GESTIONE DELLA MEMORIA

```
void * malloc (size_t size);
void   free (void * ptr);
void * calloc (size_t count, size_t size);
```

dimensione di ogni elemento
numero di elementi da allocare

ATTENZIONE: calloc inizializza a 0 tutti i byte allocati.

```
void * memset (void * buf, int c, size_t len);
```

copia il byte meno significativo di c in len bytes a partire da buf.

```
void * memcpy (void * dst, const void * src, size_t len);
```

copia len bytes da src a dst

ATTENZIONE: mai fornire due buffer che si sovrappongono in modo scorretto in caso contrario comportamento non def.

Es: src [a|b|c|d] dst [a|b|c|d]

dst [a|b|a|b|c|d] src

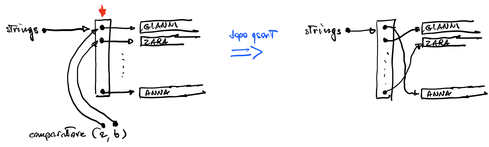
```
void * memmove ( "  "  " );
```

src non viene mai sovrascritto, anche sovrapposto a dst.

FUNZIONI DI ORDINAMENTO E RICERCA

```
#include <stdlib.h>

void qsort ( void * base,            → puntatore ad array da ordinare
             size_t numelem,         → numero di elementi in base
             size_t sizeelem,        → dimensione di ogni elemento
             int (* comparatore)(const * void, const * void)
           );
                                     → puntatore a vera funzione di comparazione

comparatore (a, b)  <0 se a precede b
                    =0 se a==b
                    >0 altrimenti
```

(Appunto su esercizio 6.a. / Es)




```
void * bsearch ( const void * key,        → elemento da cercare
                 const void * base,
                 size_t numelem,
                 size_t sizeelem,
                 int (* comparatore) (const void *, const void *)
               );
```

cerca l'elemento key in base usando la ricerca binaria

ATTENZIONE: base deve essere ordinato

FUNZIONI MATEMATICHE

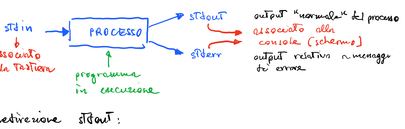
```
#include <math.h>

double  sqrt (double x);
        exp
        log
        cos
        ...
```

ATTENZIONE: per compilare si deve aggiungere lo switch "-lm"

FUNZIONI PER LA GESTIONE DEGLI STREAM

STREAM: astrazione che rappresenta un flusso di byte

Quando eseguiamo un programma il SO assegna di corrispondenti processo tre stream di default:



Redirezione stdout:

```
$ ./calcolatrice > output.txt
$ ./calcolatrice 2> log_errori.txt
```

Per gestire gli stream da codice ho due opzioni: ① libc ② syscall POSIX

① LIBC

```
#include <stdio.h>

FILE * fopen ( const char * path, const char * mode );  // apre un file come stream
```

- puntatore a struct di tipo FILE
- percorso sul file system (fisico) attraverso cui raggiungere il file. Es: "/home/studente/a.txt"
- stringa che rappresenta le operazioni da svolgere sul file:
  - r → leggi
  - w → scrivi
  - a → scrivi in coda (aggiungo)

```
int ferror (FILE * f)
```

Restituisce != 0 se la precedente funzione ha settato un flag di errore

```
void clearerr (FILE * f)
```

Pone pari a 0 il flag di errore

```
int fclose (FILE * f)  // chiude lo stream associato al file
int fprintf ( FILE * f, const char * format, ... )
int fscanf ( "  "  " )
char * fgets (char * str, int size, FILE * f);
int feof (FILE * f)  // indica se è stata raggiunta la fine del file
size_t fread (void * ptr, size_t size, size_t numeltems, FILE * f);
```

- numero di blocchi da leggere
- dimensione blocchi da leggere
- puntatore a buffer dove scrivere quanto letto dal file

```
size_t fwrite (const void * ptr, size_t size, size_t numeltems, FILE * f);
int fseek (FILE * f, long offset, int whence);  // sposta il puntatore di prossimo byte da leggere o scrivere in f
```

- spostamento da applicare al puntatore espresso in byte rispetto alla posizione indicata da whence
- whence:
  - SEEK_CUR : posizione corrente
  - SEEK_END : fine del file
  - SEEK_SET : inizio del file

Es.:
- fseek (f, 10, SEEK_CUR) // sposto in avanti di 10 byte rispetto alla posizione corrente
- fseek (f, 5, SEEK_SET) // sposto al sesto byte dall'inizio del file
- fseek (f, -2, SEEK_END) // sposto al penultimo byte del file

```
long ftell (FILE * f);  // restituisce il corrente offset in byte del puntatore da inizio file
```

② SYSCALL POSIX

```
#include <fcntl.h>

int open (const char * path, int flags, mode_t mode);
```

- file descriptor
- percorso
- modalità di apertura e uso del file
- permessi associati al file

Es.: `int fd = open ("miofile.txt", O_CREAT | O_WRONLY | O_TRUNC, 0640);`

- Se il file esiste, troncalo
- Accetto solo scritture
- Se il file non esiste, crealo

```
ssize_t write (int fd, const void * buf, size_t count);
  "  read  "  "  "  "
off_t lseek (int fd, off_t offset, int whence);
```

invece di ftell si può usare lseek(fd, 0, SEEK_CUR)