========== LIBRERIE ==========

```c
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <sys/ipc.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/wait.h>
#include <errno.h>

#include <pthread.h> // -pthread
```

========== DEFINE ==========

```c
#define RED "\033[0;31m"
#define GREEN "\033[0;32m"
#define GREEN "\033[0;33m"
#define RED "\033[0m"
```

========== FUNZIONI UTILI ==========

```c
memset(void *s, int c, size_t n)                // Riempie i primi n caratteri di s con c.
                                                // Per le stringhe uso c:'\0', per gli interi uso c:-1

fgets(char *s, int n, FILE *stream)             // Per leggere un input contenente spazi (lettura di una stringa)

sscanf(char *s, const char *format, ...)        // Permette di leggere una stringa avente una certa formattazione
                                                // (ritorna 0 in caso di errore)

strcmp(char *s1, char *s2)                      // Restituisce 0 se le due stringhe sono uguali

strcpy(char *dest, const char *src);            // Copia la stringa src sull'area puntata da dest

fprintf(FILE *stream, const char *format, ...)  // Permette di stampare su stderr, stdin, stdin, ptr ecc...

sprintf(char *str, const char *format, ...)     // Permette di stampare su str un output formattato

mkdir(char *path, mode_t mode)                  // Crea una cartella con le proprietà specificate da mode (0755)
```

```
creat(char *path, mode_t mode)                  // Crea un file con le proprietà specificate da mode (0755)

fflush(FILE *stream)                            // Forza l'output

atoi(const char *str)                           // Per convertire la stringa *str in un intero

exit(n)                                         // 0 ha valore di EXIT_SUCCESS
                                                // 1 ha valore di EXIT_FAILURE

fork()                                          // x Padre
                                                // 0 Figlio
```

========== SNIPPETS DI CODICE ==========

CREAZIONE E KILL DI TUTTI I PROCESSI

```
// creazione figli
for(int i=0; i<n_child; i++){
    int isChild=fork();
    if(isChild==0){
        ...
    }else{
        ...
    }
}
// Kill dei processi
kill(0, SIGTERM);
while(wait(NULL) > 0);
exit(0);
```

VERIFICA SE UN FILE ESISTE

```
int fileExists(char *fileName)
{
    FILE *ptr = fopen(fileName, "r");
    //int result = (ptr) ? 1 : 0;
    int result;
    if(ptr != 0)
        result = 1;
    else
        result = 0

    if(result != 0)             // result ot ptr
        fclose(ptr);

    return result;
}
```

VERIFICA SE UNA CARTELLA ESISTE

```
int folderExists(char *folder)
{
```

```
    char current_working_dr[MAX];
    getcwd(current_working_dr, MAX);
    //int result = (chdir(folder) == 0) ? 1 : 0;
    int result;
    if(chdir(folder) == 0)
        result = 1;
    else
        result = 0;

    chdir(current_working_dr);
    return result;
}
```

FUNZIONE STAMPA ERRORI

```
void perror(char *msg){
    fprintf(stderr, "?%s\n", msg);
}
```

PIPE ANONIME

```
#define READ 0      // READ è sempre 0
#define WRITE 1     // WRITE è sempre 1

int fd[2];
pipe(fd);
```

PIPE CON NOME

```
mkfifo("/tmp/...", S_IRUSR | S_IWUSR)   // Nelle pipe con nome, il file deve SEMPRE essere sotto tmp
                                        // (non è necessario che esista)
```

SE UNA CODA ESISTE, LA CANCELLA E LA RICREA

```
key_t queueKey = ftok(key, 32);
int queueId = msgget(queueKey, 0777);

if (queueId != -1)                      // Se la queue esiste già
    msgctl(queueId, IPC_RMID, NULL);    // La cancello...

queueId = msgget(queueKey, 0777 | IPC_CREAT);   // ...creo la queue
```

INVIO MESSAGGIO TRAMITE CODA

```
typedef struct msg_buffer{
    long mtype;
    char tetx[MAX];
}msg_buffer;

int send_msg(){
    msg_buffer msg;
    memset(msg.tetx, '\0', MAX);
    strcpy(msg.text,"ciao");
```

```
    // sprintf(msg.msg, "%d\n", getpid());
    msg.type = 1;

    return msgsnd(queueId, &msg, sizeof(msg.text), 0);
}
```

STRUTTURA TIPICA DELLE CODE

```
typedef struct Msg
{
    long mtype; // Serve per interagire con la coda (invio e ricezione). Così quando dobbiamo leggere
                // un messaggio, possiamo filtrare in base al tipo (come se fosse una categoria)
    ...         // Altri attributi
} Msg;
```

CREAZIONE THREAD

```
void myHandler(int sigNum){
    pthread_t t_id;
    pthread_create(&t_id, NULL, send_msg, (void *)&sigNum);     // Creo thread con t_id che esegue
                                                                // la funzione send_msg
}
```

LETTURA FILE

```
char c;
while ((c = fgetc(FILE *stream)) != EOF) {...}
```

ERRORI

```
external int errno;         // Variabile esterna che cattura i codici di errore
perror(char *s);            // Stampa l'ultimo errore generato preceduto dalla stringa s
strerror(errno);            // Converte errno in stringa
```

========== BASH ==========

```
>&1 echo ...            //output su stdout
>&2 echo ...            //output su stderr

1> file.txt             // stdoutput su file.txt
2> file.txt             // stderr su file.txt

#!/bin/bash             // hash-bang
```

========== PARTE SIMO ==========

PIPE FIFO

```
char *nomeFifo = "/tmp/nomeFifo";   //Importante il tmp senno non funzia

    unlink(nomeFifo);  //Elimino se esiste già
    mkfifo(fifoMaster, 0666);    //Creo la fifo

    fifoFD = open(nomeFifo[i], O_WRONLY | O_RDONLY | O_RDWR);   //Apro

    write(fifoFD, str, strlen(str));    //Scrivo

    int r = read(fifoFD,buf,MAX); //Leggo
    if(r>0){
        buf[r] = 0;
    }
}
```

PIPE SENZA NOME

```
//Create pipes
int pipe1[2]; int pipe2[2];
pipe(pipe1); pipe(pipe2);   //Creazione
if(!fork()){
    close(pipe1[WR]);
    read(pipe1[RD],&buf,MAX);   //Lettura
    close(pipe2[RD]);
    write(pipe2[WR],"msg",MAX); //Scrittura
}
else{
    close(pipe1[RD]);
    write(pipe1[WR],"msg",MAX); //Scrittura
    close(pipe2[WR]);
    read(pipe2[RD],&buf,MAX);   //Lettura
}
```

MESSAGE QUEUE

```
typedef struct msg{
        long mtype;
        char mtext[MAX];
}msg;

//Creo la queue
key_t queueKey=ftok("/tmp/unique",1);
int queueId=msgget(queueKey,0777 | IPC_CREAT);

//Invio
msg messaggio;
messaggio.mtype=command-1;
strcpy(messaggio.mtext,"mexx");
msgsnd(queueId,&messaggio,sizeof(messaggio.mtext),0);

//Ricezzione
```

```
msg msg_rcv;
msgrcv(queueId,&msg_rcv,sizeof(msg_rcv.mtext),type,0);
```

LISTA SEGNALI

```
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
```

SEGNALI NORMALI

```
void handler(int signo){}
signal(SIGUSR1,handler);
```

SIGACTION

```
 struct sigaction sa;  //Dichiarazione

void set_action(){
    sa.sa_flags = SA_SIGINFO;   // Ricevo infoemazioni da chi manda il
                                // segnale tipo il PID
    sa.sa_sigaction = handler; //Uso sa_sigaction perchè posso usare info
    sigaction(SIGUSR1,&sa,NULL);    //Blocco i segnali andranno in handler
    ...
}

void handler(int signo, siginfo_t *info,void *sium){
    int sender = (int)info->si_pid; //Prendo il PID di chi invia il segnale
}
```

TERMINAZIONE FIGLI

```
// Utile per terminare i figli
signal(SIGINT,signalIntHandler); //Override handler to kill all children

//Termination function
void signalIntHandler(int signo){
    //Issue termination signal to all children
    kill(0,SIGTERM);

    //Wait children termination
    while(wait(NULL)>0);
    printf("Terminating");

    //Terminate
    exit(0);
}

//Versione con piu figli e pid salvati
void signalIntHandler(int signo){

    if(getpid()!=pidFather){
        return;
```

```
    }

    //Issue termination signal to all children
    for(int i=0;i<tot_figli;i++){
        printf("kill: %d\n",children[i]);
        kill(children[i],SIGTERM);
    }

    //Wait children termination
    while(wait(NULL)>0);
    printf("Terminating\n");

    //Terminate
    exit(0);
}
```

LETTURA E SCRITTURA SU FILE

```
//Scrittura con file descriptor
    int openedFile=open("tmp/log.txt",O_CREAT|O_RDWR|O_APPEND,S_IRUSR|S_IWUSR); //Apro il file
    write(openedFile,messaggio,strlen(messaggio));
    close(openedFile);

//Lettura con file descriptor
    int openedFile=open("tmp/log.txt",O_RDONLY); //Apro il file
    if(errno!=0){   //Check errori
        return 1;
    }
    read(openedFile,buf,MAX); //leggo
    close(openedFile);

//scrittura con streams
    FILE *ptr = fopen("/tmp/log.txt", "a"); //a=write at end
    fprintf(ptr, messaggio);
    fclose(ptr);

//Lettura con streams
    FILE *ptr = fopen(fileName, "r");
    char mex[MAX];
    while (!feof(ptr))
    {
        fscanf(ptr,"%s",mex);
    }
    fclose(ptr);
```

THREAD

```
pthread_t t_id; //Id thread
int arg=10;
pthread_create(t_ids, NULL, ascoltoSlave, (void *) &arg);   //Faccio partire il thread
//pthread_create(t_ids, NULL, ascoltoSlave,NULL);

void* my_fun(void* param){
    //return (void *)3; //non necessario
}
```

========== ESERCIZI ==========

Stampa “T” (per True) o “F” (per False) a seconda che il valore rappresenti un file o cartella esistente

```
[[ -f $DATA ]] && echo T; [[ -d $DATA ]] && echo F;
```

Stampa “file”, “cartella” o “?” a seconda che il valore rappresenti un file (esistente), una cartella (esistente) o una voce non presente nel file-system

```
([[ -f $DATA ]] && [[ -e $DATA ]] && echo file ) || ([[ -d $DATA ]] && [[ -e $DATA ]] && echo cartella) || echo ?
```

Stampa il risultato di una semplice operazione aritmetica (es: ‘1 < 2’) contenuta nel file indicato dal valore di DATA, oppure “?” se il file non esiste

```
[[ $(cat $DATA) ]]  && echo ?;
```

Scrivere uno script che dato un qualunque numero di argomenti li restituisca in output in ordine inverso.

```
lista=()

while [[ $1 != "" ]]; do
    lista[${#lista[@]}]=$1
    shift
done

for i in ${!lista[@]}; do
    echo ${lista[${#lista[@]}-$i-1]}
done
```

Scrivere uno script che mostri il contenuto della cartella corrente in ordine

```
lista=($(ls))
for i in ${!lista[@]}; do
    echo ${lista[${#lista[@]}-$i-1]}
done
```

Creare un “alias” in bash denominato “feedback” che se invocato attende dall’utente un input proponendo il prompt “Come ti chiami?” e rispondendo con “Ciao !” (dove è l’input immesso) senza però sovrascrivere o impostare alcuna nuova variabile nella shell attiva.

```
alias feedback='echo Come ti chiami?; read var; echo Ciao $var;'
```

Creare un “alias” in bash denominato “somma” che legge un numero (intero con segno) alla volta (numero+INVIO, numero+INVIO, …) e alla fine (immissione vuota premendo solo INVIO) stampa la somma dei numeri inseriti.

```
alias somma='var="."; somma="0"; while [[ $var != "" ]]; do read var; somma="$somma+$var"; done; somma="$somma 0"; echo $(( $somma ));'
```