C secondo sCagnetto

concetti basilari (tipi, sintassi, ecc...)

```
// Tipi
char, short, int, long , float, double // in ordine di promozione, (tipo) per il casting
```

- **input/output standard**

```
#include <stdio.h>

char getchar(); // legge carattere dallo standard input
void putchar(c); // stampa carattere nello standard output
```

- **operazioni su stringhe**

```
#include <string.h>

char message[] = "testo della stringa";
char message[] = {'t', 'e', 's', ..., 'g', 'a'}

int strlen(char *s); // restituisce la lunghezza di una stringa
int strncmp(char *s1, char *s2, unsigned len); //confronto lessicografico (0 uguali, -1 altrimenti)
int strcmp(char *s1, char *s2); // versione meno sicura senza lunghezza
char *strncpy(char *dest, char *source, unsigned len); // copia i primi len caratteri di source in a
char *strncat(char *dest, char *source, unsigned len); // concatena i primi len caratteri di source
long long strtoll(const char *str, char **endptr, int base); // converte stringa in long long

ERRORE: char *stringa = "testo della stringa";
```

- **stampa**

```
// Stampa
printf("stringa", %1, %2, ...); // stampa stringa nello standard output, sostituendo ogni % nella st
scanf("formato", &var1, &var2, ...); // legge input da standard input e lo memorizza nelle variabili

sscanf(stringa, "formato", &var1, &var2, ...); // legge da stringa fornita come parametro invece che
sprintf(stringa, "formato", var1, var2, ...); // stampa su una stringa invece che su standard output
snprintf(stringa, dimensione, "formato", var1, var2, ...); // ulteriore argomento lunghezza massima
```

- **memoria dinamica**

```c
// Memoria dinamica
void *malloc(unsigned n); // serve ad allocare n byte contigui, void * è un puntatore di qualsiasi t
void free(void *ptr); // serve a liberare la memoria allocata con malloc (buona norma usarlo sempre)
void *realloc(void *ptr, unsigned new_size); // funzione ritorna un nuovo puntatore
void *calloc(unsigned count, unsigned size); // alloca della memoria azzerandola precedentemente
```

- **chiamate ISO**

```c
// Chiamate ISO
FILE *stdout; // standard output (FILE in scrittura)
FILE *stdin; // standard input (FILE in lettura)
FILE *stderr; // standard error (FILE in scrittura)

FILE *fopen(char *name, char *mode); // restituisce il puntatore del file su cui operare. *mode può
int fclose(FILE *fp); // chiude il file pointer (buona norma utilizzarlo sempre)

int fprintf(FILE *fp, char *format, ...); // analogo a printf()
int fscanf(FILE *fp, char *format, ...); // analogo a scanf()
int fgetc(FILE *fp); // analogo a getchar()
int fputc(int c, FILE *fp); // analogo a putchar()

size_t fread(void *ptr, size_t size, size_t nitems, FILE *file); // legge (size * nitems) byte da fi
size_t fwrite(void *ptr, size_t size, size_t nitems, FILE *file); // scrive (size * nitems) byte da

int feof(FILE *fp); // restituisce vero (n > 0) se lettura è arrivata alla fine
int ferror(FILE *fp); // restituisce vero (n > 0)

char *strerror(int errno); // ritorna stringa di descrizione corrispondente al valore di errno, cioè
void perror(const char *__s); // stampa messaggio che descrive il valore di errno

int fseek(FILE *fp, long offset, int whence); // imposta la posizione attuale a _offset_ byte da pos

// whence = SEEK_SET (inizio file), SEEK_CUR (posizione corrente), SEEK_END (fine file)

long ftell(FILE *file); // restituisce posizione attuale
```

- **CHIAMATE POSIX**

```c
#include <unistd.h>
#include <fcntl.h>

// Standard file descriptors
#define STDIN_FILENO 0 // standard input
#define STDOUT_FILENO 1 // standard output
#define STDERR_FILENO 2 // standard error

int open(const char *pathname, int openflags); // restituisce il file descriptor
int creat(const char *pathname, mode_t mode); // equivalente a open() con flag O_CREAT | O_WRONLY |
int close(int fd); // chiude il file descriptor
ssize_t read(int fd, void *buffer, size_t nbytes); // legge
ssize_t write(int fd, const void *buffer, size_t n); // scrive
off_t lseek(int fd, off_t offset, int whence); // imposta la posizione
int unlink(const char *pathname); // rimuove un file
int fcntl(int fd, int cmd, ... /* arg */); // manipola il file descriptor
int chmod(const char *pathname, mode_t mode); // cambia i permessi del file

#include <sys/types.h>
#include <sys/stat.h>

int stat(const char *pathname, struct stat *out); // ottiene informazioni sul file
int fstat(int filedes, struct stat *out); // ottiene informazioni sul file descriptor

struct stat {
    dev_t st_dev;       // device id
    ino_t st_ino;       // inode number
    mode_t st_mode;     // tipo di file e permessi
    nlink_t st_nlink;   // numero di link non simbolici
    uid_t st_uid;       // UID
    gid_t st_gid;       // GID
    dev_t st_rdev;      // device type
    off_t st_size;      // dimensione del file
    time_t st_atime;    // tempo di ultimo accesso
    time_t st_mtime;    // tempo di ultima modifica
    time_t st_ctime;    // tempo di creazione
    long st_blksize;    // dimensione del blocco
    long st_blocks;     // numero di blocchi
};
```

- **processi**

```c
#include <sys/types.h>

// Processi
pid_t getpid(void); // restituisce il PID
pid_t getppid(void); // restituisce il PPID
pid_t getpgrp(void); // restituisce il gruppo

pid_t fork(void); // duplica il processo, restituisce PID con padre, 0 con figlio

int execve(const char *pathname, char *const argv[], char *const envp[]); // esegue un nuovo program

pid_t wait(int *stat_loc); // aspetta la terminazione di un processo figlio

void exit(int status); // termina il processo

char *getenv(const char *name); // ottiene il valore di una variabile d'ambiente
int setenv(const char *name, const char *value, int overwrite); // setta una nuova variabile d'ambie
```

- **pipe**

```c
// Pipe
int pipe(int pipefd[2]); // crea una pipe
int dup2(int oldfd, int newfd); // duplica un file descriptor
```

- **segnali**

```c
#include <signal.h>

// Segnali
int kill(pid_t pid, int sig); // invia il segnale _sig_ al processo con PID _pid_
int raise(int sig); // invia il segnale _sig_ a se stessi
unsigned int alarm(unsigned int secs); // ricezione di SIGALRM dopo int secs
```

- **socket**

```c
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

// Socket
int socket(int domain, int type, int protocol); // crea un socket
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen); // associa un indirizzo al soc
int listen(int sockfd, int backlog); // mette il socket in ascolto
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen); // accetta una connessione
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen); // connette il socket a un
ssize_t send(int sockfd, const void *buf, size_t len, int flags); // invia dati
ssize_t recv(int sockfd, void *buf, size_t len, int flags); // riceve dati
int close(int sockfd); // chiude il socket
```

- **multithreading**

```c
#include <pthread.h>

// Multithreading
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), vo
int pthread_join(pthread_t thread, void **retval); // aspetta la terminazione di un thread
int pthread_mutex_lock(pthread_mutex_t *mutex); // blocca un mutex
int pthread_mutex_unlock(pthread_mutex_t *mutex); // sblocca un mutex
```