C Cheat Sheet

Index

Libraries

```
// basic
#include <unistd.h> // fork, pipe and I/O primitives (read, write, close, etc.)
+ primitve types like uid_t, pid_t etc
#include <stdlib.h> // standard lib, contains primitves for number conversion
and memory allocation
#include <stdio.h> // basic i/o lib: printf etc
#include <string.h> // string manipulations
#include <time.h> // time related functions
#include <signal.h> // signal handling
#include <stdbool.h> // boolean type
#include <math.h> // math functions

// advanced
#include <sys/socket.h> // socket connections
#include <sys/types.h> // primitive types like uid_t, pid_t etc
#include <netinet/in.h> // internet address family
#include <arpa/inet.h> // definitions for internet operations
#include <pthread.h> // threads
#include <stdatomic.h> // mutual exclusion locks
```

Format Type Specifiers

```
specifier Output
%c        Character
%s        String of characters
%d        or i Signed decimal integer
%f        Decimal floating point
%llu      unsigned long long
%o        Signed octal
%u        Unsigned decimal integer
%x        Unsigned hexadecimal integer
%p        Pointer address
```

File Modes

```
mode    Description
"r"     Opens a file for reading. The file must exist.
"w"     Creates an empty file for writing. If a file with the same name already
exists, its content is erased and the file is considered as a new empty file.
"a"     Appends to a file. Writing operations, append data at the end of the
file. The file is created if it does not exist.
"r"     Opens a file to update both reading and writing. The file must exist.
"w"     Creates an empty file for both reading and writing.
"a"     Opens a file for reading and appending.
```

Important Function Signatures and Return Values

Random Numbers

```
// seed with current time:
// time_t t;
// srand(time(&t));
void srand(unsigned seed);

int rand(void);
```

Signals

```
sig_t signal(int sig, sig_t func);

int raise(int sig);

int kill(pid_t pid, int sig);
```

Sorting

```
void qsort(void* values, size_t num_items, size_t item_size, int (*comparefunc)
(const void*, const void*));
```

Strings

```
// SUCCESS: pointer to destination string
char *strcpy(char *dest, const char *src);

// result == 0 -> strings are equal
// result < 0 -> str1 less than str2
// result > 0 -> str2 less than str1
int strcmp(const char *str1, const char *str2);;


// result == NULL -> no tokens left to retrieve
// else: pointer to last token found
char *strtok(char *str, const char *delim);

// NULL if no match
// else: pointer to first occurence in string
char *strstr(const char *string, const char *substring);

// ERROR: result == 0
int atoi(const char *str);
```

Inter Process Communication

```
// ERROR: result < 0
// result = 0 inside child
// result > 0 inside parent
pid_t fork(void);

// ERROR: result < 0
// SUCCESS: result == 0
int pipe(int fd[2]);
```

File IO

```
// ERROR: result == NULL (also when EOF is reached)
char *fgets(char *str, int strlen, FILE *stream);

// ERROR: result == NULL
FILE *fopen(const char *filename, const char *mode);

// result == EOF when finished reading the stream
// SUCCESS: number of matched items on success
int fscanf(FILE *stream, const char *format, ...);

// ERROR: result == EOF
// SUCCESS: result == 0
int fclose(FILE *stream);

// ERROR: result == -1
// SUCCESS: number of bytes written
ssize_t write(int fildes, const void *buf, size_t nbyte);

// ERROR: result == -1
// EOF when finished reading
// SUCCESS: number of bytes read
ssize_t read(int fildes, void *buf, size_t nbyte);

// SUCCESS: result > 0
// ERROR: result == EOF
int fputs(const char *restrict s, FILE *restrict stream);
```

Shared Memory

```c
// ERROR: result < 0
// SUCCESS: result == shmid
int shmget(key_t key, size_t size, int shmflg);

// ERROR: result == NULL
void *shmat(int shmid, const void *shmaddr, int shmflg);

// ERROR: result < 0
// SUCCESS: result == 0
int shmdt(const void *shmaddr);
```

Networking

```c
// ERROR: result == -1
// example: sock = socket(AF_INET, SOCK_STREAM, 0);
int socket(int domain, int type, int protocol);

// sockaddr_in struct (man ip 4)
struct sockaddr_in server;
server.sin_family = PF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(8080);

// ERROR: result < 0
// SUCCESS: result == 0
// example: bind(sock, (struct sockaddr *) &server, sizeof(server))
int bind(int sockfd, const struct sockaddr *my_addr, socklen_t addrlen);

// ERROR: result < 0
// SUCCESS: filedeskriptor for accepted socket
// example: fd = accept(sock, (struct sockaddr *) &client, &client_len);
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

// SUCCESS: result == 0
// ERROR: result == -1
int listen(int socket, int backlog);
```

Threads

```c
// ERROR: result > 0
// SUCCESS: res == 0
// example: pthread_create(&threads[t], NULL, printHello, (void*)t);
int pthread_create(pthread_t *restrict thread, const pthread_attr_t *restrict
attr, void *(*start_routine)(void *), void *restrict arg);

// SUCCESS: result == 0
// ERROR: result > 0
int pthread_join(pthread_t tid, void **ret);

void pthread_exit(void *value_ptr);
```