all'interno dell'header **<stdio.h>**. Tali funzioni rendono possibile la lettura/scrittura in modo indipendente dalle caratteristiche proprie dei dispositivi di I/O.

Le stesse funzioni possono essere utilizzate, ad esempio, sia per leggere un valore dalla tastiera sia per leggere un valore da un dispositivo di memoria di massa. Lo stesso vale per le funzioni di scrittura: le stesse funzioni possono essere utilizzate sia per la visualizzazione sullo schermo sia per scrivere un valore su un disco o una stampante, operando su sistemi operativi differenti. Ciò è possibile poiché il sistema di I/O di C è caratterizzato da un'interfaccia, lo *stream*, indipendente dal dispositivo effettivo, che si interpone tra il programmatore e il dispositivo stesso.

L'header **<stdio.h>** definisce inoltre diverse *macro* e dichiara alcuni tipi specifici.

> La direttiva **#define** viene generalmente utilizzata per associare identificatori significativi a costanti, parole chiave e istruzioni o espressioni di uso comune. Gli identificatori che rappresentano costanti sono denominati *costanti simboliche*. Gli identificatori che, invece, rappresentano istruzioni o espressioni sono denominati **macro** (da *macroistruzione*).

Tra le macro citiamo le seguenti:

- **NULL**, che indica una costante di tipo puntatore nullo;
- **EOF**, che indica la fine di un file (*end of file*);
- **stdin**, che indica un puntatore verso lo *standard input*;
- **stdout**, che indica un puntatore verso lo *standard output*;
- **stderr**, che indica un puntatore verso lo *standard error*.

Per quanto riguarda i tipi abbiamo **FILE**, utilizzato per contenere informazioni su un file.

2. File e stream

Un **file** è un insieme, non necessariamente omogeneo, di dati correlati, identificato da un nome, memorizzato permanentemente su un supporto di memoria non volatile e avente vita indipendente dal programma (o dai programmi) utilizzato/i per la sua creazione e/o modifica. I file possono essere classificati in due categorie fondamentali: i file di testo e i file binari.

Un **file di testo** è una sequenza di linee; ogni linea possiede zero o più caratteri ed è terminata da un marcatore di *end of line* che è specifico del sistema operativo in uso. Per esempio, nei sistemi Windows tale marcatore è rappresentato da un carattere di *carriage return* ('\r', in ASCII 0x0D) cui segue un carattere di *new line* o *line feed* ('\n', in ASCII 0x0A). Nei sistemi Unix, invece, tale marcatore è rappresentato dal solo carattere di *new line*.

Un **file binario** è un file i cui byte non rappresentano necessariamente dei caratteri ma possono rappresentare codice macchina, dati numerici, codifiche di immagini, di suoni, di video e così via. Questi byte non sono dunque leggibili in modo significativo da un essere umano oppure modificabili con un semplice editor di testo. Inoltre, un file binario non è diviso in righe di testo; un file binario viene letto a blocchi di byte che sono interpretati in funzione del tipo di dato che essi rappresentano dall'applicazione per la quale il file viene creato.

Le operazioni fondamentali che possono essere eseguite sui file sono:

- **apertura**: stabilisce un collegamento tra il file registrato in memoria di massa e l'area di memoria centrale (buffer) ad esso associata per consentire le operazioni di I/O;
- **chiusura**: completa il trasferimento dei dati dal buffer verso la memoria di massa e interrompe il collegamento liberando la memoria centrale.
- **lettura** (*r*): i dati memorizzati sulla memoria di massa vengono trasferiti nel buffer di memoria
- centrale per l'elaborazione;
- **scrittura** (*w*): i dati contenuti nel buffer vengono memorizzati sul supporto fisico;
- **aggiornamento**: i dati possono essere modificati durante l'elaborazione e le modifiche vengono salvate sul supporto fisico.

> Un **buffer** è un'area di memoria temporanea utilizzata, in caso di operazioni di input e output, per memorizzare dei dati prima che questi vengano effettivamente processati (per esempio, visualizzati su uno schermo oppure scritti in un file su un hard disk). In pratica, i byte da leggere o da scrivere sono raggruppati in "blocchi", e solo dopo il verificarsi di certe condizioni (per esempio, quando il buffer è pieno) quei blocchi sono realmente letti o realmente scritti.

Uno **stream** (flusso) è una sorgente (input source o input stream) o una destinazione (output source o output stream) di dati che possono essere associati ad un disco oppure ad altre periferiche. Uno stream viene connesso ad un file o a un device tramite un'operazione di *apertura*; la connessione viene interrotta *chiudendo* lo stream. L'apertura di un file fornisce un puntatore a una struttura di tipo FILE, che contiene tutte le informazioni necessarie per la gestione dello stream.

Questa connessione logica permette di trattare in modo uniforme le varie tipologie di input o di output che possono avere delle caratteristiche proprie e differenti (per esempio, un input può provenire da una comune tastiera, da un sintetizzatore vocale, da uno scanner, da un file e così via; allo stesso modo, un output può fluire verso un comune monitor, una stampante, un plotter, un file e così via).

Di default, l'header <stdio.h> fornisce tre stream predefiniti, indicati come *standard stream*, che sono già disponibili (non è necessario aprirli o chiuderli) e che sono rappresentati dalle macro già citate semplici:

- **standard input**: stream associato a una sorgente di input convenzionale che tipicamente è la tastiera;
- **standard output**: stream associato a una destinazione di output convenzionale che tipicamente è lo schermo;
- **standard error**: stream associato a una destinazione di output utilizzata per scrivere messaggi di diagnostica che tipicamente è lo schermo.

Nei moderni sistemi operativi è possibile "forzare" gli stream citati, associandoli a dei file piuttosto che ai device prima menzionati, mediante un procedimento noto con il termine di **redirezione dell'input** o **redirezione dell'output**.

3. Apertura e chiusura di file

Il tipo **FILE** è definito nell'header <stdio.h> mediante un *typedef* di una struttura, la quale è costituita da una serie di membri deputati a contenere informazioni essenziali per il controllo di uno stream quali: un indicatore di posizione del carattere corrente, un puntatore associato a un eventuale buffer, un indicatore di stato per un eventuale errore di lettura o di scrittura, un indicatore che specifica se si è raggiunta la fine di un file e così via.

Per implementare un file in C è quindi necessario dichiarare un puntatore di tipo FILE. La sintassi è la seguente:

```
FILE *puntatore;
```

Prima di accedere ad un file è necessario aprirlo: l'operazione di apertura compie le azioni preliminari necessarie affinché si possa avere accesso al file (in lettura o in scrittura). L'operazione di apertura inizializza il puntatore al file ed avviene tramite la funzione:

```
FILE *fopen(char *name, char *mode);
```

dove:

- **name** è una stringa che rappresenta il *pathname* (assoluto o relativo) del file nel file system;
- **mode** esprime la modalità di accesso scelta: ▶

Modo	Significato
"r"	Apre un file di testo per la lettura.
"w"	Crea un file di testo per la scrittura oppure lo svuota se già esiste.
"a"	Apre o crea un file di testo per la scrittura a partire dalla sua fine (*append mode*).
"rb"	Apre un file binario per la lettura.
"wb"	Crea un file binario per la scrittura oppure lo svuota se già esiste.
"ab"	Apre o crea un file binario per la scrittura a partire dalla sua fine (*append mode*).
"r+"	Apre un file di testo per l'aggiornamento (lettura e scrittura).
"w+"	Crea un file di testo per l'aggiornamento oppure lo svuota se già esiste.
"a+"	Apre o crea un file di testo per l'aggiornamento a partire dalla sua fine (*append mode*).
"rb+"	Apre un file binario per l'aggiornamento (lettura e scrittura).
"wb+"	Crea un file binario per l'aggiornamento oppure lo svuota se già esiste.
"ab+"	Apre o crea un file binario per l'aggiornamento a partire dalla sua fine (*append mode*).

Se eseguita con successo, l'operazione di apertura restituisce come risultato un puntatore al file aperto. Se l'operazione di apertura fallisce (ad esempio, perché il file non esiste oppure perché non si hanno i permessi necessari per eseguirla), *fopen* restituisce il valore *NULL*, quindi è possibile effettuare un controllo sul puntatore per interrompere l'esecuzione del programma.

Al termine di una sessione di accesso al file, esso deve essere chiuso. L'operazione di chiusura è indispensabile per svuotare il buffer associato al file e si realizza con la funzione **fclose**:

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

dove *fp* rappresenta il puntatore al file da chiudere.

Se l'operazione di chiusura avviene in modo regolare, la funzione restituisce il valore 0. Se la chiusura non è andata a buon fine, restituisce la costante *EOF*.

Esempio:

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

int main(void)
{
    FILE *fp = fopen("test.txt", "r");

    if (fp == NULL) {
        printf("File opening failed");

        return EXIT_FAILURE;
    }

    fclose(fp);

    return 0;
}
```

4. File di testo

4.1 Lettura e scrittura di variabili

Per leggere dati da un file, si usa la funzione **fscanf**:

```
int fscanf (FILE *fp, formato, variabili);
```

dove:

- **fp** è il puntatore al file
- **formato** indica l'elenco di formati dei dati da leggere
- **variabili** è la lista degli indirizzi delle variabili a cui assegnare i valori letti.

La funzione può restituire il numero di elementi letti, oppure un valore negativo in caso di errore.

test.txt:

```
33 M 173.5
34 F 165.5
23 M 181.0
44 F 175.5
```

Esempio di lettura di una sola riga:

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

int main(void)
{
    FILE *fp = fopen("test.txt", "r");
    int i;
    char c;
    float f;

    if (fp == NULL) {
        printf("File opening failed");

        return EXIT_FAILURE;
    }
    else {
        fscanf(fp, "%d %c %f", &i, &c, &f);
        printf("%d %c %.2f", i, c, f);
    }

    fclose(fp);

    return 0;
}
```

Quando apriamo un file, il relativo stream associato ha un indicatore di posizione, una sorta di marcatore o puntatore che tiene traccia di dove, all'interno di quel file, avverrà la prossima operazione di lettura o di scrittura.

Normalmente, quando si apre un file per una lettura o per una scrittura, questo indicatore ha un valore impostato al suo inizio; quando si apre, invece, per un'operazione di *append*, l'indicatore ha un valore impostato alla sua fine.

In seguito, qualsiasi operazione si effettui con quel file, per esempio una lettura, la stessa avverrà in modo sequenziale, ossia sarà letto un carattere dopo l'altro o una riga dopo l'altre, e l'indicatore di posizione si aggiornerà, incrementandosi automaticamente.

Durante la fase di accesso ad un file è possibile verificare il raggiungimento della fine del file con la funzione **feof**:

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

dove **fp** rappresenta il puntatore al file. La funzione controlla se è stata raggiunta la fine del file nell'operazione di lettura o scrittura precedente. Restituisce il valore 0 se non è stata raggiunta la fine del file, altrimenti un valore diverso da zero.

Esempio di lettura dell'intero file:

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

int main(void)
{
    FILE *fp = fopen("test.txt", "r");
    int i;
    char c;
    float f;

    if (fp == NULL) {
        printf("File opening failed");

        return EXIT_FAILURE;
    }
    else {
        while (!feof(fp)) {
            fscanf(fp, "%d %c %f", &i, &c, &f);
            printf("%d %c %.2f\n", i, c, f);
        }
    }

    fclose(fp);

    return 0;
}
```

Oppure:

```
    ...
    while (fscanf(fp, "%d %c %f", &i, &c, &f) != EOF) {
           printf("%d %c %.2f\n", i, c, f);
    }
    ...
```

EOF è una macro definita nell'header <stdio.h> che deve espandersi con un'espressione costante intera di tipo int con un valore negativo (per esempio, in GCC, EOF si espande come -1). EOF è ritornata da diverse funzioni di I/O per indicare che non vi è più nessun input da processare dal corrente stream.

La funzione **fprintf**, invece, scrive sul file una lista di variabili o espressioni.

```
int fprintf (FILE *fp, formato, variabili);
```

dove:

- **fp** è il puntatore al file
- **formato** indica il formato dei dati da scrivere
- **variabili** è la lista dei valori (o espressioni) da scrivere.

Anche in questo caso, la funzione restituisce il numero di elementi scritti, oppure un valore negativo in caso di errore.

Nota bene: gli stream C su sistema operativo Windows convertono automaticamente '\n' in '\r\n' in output e convertono '\r\n' in '\n' in input).

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 3

int main(void)
{
    FILE *fp = fopen("result.txt", "w");
    int i, age;
    char sex;

    if (fp == NULL) {
        printf("File opening failed");

        return EXIT_FAILURE;
    }
    else {
        for (i = 0; i < MAX; i++) {
            printf("Sex: ");
            scanf("%c", &sex);

            printf("Age: ");
            scanf("%d", &age);

            getchar();

            fprintf(fp, "Sex: %c - age: %d\n", sex, age);
        }
    }

    fclose(fp);

    return 0;
}
```

4.2 Lettura e scrittura di caratteri

La funzione **fgetc** legge un carattere dallo stream indicato come parametro. Ritorna il carattere letto oppure EOF se è stato riscontrato un errore o la fine del file e poi sposta il cursore avanti di un carattere. Il tipo ritornato è un intero poiché la funzione, oltre a ritornare ogni carattere del tipo *char*, deve ritornare anche EOF.

```
int fgetc(FILE *fp);
```

La funzione **fputc** scrive il carattere indicato come primo parametro sullo stream indicato come secondo parametro, eseguendo un cast a unsigned char. Ritorna il carattere scritto se la funzione ha successo, altrimenti EOF in caso di errore.

```
int fputc(int c, FILE *fp);
```

4.3 Lettura e scrittura di stringhe

La funzione **fgets:**

```
char *fgets(char *str, int count, FILE *stream );
```

legge dallo stream puntato da *stream* al massimo il numero di caratteri specificati da *count* (meno 1) e li memorizza nell'array puntato da *str*. Se durante la lettura è incontrato un carattere di *new line* oppure un'*end of file* la lettura è interrotta, ossia nessun carattere è più parte dell'input; l'eventuale carattere *new line* è comunque trattenuto. In più, il carattere nullo '\0' è sempre posto nell'array str dopo la lettura dell'ultimo carattere.

Ritorna un puntatore a *str* se la lettura è occorsa correttamente; un puntatore nullo se è occorso un'*end-of-line* e nessun carattere è stato posto nell'array oppure vi è stato un errore di lettura. In definitiva questa funzione è utile per leggere un'intera riga di testo.

La funzione **fputs**:

```
int fputs(const char *str, FILE *stream);
```

scrive la stringa *str* su *stream*, ignorando il terminatore. Ritorna un numero non-negativo in caso di successo e EOF in caso di errore.

cat.c - simulazione del comando *cat* della BASH:

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

#define MAX 80

int main(int argc, char *argv[])
{
    FILE *fp = fopen(argv[1], "r");
    char buf[MAX];

    if (fp == NULL) {
        printf("File opening failed");

        return EXIT_FAILURE;
    }
    else {
        while (fgets(buf, MAX, fp) != NULL) {
            printf("%s", buf);
        }
    }

    fclose(fp);

    return 0;
}
```