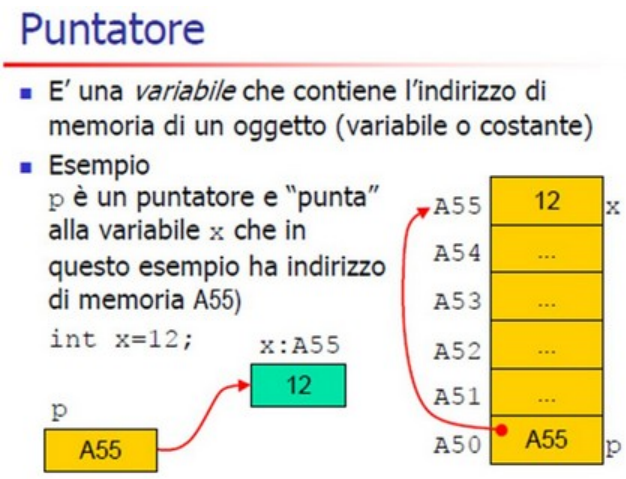
File di testo in C

Il linguaggio C offre due tipi di accesso ai file; uno definibile come «normale» e un altro a basso livello, per permettere l'accesso a funzioni del sistema operativo (chiamate di sistema open/read/write). Vedremo solo l'utilizzo normale dei file.

FILE come tipo di dati
Nel linguaggio C, i file vengono trattati come un tipo di dati derivato, cioè ottenuto dai tipi elementari esistenti. In pratica, quando si apre e si gestisce un file, si ha a che fare con un puntatore al file, o *file pointer*, che è una variabile contenente un qualche riferimento univoco al file stesso.

Per gestire i puntatori ai file in C, occorre dichiarare una variabile come **puntatore** al tipo derivato `FILE`, come nell'esempio seguente:

```
#include <stdio.h>
...
int main ()
  {
    FILE *pf;
    ...
  }
```



Il fatto che il nome utilizzato per questo tipo di dati, `FILE`, sia scritto utilizzando solo lettere maiuscole, lascia intendere che si tratta di una macro che si traduce in qualcosa di adatto al tipo particolare di piattaforma utilizzato. Per questo stesso motivo, l'utilizzo di tale tipo richiede l'inclusione del file di intestazione `stdio.h`.

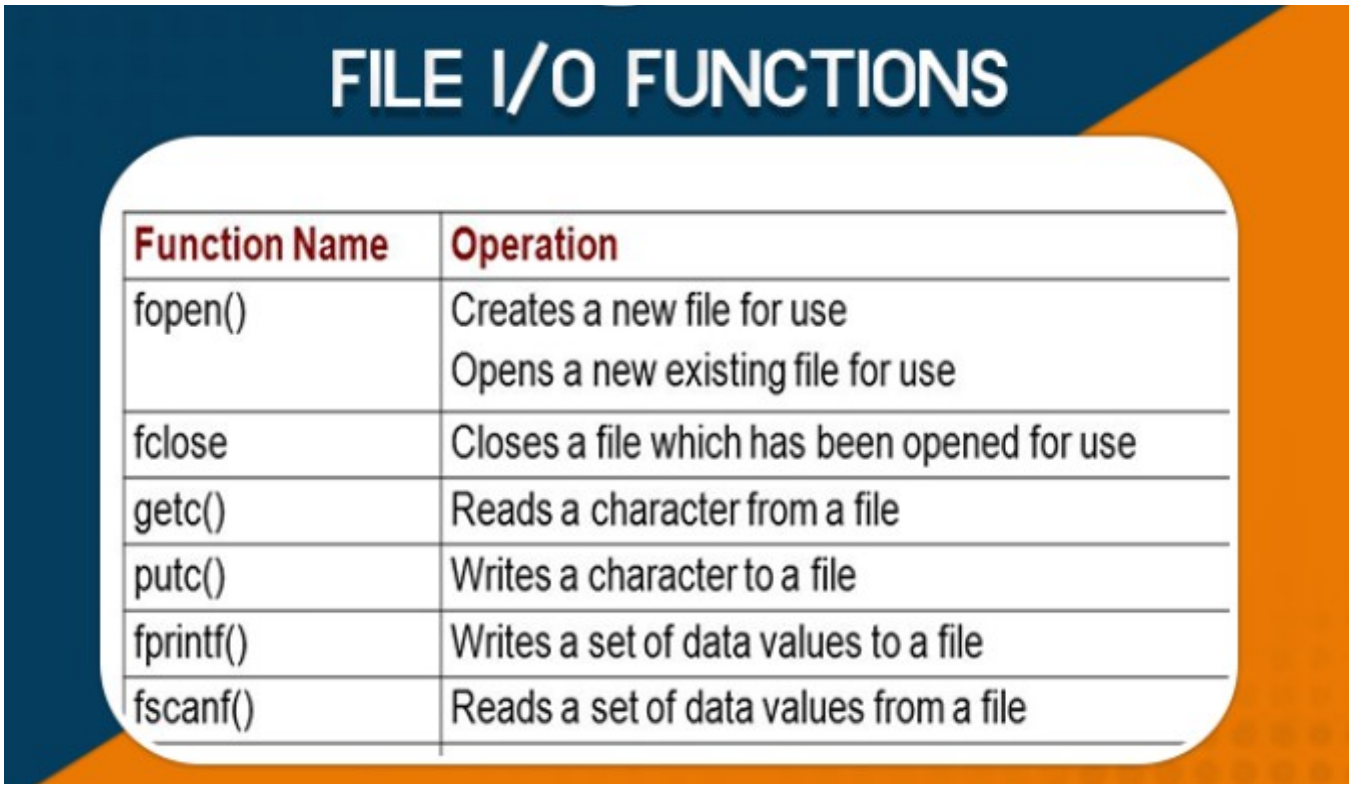
EOF

Oltre alla macro `FILE`, è bene considerarne un'altra, anch'essa molto importante: `EOF`. Si tratta di un valore, definito in base alle caratteristiche della piattaforma, utilizzato per rappresentare il raggiungimento della fine del file. Anche questa macro è definita all'interno del file `stdio.h`..

Abbiamo già visto l'EOF nel ciclo per leggere da tastiera o con redirezione input/output:

while((c=getchar())!=EOF) { ….. };

Iniziamo quindi con i File in C:



Apertura e chiusura file

L'apertura dei file viene ottenuta normalmente con la funzione `fopen()` che restituisce il puntatore al file, oppure il puntatore nullo, `NULL (vale 0 )` in caso di fallimento dell'operazione. L'esempio seguente mostra l'apertura del file `mio_file` contenuto nella directory corrente, con una modalità di accesso in sola lettura.

```
#include <stdio.h>
...
int main ()
  {
    FILE *pf;
    ...
    pf = fopen("miofile", "r");
    ...
  }
```

Estensioni comuni in Windows

- Per individuare un file: cammino assoluto
- Un file system per ogni disco → anche nome del disco
- Esempio: C:\Dir1\Dir2\Dir3\file.txt
- Estensioni per file:
  - .exe per programma eseguibile
  - .sys per driver di sistema
  - .txt per file di testo
  - .c per programma in C
  - .doc per documento Word

Come si vede dall'esempio, è normale assegnare il puntatore ottenuto a una variabile adatta, che da quel momento identificherà il file, finché questo resterà aperto.

`fclose (pf);`	La chiusura del file avviene in modo analogo, attraverso la funzione `fclose()`, che restituisce zero se l'operazione è stata conclusa con successo, oppure il valore rappresentato da `EOF`. L'esempio seguente ne mostra l'utilizzo.

La chiusura del file conclude l'attività con questo, dopo avere scritto tutti i dati eventualmente ancora rimasti in sospeso (se il file era stato aperto in scrittura).

Normalmente, un file aperto viene definito come ***flusso***, o *stream*; così, nello stesso modo viene identificata la variabile puntatore che vi si riferisce. In effetti, lo stesso file potrebbe anche essere aperto più volte con puntatori differenti, quindi è corretto distinguere tra file fisici su disco e file aperti, o flussi.

fopen() → `FILE *fopen (char nome`*`file[]`*`, char *`*`modalità`*`)`

La funzione `fopen()` permette di aprire il file indicato attraverso la stringa fornita come primo parametro, tenendo conto che tale stringa può contenere anche informazioni sul percorso necessario per raggiungerlo, secondo la modalità stabilita dal secondo parametro, anch'esso una stringa.La stringa fornita come secondo parametro, contenente l'informazione della modalità, può utilizzare i simboli seguenti:

Parametro x fopen	Descrizione
`r`	apre il file in sola lettura, posizionandosi all'inizio del file;
`r+`	apre il file in lettura e scrittura, posizionandosi all'inizio del file;
`w`	apre il file in sola scrittura, creandolo se necessario, o troncandone a zero il suo contenuto se questo esisteva già;
`w+`	apre il file in scrittura e lettura, creandolo se necessario, o troncandone a zero il suo contenuto se questo esisteva già;
`a`	apre il file in scrittura in aggiunta (*append*), creandolo se necessario, o aggiungendovi dati a partire dalla fine e, di conseguenza, posizionandosi alla fine dello stesso;
`a+`	apre il file in scrittura in aggiunta e in lettura, creandolo se necessario, o aggiungendovi dati a partire dalla fine e, di conseguenza, posizionandosi alla fine dello stesso.

La funzione fopen restituisce un puntatore al tipo `FILE`, riferito al file aperto, oppure si tratta del puntatore nullo (`NULL`) in caso di insuccesso.

`fclose()` → `int fclose (FILE *`*`stream`*`)`

La funzione `fclose()` chiude il file rappresentato dal puntatore indicato come parametro della funzione. Se il file era stato aperto in scrittura, prima di chiuderlo vengono scaricati i dati eventualmente accumulati nella memoria tampone (i *buffer*).

La funzione restituisce il valore zero se l'operazione si conclude normalmente.

Lettura e scrittura
Le operazioni di lettura e scrittura dipendono da un **indicatore riferito a una posizione**, espressa in byte, del contenuto del file stesso. A seconda di come viene aperto il file, questo indicatore viene posizionato nel modo più logico (vedi sopra le modalità di apertura della funzione `fopen()`). Questo viene spostato automaticamente in avanti a seconda delle operazioni di lettura e scrittura che si compiono.

File di testo : fprintf() ed fscanf()

Se desidero leggere / scrivere nei file di testo come naturalmente faccio da tastiera, uso

fprintf() / fscanf()

Le funzioni **fprintf() ed fscanf()**, sono praticamente uguali alle funzioni **printf()** e **scanf()**:

hanno solo in aggiunta il primo parametro che indica il file (variabile di tipo FILE *, cioè puntatore a FILE).

Esempio:

data le variabilI

FILE *fd;

int n;

se il file e' aperto in lettura, e' possibile leggere dal file di testo un intero proprio come se fosse letto da tastiera:

fscanf(fd,"%d",&n); /// da tastiera era scanf("%d",&n);

oppure, se il file fosse aperto in scrittura, ci si puo' scrivere 10 linee (tabellina del 7) cosi:

for(i=1; i <=10; i++)

fprintf(fd,"7 per %d fa %d\n",7*i);

/// su schermo era: printf("7 per %d fa %d\n",7*i);

/// Esercizi: (nei seguenti file di testo il primo valore indica il numero di linee del file)

A) Dato un file contenente le spese della settimana, visualizzare la media giornaliera, la spesa massima e la minima

B) Creare un file contenente numeri N numeri pseudocasuali fra 1 ed M. (leggere M ed N da tastiera)

C) Dato il file del punto sopra, ed un valore V letto da tastiera, verificare quante volte il valore V compare nel file, visualizzando anche il numero di linea in cui compare.

D) Letto un file di numeri double, caricare i valori in un vettore (MAX 100 elementi). Poi visualizzare sullo schermo la differenza, per ogni valore, dalla media.

E) Scrivere i valori del file di testo del punto D) in un file HTML, in tabella, in ordine inverso.

Approfondimento: fgets() ed fputs()

I file di testo possono essere gestiti anche attraverso due funzioni: **`fgets()`** e **`fputs()`**. Queste permettono rispettivamente di ***leggere e scrivere un file una riga alla volta***, intendendo come riga una porzione di testo che termina con il carattere di interruzione di riga ('\n').

La funzione `fgets()` permette di leggere una riga di testo di una data dimensione massima. Si osservi l'esempio seguente:

```
fgets (str, 100, pf);
```

In questo caso, viene letta una riga di testo di una dimensione massima di 99 caratteri, dal file rappresentato dal puntatore `pf`. Questa riga viene posta all'interno dell'array str`[]`, con l'aggiunta di un carattere `\0` finale. Questo fatto spiega il motivo per il quale il secondo parametro corrisponde a 100, mentre la dimensione massima della riga letta è di 99 caratteri. In pratica, l'array di destinazione è sempre una stringa, terminata correttamente.

Nello stesso modo funziona `fputs()`, che però richiede solo la stringa e il puntatore del file da scrivere. Dal momento che una stringa contiene già l'informazione della sua lunghezza perché possiede un carattere di conclusione, non è prevista l'indicazione della quantità di elementi da scrivere.

```
fputs (str, pf);
```

fgets() → `char *fgets (char` *`stringa[]`*`, int` *`dimensione`*`, FILE *`*`stream`*`)`
La funzione `fgets()` permette di leggere una stringa corrispondente a una riga di testo da un file. `fgets()` impone l'indicazione della dimensione massima della riga, dal momento che questa deve poi essere contenuta nell'array indicato come primo parametro. La dimensione massima, indicata come secondo parametro, rappresenta la dimensione dell'array di caratteri che deve riceverlo. Dal momento che per essere una stringa, questa deve essere terminata, allora la dimensione massima della riga sarà di un carattere in meno rispetto alla dimensione dell'array.

Pertanto, la lettura del file avviene fino al raggiungimento della dimensione dell'array (meno uno), oppure fino al raggiungimento del codice di interruzione di riga, che viene regolarmente incluso nella riga letta, oppure anche fino alla conclusione del file.

Se l'operazione di lettura riesce, `fgets()` restituisce un puntatore corrispondente alla stessa stringa (cioè l'array di caratteri di destinazione), altrimenti restituisce il puntatore nullo, `NULL`, per esempio quando il file ha raggiunto la fine.

fputs() → `int fputs (const char` *`stringa[]`*`, FILE *`*`stream`*`)`
La funzione `fputs()` permette di scrivere una stringa in un file di testo. La stringa viene scritta senza il codice di terminazione finale, `\0`.

Il valore restituito è un valore positivo in caso si successo, altrimenti `EOF`.

I/O standard : Ci sono **tre file che risultano aperti automaticamente**:

stdin → standard input, corrispondente normalmente alla tastiera;

stdout → standard output, corrispondente normalmente allo schermo del terminale;

stderr → standard error, anch'esso corrispondente normalmente allo schermo del terminale.

Si accede a questi flussi attraverso funzioni standard (es. printf(), scanf(), getchar(), putchar()), ma si potrebbe accedere anche attraverso le funzioni di accesso ai file, utilizzando per identificare i flussi i nomi: `stdio`, `stdout` e `stderr`.

Esempi:

fprintf(stdout,"Ciao a tutti\n"); equivale a **printf**("Ciao a tutti\n");

fscanf(stdin,"%d",&n); equivale a **scanf("%d",&n)**;

c = getc(stdin); equivale a **c = getchar()**;

putc(c,stdout); equivale a **putchar(c);**

Esercizi:

A) Si modifichi uno o più dei programmi fatti con getchar() e putchar() in modo che possano essere usati senza la redirezione I/O (simboli > e < da linea comando), leggendo prima da tastiera il nome del file da aprire:

```
char nomefile[80];

….....

scanf("%s",nomefile);

fd = fopen(nomefile,"r");

while((c=getc(fd))!=EOF)  ….....
```