

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 sottostante. Per ogni versione C esistono funzioni per il trattamento dei file (**Standard Library**) che tengono conto delle funzionalita` del S.O.

In C i file vengono distinti in due categorie:

- **file di testo**, costituiti da caratteri ed 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 e' terminata da una marca di fine linea (***newline***, carattere '\n').

```
Egregio Sig. Rossi,
                        con la presente
le rendo noto di aver provveduto
...
```

☞ Il ***record logico*** puo' essere il singolo carattere oppure la singola linea.

Organizzazione degli archivi

Dispositivi di memoria di massa:

- **Ad accesso sequenziale**, per accedere ad un blocco occorre scandire tutti quelli che lo precedono (ad esempio, nastri).
- **Ad accesso diretto**, ciascun blocco e' indirizzabile (ad esempio, dischi).

I dati sono organizzati in memoria di massa tenendo conto di due obiettivi:

- memorizzazione compatta di molti dati su dispositivi di capacita` limitata;
- capacita` di recuperare in modo efficiente i dati.

Metodi base per l'organizzazione degli archivi (**strutture dati**):

- Sequenziale
- Ad accesso diretto (o casuale)
- Ad indice

A ciascuna struttura dati corrispondono diversi modi nei quali è possibile utilizzarla (ad esempio, diversi **metodi di elaborazione**).

Gestione di file in C

Sono *file sequenziali*, con possibilita' di *accesso diretto*.

Apertura di un file:

- ☞ Prima di accedere ad un file e' necessario **aprirlo**:
l'operazione di apertura compie le azioni preliminari necessarie affinché il file possa essere acceduto (in lettura o in scrittura)

Per aprire un file in C:

- e' necessario definire una variabile di tipo *puntatore a FILE*:
`FILE *fp;`
- ☞ il tipo FILE e' un tipo non primitivo dichiarato nel file **stdio.h**.
- si utilizza la specifica **funzione di apertura**:

FILE *fopen(char *name, char *mode);

Gestione di file in C

Accesso ad un file:

Dipendentemente dalla modalita' di apertura scelta il file puo' essere acceduto con diversi strumenti:

fprintf
fscanf
fread
fwrite
etc.

Chiusura di un file:

Alla fine di una sessione di accesso ad un file e' necessario chiudere il file per memorizzare permanentemente il suo contenuto in memoria di massa:

- si utilizza la specifica **funzione di apertura**:

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

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", in lettura (read)
 - "w", in scrittura (write)
 - "a", scrittura, aggiunta in fondo (append)
 - "b", a fianco ad una delle precedenti,
 indica che il file e` binario
 - "t", a fianco ad una delle precedenti,
 indica che il file e` di testo

L'operazione di apertura ritorna come risultato un ***puntatore*** al file aperto:

FILE ***fopen(char **name*, char **mode*);**

Il file deve esistere in memoria di massa. Altrimenti restituisce il valore NULL.

Ad 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 a 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 ritorna 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, ritorna la costante EOF.

- ☞ **fclose** assegna il valore NULL al puntatore fp.

Esempio:

```
#include <stdio.h>
main()
{
    FILE *fp;
    fp = fopen("prova.dat", "w")
    ...
    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:

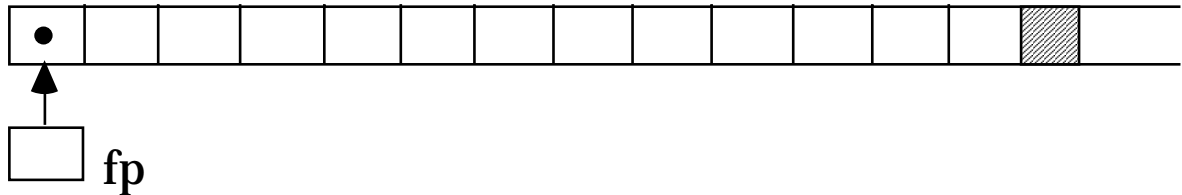
int feof(FILE *fp);

- ☞ feof(fp) controlla se e' stata raggiunta la fine del file fp nella 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).

Apertura in lettura

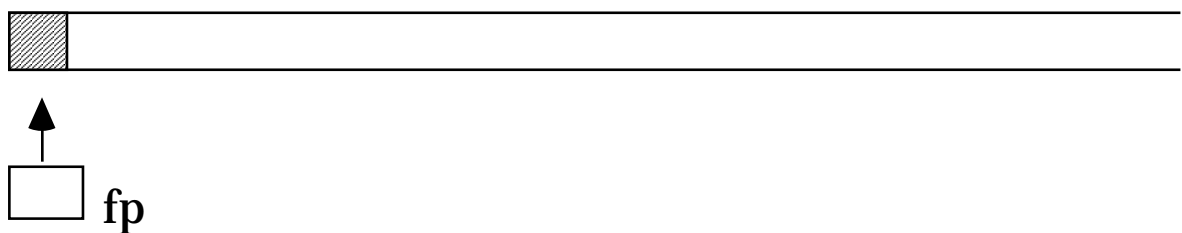
```
fp = fopen("filename", "r")
```

- Se il file non e` vuoto:



feof(fp) ha valore falso (valore zero).

- Se il file e` vuoto:

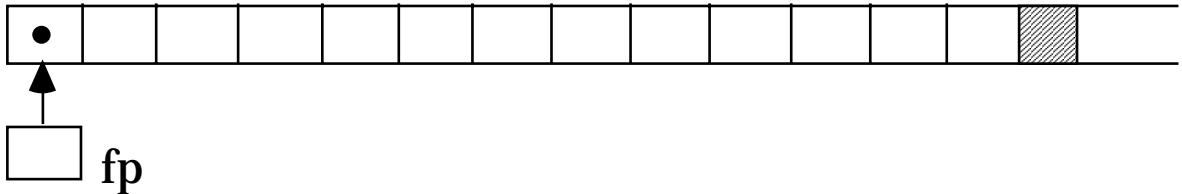


feof(fp) ha valore vero (valore diverso da zero).

Apertura in scrittura

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

- Anche se il file non e' vuoto:

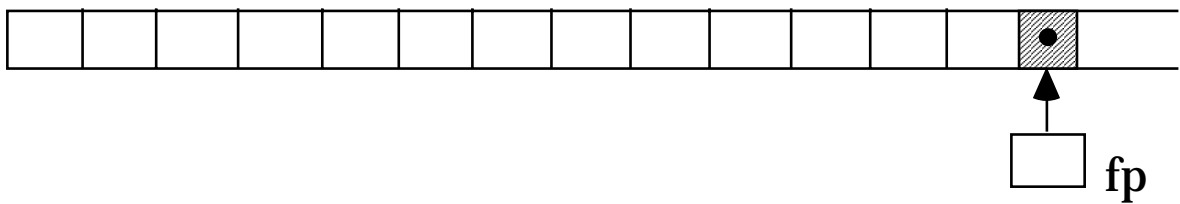


➤ Se il file esisteva già, il suo contenuto viene **perso**.

👉 **feof(f)** ha sempre valore vero.

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 esisteva già, il suo contenuto non viene perso.

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.

Quattro diverse modalita':

- Lettura/scrittura **con formato**
- Lettura/scrittura di **caratteri**
- Lettura/scrittura di **stringhe di caratteri**
- Lettura/scrittura di **blocchi** (file **binari**)

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:

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

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

Restituiscono il numero di elementi scritti o letti, oppure un valore negativo in caso di errore.

 **notare** che:

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

equivalgono a:

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

Esempio:

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

```
void riempifile(char *nome, char *str,  
int n);  
void copiafile(char *nome1, char *nome2);  
void vedifile(char *nome);
```

```
main()  
{  
char buf[20]="ciao!";  
  
riempifile("Pippo", buf, 2);  
vedifile("Pippo");  
copiafile("Pippo", "Pluto");  
vedifile("Pluto");  
}
```

```
void riempifile(char *nome, char *str,  
int nvolte)  
{  
FILE *fp;  
int i;  
fp=fopen(nome, "w");  
for (i=0; i<nvolte; i++)  
    fprintf(fp, "%s\n", str);  
fclose(fp);  
}
```

```

void copiafile(char *nome1, char *nome2)
{
    FILE *fp1, *fp2;
    char C;

    fp1=fopen(nome1, "r");
    fp2=fopen(nome2, "w");

    while (!feof(fp1))
    {
        fscanf(fp1, "%c", &C);
        fprintf(fp2, "%c", C);
    }

    fclose(fp1);
    fclose(fp2);
}

```

```

void vedifile(char *nome)
{
    FILE *fp;
    char C;
    fprintf(stdout, "contenuto di %s:\n", nome);
    fp=fopen(nome, "r");
    while (!feof(fp))
    {
        fscanf(fp, "%c", &C);
        fprintf(stdout, "%c", C);
    }
    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>

main(int argc, char **argv)
{
    FILE *fp;
    void filecopy(FILE *, FILE *);

    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:

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

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.

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).

Il test di fine del file viene ottenuto con la funzione feof (non e' piu' possibile il confronto tra il carattere letto ed EOF).

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

☞ feof(fp) controlla se e' stata raggiunta la fine del file fp nella operazione di lettura o scrittura precedente. Restituisce il valore 0 se non e' stata raggiunta la fine del file, altrimenti un valore diverso da zero.

Lettura/Scrittura a Blocchi

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).

Esempio:

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

```
#include <stdio.h>

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

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

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 piu' 40 elementi).

```
#include <stdio.h>
#define MAX 40

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

 if (argc==2)
 {if ((file=fopen(argv[1], "rb"))==NULL)
 stop("Impossibile aprire file
 d'ingresso\n");
 }
 else stop("Manca il nome del 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');
 exit(0);
}

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

Esempio:

File di record.

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

Persona P[DIM];

main(int argc, char **argv)
{
    int crea_vettore(Persona V[], int dim);
    int i, n;
    FILE *file;

    if (argc==2)
        {n=crea_vettore(P,DIM);

            if ((file=fopen(argv[1], "w"))==NULL)
                printf("Impossibile aprire
                        file\n");
            else
                {
                    fwrite(P,sizeof(Persona),n,file);
                    fclose(file);
                }
        }
    else printf("Manca qualche parametro\n");
}
```

```
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(f);
```

equivale a:

```
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***.

☞ Nella chiamata alla funzione ***fseek***, per un ***file testo*** è obbligatorio che il parametro *offset* sia 0.

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);
            /* fseek(file, ftell(file), SEEK_SET); */
            }
    fclose(file);
    exit(0);
}
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).

Altre funzioni relative ai file

- `int remove (char *name);`
Cancella il file che ha per nome la stringa costante *name*. Restituisce il valore 0 se l'operazione di chiusura è eseguita correttamente, altrimenti un valore diverso da zero.
- `int rename (char *oldname, char *newname);`
Modifica il nome di un file da *oldname* a *newname*. Restituisce il valore 0 se l'operazione di chiusura è eseguita correttamente, altrimenti un valore diverso da zero.
- `int ferror (FILE *fp);`
Controlla se è stato commesso un errore nella operazione di lettura o scrittura precedente. Restituisce il valore 0 se non c'è stato errore, altrimenti un valore diverso da zero.
- `void clearerr (FILE *fp);`
Riporta al valore di default lo stato delle informazioni (*eof* ed *error*) di fine file ed errore associate al file.