Gestione dei File

- Per mantenere disponibili i dati tra le diverse esecuzioni di un programma, questi si devono archiviare in file su memoria di massa (dati persistenti):
  - dischi
  - nastri
  - cd
- Un file è un'astrazione fornita dal sistema operativo per memorizzare informazioni su memoria di massa, ad **accesso sequenziale**.
- I file possono essere manipolati all'interno di un programma in C attraverso numerose funzioni disponibili nella libreria standard (<stdio.h>)

Apertura e chiusura di File

- Per agire su un file dall'interno di un programma si deve stabilire una ***corrispondenza*** fra:
  - il nome del file, come risulta al sistema operativo
  - un nome di variabile, definita nel programma
- Questa operazione si chiama **Apertura del file**

- Al termine delle operazioni, la **Chiusura del file** elimina la corrispondenza tra nome del file e variabile definita nel programma.

Accesso al File

- Tutte le operazioni sul file verranno effettuate riferendo la variabile definita all'interno del programma: il S.O. provvederà ad effettuare realmente sul file le operazioni richieste.

- Una **testina di lettura/scrittura** (virtuale) indica in ogni istante la posizione corrente:
  - inizialmente si trova in prima posizione
  - dopo ogni operazione di lettura/scrittura, si sposta nella posizione successiva.

- Non è possibile operare oltre la FINE DEL FILE.

FILE in C

- Per gestire i file, nella libreria <stdio.h> è definito il tipo **FILE**

- FILE: struttura con diverse informazioni relative al file (ad esempio la dimensione).

- La struttura FILE non va MAI gestita direttamente, ma solo attraverso le funzioni di libreria

- L'utente definisce ed usa nei programmi solo puntatori a FILE (*FILE * fp;*)

Input e Output su FILE

- std**io:**
  - **i**nput: avviene da un canale (di input) associato a un file *aperto in lettura*
  - **o**utput avviene su un canale (di output) associato a un file *aperto in scrittura*

- 2 tipi di file: **file binario** e **file di testo**
  - basterebbero i file binari, ma per fare tutto con questi sarebbe scomodo
  - file di testo: non indispensabili, ma rispondono ad esigenze pratiche.

Apertura

- Per aprire un file si usa la funzione:

  FILE* fopen(char nomefile[], char modo[])

  nomefile → stringa con il nome del file da aprire

  modo → le modalità per aprire un file sono:

  r Apre un file di testo in lettura

  w **Crea** un file di testo per scriverci (se esiste lo sovrascrive)

  a Apre un file in scrittura e si posiziona alla fine (append)

  r+ Apre un file di testo in lettura/scrittura

  w+ Crea un file di testo in lettura/scrittura

  a+ Apre o crea(se non esiste) un file di testo in lettura/scrittura

Apertura e Chiusura

- La funzione fopen restituisce un puntatore a FILE che deve essere utilizzato per tutte le successive operazioni sul file:
  - se è NULL l'apertura è fallita: se si cerca di aprire in lettura un filel che non esiste, o in scrittura un file senza avere i permessi
  - è necessario controllarne SEMPRE il valore, per sapere se il file è stato aperto correttamente.

- Per chiudere un file si usa la funzione

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

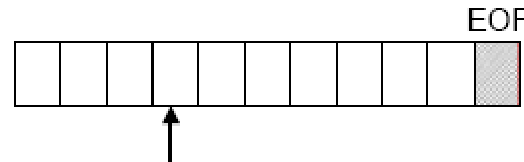
- I 3 canali predefiniti stdin, stdout e stderr sono dei file aperti automaticamente all'inizio dell'esecuzione di ogni programma (di tipo FILE*)

File di Testo

- Conoscete le istruzioni di input e output su tastiera/video.
- Esistono le stesse operazioni per leggere e scrivere su un file sequenze di caratteri.
- Questi file si dicono **FILE DI TESTO**.

File di Testo

- La lunghezza del file è sempre registrata dal sistema operativo: è indicata in modo esplicito dalla presenza del carattere **EOF** (End OF File)



- La fine del file può essere rilevata:
  - in base all'esito delle operazioni di lettura
  - perchè si intercetta il carattere di EOF

File di Testo

- I canali di I/O standard sono file **di testo** aperti automaticamente:
  - **stdin**: file di testo aperto in lettura ed associato alla tastiera
  - **stdout**: file di testo aperto in scrittura ed associato al video
  - **stderr**: file di testo aperto in scrittura ed associato al video

- Le funzioni disponibili per i file di testo sono una generalizzazione di quelle già note per i canali di I/O standard

Stringhe su stdio

- char *gets(char *s)
  - Memorizza nella stringa puntata da s i caratteri letti da stdin (fino a \n o a fine file)
  - Restituisce un puntatore al primo elemento della stringa (NULL in caso di errore)

- int puts(char *s)
  - Scrive su stdout la stringa puntata da s (seguita da \n)
  - Restituisce il numero di caratteri scritti (**zero** in caso di errore)

File di Testo

Funzione da console	*Funzione da file*
`int getchar(void);`	`int` **`fgetc`**`(`**`FILE*`** `f);`
`int putchar(int c);`	`int` **`fputc`**`(int c,` **`FILE*`** `f);`
`char* gets(char* s);`	`char*` **`fgets`**`(char* s, int n,` **`FILE*`** `f);`
`int puts(char* s);`	`int` **`fputs`**`(char* s,` **`FILE*`** `f);`
`int printf( ... );`	`int` **`fprintf`**`(`**`FILE*`** `f, ... );`
`int scanf( ... );`	`int` **`fscanf`**`(`**`FILE*`** `f, ... );`

Tutte le funzioni che gestiscono i file hanno davanti al nome una f

Tutte le funzioni che gestiscono i file hanno un parametro aggiuntivo: il puntatore a FILE

Lettura e scrittura su file

- int **fgetc**(FILE * fp);
  - legge il byte successivo dal file puntato da fp e lo restituisce come intero (il valore ASCII corrispondente al carattere);
  - se si verifica un errore o termina il file la funzione restituisce EOF.

- int **fputc**(int c, FILE * fp);
  - scrive un byte corrispondente al carattere ricevuto in ingresso come primo parametro nel file puntato da fp.
  - il tipo del primo parametro è un intero, ma la funzione può ricevere in ingresso un carattere (si ricordi sempre il legame tra un carattere e il suo codice ASCII)
  - restituisce il carattere scritto nel caso non vi siano problemi, altrimenti restituisce EOF.

Lettura e scrittura su file

- char *__fgets__(char *s, int L, FILE* fp)
  - Memorizza nella stringa puntata da s i caratteri letti dal file puntato da fp (fino a \n o a fine file o fino a L-1 caratteri)
  - Restituisce un puntatore al primo elemento della stringa (NULL in caso di errore)

- int **fputs**(char *s, FILE* fp)
  - Scrive sul file puntato da fp la stringa puntata da s (seguita da \n)
  - Restituisce il numero di caratteri scritti (**zero** in caso di errore)

Esempio

- Dato un file “origine.txt”, creare un file “destinazione.txt” con lo stesso contenuto ma con spaziatura doppia.

- Algoritmo
  - Aprire il file origine.txt e destinazione.txt (ricordarsi i controlli)
  - Finchè si possono leggere righe dal file origine
    - Scrivere la riga letta nel file destinazione
    - Scrivere un \n nel file destinazione

Esempio

```
#include <stdio.h>
#define L 200

main() {
  char linea[L];
  FILE *FIN, *FOUT;

  if ( (FIN=fopen("origine.txt","r")) == NULL )
    printf("Non posso aprire il file di origine");
  else
    if ( (FOUT=fopen("destinazione.txt","w")) == NULL )
       {
      printf("Non posso aprire il file destinazione");
      fclose(FIN);
    }
    else {
       <lettura e scrittura di 1 riga >
       fclose(FIN);
       fclose(FOUT);
 }
}
```

Controllare SEMPRE la corretta apertura dei file!!!

```
while ( fgets(linea, L, FIN) != NULL ) {
    fputs( linea, FOUT);
    fputs( "\n", FOUT );
  }
```

strcmp → confronto

- Il confronto tra stringhe si effettua tramite la funzione strcmp (nella libreria standard, dichiarata in <string.h>)

- #include <string.h>
- int **strcmp**(char *s1, char *s2)
  - Restituisce un intero:
    - < 0 se la stringa 1 e' minore della stringa 2
    - 0 se la stringa 1 e' uguale alla stringa 2
    - > 0 In questo caso la stringa 1 e' maggiore della 2
  - Il confronto tra stringhe è **CASE SENSITIVE**

Esercizio

- Scrivere l'algoritmo ed il codice del programma che esegua le seguenti operazioni:

  – Confrontare il contenuto di 2 file di testo e stampare a video la prima linea di entrambi in cui differiscono.

  – Ripetere l'esercizio stampando su un terzo file tutte le righe di entrambi in cui differiscono.

I/O formattato su file

- Avviene con le stesse modalità già note per I/O standard, con scanf e printf

- **fscanf**: legge da file, formatta e memorizza
- **fprintf**: converte, formatta e scrive su file

- Invece di ridirigere le operazioni di ingresso/uscita verso lo std i/o, operano sul file specificato dal puntatore al file ricevuto come parametro.
- Rendono estremamente semplice scrivere una grande quantità e varietà di dati formattati su un file di testo.

I/O formattato su file

- int **fscanf**(FILE * fp, char * format, …);
  - Legge il file puntato da fp e converte il contenuto nel formato indicato
  - Restituisce il numero di letture effettuate o EOF se il file è terminato

- int **fprintf**(FILE * fp, char * format, …);
  - esattamente come la printf …

Funzioni per i file

- int **feof**(FILE * fp);
  - Restituisce un valore vero (true) se il file puntato da fp raggiunge la fine del file EOF;

- int **fseek**(FILE * fp, long offset, int origine);
  - Sposta la testina di lettura/scrittura del file nella posizione specificata: la prossima operazione di I/O verrà eseguita dalla nuova posizione impostata.
  - La posizione è calcolata aggiungendo **offset** (che può assumere anche valori negativi) a **origine**.
  - **origine** può valere:
    - 0 dall'inizio del file
    - 1 dalla posizione corrente
    - 2 dalla fine del file
  - In caso di successo, fseek() ritorna 0, in caso di fallimento ritorna -1.

Esempio

Stampare a video il contenuto del file di testo prova.txt

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

main() {
  char *s;
  FILE *fp;
  if ( (fp=fopen(“prova.txt","r") )== NULL )
    printf("Non posso aprire il file prova.txt");
  else{
     while ( !feof(fp) ) {
             fscanf(fp, “%s\n”, s);
             printf(“%s\n”,s);
     }
  }
}
```

Controllare SEMPRE la corretta apertura dei file!!!

Esempio di file di testo

Dato un file di testo persone.txt, in cui ogni riga contiene il cognome di una persona, si vuole scrivere un programma che

- legga riga per riga il file
- memorizzi i dati in un vettore
- elabori i dati letti (per es. ordinarli)

persone.txt

```
Rossi
Bolognesi
Bianchi
Verdi
…
```

Esempio di file di testo

- Algoritmo:
  - Definire un array
  - Aprire il file in lettura
  - Finché ci sono righe nel file:
    - leggere una riga per volta
    - memorizzare il dato nell’array
  - Chiudere il file

Esempio di file di testo

```
#include <stdio.h>
#define DIM 100
typedef char * Stringa;

int main () {
    int k=0;   Stringa v[DIM];   FILE * fp;

    fp=fopen(“persone.txt”, “r”);
    if (fp==NULL) {
            printf(“Errore durante l’apertura del file”);
            return -1;  /* fine del programma */
    }

    // finché ci sono dati, lettura dal file e inserimento nell’array
    while (fscanf(fp,”%s\n”,v[k]) != EOF)  k++;
    fclose(fp);
}
```

Esempio di file di testo

- Se il file di testo contiene 2 stringhe (es cognome e nome), si può fare una sola lettura:

```
fscanf(fp,"%s%s\n",cognome[k], nome[k])
```

persone.txt

```
Rossi       Mario
Bolognesi Giovanni
Verdi    Franco
```

- fscanf <u>considera finita una stringa al primo spazio che trova</u> (attenzione: non si può usare per leggere stringhe contenenti spazi)
- fscanf <u>elimina automaticamente gli spazi</u> che separano una stringa dall'altra (NON si devono inserire gli spazi nella stringa di formato)

Esempio di file di testo - variante

- Sia dato un file di testo persone1.txt, in cui ogni riga contiene:
  - cognome (esattamente 10 caratteri)
  - nome (esattamente 15 caratteri)

  Attenzione:
  - Entrambi i campi possono contenere spazi
  - non sono previsti spazi espliciti di separazione

persone1.txt

```
Rossi     Mario
Della PaceGiovanni
Verdi     Gian Giacomo
….        …..
```

Esempio di file di testo - variante

- Non si può utilizzare fscanf(fp,”**%s%s**...
  - Si fermerebbe al primo spazio
  - Potrebbe interpretare come parola unica, due senza spazi in messo

- Possiamo specificare quanti caratteri leggere:

  fscanf(fp,”**%10c%15c\n”**, …)

  Si leggono 10 caratteri per il cognome e 15 per il nome

File binari

- Un file binario è una sequenza di byte: come tale può essere usato per memorizzare su memoria di massa *qualunque tipo di informazione*

- Input e output avvengono sotto forma di sequenza di byte

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

Lettura e scrittura su file binari

- int fread(buffer, dim, elementi, fp)
  - buffer: vettore dove verranno trasferiti i byte letti
  - dim: dimensione in byte di ogni elemento del vettore (si può usare la funzione sizeof() per conoscere la dimensione di ogni tipo di dato)
  - elementi:indica il numero di elementi del vettore
  - fp: puntatore al FILE
- int fwrite(buffer, dim, elementi, fp)
  - buffer: vettore che contiene i byte da trasferire
  - dim: dimensione in byte di ogni elemento del vettore (si può usare la funzione sizeof() per conoscere la dimensione di ogni tipo di dato)
  - elementi:indica il numero di elementi del vettore
  - fp: puntatore al FILE

Lettura e scrittura su file binari

- Il valore intero di ritorno di fread e fwrite indica il **numero di byte letti/scritti** dal/nel file
- Se il valore di ritorno è **negativo** significa che si è verificato **qualche errore** (per es. il file non è stato aperto correttamente)
- L'esecuzione della funzione fread/fwrite sposta l'indicatore di posizione avanti di un numero di byte pari a quelli letti/scritti
- Quando tutto il file è stato letto e si raggiunge l'**EOF la funzione fread restituisce 0** (e continuerà a restituire 0 ad ogni successiva lettura)

Lettura e scrittura su file binari

```
…
FILE *f1;
FILE *f2;
int a[20];
int b[30];
int r;
…
r = fread(a, sizeof(int), 20, f1);
…
r = fwrite(b, sizeof(int), 10, f2);
```

Esempio di file di binario

Scrivere un programma che crei un file binario persone.dat: ogni riga contiene il cognome di una persona.
Poi un programma che legga riga per riga il file (come nell'esempio precedente)

- Il cognome di ogni persona da scrivere nel file viene richiesto all'utente da stdin

Esempio di file di binario

Scrittura su file binario

```
#include <stdio.h>
main() {
FILE *fp; char c[30]; int fine=0;
if ( (fp= fopen(“persone.dat”,”wb”))==NULL) return -1;
while (!fine) {
    printf(“cognome: “);   scanf(“%s”,c);
    fwrite(c, sizeof(char), 30, fp);
    printf(“Fine? (SI=1,  NO=0) “);
    scanf(“%d”,&fine);
}
fclose(fd);
}
```

Esempio di file di binario

Lettura da file binario

```
#include <stdio.h>
main() {

FILE *fp; char *v[100]; int i=0;
if ( (fp= fopen("persone.dat","rb"))==NULL) return -1;
while ( fread(&v[i], (30*sizeof(char)), 1, fp) >0) {
    i++;
}
fclose(fd);
}
```

Esempio di file di binario

- Se sapessimo con certezza il numero di cognomi scritti nel file (es DIM), potrei fare una sola lettura: invece di leggere 1 elemento ne potrei leggere DIM

- Per DIM volte leggo 1 elemento

```
while ( fread(&v[i], (30*sizeof(char)), 1, fp) >0) {
    i++;
}
```

- Leggo DIM elementi in 1 volta sola

```
fread(v, (30*sizeof(char)), DIM, fp)
```