

Il linguaggio C

Breve panoramica su stdio.h

Input/Output: `stdio.h`

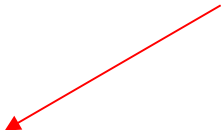
- Contiene definizioni di costanti legate all' I/O
 - es. EOF (end of file)
#define EOF (-1)
valore restituito alla fine di un file
- Contiene la definizione della struttura che descrive un file generico (**FILE**)
 - il formato dipende dal sistema e contiene: posizione corrente, indicatori di errore l/s, indicatori di fine file raggiunta etc

Input/Output: `stdio.h` (2)

- Come avviene la lettura di un file:
 - prima il file viene ‘aperto’, cioè si crea una struttura `FILE f` per esso
 - generalmente c’è un limite al numero di file aperti
 - poi si accede al file usando la funzioni di libreria passando `f` come parametro
 - infine il file viene ‘chiuso’ (`f` viene deallocata) il contenuto del file non è più accessibile da programma

```
#include <stdio.h>
int main (void){
    int a, sum = 0;
    FILE * ifp, * ofp;
    ifp = fopen ("inputfile", "r");
    ofp = fopen ("outputfile", "w");
    /* finchè il numero di conversioni operate con
    successo è uguale a 1 */
    while (fscanf(ifp, "%d", &a)==1)
        sum+=a;
    fprintf(ofp, "la somma è %d", &sum);
    fclose(ifp);
    fclose(ofp);
    return 0;
}
```

Dichiaro i
Puntatori a FILE



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

```
int main (void){
```

```
    int a, sum = 0;
```

```
    FILE * ifp, *ofp;
```

```
    ifp = fopen ("inputfile", "r");
```

```
    ofp = fopen ("outputfile", "w");
```

```
    /* finchè il numero di conversioni operate con successo è  
    uguale a 1 */
```

```
    while (fscanf(ifp, "%d", &a)==1)
```

```
        sum+=a;
```

```
        fprintf(ofp, "la somma è %d", &sum);
```

```
        fclose(ifp);
```

```
        fclose(ofp);
```

```
        return 0;
```

```
    }
```

Apro il file inputfile
In sola lettura



```
#include <stdio.h>
int main (void){
    int a, sum = 0;
    FILE * ifp, *ofp;
    ifp = fopen ("inputfile", "r");
    ofp = fopen ("outputfile", "w");
    /* finchè il numero di conversioni operate con successo è
    uguale a 1 */
    while (fscanf(ifp, "%d", &a)==1)
    sum+=a;
    fprintf(ofp, "la somma è %d", &sum);
    fclose(ifp);
    fclose(ofp);
    return 0;
}
```

Apro il file outputfile
In sola scrittura

```
#include <stdio.h>
int main (void){
    int a, sum = 0;
    FILE * ifp, *ofp;
    ifp = fopen ("inputfile","r");
    ofp = fopen ("outputfile","w");
    /* finchè il numero di conversioni operate con
    successo è uguale a 1 */
    while (fscanf(ifp, "%d", &a)==1)
        sum+=a;
    fprintf(ofp, "la somma è %d", &sum);
    fclose(ifp);
    fclose(ofp);
    return 0;
}
```

Leggo: funziona come
Scanf ma legge dal FILE
Puntato da **ifp**

```
#include <stdio.h>
int main (void){
    int a, sum = 0;
    FILE * ifp, *ofp;
    ifp = fopen ("inputfile", "r");
    ofp = fopen ("outputfile", "w");
    /* finchè il numero di conversioni operate con
    successo è uguale a 1 */
    while (fscanf(ifp, "%d", &a)==1)
        sum+=a;
    fprintf(ofp, "la somma è %d", &sum);
    fclose(ifp);
    fclose(ofp);
    return 0;
}
```

Scrivo: funziona come
Printf ma scrive sul FILE
Puntato da **ofp**

fprintf(ofp, "la somma è %d", &sum);


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

```
int main (void){
```

```
    int a, sum = 0;
```

```
    FILE * ifp, *ofp;
```

```
    ifp = fopen ("inputfile", "r");
```

```
    ofp = fopen ("outputfile", "w");
```

```
    /* finchè il numero di conversioni operate con  
    successo è uguale a 1 */
```

```
    while (fscanf(ifp, "%d", &a)==1)
```

```
        sum+=a;
```

```
    fprintf(ofp, "la somma è %d", &sum);
```

```
    fclose(ifp);
```

```
    fclose(ofp);
```

```
    return 0;
```

```
}
```

Chiudo e dealloco

Input/Output: `stdio.h` (3)

- `ifp = fopen("inputfile", "r");`

Nome del
file

File
binari

Modo:

"r" - read

"w" - write

"a" - append

"rb"

"wb"

"ab"

Input/Output: `stdio.h` (4)

- `ifp = fopen("inputfile", "r");`
 - interagisce con SO per reperire informazioni sul file e controllare che l'accesso sia consentito in lettura (chiamata la SC `open()` che vedremo nella seconda parte del corso)
 - se OK alloca una struttura `FILE` (inizializzandola) e restituisce il puntatore
 - se fallisce restituisce `NULL` e setta la variabile globale **`errno`**

Input/Output: `stdio.h` (6)

-bufferizzazione ...
 - tipicamente tutto l'output viene bufferizzato
 - Si può bufferizzare una linea (fino a '\n') o di più
 - questo è il motivo per cui alcune volte i caratteri stampati con `printf()` non appaiono subito
- **`int fflush(FILE * ifp)`**
 - svuota immediatamente i buffer relativi al file `ifp`
 - `fflush(NULL)` svuota tutti i buffer
 - è chiamata dalla `fclose()`
 - restituisce 0 se è andato tutto bene e EOF altrimenti

Input/Output: `stdio.h` (7)

- `stdio.h` contiene definizioni di strutture `FILE` per i file ‘speciali’ usati per leggere e scrivere sui terminali collegati a tastiera e schermo
 - es. ogni finestra corrisponde a un terminale
 - **FILE * `stdin`** : standard input, la tastiera
 - **FILE * `stdout`** : standard output, lo schermo
 - **FILE * `stderr`** : standard error,

Input/Output: `stdio.h` (8)

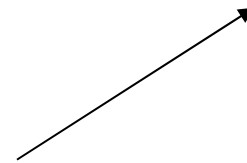
Esempi:

- `fprintf(stdout, "la somma è %d", sum);`
 - scrive sullo standard output
 - equivale a `printf("la somma è %d", &sum);`
- `fscanf(stdin, "%d", &a)`
 - legge dallo standard input
 - equivale a `scanf("%d", &a)`

Input/Output: `stdio.h` (9)

Modificare la posizione corrente di un file:

```
int fseek(FILE *fp, long offset, int place)
```



SEEK_SET	-	inizio file
SEEK_CUR	-	posizione corrente
SEEK_END	-	fine file

Posizione di partenza

Input/Output: `stdio.h` (10)

Modificare la posizione corrente di un file:

– ritorna all'inizio

```
void rewind(FILE *fp);
```

```
rewind (fp);
```

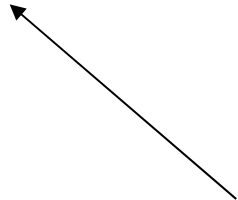
equivale a

```
lseek(fp, 0, SEEK_SET);
```


Input/Output: `stdio.h` (11)

Leggere la posizione corrente di un file:

```
int ftell(FILE *fp)
```



**Posizione corrente: numero di byte
dall'inizio del file**

Input/Output: `stdio.h` (12)

Gestione degli errori. Es.

– `int feof(FILE *fp)`

- 0 se l'indicatore di fine file è attivo e diverso da 0 altrimenti

– `void perror (const char * s)`

- la maggior parte delle funzioni di libreria (e tutte le SC) se in errore settano la variabile globale predefinita **errno** con un codice che dipende dall'errore
- la **perror()** invocata immediatamente dopo la funzione che ha dato errore converte un valore numerico in un messaggio su **stderr** (schermo)
- attenzione!!!! chiamate successive sovrascrivono **errno**!

Input/Output: `stdio.h` (13)

Esempio apertura di un file inesistente:

```
.....  
if ((fp = fopen("pippo", "r")) ==  
    NULL) {  
    perror ("Aprendo pippo");  
    return -1;  
}  
.....
```

Input/Output: `stdio.h` (14)

All'esecuzione :

```
$$ prova
```

```
Aprendo pippo: No such file or  
directory
```

```
$$
```

Input/Output: `stdio.h` (15)

Input ed output di caratteri e stringhe

- moltissime funzioni, ne citeremo alcune
- alcune funzioni sono *pericolose*, perché possono generare **buffer overrun** (scrittura caratteri oltre la dimensione del buffer)
- solitamente per ogni funzione pericolosa ne esiste una che funziona correttamente (magari con più parametri)
- controllare sempre il man e utilizzare le versioni innocue!

Input/Output: `stdio.h` (16)

Esempio: `gets` ed `fgets`

– `char* gets (char* s);`

- legge una linea dello ***stdin*** nel buffer puntato da **s** continua a leggere fino a primo ‘\n’ o all’EOF, che vengono rimpiazzati da un terminatore di stringa (‘\0’)
- pericolosa non permette il check del *buffer overrun*
- *restituisce* **s** in caso di successo e NULL in caso di fallimento o se viene letto EOF prima di un qualsiasi carattere valido

Input/Output: `stdio.h` (17)

Esempio: `gets` ed `fgets`

- `char* fgets(char*s, int n, FILE*fp);`
 - legge al più `n-1` caratteri dal file `fp` e li copia nel buffer `s`
 - continua a leggere fino a primo `'\n'` o all'EOF, il `'\n'` viene copiato nel buffer dopo il quale viene inserito un `'\0'`
 - diverso da `gets()`
 - basta passare la *lunghezza del buffer -1* per essere sicuri
 - *restituisce* `s` in caso di successo e `NULL` in caso di fallimento o se viene letto EOF prima di un qualsiasi carattere valido

```
#include <stdio.h>
/* max lung stringa */
#define N 20
int main (void){
    char s[N+2];
    FILE * ifp;
    ifp = fopen ("inputfile","r");
    /* finchè non raggiungiamo la fine del file */
    while ( (fgets(s,N+2,ifp))!=NULL)
        /* .... Elaborazione ...*/

    fclose(ifp);
    return 0;
}
```


Input/Output: `stdio.h` (18)

Leggere e scrivere stringhe:

- **`int sscanf(const char *s, const char *format, ...);`**
 - funziona come `scanf()`, `fscanf()`
 - **`s`** è la stringa da cui leggere

- **`int sprintf(const char *s, const char *format, ...);`**
 - funziona come `printf()`, `fprintf()`
 - **`s`** è la stringa da cui leggere

Input/Output: `stdio.h` (19)

- Ci sono molte più funzioni
- lettura di byte non formattati (file binari)
 - **`fread()`**, **`fwrite()`**
- rimozione e ridenominazione di file
- l'appendice A del testo Kelley Pohl fornisce una panoramica delle principali funzioni nelle librerie standard
- per la documentazione però è meglio consultare sempre il man
 - più aggiornato e preciso