## Premessa

## Input /output e gestione dei file

Iniziamo la lezione dicendo subito che per INPUT/OUTPUT si intende l' insieme delle operazioni di ingresso ed uscita, ossia lo scambio di informazioni tra il programma e le unita' periferiche del calcolatore.

Nel linguaggio C l' I/O e' interamente implementato mediante funzioni della libreria standard. Il linguaggio C vede i file come un flusso (stream) sequenziale di byte. Uno stream dal punto di vista tecnico e' una implementazione software in grado di gestire le informazioni relative all'interazione a basso livello con la periferica associata, in modo che il programma possa trascurare di che periferica si tratti.

Ogni file termina con un marcatore di fine file detto **EOF**( **end-of-file**) definito in **<stdio.h>** come una costante simbolica uguale a -1.

## I file e gli stream

Un file e' una sequenza di lunghezza non prefissata di valori dello stesso tipo.

Quando un file viene aperto gli sara' associato il relativo **stream** .

Tre file e i loro rispettivi stream, lo standard di input, lo standard di output, lo standard error (**stdin**, **stdout**, **stderr**) , sono aperti automaticamente quando inizia l'esecuzione di un programma.

Lo stream dello standard di input (**stdin**) consente ad un programma di ricevere input dalla tastiera (puo'essere rediretto su altre periferiche).

Lo stream dello standard di output (**stdout**) consente ad un programma di di scrivere sul video della macchina (puo' essere rediretto su altre periferiche).

Lo stream dello standard di errore (**stderr**) consente ad un programma di scrivere sul video della macchina (non puo' essere rediretto su altre periferiche).

Un programma C puo' servirsi degli stream standard ; basta che vi compaia la direttiva

**#include<stdio.h>**

Gli stream forniscono un canale di comunicazione tra i file ed i programmi.

L'apertura di un file restituisce un puntatore ad una struttura FILE (definita in <stdio.h>) che contiene le informazioni utilizzate per elaborare il file. La struttura FILE ha un file descriptor (descrittore di file), cioe' l'indice di un vettore del S.O. chiamata **open file table** (tabella dei file aperti).

Ogni elemento del vettore contiene un file control block o FCB (blocco di controllo del file) il quale viene utilizzato dal S.O. per amministrare un particolare file.

Consideriamo due tipi di file i file di testo ed i file binari; i primi utilizzati quando l'informazione da memorizzarvi e' di tipo testo cioe' sequenze di caratteri; i secondi quando si vuole memorizzare nel file dati di un generico tipo (interi,strutture ecc…).

## Il puntatore al file

Attraverso la definizione

**FILE *fp;**

viene permessa l'associazione di un identificatore del programma ad un file, che si trova in memoria secondaria.

Con la sola definizione, fp e' soltanto una "variabile di tipo FILE"; l'effettiva associazione avviene in fase di apertura del file.

Il puntatore fp punta ad informazioni che descrivono alcune caratteristiche dei file come il nome, lo stato e la posizione corrente.

Scopo del puntatore al file e' quello di identificare uno specifico file su disco e viene usato dal flusso associato per svolgere in modo corretto le operazioni di input/output.

# I File

Il file e` l'unita` logica di memorizzazione dei dati su memoria di massa.

- Consente una ***memorizzazione persistente*** dei dati, ***non limitata*** dalle dimensioni della memoria centrale.
- Generalmente un file e` una sequenza di componenti omogenee (***record logici***).
- I file sono gestiti dal Sistema Operativo. Per ogni versione C esistono funzioni per il trattamento dei file (***Standard Library***) che tengono conto delle funzionalita` del S.O ospite.

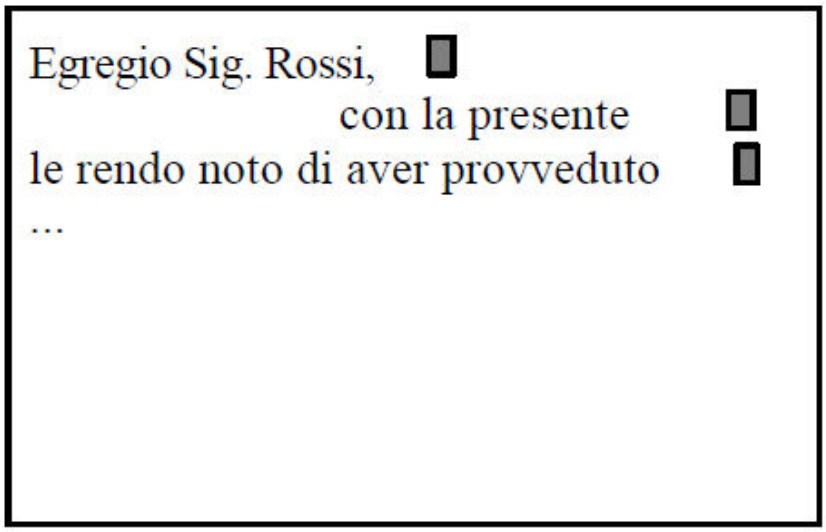
**In C i file vengono distinti in due categorie:**

- ***file di testo***, trattati come sequenze di caratteri. organizzati in linee (ciascuna terminata da '\n')
- ***file binari***, visti come sequenze di bit

Si noti che la creazione del file binario deve essere fatta da programma, mentre per i file di testo può essere fatta con un text editor.

# File di testo

Sono file di caratteri, organizzati in linee.
Ogni linea e` terminata da una marca di fine linea ( ***newline***, cara ttere '\n').




- Il ***record logico*** puo` essere il singolo carattere oppure la singola linea.

# Gestione di file in C

I file hanno una struttura ***sequenziale:***

- i record logici sono organizzati in una sequenza
- per accedere ad un particolare record logico, e` ne cessario "scorrere" tutti quelli che lo precedono.


Per accedere ad un file da un programma C, e` necessario predisporre una variabile che lo rappresenti (**puntatore a file)**

**Puntatore a file:**

e` una variabile che viene utilizzata per riferire un file nelle operazioni di accesso (lettura e scrittura.).
Implicitamente essa indica:

- il file
- l'elemento corrente all'interno della sequenza

**Ad esempio:**

```
FILE *fp;
```

➳ il tipo FILE e` un tipo non primitivo dichiarato nel file **stdio.h.**

# Gestione di file in C

## Apertura di un file:

Prima di accedere ad un file e` necessario ***aprirlo:*** l'operazione di apertura compie le azioni preliminari necessarie affinche` il file possa essere acceduto (in lettura o in scrittura). L'operazione di apertura inizializza il puntatore al file.

## Accesso ad un file:

Una volta aperto il file, e` possibile leggere/scrivere il file, riferendolo mediante il puntatore a file.

## Chiusura di un file:

Alla fine di una sessione di accesso (lettura o scrittura) ad un file e` necessario chiudere il file per memorizzare permanentemente il suo contenuto in memoria di massa:

# Apertura di un File

**FILE *fopen(char *_name_, char *_mode_);**

dove:

- ***name*** e` un array di caratteri che rappresenta il nome (assoluto o relativo) del file nel file system
- ***mode*** esprime la modalita` di accesso scelta.

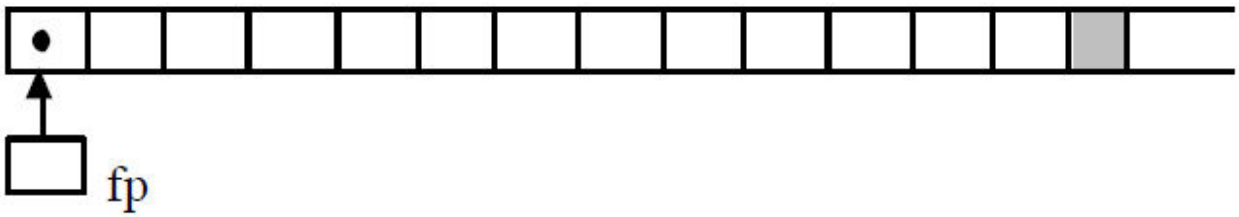
- ✓ "**r**" Apre un file testuale esistente per sola lettura, posizionando il cursore all'inizio del file;
- ✓ "**w**" Apre un file testuale per sola scrittura: se il file gia' esiste i dati in esso contenuti vengono perduti; se il file non esiste ne viene creato uno nuovo;
- ✓ "**a**" Apre un file di testuale esistente per scrittura, in modalita' append; il cursore viene posizionato alla fine del file e si puo' scrivere nel file solo in coda ai dati gia' presenti; i dati gia presenti non vengono cancellati;
- ✓ "**r+**" Apre un file testuale esistente in lettura e scrittura; il cursore viene posto all'inizio;
- ✓ "**w+**" Apre un file testuale per lettura e scrittura; se il file gia' esiste i dati in esso contenuti vengono perduti; se il file non esiste, ne viene creato uno nuovo;
- ✓ "**a+**" Apre un file testuale in modalita' append; se il file non esiste, ne viene creato uno nuovo; si possono leggere tutti i dati del file; si possono scrivere dati solo in fondo al file.
- ✓ "**rb**" Apre un file binario esistente per sola lettura, posizionando il cursore all'inizio;
- ✓ "**wb**" Apre un file binario per sola scrittura; se il file gia' esiste i dati in esso contenuti vengono persi; se il file non esiste , ne viene creato uno nuovo;
- ✓ "**ab**" Apre o crea un file binario per scrittura in modalita' append; il cursore viene posizionato alla fine del file e si puo' scrivere nel file solo in coda ai dati gia' presenti;
- ✓ "**rb+**" Apre un file binario esistente in lettura e scrittura
- ✓ "**wb+**" Apre o crea un file binario in lettura e scrittura
- ✓ "**ab+**" Apre o crea in modalita' append un file binario.

- **Se eseguita con successo, l'operazione di apertura ritorna come risultato un *puntatore* al file aperto**
- **Se, invece, l'apertura fallisce (ad esempio, perche` il file non esiste), fopen restituisce il valore NULL.**

# Apertura in lettura

```
fp = fopen("filename", "r")
```

**Applicabile se il file non e` vuoto (altrimenti NULL)**




**Ad esempio:**

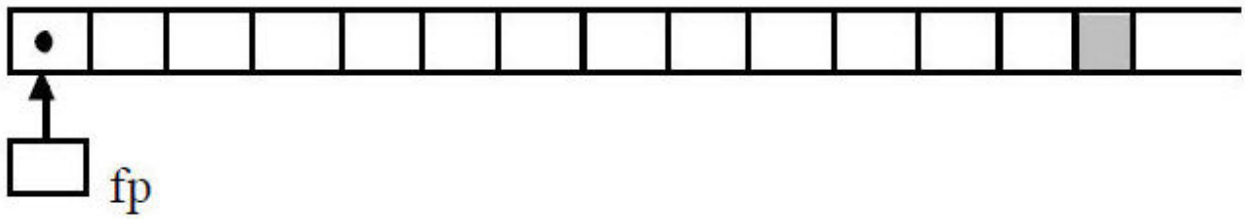
```
File *fp;
.........
fp=fopen("c:\dati.dat", "r");
<uso del file>
.........
```

- fp rappresenta, dall'apertura in poi, il riferimento da utilizzare nelle operazioni di accesso a c:\dati.dat
  Esso individua, in particolare:
  - il file
  - l'elemento corrente all'interno del file

# Apertura in scrittura

```
fp = fopen("filename", "w")
```

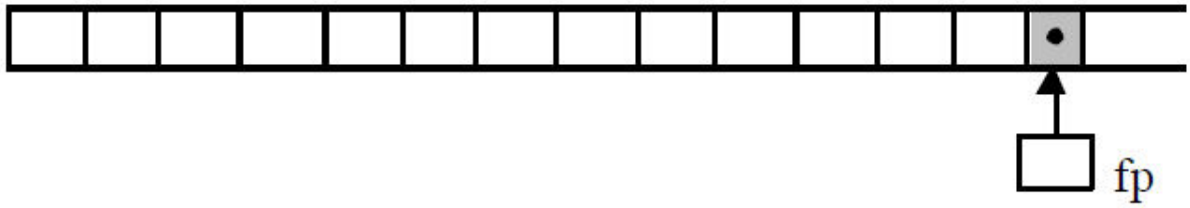
**Anche se il file non e` vuoto:**




- Se il file esiste gia`, il suo contenuto viene **perso**.

# Apertura in aggiunta (append)

```
fp = fopen("filename", "a")
```




Il puntatore al file si posiziona sull'elemento successivo all'ultimo significativo del file

➠ se il file esiste gia`, il suo contenuto non viene perso.

# Chiusura di un File

Al termine di una sessione di accesso al file, esso deve essere **chiuso**.

L'operazione di chiusura si realizza con la funzione **fclose**:

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

dove:

- fp rappresenta il puntatore al file da chiudere.

**fclose ritorna come risultato un intero:**

- Se l'operazione di chiusura e` eseguita correttamente restituisce il valore 0
- se la chiusura non e` andata a buon fine, ritorna la costante EOF.

**Esempio:**

```
#include <stdio.h>
int main()
{
  FILE *fp;
  fp = fopen("prova.dat", "w")
  ...
   /* scrittura di prova.dat */
  ...
  fclose(fp);
  return 0;
}
```

# File standard di I/O

Esistono tre file testo che sono aperti automaticamente all'inizio di ogni esecuzione:

- ***stdin***, standard input (tastiera), aperto in lettura
- ***stdout***, standard output (video), aperto in scrittura
- ***stderr***, standard error (video), aperto in scrittura

☞ *stdin*, *stdout*, *stderr* sono variabili di tipo *puntatore a file* automaticamente (ed **implicitamente**) definite ➡ **non vanno definite**.

# Funzione feof()

Durante la fase di accesso ad un file e` possibile verificare la presenza della marca di fine file con la funzione di libreria:

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

- feof(fp) controlla se e` stata raggiunta la fine del file puntato da fp operazione di lettura o scrittura **precedente**. Restituisce il valore 0 (falso logico) se non e` stata raggiunta la fine del file, altrimenti un valore diverso da zero (vero logico).

# Lettura e Scrittura di file

Una volta aperto un file, su di esso si puo` accedere in lettura e/o scrittura, compatibilmente con quanto specificato in fase di apertura.

**Per file di testo sono disponibili funzioni di:**

- Lettura/scrittura **con formato**
- Lettura/scrittura di **caratteri**
- Lettura/scrittura di **stringhe di caratteri**

**Per file binari si utilizzano funzioni di:**

- Lettura/scrittura di **blocchi**

↣ Ogni operazione di lettura (o scrittura) provoca l'avanzamento del puntatore al file al primo record logico (carattere , linea o blocco) non letto (o libero, nel caso di scrittura).

# Accesso a file di testo: Lettura/scrittura con formato

Funzioni simili a **scanf** e **printf**, ma con un parametro aggiuntivo rappresentante il puntatore al file **di testo** sul quale si vuole leggere o scrivere:

## Lettura con formato:

Si usa la funzione fscanf:

**int** fscanf **(FILE *fp,** ***stringa-controllo*****,** ***ind-elem*****);**

dove:

- fp e` il puntatore al file
- *stringa-controllo* indica il formato dei dati da leggere
- *ind-elem* e` la lista degli indirizzi delle variabili a cui assegnare i valori letti.

**Esempio:**

```
FILE *fp;
int A; char B; float C;
fp=fopen("dati.txt", "r");
fscanf(fp, "%d%c%f", &A, &B, &C);
...
fclose(fp);
```

Restituisce il numero di elementi letti, oppure un valore negativo in caso di errore.

## Scrittura con formato:

Si usa la funzione fprintf::

**int** fprintf **(FILE *fp, *stringa-controllo, elementi*);**

- fp e' il puntatore al file
- *stringa-controllo* indica il formato dei dati da scrivere
- *elementi* e' la lista dei valori (espressioni) da scrivere.

## Esempio:

```
FILE *fp;
float C=0.27;
fp=fopen("risultati.txt", "w");
fscanf(fp,"Risultato: %f", c*3.14);
...
fclose(fp);
```

Restituisce il numero di elementi scritti, oppure un valore negativo in caso di errore.

## printf/scanf e fprintf/fscanf:

Notiamo che:

printf **(stringa-controllo, elementi)**
scanf **(stringa-controllo, ind-elementi);**

**equivalgono a:**

fprintf **(stout, stringa-controllo, elementi);**
fscanf **(stdin, stringa-controllo, ind-elementi);**

**Esempio:**

Visualizzazione del contenuto di un file di testo:

```
#include <stdio.h>
int main()
{
char buf[80];
FILE *fp;
fp=fopen("testo.txt", "r");
fscanf(fp,"%s",buf);
while(!feof(fp))
   {
   printf("%s", buf);
   fscanf(fp,"%s",buf);
   }
fclose(fp);
return 0;
}
```

**oppure:**

```
#include <stdio.h>
int main()
{
char buf[80];
FILE *fp;
fp=fopen("testo.txt", "r");
while (fscanf(fp,"%s",buf)>0)
    {
    printf("%s", buf);
    }
fclose(fp);
return 0;
}
```

# Lettura/scrittura di caratteri:

Funzioni simili a **getchar** e **putchar**, ma con un parametro aggiuntivo rappresentante il puntatore al file (di testo) sul quale si vuole leggere o scrivere:

```
int   getc (FILE *fp);

int   putc (int c, FILE *fp);

int   fgetc (FILE *fp);

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

☞ In caso di esecuzione corretta, restituiscono il carattere letto o scritto come intero, altrimenti EOF.

## Esempio:

Programma che concatena i file dati come argomento in un unico file (***stdout***)

```
#include <stdio.h>
void filecopy(FILE *, FILE *);

int main(int argc, char **argv)
{
FILE *fp:

if (argc==1)
   {
   filecopy(stdin, stdout);
   }
else
   {
   while (--argc>0)
     {
     if ((fp=fopen(*++argv, "r"))==NULL)
         {
         printf("\nImpossibile aprire il file %s\n", *argv)
         }
    else
         {
         filecopy(fp, stdout);
         fclose(fp);
         }
     }
return 0;
}


void filecopy(FILE *inputFile, FILE *outputFile)
{
int c;
while((c=getc(inputFile))!=EOF)
   {
   putc(c, outputFile);
   }
return;
}
```

## Note sull'esempio:

- se non ci sono argomenti (argc=1), il programma copia lo standard input nello standard output;
- la funzione **filecopy** effettua la copia del file carattere per carattere;
- se uno dei file indicati come argomento non esiste, la funzione **fopen** fallisce, restituendo il valore NULL. In questo caso il programma termina (**exit**) restituendo il valore 1 e stampando un messaggio di errore;
- sarebbe meglio scrivere i messaggi di errore sullo standard error (***ridirezione***).

```
printf(stderr,"\nImpossibile aprire il file %s\n", *argv);
```

- il ciclo:

```
while((c=getc(inputFile))!=EOF)
  {
  putc(c, outputFile);
  }
```

poteva essere scritto anche come:

```
c=getc(inputFile);
while(!feof(inputFile))
  {
  putc(c,outputFile);
  c=getc(inputFile);
  }
```

# Lettura/scrittura di stringhe

Funzioni simili a **gets** e **puts**:

```
char      *fgets (char  *s, int n, FILE *fp);
```

Trasferisce nella stringa s i caratteri letti dal file puntato da fp, fino a quando ha letto n-1 caratteri, oppure ha incontrato un newline, oppure la fine del file. La **fgets** mantiene il newline nella stringa s.

Restituisce la stringa letta in caso di corretta terminazione; '\0' in caso di errore o fine del file.

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

Trasferisce la stringa s (terminata da '\0') nel file puntato da fp. Non copia il carattere terminatore '\0' ne' aggiunge un newline finale.

Restituisce l'ultimo carattere scritto in caso di terminazione corretta; EOF altrimenti.

# Accesso a file binari: Lettura/scrittura di blocchi

Si puo` leggere o scrivere da un file binario un intero blocco di dati (binari).

Un file binario memorizza dati di qualunque tipo, in particolare dati che non sono caratteri (interi, reali, vettori o strutture).

Per la lettura/scrittura a blocchi e` necessario che il file sia stato aperto in modo ***binario*** (modo "b").

## Lettura:

**int fread (void **vet*, int *size*, int *n*, FILE **fp*);**

Legge (al piu`) *n* oggetti dal file puntato da *fp*, collocandoli nel vettore *vet*, ciascuno di dimensione *size*. Restituisce un intero che rappresenta il numero di oggetti effettivamente letti.

## Scrittura

**int fwrite (void **vet*, int *size*, int *n*, FILE **fp*);**

Scrive sul file puntato da *fp*, prelevandoli dal vettore *vet*, *n* oggetti, ciascuno di dimensione *size*. Restituisce un intero che rappresenta il numero di oggetti effettivamente scritti (inferiore ad *n* solo in caso di errore, o fine del file).

## Esempio:

Programma che scrive una sequenza di record (dati da input) in un file binario:

```
#include<stdio.h>
typedef struct
    {
    char nome[20];
    char cognome[20];
    int reddito;
     }persona;
int main()
{
FILE *fp;
persona p;
int fine=0;

fp=fopen("archivio.dat","wb");
do
    {
    printf("Dati persona?");
    scanf("%s%s%d ",p.nome,
    p.cognome,&p.reddito);
    fwrite(&p,sizeof(persona),1,fp);
    printf("Fine (si=1,no=0)?");
    scanf("%d", &fine);
     }
while(!fine);
fclose(fp);
return 0;
}
```

## Esempio:

Programma che legge e stampa il contenuto di un file binario:

```
#include<stdio.h>
typedef struct
    {
    char nome[20];
    char cognome[20];
    int reddito;
     }persona;
int main()
{
FILE *fp;
persona p;

fp=fopen("archivio.dat","rb");
fread(&p, sizeof(persona),1, fp);
while (!feof(fp))
   {
    printf("%s%s%d",p.nome,p.cognome,
            p.reddito);
   fread(&p,sizeof(persona),1,fp);
   }
fclose(fp);
return 0;
}
```

## Esempio:

Programma che richiede in input il nome di un file e scrive in questo file binario un vettore di interi.

```
#include <stdio.h>
#define N 5
int main()
{
FILE *fp;
int i, tab[N]={1,2,3,4,5};
char fname[20];

printf("Inserire il nome del file: ");
scanf("%s",fname);
if ((fp=fopen(fname,"wb"))==NULL)
    {
    printf("Impossibile aprire file
            di uscita\n");
    }
else
    {
    for(i=0;i<N;i++)
      fwrite(&tab[i], sizeof(int), 1, fp);
    fclose(fp);
    }
return 0;
}
```

## Esempio:

Programma che richiede in input il nome di un file binario di interi e memorizza il contenuto di questo file in un vettore di interi (di al più 40 elementi).

```
#include <stdio.h>
#define MAX 40
int main()
{
FILE *fp;
char fname[20];
int i, j, tab[MAX];

/* acquisizione del nome del file in fname */
if ((fp=fopen(fname, "rb"))==NULL)
   {
   printf("Impossibile aprire file di ingresso\n");
   }
else
   {
    i=0;
    fread(&tab[i++], sizeof(int), 1, fp);
    while(!feof(fp) && i<MAX)
       fread(&tab[i++], sizeof(int), 1, fp);
    fclose(fp);
    /* il valore di i viene incrementato
    anche per la marca di fine file EOF */
    i--;
    for(j=0;j<i;j++)
       printf("%d ", tab[j]);
    }
return 0;
}
```

**Esempio:**

Scrittura di record da vettore a file binario (nome fornito in input).

```
#include <stdio.h>
#define DIM 5
typedef struct
    {
    char nome[15],
    cognome[15],
    via[10];
    int eta;
    } Persona;
Persona v[DIM];

int main()
{
int i, n;
FILE *fp;
char fname[20];
/* Acquisizione di n record da input memorizzati */
/* nel vettore v (fname è il file di uscita) */
if ((fp=fopen(fname, "wb"))==NULL)
    {
    printf("Impossibile aprire il file\n");
    }
else
    {
    fwrite(v,sizeof(Persona),n,fp);
    fclose(fp);
    }
return 0;
}
```

# File ad accesso diretto

Il C consente di gestire i file non solo come sequenziali, ma anche come file ad accesso diretto.

## Posizionamento in un file:

La funzione ***fseek*** della Standard Library consente il posizionamento del puntatore al file su un qualunque byte.

**int fseek (FILE *f, long *offset*, int *origin*)**

si sposta di *offset* byte a partire dalla posizione *origin* (che vale 0, 1 o 2).

Restituisce:

- 0 se ha spostato la posizione sul file
- un valore diverso da 0, altrimenti.

## Origine dello spostamento:

| | |
|---|---|
| SEEK_SET 0 | inizio file |
| SEEK_CUR 1 | posizione attuale nel file |
| SEEK_END 2 | fine file |

Per posizionarsi all'inizio di un file gia' aperto e' possibile utilizzare anche la funzione ***rewind***:

**void rewind (FILE *f)**

```
file=fopen(argv[1], "r+");
...;
rewind(file);
```

equivale a:

```
fseek(f,0,SEEK_SET);
```

## Posizione corrente nel file:

La funzione ***ftell*** restituisce la posizione del byte sul quale si e' posizionati nel file al momento della chiamata della funzione (restituisce -1 in caso di errore):

**long ftell (FILE *f)**

Il valore restituito da ***ftell*** puo' essere utilizzato in una chiamata ad ***fseek***.

## Esempio:

Programma che sostituisce tutte le minuscole in maiuscole in un file testo dato come (unico) argomento.

```
#include <stdio.h>
#include <ctype.h>
main(int argc, char **argv)
{   FILE *file;
    void stop(char *);
    int ch;
if (argc==2)
 {if ((file=fopen(argv[1], "r+"))==NULL)
     stop("Impossibile aprire file
          d'ingresso\n");
 }
else stop("Manca qualche parametro\n");
while((ch=getc(file))!=EOF)
  if(islower(ch))
   {fseek(file, ftell(file)-1, SEEK_SET);
    putc(toupper(ch), file);
    fseek(file, 0, SEEK_CUR);
}
fclose(file);
exit(0);
}
void stop(char *msg)
{fprintf(stderr,msg);
 exit(1);
}
```

## Note sull'esempio:

- Il file e` aperto con modalita` "r+" (aggiornamento, ma posizione all'inzio del file).

- Il programma legge ad uno ad uno i caratteri del file e, quando trova una lettera minuscola (funzione ***islower*** della libreria ***ctype***), retrocede con ***fseek*** di una posizione e la sostituisce con la corrsipondente maiuscola (funzione ***toupper***)

- L'utilizzo della funzione ***fseek*** e` utilizzata per riposizionarsi sul carattere appena letto, se questo e` una lettera minuscola:

```
fseek(file, ftell(file)-1, SEEK_SET);
```

- E` inoltre ***obbligatoria*** per poter alternare scritture e letture su file:

```
fseek(file, 0, SEEK_CUR);
```

- L'apertura di un file in modo di aggiornamento "+" (abbinato ad uno qualunque tra "r", "w", "a") richiede esplicitamente che, dopo una sequenza di letture, prima di iniziare qualunque scrittura venga usata una delle funzioni di posizionamento su file (e analogamente per scritture seguite da letture).