Stdio.h

Input / Output

- Come già detto, input e output sono realizzati in C da funzioni di **`stdio.h`** all'interno della libreria standard
  - Sia i file che i dispositivi (tastiera, schermo ...) sono visti come una successione (*stream*) di caratteri
  - Quando in programma C va in esecuzione le connessioni a tre stream sono preconfigurate
    - Lo standard input (di solito la tastiera)
    - Lo standard output (di solito lo schermo)
    - Lo standard error (di solito lo schermo)
  - Abbiamo già visto come funzionano **`printf()`** e **`scanf()`** che lavorano su standard input ed output, vediamo adesso una panoramica sulle altre funzioni utili della libreria
  - Lo standard error viene usato per i messaggi di errore in modo da non mischiarli con l'output del programma

Input/Output: `stdio.h`

- Contiene definizioni di costanti legate all' I/O

  es. EOF (end of file)

  `#define EOF (-1)`

  valore restituito alla fine di uno stream

- Contiene la definizione della tipo che descrive un file generico
  - **`FILE`** è una *struttura* il formato dipende dal sistema
  - contiene: posizione corrente, indicatori di errore l/s, indicatori di fine file raggiunta etc

Input/Output: `stdio.h`

Come avviene la lettura di un file:

- prima il file viene ‘aperto’, cioè si cerca nel file system e si crea una struttura **`FILE f`** con le informazioni relative al file
  - generalmente c’è un limite al numero di file aperti
- poi si accede al file usando la funzioni di libreria passando **`&f`** come parametro
- infine il file viene ‘chiuso’ (**`f`** viene deallocata) il contenuto del file non è più accessibile da programma

Esempio: somma di interi di un file

Problema:

- Leggere il contenuto del file **`inputfile`**
- Convertire ogni riga in un **`int`**
- Sommare tutti i numeri letti
- Scrivere la somma totale in un nuovo file **`outputfile`**
- Se **`outputfile`** esiste vogliamo semplicemente sovrascriverlo

```
#include <stdio.h>
#include <stdlib.h>
int main (void){
   int a, sum = 0;
   FILE * ifp, *ofp;
```

Dichiaro i Puntatori a FILE per il file di ingresso e il file di uscita

```
#include <stdio.h>
#include <stdlib.h>
int main (void){
   int a, sum = 0;
   FILE * ifp, *ofp;
   ifp = fopen ("./inputfile","r");
```

Apro il file inputfile in lettura fopen restituisce NULL se c'e' stato un errore oppure restituisce il puntatore ad una Struttura di tipo FILE con le informazioni di accesso

```
#include <stdio.h>
#include <stdlib.h>
int main (void){
   int a, sum = 0;
   FILE * ifp, *ofp;
   ifp = fopen ("./inputfile","r");
   if ( ifp == NULL ) {
        perror("fopen: inputfile");
        return EXIT_FAILURE;
}
```

Se c'e' stato un errore chiamo la funzione **perror()**
Che stampa su standard error informazioni sull'errore

Utilizza il codice numerico che la funzione ha lasciato in una variabile condivisa predefinita **errno**

```
#include <stdio.h>
#include <stdlib.h>
int main (void){
   int a, sum = 0;
   FILE * ifp, *ofp;
   ifp = fopen ("./inputfile","r");
   if ( ifp == NULL ) {
        perror("fopen: inputfile");
        return EXIT_FAILURE;
}
```

Ad esempio se il file non esiste stampa

"fopen: inputfile: no such file or directory"

```
#include <stdio.h>
#include <stdlib.h>
int main (void){
   int a, sum = 0;
   FILE * ifp, *ofp;
   ifp = fopen ("./inputfile","r");
   if ( ifp == NULL ) {
        perror("fopen: inputfile");
        return EXIT_FAILURE;
}
```

EXIT_FAILURE
Valore predefinito (diverso da 0)
Indica terminazione con errore
(in stdlib.h)

```
#include <stdio.h>
#include <stdlib.h>
int main (void){
   int a, sum = 0;
   FILE * ifp, *ofp;
   ifp = fopen ("./inputfile","r");
   if ( ifp == NULL ) { ....}
   ofp = fopen ("./outputfile","w");
   if ( ofp == NULL ) { ....}
```

Facciamo lo stesso per il file destinazione,
specificando "w" come diritti chiediamo di accedere
in scrittura sovrascrivendo il file se esiste, altrimenti viene creato

```
#include <stdio.h>
#include <stdlib.h>
int main (void){
  int a, sum = 0;
  FILE * ifp, *ofp;
  ifp = fopen ("./inputfile","r");
   if ( ifp == NULL ) { ....}
  ofp = fopen ("./outputfile","w");
   if ( ofp == NULL ) { ....}
/* finchè il numero di conversioni operate con successo è
  uguale a 1 */
  while ( fscanf(ifp, "%d", &a) == 1 )
          sum+=a;
  fprintf(ofp, "la somma è %d\n", sum);
  fclose(ifp); fclose(ofp);
  return 0;
}
```

Leggo: funziona come scanf ma legge dal FILE Puntato da **ifp**

```
#include <stdio.h>
#include <stdlib.h>
int main (void){
  int a, sum = 0;
  FILE * ifp, *ofp;
  ifp = fopen ("./inputfile","r");
   if ( ifp == NULL ) { ....}
  ofp = fopen ("./outputfile","w");
   if ( ofp == NULL ) { ....}
/* finchè il numero di conversioni operate con successo è
  uguale a 1 */
  while (fscanf(ifp, "%d", &a)==1)
          sum+=a;
  fprintf(ofp, "la somma è %d\n", sum);
  fclose(ifp); fclose(ofp);
  return 0;
}
```

Scrivo: funziona come Printf ma scrive sul FILE Puntato da **ofp**

```
#include <stdio.h>
#include <stdlib.h>
int main (void){
  int a, sum = 0;
  FILE * ifp, *ofp;
  ifp = fopen ("./inputfile","r");
  if ( ifp == NULL ) { ....}
  ofp = fopen ("./outputfile","w");
  if ( ofp == NULL ) { ....}
/* finchè il numero di conversioni operate con successo è
  uguale a 1 */
  while (fscanf(ifp, "%d", &a)==1)
          sum+=a;
  fprintf(ofp, "la somma è %d\n", sum);
  fclose(ifp); fclose(ofp);
  return 0;
}
```

Chiudo i file: questo dealloca anche le strutture

Eseguiamo....

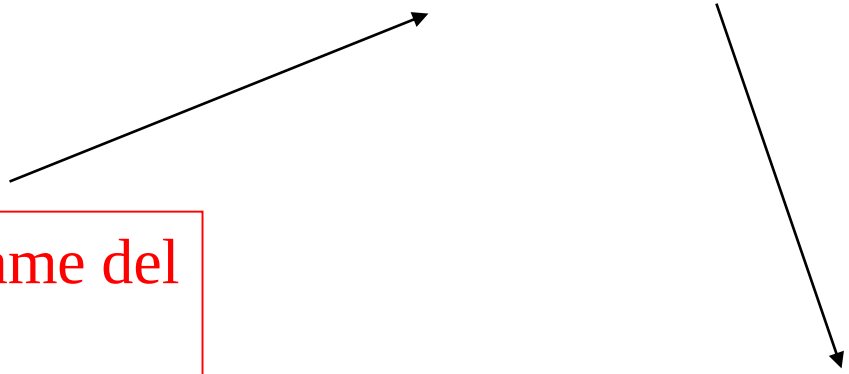
```
$ ls -l ./inputfile
-rw-r—r- 1 susanna usr 214 1 Apr 2015 18:17 ./inputfile
$ ls -l ./outputfile
ls: Impossibile accedere a ./outputfile: No such file or
  directory
$ cat inputfile
34
56
1
2
3
4
$
```

Eseguiamo....

```
$ ./leggi
$ ls -l ./outputfile
-rw-r—r- 1 susanna usr 4 1 Apr 2015 18:23 ./outputfile
$ cat ./ouputfile
la somma è 100
$ rm ./inputfile
$ ./leggi
fopen: inputfile: No such file or directory
$
```

Input/Output: `stdio.h`

```
ifp = fopen("./inputfile","r");
```



Pathname del file

Modo:
- **r** - read
- **w** - write
- **a** - append
- **rb**
- **wb**
- **ab**
- **r+**
- **w+**
- **a+**

Input/Output: `stdio.h`

```
ifp = fopen("./inputfile","r");
```

- interagisce con il sistema operativo per controllare se il file esiste e se il programma ha permesso di leggerlo
  - Deve avere il permesso **`r`** (si può controllare con il comando "**`ls -l inputfile`**" da shell) per l'utente o il gruppo che esegue il programma
- se il file esiste ed abbiamo il permesso di leggerlo la funzione **`fopen()`** alloca una struttura **`FILE`** , ci inserisce tutte le informazioni che servono per utilizzare il file e restituisce il puntatore

Input/Output: `stdio.h`

```
ifp = fopen("./inputfile","r");
```

- se c'è un problema (ad esempio il file non esiste) la funzione **`fopen()`**
  1. restituisce il puntatore NULL e
  2. mette in una variabile globale (**`errno`**) il codice dell'errore che si è verificato
- Posso utilizzare la funzione di libreria **`perror()`** per fare stampare a schermo un messaggio di errore significativo
  - **`perror()`** legge il codice in **`errno`** e stampa la frase che corrisponde all'errore, ad esempio "No such file or directory"
  - Molte funzioni di libreria usano **`errno`** per questo scopo, quindi **`perror()`** va chiamata subito dopo la **`fopen()`**

Input/Output: `stdio.h`

- `stdio.h` contiene delle strutture FILE predefinite e dei puntatori predefiniti a queste strutture

  `FILE * stdin` : standard input, la tastiera
  `FILE * stdout` : standard output, lo schermo
  `FILE * stderr` : standard error, lo schermo

- Questi puntatori possono essere usati direttamente senza bisogno di usare **`fopen()`** e non devono essere chiusi con **`fclose()`**

```
#include <stdio.h>
#include <stdlib.h>
/* scrivo sullo standard output invece che sul file
   ./outputfile */
int main (void){
   int a, sum = 0;
   FILE * ifp;
   ifp = fopen ("inputfile","r");
   if ( ifp == NULL ) { ....}
   /* finchè il numero di conversioni operate con successo è
   uguale a 1 */
   while (fscanf(ifp, "%d", &a)==1)
           sum+=a;
   fprintf(stdout, "la somma è %d\n", sum);
   fclose(ifp);
   return 0;
}
```

Eseguiamo....

```
$ ./leggi
la somma e' 100
$
```

```
#include <stdio.h>
#include <stdlib.h>
/* scrivo sullo standard output invece che sul file
   ./outputfile */
int main (void){
   int a, sum = 0;
   FILE * ifp;
   ifp = fopen ("inputfile","r");
   if ( ifp == NULL ) { ....}
   while (fscanf(ifp, "%d", &a)==1)
          sum+=a;
   fprintf(stdout, "la somma è %d\n", sum);
   fprintf(stderr, "..sto terminando...\n");
   fclose(ifp);
   return 0;
}
```

Eseguiamo....

```
$ ./leggi
La somma e' 100
..sto terminando...
$ ./leggi 1> out 2> err
$ more out
La somma e' 100
$ more err
..sto terminando...
$
```

Input/Output: `stdio.h`

Esempi:

`fprintf(stdout, "la somma è %d", sum);`

scrive sullo standard output

equivale a **`printf("la somma è %d", sum);`**

`fscanf(stdin, "%d", &a)`

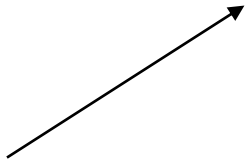
legge dallo standard input

equivale a **`scanf("%d", &a)`**

Input/Output: `stdio.h`

Modificare la posizione corrente di un file:

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



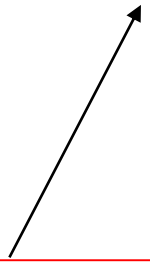
SEEK_SET - inizio file
SEEK_CUR - posizione corrente
SEEK_END - fine file

Posizione di partenza

Input/Output: `stdio.h`

Modificare la posizione corrente di un file:

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



numero di byte di cui mi voglio spostare (anche negativo)

Input/Output: `stdio.h`

Modificare la posizione corrente di un file:

- Funzione per ritornare ad all'inizio del file

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

Esempio:

```
rewind (fp);
```

equivale a

```
fseek(fp, 0, SEEK_SET);
```

Input/Output: `stdio.h`

Capire se siamo a fine file:

- **`int feof(FILE *fp)`**
  - La funzione restituisce 0 se l'indicatore di fine file è attivo e un valore diverso da 0 altrimenti
  - Vediamo un esempio di uso ...

```
#include <stdio.h>
#include <stdio.h>
/* stampa tutti i valori pari e poi tutti i dispari */
int main (void){
  int a; FILE * ifp;
  ifp = fopen ("inputfile","r");
  if ( ifp == NULL ) { ....}
  fprintf(stdout,"Valori pari:\n");

  while (!feof(ifp)) {
        fscanf(ifp, "%d", &a);
        if ( a % 2 == 0 ) printf("%d\n",a);
       }
  fprintf(stdout,"Valori dispari:\n");
  rewind(ifp);
  .... (segue)
```

```
#include <stdio.h>
#include <stdio.h>
/* stampa tutti i valori pari e poi tutti i dispari */
int main (void){
.....
  rewind(ifp);
  fprintf(stdout,"Valori dispari:\n");

while (!feof(ifp)) {
     fscanf(ifp, "%d", &a);
     if ( a % 2 != 0 ) printf("%d\n",a);
   }
   fclose(ifp);
   return 0;
 }
```

Eseguiamo....

```
$ ./stampaparidispari
Valori pari:
34
56
2
4
Valori dispari:
1
3
$
```

Input/Output: `stdio.h`

Leggere e scrivere stringhe:

```
int sscanf(const char *s,
                  const char *format, …);
```

funziona come **`scanf()`, `fscanf()`**

`s` è la stringa da cui leggere

```
int sprintf(const char *s,
                   const char *format, …);
```

funziona come **`printf()`, `fprintf()`**

`s` è la stringa su cui scrivere

Input/Output: `stdio.h`

.....bufferizzazione …

tipicamente tutto l'output viene bufferizzato

Si può bufferizzare una linea (fino a '\n') o di più

questo è il motivo per cui alcune volte i caratteri stampati con **`printf()`** non appaiono subito

`int fflush(FILE * ifp)`

svuota immediatamente i buffer relativi al file `ifp`

`fflush(NULL)` svuota tutti i buffer

è chiamata dalla `fclose()`

```
#include <stdio.h>
/* esempio di bufferizzazione, lo standard output
se collegato a terminale è bufferizzato fino a \n */
int main (void){
  int a;
  fprintf(stdout,"Prova buffer:"); /* non ho \n */
  getchar(); /* si blocca senza stampare niente */
  fprintf(stdout," ..fine prova ....");
  return 0;
}
```

Eseguiamo....

```
$ ./provabufferIO
```

Non stampa niente e si blocca su getchar().

Se digito un qualsiasi carattere ....

```
$ ./provabufferIO
Prova buffer: .. fine prova ...
$
```

Viene stampato tutto ...

La fclose() (chiamata automaticamente alla chiusura del main) svuota il buffer anche se la seconda fprintf() non termina con \n

```
#include <stdio.h>
/* esempio di bufferizzazione, lo standard output
Se collegato a terminale è bufferizzato fino a \n */
int main (void){
  int a;
  fprintf(stdout,"Prova buffer:"); /* no \n */
  fflush(stdout); /* svuoto il buffer */
  getchar(); /* si blocca senza stampare niente */
  fprintf(stdout," ..fine prova ....");
  return 0;
}
```

Eseguiamo....

```
$ ./provabufferIO
Prova buffer:
```

La stampa viene effettuta prima di bloccarsi in attesa di un carattere. Se digito un qualsiasi carattere ....

```
$ ./provabufferIO
Prova buffer: .. fine prova ...
$
```

Si stampa anche la seconda (come prima)

Scrittura/Lettura di strutture da file

- Possiamo trasformare la struttura in una stringa secondo una certa codifica
- Ma possiamo anche scrivere direttamente la sua rappresentazione binaria
  - Facciamo una esempio .....

Scrittura su file (binario)

```
#define MAXLEN 40
typedef struct studente {
   char nome[MAXLEN+1];
   char cognome[MAXLEN+1];
   unsigned matricola;
} studente_t ;
```

Scrittura su file (binario)

```
#define MAXLEN 40
#define NSTUD 3
typedef struct studente { ... } studente_t ;

int main (void) {
studente_t corsoA[NSTUD];
FILE * outf;
/* parte che elabora gli studenti .... la vediamo in lab*/
if ( ( outf = fopen("outfile","w") ) == NULL ) {
 perror("fopen: outfile"); exit(EXIT_FAILURE); }
scrivi_stud(corsoA, NSTUD, outf);
return 0 ;
}
```

Scrittura su file (binario)

```
#define MAXLEN 40
typedef struct studente { ... } studente_t ;

/* scriviamo direttamente i byte della rappresentaizone
   (attenzione a cambiare computer ....) */
void scrivi_stud (studente_t * a, int n, FILE* f) {
  /* scrive n elementi a partire da a ognuno lungo
  sizeof(studente) byte nel file f */
  fwrite( a,sizeof(studente_t), n, f );

}
```

Scrittura su file (testo)

```
#define MAXLEN 40
typedef struct studente { ... } studente_t ;

/* serializzazione: trasformazione in stringa e scrittura
  in file di testo
*/
void scrivi_stud (studente_t * a, int n, FILE* f) {
  int i;
  for (i=0; i>n; i++)
    fprintf(f,"%s:%s:%u\n", a[i].nome, a[i].cognome,
                    a[i].matricola);

}
```

Esempio: lettura (binario)

```
#include <stdio.h>
#define MAXLEN 40
#define NSTUD 3
typedef struct studente {...} studente_t;
/* legge da file binario nell'array x e stampa su stdout */
int main (void) {
studente_t x[NSTUD]; int i; FILE* ifp;
ifp = fopen("studenti.out","r");
fread(x,sizeof(studente_t),NSTUD,ifp);
for (i=0;i<NSTUD;i++)
  printf("%d: %s %s %u\n",i, x[i].nome, x[i].cognome, x[i].matricola);
return 0;
}
```