Gestione dei file

 **File** → Sono strutture dati persistenti (dischi, etc...)

- Si leggono e si scrivono con opportune istruzioni.

Grazie ai file, i dati possono sopravvivere al termine dell'esecuzione del programma. Si usano anche per memorizzare i programmi.

- Quando se ne chiede l'esecuzione, il sistema operativo copia il programma (eseguibile, conservato in un file) in memoria centrale, e inizia a eseguirlo.

Sono strutture dati **sequenziali** → si leggono e si scrivono gli elementi del file in sequenza.

- Un file **binario** è una sequenza di byte che non è "*interpretata*" in alcun modo;
- Un file di **testo** è una sequenza di caratteri "*interpretata*":
  - Alcuni caratteri rappresentano separatori: Come il carattere "newline" che è interpretato come "*salto alla riga successiva e posizionamento della testina a inizio rigo*"

File e Sistema operativo

I file sono gestiti dal sistema operativo. In C per gestire i file dobbiamo creare una *variabile di collegamento* che ci permette di accedere a un dato sul disco.

In C le variabili devono essere di tipo **FILE ***

- Esse consentono di inizializzare un *flusso di comunicazione* tra il nostro programma e il file

Esempio: FILE *fp;

Per poter usare un file all'interno di un programma in C:

- Dichiarare una variabile di tipo **FILE *nome;**
- Facendo uso di una oppurtuna funzione di libreria, dobbiamo cioè **aprire il file;**
- Facendo uso di altre funzioni di libreria possiamo aggiornate il documento o leggerne il contenuto;

- Infine prima della fine del programma, dobbiamo dichiarare concluso l'uso del documento sull'hard disk - dobbiamo cioè **chiudere il documento** (chiudere **il file**).

 Le funzioni di libreria sono incluse nella libreria #include <stdio.h>

In C, tutte le periferiche sono viste come file

- **stdin** (tastiera);
- **stdout** (monitor);

In C possiamo ***leggere*** e ***scrivere*** su ogni periferica di I/O con le stesse modalità utilizzate per i **file.**

 **FILE * nome** → è un puntatore a un tipo di dato struct di nome FILE, che contiene tutti gli attributi necessari alla gestione dei file.

Rappresentazione: il tipo di dato FILE

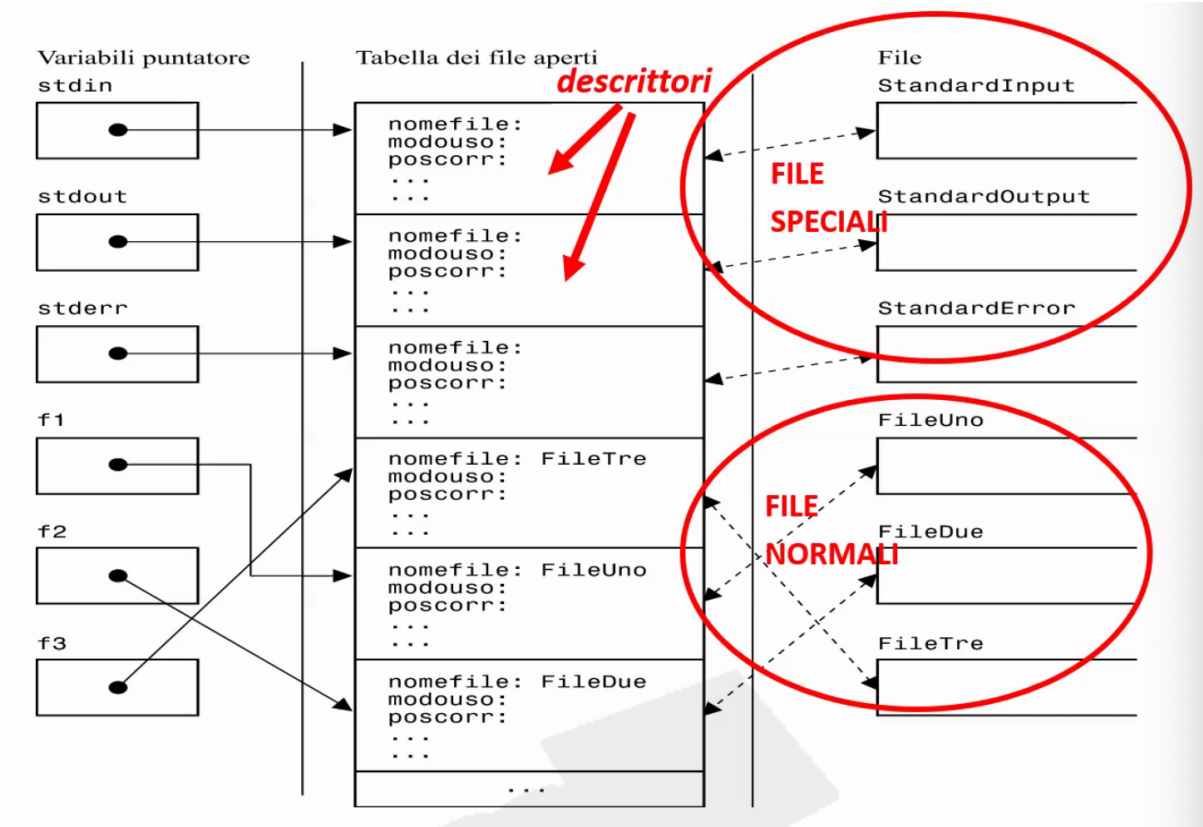
Il Sistema Operativo gestisce per ogni file un **descrittore** cioè una variabile di tipo **struct FILE**

- Risiede nella tabella dei file aperti, una delle strutture dati che il S.O associa ai programma in esecuzione.

Il descrittore memorizza:

- Il **nome** del file;
- La **modalità** d'uso (*read, write*);
- Un indicatore della **posizione** corrente nel file (testina);
- Dimensione del file (per stabilire se la testina è alla fine del documento (**eof**) end-of-file);
- L'indicatore di un eventuale **errore.**

L'apertura del file all'interno di un programma C restituisce un **descrittore**, per la precisione un **puntatore** a un descrittore: **FILE ***.



Le periferiche sono dette anche **file speciali.**

Dichiarare e aprire un file

```
FILE *fp
fp = fopen("nomefile", "modalità");
```


Se si verifica un errore, **fopen**() restituisce **NULL**. → ERRORI NELL'APRIRE IL FILE

La variabile **nomefile** può indicare:

- Il nome di un documento con anche l'indicazione del percorso completo sull'hard disk dove trovarlo (path)
- Se il percorso completo non è indicato, il nome di un documento che si trova nella stessa cartella da cui si è eseguito il programma.

La variabile **modalità** può contenere:

- "**r**" → lettura modalità testo, posizionamento inizio file (**read**);
- "**w**" → scrittura modalità testo, posizionamento inizio file (**write**)**,** si cancella ciò che c'è nel file**;** Se il file non esiste, esso viene creato.

- "**a**" → scrittura in modalità testo, posizionamento fine file (*append*)**, NON** si cancella ciò che c'è nel file**;**
- "**rb**", "**wb**" e "**ab**" → (corrispondono ad "**r**", "**w**" e "**a**", ma con i file binari).

Se voglio aprire un file sia in lettura che in scrittura, mi basterà inserire "**rw**" (lettura e scrittura), "**ra**" (lettura e scrittura in append → scrivo alla fine del file), oppure "**r+**" (lettura e scrittura).

Creare o Aprire un File

```
fp = fopen("nomefile", "modalità");
```

Apre il file (oppure ne **crea** uno nuovo, se non lo trova).

Cancellare

Cancella file nomefile;

```
int remove (char *nomefile)
```

Restituisce **0** se l'operazione va a buon fine, **≠ 0** se non va a buon fine.

Ridenominare

Cambia nome al file;

```
int rename (char *oldname, char *newname)
```

Restituisce **0** se l'operazione va a buon fine, **≠ 0** se non va a buon fine.

Chiudere

fp diventa NULL, descrittore di tipo FILE rilasciato

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

Restituisce **0** se l'operazione va a buon fine, se non va a buon fine EOF.

Gestione degli errori - File già aperto

```
int ferror (FILE *fp)
```

Restituisce **0** se non si verificano errori, se non va a buon fine restituisce un codice specifico dell'errore.

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

Restituisce **0 (falso)** se NON si è alla fine**.**

Lettura e Scrittura

Si opera sui file in quattro modi possibili

Tre modi per i **file di testo**:

- Precisando la **formattazione** dell' I/O ("à-la-scanf");
- Un **carattere** alla volta ("à-la-getc");
- Per **linee** di testo ("à-la-gets" - fino ad ogni prossimo '\n').

Un modo per i **file binari:**

- Per **blocchi** di byte
  - "à-la-sizeof" - indico solo il numero di byte da leggere.

Comandi Lettura/scrittura

scanf e *printf* fanno riferimento a **stdin** e **stdout**: non serve specificare su quale file agiscono (è implicito)

fprintf e **f**scanf fanno riferimento a file generici, ma si usano esattamente come scanf e printf.

```
int fprintf (FILE *fp, str_di_formato, espressioni);
int fscanf (FILE *fp, str_di_formato, indirizzi_di_variabili)
```

Restituiscono il numero di elementi effettivamente letti/scritti, oppure 0 in caso di errore.

- fscanf(...) restituisce -1 (costante EOF) se l'indicatore di posizione è già oltre la fine del file e non può leggere alcun valore valido.

Esempio

```
#define LUN_STRINGA (50+1)
#define NUM_STUDENTI (300)
typedef struct
```

```
{
  char nome[LUN_STRINGA]; // nome
  char cognome[LUN_STRINGA];// cognome
  unsigned int matricola; // numero di matricola (e' un intero senza segno)
  unsigned int domande[8]; // array per 8 risposte
} t_risultati;

typedef t_risultati t_parteA[NUM_STUDENTI];// tipo array con NUM_STUDENTI celle

int main()
{
  t_parteA parteA; // tabella risultati
  FILE* fp; // puntatore a file
  fp = fopen("ris.txt","r");
  if (fp == NULL) // controllo errore di apertura
  {
    printf("\n Errore durante l'apertura del file: ris.txt \n");
    return -1;
  }
  int count = 0; // indice per scorrere l'array parteA (e per contare i righi del file)
  while( feof(fp) == 0 && count < NUM_STUDENTI ) // lettura del file rigo per rigo
  {
    fscanf(fp, "%s", parteA[count].nome);
    fscanf(fp, "%s", parteA[count].cognome);
    fscanf(fp, "%u", &parteA[count].matricola); //unsigned int
    for(int i = 0; i < 8; i += 1)
      fscanf(fp, "%u", &parteA[count].domande[i]);
    count++; // incremento contatore
  }
  fclose(fp); // chiusura file
// qui possono essere elaborati i dati letti dal file
// il contatore count contiene il numero di Parti A consegnate
  return 0;
}
```

%u = unsigned int

Altri modi per scrivere/leggere su file

Lettura carattere per carattere

```
int getc (void)
```

Legge un carattere da **standard input** (tastiera), restituendolo come intero.

Lettura carattere per carattere su file

```
int fgetc(FILE *fp)
```

Legge un carattere dal file descritto da *fp restituendolo come intero.

Scrittura carattere per carattere

```
int putc (int c)
```

Scrive un carattere su **standard output** (stampa a video).

Scrittura carattere per carattere su file

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

Scrive un carattere sul file descritto da *fp restituendolo come intero. Viene scritto nel file il carattere della variabile c, preso dalla posizione meno significativa.

Esempio: Leggere e mostrare a video un file

```
#include <stdio.h>
int main()
{
  FILE *fp;
  int c;
  fp = fopen ("filechar.txt", "r"); //file di testo - lettura
  if (fp != NULL)
  {
    c = fgetc(fp);
    while (c != EOF) //oppure while (! feof (fp))
    {
      fputc(c, stdout); //scanf ("%c", c);
      c = fgetc (fp); //oppure c = (char) fgetc (fp);
    }
    fclose (fp);
  }else
    printf("Il file non può essere aperto\n");
  return 0;
}
```


Lettura per linee di testo

Su **stdin** e **stdout**:

```
char *gets(char *s)
```

- s è l'array in cui copiare la stringa letta da **stdin** (tastiera). Essa risulterà terminata da un '\0' sostituendo il '\n' finale.

- Non si può limitare la dimensione dei dati in input. La funzione non controlla che la stringa s sia sufficientemente grande.
- Restituisce s (puntatore al primo elemento dell'array di char), se eseguita con successo, mentre in caso di errore o se il file è finito, restituisce NULL.

Lettura per linee di testo da file

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

- legge al più n-1 caratteri, fino a '\n' o EOF
- se incontra '\n' lo inserisce tra gli n-1, e mette alla fine anche il terminatore '\0'
- Restituisce **s** o, in caso di errore, **NULL**

Scrittura per linee di testo

```
int puts(char *s)
```

- Scrive la stringa s, escluso il '\0'
- Al posto del '\0' che si trova nella stringa, scrive un '\n'
- Restituisce **n ≥ 0** se OK, *EOF* in caso di errore

Scrittura per linee di testo su file


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

- Come puts(...), ma **non** aggiunge il '\n', si limita a non scrivere il '\0'
- Restituisce **0** se OK, *EOF* in caso di errore

Esempio

```
#define OK 1
#define ERROR 0
#define MAXLINE 100

int copiaselettiva (char *refstr, char *filein, char *fileout);

int main()
{
  int x = copiaselettiva("ciao", "primo.txt", "secondo.txt");
```

```
}

int copiaselettiva (char *refstr, char *filein, char *fileout)
{
  char line [MAXLINE + 1];
  FILE *fin = fopen(filein, "r");
  if (fin == NULL)
    return ERROR;
  FILE *fout = fopen (fileout, "w");
  if (fout == NULL)
  {
    fclose(fin);
    return ERROR;
  }
  while (fgets(line, MAXLINE, fin) != NULL) //fgets legge da filein
  {
    if (strstr(line, refstr) != NULL) //al più MAXLINE -1 char
      fputs(line, fout);
  }
  fclose(fin);
  fclose(fout);

  return OK; //return 1
}
```

```
char *strstr(char *s, char *p)
```

Restituisce un puntatore di carattere di s da cui inizia la prima occorrenza della stringa p (se presente), NULL altrimenti.

Lettura/scrittura per blocchi di byte

Ci sono funzioni che consentono di scrivere o leggere un intero blocco di dati testuali o binari:

- Utile per esempio quando si vuole scrivere su file un'intera struct, in un "colpo solo".

```
fread(...);
fwrite(...);
```

Accesso diretto

Si può accedere ad uno specifico byte come se il file fosse un array di blocchi di byte:

```
int fseek (FILE *fp, long offset, int refpoint)
```

- Imposta la posizione corrente a un valore pari a uno **spostamento** (positivo o negativo) pari a **offset**, calcolato **rispetto** a uno dei seguenti punti di partenza:
  - L'inizio del file, se refpoint vale SEEK_SET (*costante di stdio.h*);
  - L'attuale posizione corrente, se refpoint vale SEEK_CUR;
  - La fine del file, se refpoint vale SEEK_END.
- restituisce **0** se l'operazione di spostamento va a buon fine, un valore **≠ 0** altrimenti.

```
long int ftell (FILE *fp)
```

- restituisce la posizione corrente della "testina":
  - per file binari è il numero di byte dall'inizio
  - per file testuali il numero dipende da come sono codificati i char speciali.
    - Per esempio: < INVIO > su WinOS sono 2 byte: $13_{dec}10_{dec}$; su LinuxOS, MacOS è 1 byte: 10_{dec}

```
void rewind (FILE *fp) //equivale a fseek(f,0, SEEK_SET);
```

Riavvolge il file (la posizone corrente torna all'inizio). Mette la testina all'inizio del file.