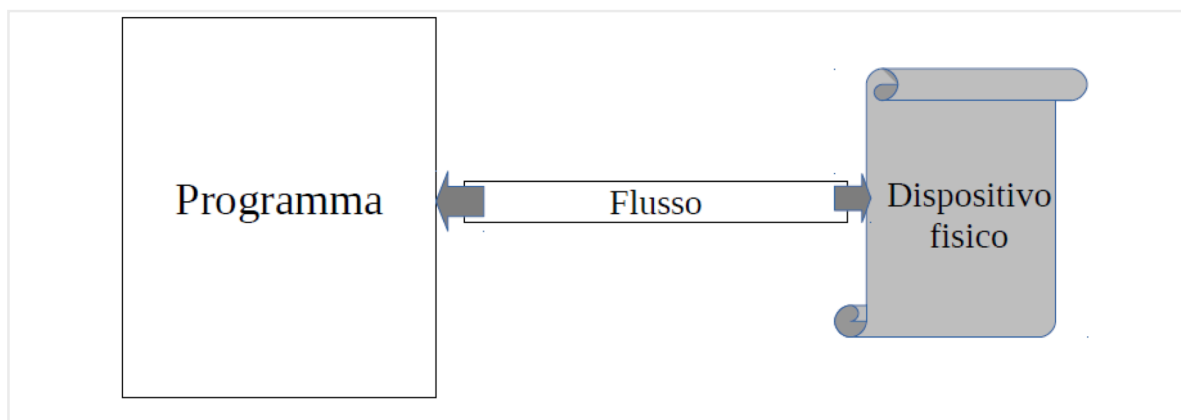


GESTIONE DEI FILE IN C

- Una stessa funzione per operazioni di input può essere utilizzata, ad esempio, sia per leggere un valore fornito tramite tastiera sia per leggere un valore da un dispositivo di memoria di massa tramite un file in esso memorizzato.
- Una funzione per operazioni di output può essere utilizzata sia per la visualizzazione sullo schermo sia per scrivere su un disco o una stampante.

Lo scambio di questi dati è realizzato attraverso uno ***stream***, un *flusso*.
(canale di comunicazione)



***Questo si comporta alla stessa maniera indipendentemente dal tipo di dispositivo fisico*

Esistono:

- **FLUSSI DI TESTO:** i dati vengono gestiti usando la loro rappresentazione caratteriale con tag di newline '\n' per riga.
- **FLUSSI BINARI:** i dati vengono gestiti nel modo in cui vengono rappresentati in memoria.

COME GESTIAMO I FLUSSI?

APERTURA -> ACCESSO -> CHIUSURA

- **Apertura del flusso:** associamo un flusso ad un dispositivo fisico
- **Accesso al flusso:** scambio dei dati
- **Chiusura del flusso:** eliminazione dell'associazione tra programma e dati

FUNZIONI:

- **fopen:** alloca dinamicamente una struttura e restituisce un puntatore a essa con i seguenti campi
 - ◆ Modalità di accesso (lettura / scrittura)
 - ◆ Posizione corrente (indicante la prossima)
 - ◆ Un indicatore *end* associato ad un byte utilizzato come marcatore di fine

FILE * fopen (char * name, const char * mode):

Name: percorso relativo o assoluto del file

Mode: modalità di accesso

*** in caso di errore verrà restituito un puntatore NULL*

example:

FILE * fp = fopen (<nome dispositivo>, "r");

Only read file

FILE * fp = fopen (<nome dispositivo>, "w");

Only write file

*** così facendo verranno sovrascritti i dati già presenti nel file*

FILE * fp = fopen (<nome dispositivo>, "a");
Only write file - APPEND

*** il puntatore si posizionerà alla fine del file*

- ◆ "r+"
- ◆ "w+"
- ◆ "a+"

Per file già esistenti

- ◆ "rb"
- ◆ "wb"
- ◆ "ab"

Per forma binaria

Flussi standard:

- *Standard input*: aperto in only read
- *Standard output*: aperto in only write
- *Standard error*: aperto in only write

Manipolati tramite: **FILE * stdin, stdout, e stderr**

- **fclose**: chiude un flusso : se un flusso non viene chiuso, l'os chiuderà il flusso solo alla fine del programma in esecuzione.

int fclose (FILE *

fp);

fp deve puntare alla struttura FILE della chiamata fopen

Successo: 0

Problema: EOF

- **feof**: per controllare se si è raggiunto il marcatore *end of file*

int feof (FILE *

fp);

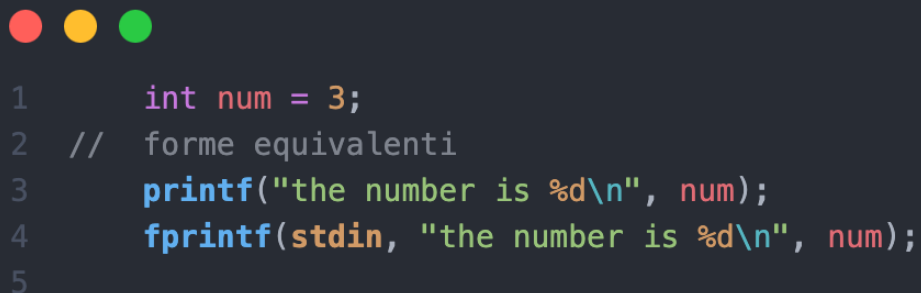
fp deve puntare alla struttura FILE della chiamata fopen

No: 0

Si: 1

fprintf: scrive su stream di testo

**int fprintf (FILE * fp, char *
stringa_di_controllo,.....);**



```
1     int num = 3;  
2  //  forme equivalenti  
3     printf("the number is %d\n", num);  
4     fprintf(stdin, "the number is %d\n", num);  
5
```

fputs: scrive su stream di testo variabili char

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



```
1 // per char
2     char c = "k";
3     fputc(c, stdout);
```

fputs: scrive su stream di testo variabili stringhe

**int fputs (char * s, FILE *
fp);**



```
1 // per stringhe
2     char s[] = "hello world";
3     fputs(s, stdout);
```

fscan: legge su stream di testo

int fscan (FILE * fp, char *

`stringa_di_controllo,.....);`



```
1     int value;  
2  //  forme equivalenti  
3     scanf("%d", &num);  
4     fscanf(stdin, "%d", &value);
```

fgetc: consente di leggere e ottenere su stream di testo variabili char

int fgetc

(FILE * fp);




```
1  char letter = fgetc(stdin);  
2  printf("%c", letter);
```

fgets: consente di leggere e ottenere su stream di testo variabili char

char * fgets (char * s, int

n, FILE * fp);



```
1 char poem[30];  
2 fgets(poem, 30, stdin);  
3 printf("%s", poem);
```

fscanf per le stringhe:

Quando fscanf incontra lo specificatore %s, legge i caratteri e li memorizza nell'array di caratteri associato finchè non incontra uno spazio, una tabulazione, un newline o un indicatore di fine file. A questo punto, la funzione inizializza l'elemento corrente dell'array di input con '\0'.

*È importante assicurarsi che il numero di caratteri processati + il carattere nullo di terminazione non superi la lunghezza del vettore di caratteri (**overflow**)*

(Ciò può essere garantito utilizzando lo specificatore di conversione %Ns)

(N = costante intera non negativa che indica il numero massimo di caratteri che possono essere letti ed inseriti nell'array di input.)

Perche utilizziamo i file binari?

I file di testo non si presentano bene alla modifica (avendo lunghezza variabile avverrebbe una sovrascrittura di una porzione di dati ben più grande di quella da modificare)

Mentre per i file binari (lunghezza fissa) è possibile aggiornare i record senza dover sovrascrivere l'intero file

Inoltre l'accesso ai record risulta essere più veloce.

Tuttavia la lunghezza di byte per tipo potrebbe non essere la stessa a seconda della macchina, quindi non tutti i computer possono leggere gli stessi file binari.

Funzioni su file binari:

Funzioni di lettura / scrittura richiederanno anche un **buffer**: indirizzo iniziale della regione di memoria che contiene i dati da scrivere nello stream puntato

size_t fwrite (void *buffer, size_t n_byte, size_t num, FILE *pf);

*Buffer avrà un'ampiezza di $n_byte * num$*

Analogamente per leggere:

size_t fread (void *buffer, size_t n_byte, size_t num, FILE *pf);