LA PERSISTENZA DEI DATI

- Le variabili usate finora sono *volatili*, cioè la loro vita termina con la terminazione dell'esecuzione del programma.
- Il concetto di *file* fornisce la possibilità di memorizzare dati in forma *persistente*, cioè in una forma che sopravvive al termine dell'esecuzione del programma che li ha generati e che ne permette l'utilizzo in tempi successivi.
- I dispositivi di *memoria di massa* (detti anche *memorie secondarie*) forniscono il supporto fisico alla memorizzazione dei file.

L'utilizzo di un file da parte di un programma richiede :

- dichiarazione di una variabile di tipo file;
- associazione della variabile ad una struttura dati del sistema operativo su cui il programma verrà eseguito;
- *apertura* del file, specificando se si intendono effettuare operazioni di scrittura (output), lettura (input) o miste;
- esecuzione delle operazioni di scrittura e lettura (*accesso*);
- chiusura del file, per terminare le operazioni.

- Un altro motivo per usare la memoria secondaria può essere la *dimensione*.

Esempio, un programma che fa uso di una variabile vettore. All'esecuzione del programma dovrà essere disponibile in memoria centrale lo spazio per *tutto* il vettore. In realtà, la memoria centrale ha sempre dimensioni ben delimitate, in generale di gran lunga inferiori a quelle della memoria secondaria, e sono assai frequenti problemi applicativi con dimensioni tali da rendere impossibile l'allocazione in memoria di un intero vettore. Anche in tale caso risulta necessario fare ricorso a strutture dati su disco.

FILE

file strutturato

è una sequenza, di lunghezza non prefissata, di valori dello stesso tipo.

differenza fra l'uso di un vettore e di un file è che per il primo l'accesso ad ogni componente è immediato; viceversa, l'accesso ad un componente di file richiede che:
1. venga individuata la posizione in memoria di massa in cui il componente risiede;
2. il componente venga *copiato* in una posizione di memoria tramite un accesso.

Nel linguaggio C

Un file è solamente una sequenza, di lunghezza non prefissata, di byte o caratteri.

File di testo → sequenza di caratteri (per memorizzare caratteri)
File binario → sequenza di byte (per memorizzare interi, record, ….)

TIPOLOGIE DI ACCESSO AI FILE

1) accesso *sequenziale*

è possibile accedere ad un componente soltanto dopo avere effettuato l'accesso a quelli che lo precedono

- al momento dell'apertura del file viene reso diponibile il primo componente, e da lì si procede in sequenza (stessa cosa in fase di scrittura)
- In ogni momento di utilizzo del file esiste quindi la nozione di *componente corrente*, che è il componente su cui avverrà il prossimo accesso (lettura o scrittura).
- Il modello dell'accesso sequenziale è quello delle unità di memoria a *nastro* o a *schede*, in cui, per motivi fisici, l'accesso poteva essere soltanto sequenziale.

2) accesso *diretto*

è possibile specificare a quale componente si intende accedere

- Il modello dell'accesso *diretto* è quello delle unità di *memoria a disco*
- uso del file in parte analogo a quello del vettore. L'analogia è soltanto parziale, in quanto l'accesso ad un componente di file richiede sempre una operazione di input/output.

FILE E PROGRAMMI C

Nel Sistema Operativo - **SO** - (e non nel programma C) è definita una struttura dati `FILE` che contiene, tra l'altro, le seguenti informazioni

- il nome del file
- un puntatore al prossimo byte del file da leggere o scrivere
- modalità di accesso al file

Il SO gestisce il seguente vettore

```
FILE TabellaFileAperti[MaxNumeroFile]
```

dove `MaxNumeroFile` è il massimo numero di file che possono essere gestiti contemporaneamente dal Sistema Operativo.

Quando un programma apre un file specificando il nome e le modalità di utilizzo, il SO
crea un nuovo elemento in `TabellaFileAperti` inizializzando opportunamente i campi di tale elemento e restituisce al programma l'indirizzo della struttura di tipo FILE. È il SO a manipolare direttamente i campi della struttura FILE, mentre il programma ne provoca la manipolazione indiretta invocando funzioni della Standard Library <stdio.h>.

I dispositivi standard di ingresso (la tastiera) e di uscita (il video) sono disponibili come file testo: quando inizia l'esecuzione di un programma, il SO apre tali file e li associa ai seguenti puntatori

`stdin` variabile puntatore al file che rappresenta la tastiera
`stdout` variabile puntatore al file che rappresenta il video

Il SO apre anche un file per la gestione degli errori:
`stderr` variabile puntatore al file che rappresenta il video

OPERAZIONI DI GESTIONE

Per poter leggere o scrivere su un file sono necessarie le seguenti operazioni

• **dichiarazione del file : `FILE *FP;`**

dichiara FP (*file pointer*) come un puntatore a FILE;

• **apertura del file : `FILE *fopen(char *NomeFile,char *Modo);`**

restituisce un puntatore al file specificato in `NomeFile` al quale sarà possibile accedere secondo la modalità riportata in `Modo`.
Se si verifica un errore, la funzione restituisce un puntatore con valore **`NULL`**.

Modalità di apertura di un file

r : sola lettura
w : sola scrittura a partire dall'inizio del file
a : sola scrittura a partire dalla fine del file (*append*)
r+: lettura e scrittura
w+:scrittura a partire dall'inizio del file
a+ : lettura e scrittura a partire dalla fine del file

- Se un file che **esiste** è aperto in modalità **w** o **w+**, il contenuto del file è perso
- Se un file che **non esiste** è aperto in modalità **w, w+, a** oppure **a+**, esso viene creato
- Se un file che **non esiste** è aperto in modalità **r** oppure **r+** viene generato un errore

• **chiusura del file :`int fclose(FILE *FP)`**

assegna a FP il valore NULL; se l'operazione è stata eseguita correttamente restituisce 0, altrimenti restituisce EOF.

ESEMPIO

```
#include <stdio.h>
main(){
 FILE *FP; // dichiarazione
 FP=fopen("prova.txt","r"); // apertura
   . . .
  fclose(FP); //chiusura
}
```

ALTRE OPERAZIONI DI GESTIONE

Funzioni	Significato
`Int remove(char *Nome)`	**Cancellazione del file** cancella il file specificato in `Nome` se l'operazione è stata eseguita correttamente restituisce 0, altrimenti restituisce un valore !=0
`Int rename(char *OldN, char *NewN)`	**Rinomina del file** modifica il nome del file `OldN` in `NewN` se l'operazione è stata eseguita correttamente restituisce 0, altrimenti restituisce un valore !=0

FILE TESTO

Per leggere/scrivere da file testo, la libreria standard fornisce funzioni analoghe a quelle utilizzate per l'input e l'output standard. Queste funzioni possono essere classificate nelle seguenti categorie:

- Lettura e scrittura a caratteri
- Lettura e scrittura a stringhe
- Lettura e scrittura formattato

I dispositivi standard di ingresso (la tastiera) e di uscita (il video) sono disponibili come file testo: quando inizia l'esecuzione di un programma, il SO apre tali file e li associa ai seguenti puntatori

`stdin` variabile puntatore al file che rappresenta la tastiera
`stdout` variabile puntatore al file che rappresenta il video

Il SO apre anche un file per la gestione degli errori:
`stderr` variabile puntatore al file che rappresenta il video

lettura e scrittura a caratteri

Funzioni	Significato
`int getc(FILE *FP)`	Restituisce, come intero, il prossimo carattere del file FP.
`int putc(int Ch, FILE *FP)`	scrive il carattere Ch nel file FP restituendo, come intero, il carattere scritto.
`int getchar(void)`	Legge da stdin il prossimo carattere e lo restituisce come intero Equivale a getc(stdin)
`Int putchar(int Ch)`	scrive il carattere Ch su stdout e lo restituisce come intero Equivale a getc(Ch,stdout)

Tutte le funzioni restituiscono il carattere **`EOF`** in caso di fine file o di errore

- Da tastiera, l'inserimento del carattere **`EOF`** avviene
  in DOS : premendo i tasti **`CONTROL + Z`** e poi **`ENTER`**
  in UNIX : premendo i tasti **`CONTROL + D`**
  in MACINTHOS : premendo i tasti **`CONTROL + D`**

ESEMPIO: LETTURA E VISUALIZZAZIONE DI UN FILE DI TESTO

- apri il file in lettura
- se il file esiste e si può aprire in lettura
- prendi come Ch il primo carattere dal file
- finchè non è stata raggiunta la fine del file
- visualizza il carattere Ch
- prendi come Ch il prossimo carattere dal file

Il controllo che non è stata raggiunta la fine del file FP viene effettuato leggendo il carattere Ch e controllando che tale carattere sia diverso da EOF

```
#include <stdio.h>
main(){
 FILE *FP;
 char Ch;
if ((FP=fopen("prova.txt","r"))!=NULL) {
  Ch=getc(FP);
  while (Ch!=EOF){
   putchar(Ch);
   Ch=getc(FP);
 }/* end while */
  fclose(FP); // chiusura del file
}/* end if */
else
  printf("Errore nell'apertura in lettura del file");
} /* end main */
```

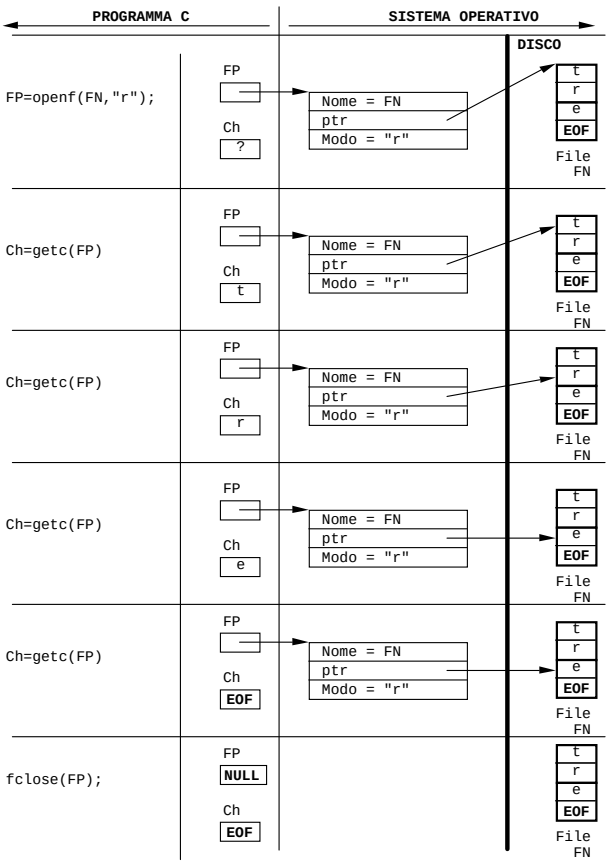
Si può effettuare la lettura del carattere direttamente nella verifica della condizione

```
while ((Ch=getc(FP))!=EOF)
 putchar(Ch);
```


ESEMPIO: SCRITTURA DI UN FILE DI TESTO (A CARATTERI)

Acquisire un testo da tastiera (fino al carattere EOF) e memorizzarlo su un file.

```
#include <stdio.h>
main(){
 FILE *FP;
 char Ch;
if ((FP=fopen("prova.txt","w"))!=NULL) {
  Ch=getchar();
  while (Ch!=EOF){
   putc(Ch,FP);
   Ch=getchar();
 }/* end while */
  fclose(FP); // chiusura del file
}/* end if */
else
  printf("Errore nell'apertura in scrittura del file");
} /* end main */
```

ESEMPIO: MODIFICA DI UN FILE DI TESTO (A CARATTERI)

GESTIONE SEQUENZIALE

Modificare un file di testo, convertendo tutti i suoi caratteri alfabetici in maiuscolo. Con l'accesso sequenziale l'unico modo di fare questa operazione è quello di usare un file temporaneo sul quale ricopiare il file modificato e quindi, dopo aver cancellato il file originale, dare al file temporaneo il nome del file originale.

```
#include <stdio.h>
main(){
 FILE *FP; // puntatore al file da modificare
 FILE *FT; // puntatore al file temporaneo
 char Ch;
if ((FP=fopen("prova.txt","r"))!=NULL) {
   if ((FT=fopen("temp.txt","w"))!=NULL) {
       Ch=getc(FP);
       while (Ch!=EOF){
         if (Ch>='a' && Ch<='z') Ch = Ch -('a'-'A');
         putc(Ch,FT);
         Ch=getc(FP);
        }/* end while */
      fclose(FT); // chiusura del file temporaneo
      fclose(FP); // chiusura del file da modificare
      remove("prova.txt"); // cancella del file da modificare
      rename("temp.txt","prova.txt"); // rinomina
    }/* end if */
  else   printf("Errore nell'apertura del file temporaneo");
  fclose(FP); // chiusura del file da modificare
}/* end if */
else  printf("Errore nell'apertura del file da modificare");
} /* end main */
```

GESTIONE DEGLI ERRORI

SERVONO PER VERIFICARE E GESTIRE SITUAZIONI DI ERRORE

Funzioni	Significato
`int ferror(FILE *FP)`	restituisce il valore 0 se non è stato commesso nessun errore nella **precedente** operazione di lettura o scrittura sul file FP altrimenti restituisce un valore maggiore di 0
`int feof(FILE *FP)`	restituisce il valore 0 se non è stata raggiunta la fine del file nella **precedente** operazione di lettura o scrittura sul file FP altrimenti restituisce un valore maggiore di 0
`void clearerr(FILE *FP)`	azzera gli errori e la condizione di fine file

Esempio di Gestione Errori

```
#include <stdio.h>
void CheckFile( FILE *fp);
main(){
 FILE *FP;
 char Ch;
 FP = fopen( "prova.txt", "w" );
 CheckFile(FP); /* Non ci sono errori. */
 c = fgetc(FP);/* questo genera un errore */
 CheckFile(FP); /* C'e' un errore */

 clearerr(FP);/* si azzera il precedente errore   */
 CheckFile(FP); /* Non ci sono errori. */

 fclose(FP);
} /* end main */

void CheckFile(FILE *fp ) {
 if ( ferror(fp) || feof(fp) )
  printf( "C'e' un errore\n" );
 else
  printf( "Non ci sono errori.\n" );
} /* end CheckFile */
```


LETTURA E SCRITTURA A STRINGHE

Funzioni	Significato
`char *fgets(char *S, int n, FILE *FP)`	Riporta nella stringa puntata da S i caratteri letti dal file puntato da FP fino a `'\n'` o al fine file oppure fino a quando non ha letto n-1 caratteri. Nella stringa viene inserito l'eventuale `'\n'` e viene terminata con `'\0'`. Se non si verificano errori restituisce un puntatore al primo elemento della stringa in cui vengono messi i caratteri letti, altrimenti restituisce **NULL**.
`int fputs(char *S, FILE *FP)`	Scrive sul file puntato da FP la stringa puntata da S, L'eventuale `'\0'` della stringa **non** viene scritto. Restituisce 0 se non si verificano errori, un valore diverso da 0 in caso contrario.
`Char *gets(char *S)`	Riporta nella stringa puntata da S i caratteri letti da `stdin` fino a `'\n'` o al fine file. Nella stringa **non** viene inserito l'eventuale `'\n'` e viene terminata con `'\0'`. Se non si verificano errori restituisce un puntatore al primo elemento della stringa in cui vengono messi i caratteri letti, altrimenti restituisce **NULL**.
`Int puts(char *S)`	Scrive su `stdout` la stringa puntata da S, seguita da `'\n'`. L'eventuale `'\0'` della stringa non viene scritto. Restituisce 0 se non si verificano errori, un valore diverso da 0 in caso contrario.

ESEMPIO: LETTURA E SCRITTURA DI UN FILE DI TESTO (A STRINGHE)

Dato un file "origine" si vuole creare un altro file, "destinazione", che ha lo stesso contenuto del file "origine" ma con una spaziatura doppia: questo si realizza riportando su "destinazione", dopo ogni linea letta da "origine", una linea contenente come unico carattere '\n'

```
#include <stdio.h>
# define MAXLINEA 100
main(){
char linea[MAXLINEA];
FILE *FIN,*FOUT;

if ((FIN=fopen("origine","r"))==NULL)
  printf("non posso aprire il file origine");
else
  if ((FOUT=fopen("destinazione","w"))==NULL)
  {
    printf("non posso aprire il file destinazione");
    fclose(FIN);
  }
  else
  { /* caso di "origine" e "destinazione" aperti corret. */
    while (fgets(linea,MAXLINEA,FIN)!=NULL)
    {
      fputs(linea,FOUT);
      fputs("\n",FOUT);
    } /* end while */
   fclose(FIN);
   fclose(FOUT);
  } /* end caso di "origine" e "destinazione" aperti corret. */
} /* end main */
```

ESEMPIO: INSERIMENTO ORDINATO IN UN FILE- GESTIONE SEQUENZIALE

```
- acquisire il nome del file NF
- finche' ci sono dati R da inserire in NF
 - inserimento ordinato di R in NF
```

L'inserimento ordinato di R in NF tramite una gestione sequenziale può essere strutturato come segue

```
- se NF non esiste
     - apri NF in scrittura
     - inserisci R in  NF
  altrimenti
     - apri NF in lettura
     - predisponi un file nuovo "temp" in scrittura
     - trasferisci tutti i dati di NF minori di R  in "temp"
    - scrivi R  in "temp"
     - trasferisci i restanti dati di NF in "temp"
     - cancella NF e rinomina "temp" come NF
```

Come ultimo livello di dettaglio, esprimiamo i trasferimenti di dati da NF al file temponaneo.

```
- finche' ci sono dati R da inserire in NF
   - se NF non esiste
      - apri NF in scrittura
      - inserisci R in  NF
     altrimenti
       - apri NF in lettura
       - predisponi un file nuovo "temp" in scrittura
       - prendi il primo dato da NF in A
       - finchè A è minore di R e NF non è finito
              - memorizza A nel file  "temp"
              - prendi il prossimo dato da NF in A
       - inserisci R in "temp"
       - se A non è minore di R
           - inserisci A in "temp"
           - prendi il prossimo dato da NF in A
           - finchè NF non è finito
                  - memorizza A nel file  "temp"
                  - prendi il prossimo dato da NF in A
       - cancella NF e rinomina "temp" come NF
```



```
/* CODICE*/

#include <stdio.h>
main()
{
 void FileShow(Stringa NomeFile);
/* funzione che stampa il contenuto di un file di caratteri */
 int getline(char *linea, int maxlun);
/* funzione per acquisire da input una stringa */
 void FileInsOrd(Stringa NomeFile, char R);
/* funzione per l'inserimento ordinato in un file di caratteri
*/

 char X;
 Stringa NF;

 printf("Nome del file > ");
  while (getline(NF,MAXSTRINGA)==0);

 while ((X=getchar())!='\n')
  FileInsOrd(NF,X);

 FileShow(NF);
} /* end main */
```



```
void FileInsOrd(char *NomeFile, char R){
 FILE *FI,*FO;
 char A;
 int EsisteFile(char *NF);
  if (!EsisteFile(NomeFile))
 { /* se il file non esiste  si crea e vi si introduce R */
    FI=fopen(NomeFile,"w");
    putc(R,FI);
    fclose(FI);
 } else
 { /* se il file esiste  si deve usare un altro file "temp" */
    FI=fopen(NomeFile,"r");
    FO=fopen("temp","w");
  /* finche' A < R e non c'e' fine del file FI
     si legge A da FI e lo si memorizza in FO */
    while (((A=getc(FI))!=EOF)&&(A<R))
            putc(A,FO);
  /* (A<R) o  fine del file FI: in ogni caso si mette R in FO */
     putc(R,FO);
  /* se si esce dal while perche' non ho (A<R) (cioe' ho
    (A!=EOF))allora ricopio A e gli altri char di FI in FO */
     if (A!=EOF)
     { putc(A,FO);
       while ((A=getc(FI))!=EOF)  putc(A,FO);     }
    fclose(FI);
    fclose(FO);
    remove(NomeFile);
    rename("temp",NomeFile);
 } } /* end  FileInsOrd */
```



```
int EsisteFile(char *NomeFile)
{
 FILE *FP;

 FP=fopen(NomeFile,"r");
 if (FP==NULL)
  return 0;
 else
 {
  fclose(FP);
   return 1;
 }
}

void FileShow(char *NomeFile){
 FILE *FP;
 char Ch;
 if ((FP=fopen(NomeFile,"r"))!=NULL)
 {
  while ((Ch=getc(FP))!=EOF)
        putchar(Ch); /* end while */
  fclose(FP);
 }
else
  printf("Errore nell'apertura del file");
} /* end FileShow */
/*
```

ACCESSO DIRETTO AL FILE

posizione-corrente

é un valore intero che rappresenta la posizione attuale del puntatore sul file: una operazione di lettura o scrittura sarà effettuata iniziando da tale posizione.

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

restituisce il valore della posizione corrente del puntatore

Il C consente l'accesso diretto al file tramite la funzione **fseek** che è in grado di modificare il valore della posizione corrente del puntatore.

```
int fseek(FILE *FP, long offset, int origine);
```

per il file FP, puntatore sul file viene spostato di **offset** byte rispetto a **origine**

origine = SEEK_SET --> **inizio del file**

origine = SEEK_CUR --> **posizione-corrente**

origine = SEEK_END --> **fine del file**

Restituisce 0 se la richiesta è corretta, altrimenti un valore diverso da zero.

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

Pone **a zero** il puntatore sul file..
Non restituisce nessun valore perchè l'eventuale errore viene azzerato.
Equivale a

```
fseek(FP, 0, SEEK_SET ); clearerr(FP);
```


ACCESSO DIRETTO AL FILE - ESEMPI

La seguente funzione determina il numero di elementi di un file, ovvero la sua dimensione. In caso di errori (esempio: file non esistente) restituisce il valore -1.

```
int DimensioneFile(char *NomeFile){
 FILE *FP;
 FP=fopen(NomeFile,"r");
 if (FP==NULL)
  return -1;
 else{
     /* posiziona il puntatore sul file alla fine de file */
  fseek(FP,0,SEEK_END);
       /* legge la posizione corrente del puntatore */
  n =ftell(FP);
  fclose(FP);
  return n;
 } }
```


ACCESSO DIRETTO AL FILE - ESEMPI

- Il seguente codice stampa in output l'ultimo carattere di ogni riga del file di testo NomeFile

```
...
if ((FIN=fopen(NomeFile,"r"))!=NULL) {
 while ((Ch=getc(FIN))!=EOF)
  if (Ch=='\n') {
/* per leggere il carattere che precedeva '\n' si deve tornare
indietro di 2 elementi*/
  fseek(FIN,-2,SEEK_CUR);
   Ch=getc(FIN);
/* dopo aver letto il carattere si deve avanzare di una
posizione, altrimenti si continua a leggere '\n'*/
   fseek(FIN,1,SEEK_CUR);
   putchar(Ch);
 } /* end while di lettura del file */
  fclose(FIN);
```


ESEMPIO: MODIFICA DI UN FILE DI TESTO (A CARATTERI) GESTIONE DIRETTA

Modificare un file di testo, convertendo tutti i suoi caratteri alfabetici in maiuscolo. Con l'accesso diretto non è più necessario usare un file temporaneo: quando si legge un carattere minuscolo ci si **sposta indietro** di una posizione e lo si **sovrascrive** con il corrispondente carattere maiuscolo.
Occorre aprire il file in modalità **r+** (lettura e scrittura); si noti che non è possibile aprirlo in modalita **w+**, in quanto il contenuto del file verrebbe perso, e neanche in modalità **a+**, in quanto, a parte il fatto che la modalità a comincia a scrivere dalla fine del file, se un file che non esiste è aperto in modalità **a+**, esso viene creato e non viene generato un messaggio di errore.

Nella modalità lettura e scrittura, per **passare dalla lettura alla scrit**tura , e viceversa, è **obbligatorio** effettuare una istruzione **fseek():** per fare una **fseek() "**senza spostarci" useremo **fseek(FN,0,SEEK_CUR)**

```
#include <stdio.h>
main(){
 FILE *FP; // puntatore al file da modificare
 char Ch;
if ((FP=fopen("prova.txt","r+"))!=NULL) {
      Ch=getc(FP);
       while (Ch!=EOF){
         if (Ch>='a' && Ch<='z') {
             Ch = Ch -('a'-'A');
              fseek(FP,-1,SEEK_CUR);
              putc(Ch,FP);
              fseek(FP,0,SEEK_CUR);          }
         Ch=getc(FP);  }/* end while */
     fclose(FP); // chiudo file da modificare  }/* end if */
 else  printf("Errore nell'apertura del file da modificare");
} /* end main */
```

RICERCA BINARIA IN UN FILE ORDINATO

Con l'accesso diretto si può accedere ad un preciso elemento del file ed è quindi possibile applicare la ricerca binaria ad un file ordinato. Per avviare la ricerca serve conoscere la dimensione del file. A tale scopo si usa la funzione **DimensioneFile**:

```
main(){
 FILE *FP;/* per aprire il file di char ordinato */
 char NF[20]; /* nome del file di char ordinato */
 char Ch;  /* carattere da ricercare */
 int dimensione; /* dimensione del file */
 int posizione  /* posizione di Ch all'interno del file */
 int DimensioneFile(char *NomeFile);
 int RicercaBinariaFile(FILE *FI,long inizio,long fine, char D);
... /* acquisisce NF e Ch */
 dimensione = DimensioneFile(NF);
 FP=fopen(NF,"r");
 posizione=RicercaBinariaFile(FP,1,dimensione,Ch);
 fclose(FP);
... /* stampa dei risultati*/
} /* end main */
```



```
int RicercaBinariaFile(FILE *FI,long inizio,long fine, char D){
  long c; /* posizione centrale */
  char Ch; /* elemento centrale */
  if (inizio>fine)
     return -1;
  else {
     c=(inizio+fine)/2;
     fseek(FI,c-1,SEEK_SET);
     Ch=getc(FI);
     if (Ch==D)
        return c;
     else
         if (Ch>D)
           return RicercaBinariaFile(FI,inizio,c-1,D);
         else
           return RicercaBinariaFile(FI,c+1,fine,D);
   }
} /* RicercaBinariaFile */
```


FILE BINARIO

Si può accedere ai dati di un file leggendo o scrivendo un intero blocco di dati specificando:

`*Ptr` l'indirizzo di un vettore per contenere gli elementi che compongono il blocco

`DimElem` le dimensioni in byte del singolo elemento

`NumElem` il numero di elementi ai quali si desidera accedere

• lettura di una struttura

```
size_t fread(void *Ptr,size_t DimElem,size_t NumElem,FILE *FP)
```

legge dal file `FP` nell'array `Ptr` al massimo `NumElem` elementi di dimensione `DimElem`
restituisce il numero di elementi letti: se tale numero è minore di `NumElem` significa che è stata raggiunta la fine del file oppure che si è verificato un errore di lettura (ciò va determinato tramite le funzioni `feof` e `ferror`).

• scrittura di una struttura

```
size_t fwrite(void *Ptr,size_t DimElem,size_t NumElem,FILE *FP)
```

scrive sul file `FP`, prendendoli dall'array `Prt`, `NumElem` elementi di dimensione `DimElem`
restituisce il numero di elementi scritti: se tale numero è minore di `NumElem` significa che si è verificato un errore di scrittura (ciò va determinato tramite la funzione `ferror`).

ESEMPIO: CREAZIONE DI UN FILE DI RECORD

- Struttura dati

```
#define MAXSTRINGA 20
#define MAXV 2 /* numero di esami */
typedef char Stringa[MAXSTRINGA];
typedef struct {
 Stringa Nome;
 int Voti[MAXV];
} Studente;
```

- Procedura per l'inserimento di un record alla fine di un file (**append**)

```
void FileAppend(Stringa NomeFile, Studente R){
 FILE *FI;
if ((FI=fopen(NomeFile,"a"))==NULL)
  printf("non posso aprire il file %s", NomeFile);
else {
 fwrite(&R,sizeof(Studente),1,FI);
 fclose(FI);
} /* end FileAppend */
```

- Accesso diretto ad un file di record. Si usa sempre la funzione

```
int fseek(FILE *FP, long offset, int origine);
```

Dato che lo spostamento specificato in **offset** è espresso in byte occorre usare l'operatore **sizeof** per accedere all'elemento desiderato

Esempio: per leggere il record in posizione **c** del file FI

```
fseek(FI,(c-1)* sizeof(Studente),SEEK_SET);
```

Si noti che per determinare il numero di record contenuti in un file si può usare

```
int DimensioneFile(char *NomeFile);
```

ESEMPIO: LETTURA SEQUENZIALE DI UN FILE DI RECORD

```
- apri il file in lettura
- se il file esiste
  - prendi  come  A il primo  record dal file
    - finchè  non è stata raggiunta la fine del file
      - elabora il record R
      - prendi come  R il prossimo record  dal file
```

Il controllo che non è stata raggiunta la fine del file FP può essere effettuato in due modi

1. si legge R e si controlla che non sia stata raggiunta la fine di FP con `feof(FP)`
2. si legge il record R e si controlla che il numero di elementi letti sia 1

Nel modo 2 si può effettuare la lettura del record direttamente nella verifica della condizione:

```
while ((fread(&R,sizeof(R),1,FI)!=0)
    {
      <elaborazione record R >
    }
```

```
void FileShow(Stringa NomeFile){
 FILE *FI;
 Studente A;
 int I;
 if ((FI=fopen(NomeFile,"r"))==NULL)
  printf("non posso aprire il file %s", NomeFile);
 else
  while (fread(&A,sizeof(Studente),1,FI)!=0){
   printf("\nNome Studente > %s\n",A.Nome);
   printf("Voti Studente: ");
   for(I=0;I<MAXV;I++)
    printf("%d  ",A.Voti[I]);
 } /* end while */
} /* end  FileShow */
```


ESEMPIO: MODIFICA SEQUENZIALE DI UN RECORD NEL FILE

La seguente funzione modifica i record con NomeStudente specificato. Riporta in uscita il numero di record modificati.
La gestione è sequenziale; si utilizza un file di appoggio temporaneo.

```
int FileUPDATE(Stringa NomeFile, Stringa NomeStudente)
{FILE *FI,*FO;
 Studente A;
 int StringaUgualeA(char *S, char *T);
 int EsisteFile(char *NF);
 int count=0;

 if (EsisteFile(NomeFile))
 {FI=fopen(NomeFile,"r");
  FO=fopen("temporaneo","w");
  while (fread(&A,sizeof(Studente),1,FI)!=0)
   {
    if (StringaUgualeA(A.Nome,NomeStudente))
    {
      < modifiche al record A >
     count++;
    }
   fwrite(&A,sizeof(Studente),1,FO);      }
 fclose(FI);
 fclose(FO);
 remove(NomeFile);
 rename("temporaneo",NomeFile);
 }
  return count;
} /* end FileUPDATE */
```


ESEMPIO: CANCELLAZIONE SEQUENZIALE DI UN RECORD NEL FILE

La seguente funzione cancella i record con NomeStudente specificato. Riporta in uscita il numero di record cancellati.
La gestione è sequenziale; si utilizza un file di appoggio temporaneo

```
int FileDELETE(Stringa NomeFile, Stringa NomeStudente)
{
 FILE *FI,*FO;
 Studente A;
 int StringaUgualeA(char *S, char *T);
 int EsisteFile(char *NF);
 int count=0;

 if (EsisteFile(NomeFile))
 {
  FI=fopen(NomeFile,"r");
  FO=fopen("temporaneo","w");
  while (fread(&A,sizeof(Studente),1,FI)!=0)
   {
    if (!StringaUgualeA(A.Nome,NomeStudente))
    fwrite(&A,sizeof(Studente),1,FO);
    else
      count++;
   }
 fclose(FI);
 fclose(FO);
 remove(NomeFile);
 rename("temporaneo",NomeFile);
 }
  return count;
} /* end  FileDELETE */
```


OUTPUT FORMATTATO – PRINTF E FPRINTF

La funzione

```
int printf(char *formato, arg1, arg2, … , argn);
```

converte, **formatta** e **stampa** sullo standard output i suoi argomenti arg_i sotto il controllo di `formato` e riporta come risultato il numero di caratteri stampati.

La stringa di formato contiene due tipi di oggetti: caratteri ordinari, che vengono semplicemente copiati sull'output, e specifiche di conversione, ciascuna delle quali provoca la conversione e la stampa del successivo argomento di printf. Ogni specifica di conversione inizia con % e termina con un carattere di conversione. Tra % ed il carattere di conversione possiamo trovare, nell'ordine

- Il segno -, che indica l'allineamento a sinistra.
- Un numero che indica l'ampiezza minima del campo.
- Un punto, che separa l'ampiezza del campo dalla precisione.
- Un numero, la precisione (per una stringa, il massimo numero di caratteri che devono essere stampati, per un float il numero di cifre dopo il punto, per un intero il numero minimo di cifre di un intero).
- Una h se l'intero deve essere stampato come short, oppure una l se l'intero deve essere stampato come long.

Principali caratteri di conversione:

carattere	Tipo Argomento	Stampato come
d,i	int	Numero decimale
o	int	Numero ottale privo di segno (senza zero iniziale)
X,X	int	Numero esadecimale privo di segno
u	int	Numero decimale privo di segno
c	Int	Carattere singolo
s	char *	Stampa caratteri fino a '\0' o al raggiungimento della precisione
f	double	m.dddddd, dove il numero delle d è dato dalla precisione (il default è 6)
p	void *	Puntatore (dipende dall'implementazione
%		Non converte nessun argomento, ma stampa %

ALTRE FUNZIONI

```
int fprintf(FILE *FP, char *formato, arg1, arg2, … , argn);
```

Scrive sul file puntato da FP, con le stesse modalità di printf

```
int sprintf(char *stringa, char *formato, arg1, arg2, … , argn);
```

Scrive in stringa, che deve essere sufficientemente lunga da poter contenere il risultato

INPUT FORMATTATO – SCANF E FSCANF

La funzione

```
int scanf(char *formato, arg1, arg2, … , argn);
```

legge caratteri dallo standard input, li interpreta in base al contenuto di `formato` e memorizza il risultato negli argomenti `argi` e riporta come risultato il numero di caratteri letti e memorizzati correttamente.

Gli argomenti `argi` devono essere dei **puntatori**, che indicano dove registrare l'input convertito.

La stringa di formato contiene specifiche di conversione, per il controllo della conversione dell'input. Tali specifiche possono contenere:

- spazi o caratteri di tabulazione, che vengono ignorati
- caratteri normali (diversi da %) che ci si aspetta corrispondano al successivo carattere non bianco del flusso di input.
- specifiche di conversione, costituite essenzialmente dal carattere % e da un carattere di conversione

Principali caratteri di conversione:

carattere	Tipo argomento	dati in input
d	* int	intero decimale
i	* int	intero, che può essere ottale (preceduto da 0) o esadecimale (preceduto da 0x o 0X)
o	* int	intero ottale (preceduta o meno dallo 0)
x	* int	intero esadecimale (preceduto o meno dallo 0x o 0X)
c	* char	i prossimi caratteri in input (1 per default); si considerano anche i caratteri di spaziatura
s	* char	stringa di caratteri (non racchiusa tra apici), che punta ad un vettore di caratteri sufficientemente grande da contenere la stringa ed uno '\0' di terminazione, che verrà aggiunto automaticamente
e, f, g	* float	numeroi reale con segno, punto decimale ed esponente opzionali
%		Non viene effettuato alcun assegnamento carattere %

ALTRE FUNZIONI

```
int fscanf(FILE *FP, char *formato, arg1, arg2, … , argn);
```

legge da file puntato da FP, con le stesse modalità di printf

```
int sprintf(char *stringa, char *formato, arg1, arg2, … , argn);
```

legge da stringa

- Nella stringa di formato di scanf possono comparire dei caratteri normali, però essi non vengono stampati, ma devono corrispondere a caratteri acquisiti da input. Questo può essere utile per leggere i dati in una certa forma, come ad esempio, leggere una data come `gg/mm/aaaa` oppure un tempo come `oo:mm`:

```
int ora, minuto;
scanf("%d:%d",&ora,&minuto);
```

- Gli argomenti `argi` di scanf devono essere dei **puntatori**:
  pertanto nel caso delle stringhe l'operatore & non va utilizzato, in quanto una stringa è un vettore e il suo nome rappresenta già un indirizzo (l'indirizzo del primo elemento):

```
int anno;
char mese[10];
int *pi;
char *pm;

scanf("%s %d",mese,&anno);
pi = &anno;
scanf("%s %d",mese,pi);
pm = mese;
scanf("%s %d",pm,pi);
```

- In realtà, nel caso delle stringhe, il tutto funziona anche se il nome del vettore è messo con &: questo è dovuto al fatto che "%s" non indica solo il tipo dell'argomento (una stringa) ma forza anche una conversione:

```
scanf("%s %d",&mese,&anno);
```

Infatti pm = &mese; non va bene, ma possiamo forzare la conversione con il cast

```
pm = (char *)&mese;
```

che va bene ed equivale a pm=mese.

ESEMPIO: CREAZIONE E LETTURA DI UN FILE DI INTERI

- Salvare su un file N numeri interi acquisiti da input
-

```
#include <stdio.h>
#define N 20

main(){
 FILE *FP;
 int I,X;

 if ((FP=fopen("interi.dat","wb"))!=NULL) {
   for(I=1;I<=N;I++){
     scanf("%d",&X);
     fwrite(&X,sizeof(int),1,FP);
   }
  fclose(FP); // chiusura del file
 }/* end if */
else
  printf("Errore nell'apertura in scrittura del file");
} /* end main */
```

- Se i valori interi sono prima memorizzati in un vettore V (ad esempio, per un loro eventuale ordinamento), possiamo scriverli con un'unica scrittura di N elementi

```
fwrite(V,sizeof(int),N,FP);
```

Essendo il **nome del vettore** V un indirizzo, **non** deve essere preceduto da &.


```
int V[N];

for(I=0;I< N;I++) scanf("%d",&V[I]);

if ((FP=fopen("interi.dat","wb"))!=NULL) {
    fwrite(V,sizeof(int),N,FP);
    fclose(FP); // chiusura del file
    ….
```

- **LETTURA** :
  memorizzare i valori letti in un vettore di N elementi.
  Valgono le stesse considerazioni fatte per la scrittura, però in questo caso è utile considerare il numero degli elementi effettivamente letti (restituito da `fread`) perché può essere minore di N se il file finisse prima:

```
k = fread(V,sizeof(int),N,FP);
```

```
if ((FP=fopen("interi.dat","rb"))!=NULL) {
    k = fread(V,sizeof(int),N,FP);
    fclose(FP); // chiusura del file
    for(I=0;I< k;I++) printf("%d",V[I]);
    ….
```