I file

Il file è l'unità 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 è una sequenza di componenti omogenei (***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 funzionalità 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;

File di testo

Sono file di caratteri, organizzati in linee.
Ogni linea è terminata da un marcatore di fine linea (***newline***, carattere `'\n'`).

Egr. Sig. Rossi,▒
▒
con la presente le rendo noto di aver▒ provveduto…

☞Il ***record logico*** può 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, è necessario "scorrere" tutti quelli che lo precedono.

				X				…

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

Puntatore a file

È una variabile che viene utilizzata per fare riferimento ad un file nelle operazioni di accesso (lettura e scrittura). Implicitamente essa indica:

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

Esempio:

```
FILE *fp;
```

☞Il tipo `FILE` è un tipo non primitivo dichiarato nel file **`stdio.h`**.

Gestione di file in C

Apertura di un file

Prima di accedere ad un file, è necessario ***aprirlo***: l'operazione di apertura compie le azioni preliminari necessarie affinché 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, è possibile leggere/scrivere il file, facendo riferimento ad esso mediante il puntatore a file.

Chiusura di un file

Alla fine di una sessione di accesso (lettura o scrittura) ad un file, è 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`** è un array di caratteri (stringa) che rappresenta il nome (assoluto o relativo) del file nel file system;
- **`mode`** è una stringa che esprime la modalità di accesso scelta:
  - **`"r"`**, in lettura (read)
  - **`"w"`**, in scrittura (write)
  - **`"a"`**, scrittura, aggiunta in fondo (append)
  - **`"b"`**, a fianco ad una delle precedenti, indica che il file è binario
  - **`"t"`**, a fianco ad una delle precedenti, indica che il file è di testo

- Se eseguita con *successo*, l'operazione di apertura restituisce come risultato un *puntatore* al file aperto.

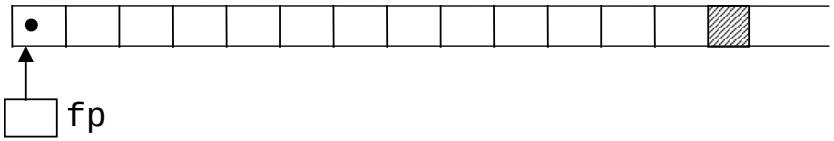
- Se, invece, l'apertura fallisce (ad esempio, perché il file non esiste), `fopen` restituisce il valore `NULL`.

Apertura

Apertura in lettura

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

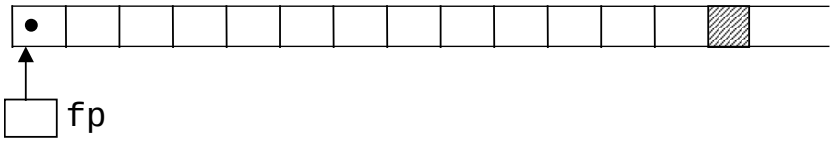
Se il file non è vuoto:



Apertura in scrittura

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

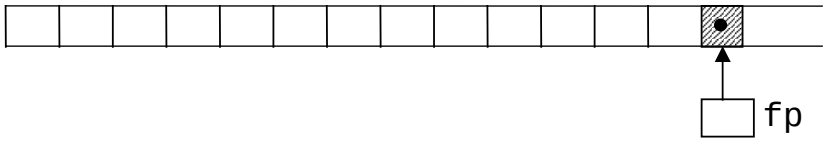
Anche se il file non è vuoto:



- Se il file esisteva già, il suo contenuto viene **perso**.

Apertura in aggiunta (append)

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



Il puntatore al file si posiziona sull'elemento successivo all'ultimo significativo del file ➠ se il file esisteva già, il suo contenuto **non** viene perso.

Esempio:

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

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

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` restituisce come risultato un intero

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

Esempio:

```
#include <stdio.h>

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 dell'esecuzione di ogni programma:

- **`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* definite nel file `stdio.h` ➠ **non vanno ri-definite**.

Funzione `feof()`

Durante la fase di accesso ad un file è possibile verificare la presenza del marcatore di fine file con la funzione di libreria:

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

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

Lettura e scrittura di file

Una volta aperto un file, ad esso si può 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` è il puntatore al file;
- *stringa-controllo* indica il formato dei dati da leggere;
- *ind-elem* è l'elenco 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);
```

`fscanf` restituisce il numero di elementi letti, oppure, in caso di errore, un valore negativo.

Scrittura con formato:

Si usa la funzione `fprintf`

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

dove:

- `fp` è il puntatore al file;
- *stringa-controllo* indica il formato dei dati da scrivere;
- *elementi* è l'elenco dei valori (espressioni) da scrivere.

Esempio:

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

`fscanf` restituisce il numero di elementi scritti, oppure, in caso di errore, un valore negativo.

`printf/scanf` e `fprintf/fscanf`:

Notiamo che:

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

equivalgono a:

```
fprintf(stdout, stringa-controllo,
elementi);
fscanf(stdin, stringa-controllo, ind-
elementi);
```


Esempio:

Visualizzazione del contenuto di un file di testo:

```
#include <stdio.h>

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);
}
```

oppure:

```
#include <stdio.h>

main() {
  char buf[80];
  FILE *fp;

  fp=fopen("testo.txt", "r");
  while(fscanf(fp, "%s", buf)>0)
    printf("%s", buf);
  fclose(fp);
}
```

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);
   exit(1);
  }
  else {
   filecopy(fp, stdout);
   fclose(fp);
  }
 return 0;
}

void filecopy(FILE *inputFile,
 FILE *outputFile) {
 int c;

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

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);
```

- per non perdere dati, la funzione **`exit`** chiude automaticamente ogni file aperto;
- 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 funzione **`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'` né 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 può 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 è 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 più) *n* oggetti, ciascuno di dimensione *size*, dal file puntato da *fp*, collocandoli nel vettore *vet*. 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;

main() {
 FILE *fp;
 persona p;
 int fine=0;

 fp=fopen("archivio.dat", "wb");
 do {
  printf("Dati persona?");
  scanf("%s%s%d%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);
}
```

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;

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);
}
```


Esempio:

Programma che riceve come argomento il nome di un file e scrive in questo file un vettore di interi.

```
#include <stdio.h>
void stop(char *);

int main(int argc, char **argv) {
 FILE *file;
 int i, n, tab[]={3, 6, -12, 5, -76, 3,
  32, 12, 65, 1, 0, -9};

 if(argc!=2) stop("Manca il nome del
  file di uscita\n");
 if((file=fopen(argv[1], "wb"))==NULL)
  stop("Impossibile aprire il file
   d'uscita\n");
 n = sizeof(tab)/sizeof(tab[0]);
 fwrite(tab, sizeof(tab[0]), n, file);
 fclose(file);
}

void stop(char *msg) {
 fprintf(stderr, msg);
 exit(1);
}
```

Esempio:

Programma che riceve come argomento il nome di un file 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
void stop(char *);

int main(int argc, char **argv) {
 FILE *file;
 int i, n, tab[MAX];

 if(argc!=2) stop("Manca il nome del
  file d'ingresso\n");
 if((file=fopen(argv[1], "rb"))==NULL)
  stop("Impossibile aprire il file
   d'ingresso\n");
 n=fread(tab, sizeof(tab[0]), MAX,
  file);
 fclose(file);

 for(i=0; i<n; i++)
  printf("%d%c", tab[i], (i==n-1)?
   '\n' : '\t');
}

void stop(char *msg) {
 fprintf(stderr, msg);
 exit(1);
}
```


Esempio:

File di record.

```
#include <stdio.h>
#define DIM 5

typedef struct {
 char nome[15],
 cognome[15],
 via[10];
 int eta;
} Persona;

int crea_vettore(Persona V[], int dim);
Persona P[DIM];

int main(int argc, char **argv) {
 int i, n;
 FILE *file;

 if(argc!=2) {
  printf("Manca qualche parametro\n");
  exit(1);
 }
 n=crea_vettore(P, DIM);
 if((file=fopen(argv[1], "wb"))==NULL){
  printf("Impossibile aprire
   il file\n");
  exit(2);
 }
 fwrite(P, sizeof(Persona), n, file);
 fclose(file);
}
```

```
int crea_vettore(Persona P[], int dim){
 int i=0, n=0; char s[80];
 while((!feof(stdin)) && (i<dim)) {
  scanf("%s\n", P[i].nome);
  scanf("%s\n", P[i].cognome);
  scanf("%s\n", P[i].via);
  scanf("%d", &(P[i].eta));
  gets(s);
  i++;
  n++;
  printf("%s\n%s\n%s\n%d\n",
   P[i].nome, P[i].cognome, P[i].via,
   P[i].eta);
 }
}
```


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 già aperto è possibile utilizzare anche la funzione **rewind**:

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

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

equivale a:

```
file=fopen(argv[1], "r+");
...;
fseek(f, 0, SEEK_SET);
```

Posizione corrente nel file:

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

```
long ftell (FILE *f);
```

Il valore restituito da **ftell** può 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>
void stop(char *);

int main(int argc, char **argv) {
 FILE *file;
 int ch;
 if(argc!=2)
  stop("Manca qualche parametro\n");
 if((file=fopen(argv[1], "r+"))==NULL)
  stop("Impossibile aprire il file
   d'ingresso\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);
}

void stop(char *msg) {
 fprintf(stderr, msg);
 exit(1);
}
```

Note sull'esempio:

- Il file è aperto con modalità `"r+"` (aggiornamento, ma posizione all'inizio 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 corrispondente maiuscola (funzione **`toupper`**).

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

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

- È 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).