Linguaggio C: File

I File

Il file e` **l'unita` logica di memorizzazione dei dati su memoria di massa**, che consente una memorizzazione **persistente** dei dati, non limitata dalle dimensioni della memoria centrale.

I programmi C possono accedere a file (leggere e scrivere) mediante le funzioni standard definite nella libreria di I/O (<stdio.h>)

Caratteristiche dell'accesso a file in C:

- Ogni programma vede il file come una **sequenza di componenti** (record logici), terminata da una "marca" di fine file (End Of File, EOF)
- I file sono gestiti dal Sistema Operativo. La realizzazione delle funzioni standard di I/O (<stdio.h>) tiene conto delle funzionalita` del S.O ospite.

I file

- 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 e` terminata da una marca di fine linea (newline, carattere '\n').

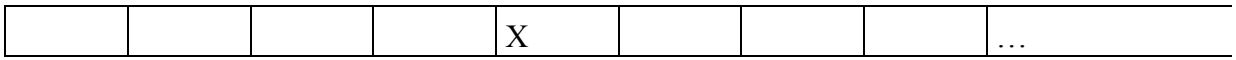
```
Gentile sig. Rossi,↲
   con la presente le comunico che↲
   è scaduto il termine per il pagamento ↲
   dell'Assicurazione della sua automobile. ↲
```

⇒Il record logico puo` essere il singolo carattere, la parola, oppure la linea.

Gestione di file in C

I file hanno una **struttura sequenziale**, cioè:

- i record logici sono organizzati in una sequenza rigidamente ordinata
- per accedere ad un particolare record logico, e` necessario "scorrere" tutti quelli che lo precedono.



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

Puntatore a file

E` una variabile che viene utilizzata per riferire un file nelle operazioni di accesso (lettura e scrittura.). Implicitamente essa indica:

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

- Ad esempio:
- FILE *fp;

- ⇒il tipo FILE e` un tipo non primitivo dichiarato nel file stdio.h.

Gestione di file in C

Apertura di un file:

- Prima di accedere ad un file e` necessario ***aprirlo***: l'operazione di apertura compie le azioni preliminari necessarie affinche` 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, e` possibile leggere/scrivere il file, riferendolo mediante il puntatore a file.

Chiusura di un file:

- Alla fine di una sessione di accesso (lettura o scrittura) ad un file e` necessario chiudere il file per memorizzare permanentemente il suo contenuto in memoria di massa:

Apertura di un File

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

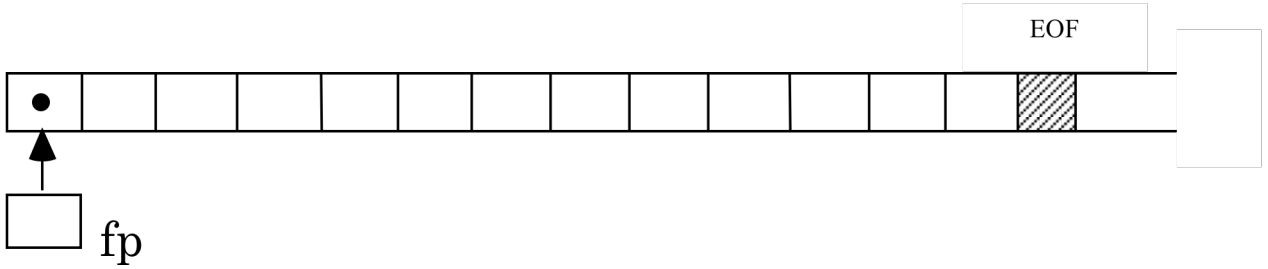
- dove:
  - **name** e` un vettore di caratteri che rappresenta il nome (assoluto o relativo) del file nel file system
  - **mode** esprime la modalita` di accesso scelta.
    - "r", in lettura (read)
    - "w", in scrittura (write)
    - "a", scrittura, aggiunta in fondo (append)
    - "b", a fianco ad una delle precedenti,indica che il file e` binario (**se non specificato, il file e` di testo**)

- Se eseguita con successo, l'operazione di apertura ritorna come risultato un puntatore al file aperto:
- Se l'apertura fallisce, fopen restituisce il valore NULL. Tipici motivi di fallimento:
  - Il file non esiste
  - Il file viene aperto in una modalita` incompatibile con le sue proprieta` (ad esempio: apro in scrittura un file a sola lettura, read only), etc.

Apertura in lettura

```
FILE *fp;
fp = fopen("filename", "r")
```

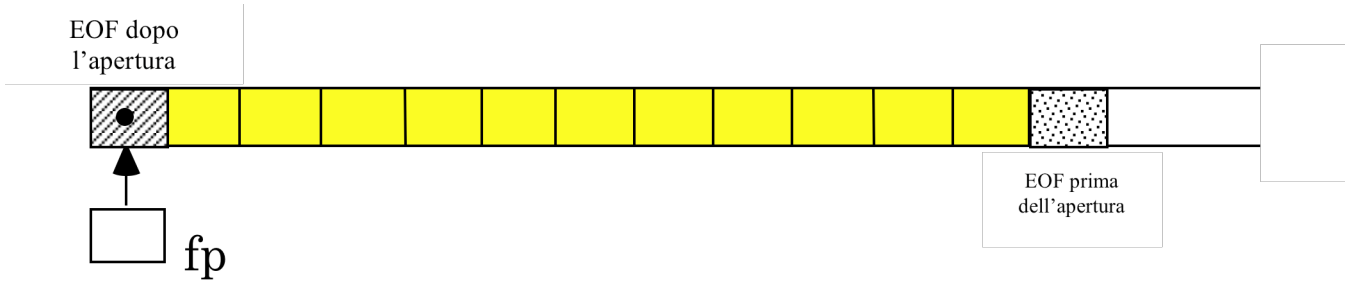
Se il file non e` vuoto:



Apertura in scrittura

```
FILE *fp;
fp = fopen("filename", "w")
```

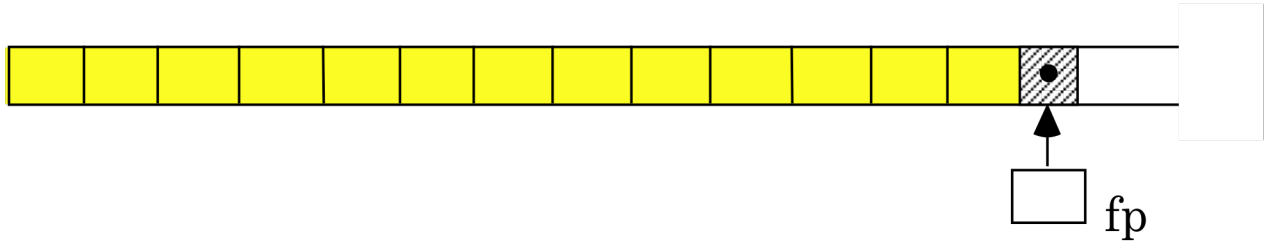
Anche se il file non e` vuoto:



Se il file esisteva gia`, il suo contenuto viene perso: si scriveranno i nuovi record logici, sopra i pre-esistenti, a partire dal primo.

Apertura in aggiunta (append)

```
FILE *fp;
fp = fopen("filename", "a")
```



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

Apertura di un file

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 e` eseguita correttamente restituisce il valore 0;
- se la chiusura non e` andata a buon fine, ritorna la costante EOF.

Esempio

```
#include <stdio.h>
main()
{
  FILE *fp;
  fp = fopen("prova.txt", "w")
  <scrittura di prova.txt>
  fclose(fp);
}
```

File standard di I/O

Esistono tre file testo che sono aperti automaticamente all'inizio di ogni esecuzione:

- **`stdin`**, standard input (es. tastiera), aperto in lettura
- **`stdout`**, standard output (es. video), aperto in scrittura
- **`stderr`**, standard error (es. video), aperto in scrittura

➔ **`stdin`**, **`stdout`**, **`stderr`** sono variabili di tipo puntatore a file automaticamente (ed implicitamente) definite ➨ non vanno definite.

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.

File di testo: sono disponibili funzioni di:

- Lettura/scrittura con formato: fscanf, fprintf
- Lettura/scrittura di caratteri: fgetc, fputc
- Lettura/scrittura di stringhe di caratteri: fgets, fputs.

File binari: si utilizzano funzioni di Lettura/scrittura di blocchi:

- fread, fwrite.

Accesso a file di testo: lettura/scrittura con formato

Funzioni simili a scanf e printf, ma con un parametro aggiuntivo per 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` e` il puntatore al file
- `stringa-controllo` indica il formato dei dati da leggere
- `ind-elem` e` la lista degli indirizzi delle variabili a cui assegnare i valori letti.

Restituisce il numero di elementi letti, oppure un valore negativo in caso di errore.

Esempio:

```
FILE *fp;
int A; char B; float C;
fp=fopen("dati.txt", "r");
fscanf(fp, "%d%c%f", &A, &B, &C);
...
fclose(fp);
```

Accesso a file di testo: lettura/scrittura con formato

Scrittura con formato: Si usa la funzione fprintf:

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

dove:

- `fp` e` il puntatore al file
- `stringa-controllo` indica il formato dei dati da scrivere
- `elem` e` la lista dei valori (espressioni) da scrivere

Restituisce il numero di elementi scritti, oppure un valore negativo in caso di errore.

Esempio:

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

printf/scanf vs 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: scrittura di un file di testo

```
#include <stdio.h>
main()
{  char item[80];
   FILE *fp;
   int fine=0;
   /* scrittura in un file della lista della spesa:*/
   fp=fopen("testo.txt", "w");
   printf(" Lista della spesa: ");
   while (!fine)
   {    printf(" \nprossimo articolo ? ");
        scanf("%s", item);
        fprintf(fp, "%s\n", item);
        printf("\nFinito (si=1, no=0)? ");
        scanf("%d ", &fine);
   }
   fclose(fp);
}
```

Esempio: lettura e stampa di un file di testo

```
#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 che rappresenta 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 copia un file in un altro file (stdout)

```
#include <stdio.h>
void filecopy(FILE *, FILE *);

main()
{  FILE *fp;
   char nome[20];
   printf("Nome del file? ");
   scanf("%s", nome);
   if ((fp=fopen(nome, "r"))==NULL)
   {   printf("\nImpossibile aprire il file %s\n",nome);
       exit(1);
   }
   else
   {   filecopy(fp, stdout);
       fclose(fp);
   }
}
```

```
void filecopy(FILE *inFile,  FILE *outFile)
{ int c;
  while((fscanf(inputFile, "%c", &c))>0)
          fprintf(outputFile, "%c", c);
  return;
}
```

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.

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.

Accesso a file binari: Lettura/scrittura di blocchi

- Si puo` leggere o scrivere da un file binario un intero blocco di dati.
- 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 e` necessario che il file sia stato aperto in modo binario (modo "b").

Lettura di file binari

```
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.

Esempio: file binario contenente interi.

```
FILE *fp;
int dati[100], k;
fp=fopen("dati.bin", "rb");
k=fread(dati, sizeof(int),100, fp);
printf("numero di valori effettivamente letti: %d\n",
   k);
…
```

Scrittura di file binari

```
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: file binario contenente i primi 100 interi.

```
FILE *fp;
int i;
fp=fopen("dati.bin", "wb");
for(i=0; i<100; i++)
   fwrite(i, sizeof(int),1, fp);
fclose(fp);
…
```

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");

  while( fread(&p, sizeof(persona),1, fp)>0)
     printf("%s%s%d",p.nome,p.cognome,
                     p.reddito);

  fclose(fp);
}
```

Esempio: file di record

- File di record: scrive il contenuto di un vettore di record (inizializzato con dati forniti da stdin) in un file binario dato da input.

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

Persona P[DIM];
```

```
main()
{ int crea_vettore(Persona V[], int dim);
  int i, n;
  FILE *file;
  char nome[30];

  gets(nome);
  n=crea_vettore(P,DIM);
  if ((file=fopen(nome, "wb"))==NULL)
  {           printf("Impossibile aprire file%s\n", nome);
              return 1;
  }
  fwrite(P,sizeof(Persona),n,file);
  fclose(file);
}
```

```
int crea_vettore(Persona P[], int dim)
{ int i=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++;
     printf("%s\n%s\n%s\n%d\n",P[i].nome,
     P[i].cognome,P[i].via,P[i].eta);
   }
  return i;
}
```