Standard Librabry

```
#include <...>
```

<stdio.h>

- printf, scanf, fprintf
- fopen, fclose
- puts, -char, getChar

<stdlib.h>

- EXIT_SUCCESS = 0
- EXIT_FAILURE = -1, 1...

<stdint.h>

- int8_t, int16_t, ..., uint8_t, uint16_t, ...

<stddef.h>

- size_t → array size
- ptrdiff_t → pointer size

<stdbool.h>

- bool b = true/false;

<string.h>

- strlen
- strcpy
- strcat
- strcmp

<math.h>

- sqrt

<assert.h>

- assert(boolean expression)

C Snippets

Code Structure

Main

```
int main(void)
{
    ...
    return EXIT_SUCCESS;
}
```

Preprocessor

```
#define LENGTH 100
...
int length = LENGTH;

#define SQUARE(x) ((x) * (x))
...
double area = 3.141 * SQUARE(5)

#define DEBUG
...
#if defined DEBUG
#elif ...
#else ..
#endif ...

#ifdef DEBUG
#ifndef DEBUG
```

```
#undef DEBUG
```

Header Guard in File name.h

```
#ifndef NAME_H
#define NAME_H
... declarations of name...
#endif

#include "name.h"
```

Conditionals

```
if (...) {

} else if (...) {

} else {

}

int x = a > b ? c : d;

switch(a) {
    case 1:
        ...
        break;
    case 2: case 3:
        ...
        break;
    default:
        ...
        break;
}

switch(myenum) {
    case ENUMVALUE:
        ...
        break;
    default:
        ...
        break;
}
```

Loops

```
for (int i = 1; i < 5; i++) {

}

while (i < 5) {
    i++;
}

do {

} while (i < 5)
```

Functions

```
int max(int a, int b); // allowed multiple times across files

int max(int a, int b) { // only once
    return a; // arrays cannot be returned
}
...
int m = max(1, 2);

(void) max(1, 2);

int increment(a) {
    return ++a;
}
int a = 1;
increment(a); // a = 1
a = increment(a); // a = 2

int myVoid(void); // parameter check is skipped
```

```
void write_int(const int a); // a is read only for this function
```

Dynamic amount of parameters

```
#include <stdarg.h>

void func(unsigned amount, ...) {
    va_list args;
    va_start(args, amount);
    int value_i = va_arg(args, int);
    va_end(args);
}
```

Bit operations

```
unsigned int number = 0x75;
unsigned int bit = 3; // bit position

// Setting a bit
number |= (1<<bit);

// Clearing a bit
bit = 1;
number &= ~(1<<bit);

// Toggling a bit
bit = 0;
number ^= (1 << bit);

// swap integers
a = a ^ b;
b = a ^ b;
a = a ^ b;

int is_power_of_two(int value) {
    int bits = sizeof(value) * 4;
    int set = 0;
    for (int i=0; i < bits; i++) {
        int mask = (0x01<<i);
        if ((value&mask) == mask) {
              set ++;
        }
        if (set > 1) {
            return EXIT_FAILURE;
        }
    }
    return EXIT_SUCCESS;
}

void print_binary(unsigned int value, int print_new_line) {
    int bits = sizeof(value) * 8 -1;
    for (int i=bits; i >= 0; i--) {
        int is_set = ((value >> i) & 0x01) == 0x01;
        printf ("%d", is_set >= 1 ? 1 : 0);
        if (i%8 == 0 && i > 0) {
            printf("'");
        }
    }
    if (print_new_line) {
        printf("\n");
    }
}
```

Data Types

Type declarations

```
double a;
int b, c;
int d = 1;
const int e;
bool b;
typedef int f;
f g;
int globalVariable = 1; /* überall sichtbar */
static int globalVariable; /* nur innerhalb datei */
... {
    static int max = 0; /* bei nächstem Funktionsaufruf noch da */
}
```

```
auto int a; /* in stack */
register int a; /* in register, tipp zur optimierung */
static int a; /* als 0 initiiert, lebt bis programmende */
extern int a; /* " */
```

Variable Sizes

- int = 4 Bytes
- short = 2 Bytes
- char = 1 Byte
- pointer = 4 Bytes

Type cast

```
int a = 1;
double b = (double) a / 3
```

Enum

```
enum A {a, b, c}; /* 0, 1, 2 */
enum B {a = 1, b, c}; /* b = 2, c = 3 */
enum B {a, b = 2, c}; /* a = 0, c = 3 */
typedef enum {a, b, c} C;
...
int d = a;
int e = b;
enum A f = a;
C g = a;
```

Struct

```
struct a {
    int x;
    int y;
    int z;
};
...
struct a myA = {1, 2, 3};
struct a myB;
myB = myA; /* kopiert */
myB.x = 4;

typedef struct {
    int x;
    int y;
    int z;
} B;
...
B myA = {5, 6, 7}

typedef struct C C;
struct C {
    int x;
    int y;
    int z;
};

void print_struct(const struct a *p) {
    printf("%d", p->x);
}
print_struct(&myA);
```

Strings

End of string = \0 = 0000000 → Space = Amount of chars + 1

```
char *mystring = "hello";
char mystring[] = "hello";
char mystring[5] = "hello"; // \0 is missing, printf keeps printing until \0
char mystring[6] = "hello";
char mystring[50] = "hello"; // filled up with \0

char mystring[3];
mystring = "hi"; // compile error
mystring[0] = 'h'; // ok
...
mystring[2] = '\0'; // now it is a string
```

```
strlen(str); /* 5 */
sizeof(str); /* 6 */

int comparison = strcmp("hans", "haus"); // n < u -> <0

char[] source[10] = "Hellooooo";
char[] dest[10];
char[] mydest = strcpy(dest, source);

char[5] s1 = "hi"; // needs to
be large enough
char[3] s2 = "hi";
char[5] mycat = strcat(s1, s2); // "hihi"

const char toLower(const char c) {
    return c | (1 << 5);
}

const char toUpper(const char c) {
  return c & ~(1 << 5);
}

if (strcmp(string1, string2) == 0) {
    // strings are equal
}

// convert integer to string
char str[8];
int i = 5;
sprintf(str, "%d",i);

// convert string to integer
i = atoi(str);
```

Booleans

```
#define bool int
#define false 0
#define true 1
// or
#include <stdbool.h>

bool b1 = true;
bool b2 = false;
```

IO

Basic Methods

```
puts("HELLO"); // output
putChar("A");
getchar(); // read one charater

(void) printf("%i", myInt)

int day, month, year;
a = scanf("%d%d%d", &day, &month, &year);
```

Formatting

- %d, %i = int
- %u = unsigned int
- %c = char
- %s = string bis \0
- %20s = string of 20 chars
- %f = double, float (nur printf)
- %5.4f = 5 lang vor komma, 4 lang nach komma
- %lf = %f für double für scanf
- %g
- %zd
- %x, %X
- %o

User Input with scanf

```
double my_value = 1;
printf("Enter double value: ");
```

```c
scanf("%lf", &my_value);

if (scanf("%lf", &my_double) != 1) {
    // invalid, should be the number of converted values
}
scanf("%u", &unsigned_integer);
```

Date input

```c
typedef struct Date {
    int Year;
    int Month;
    int Day;
} date_t;

int main(int argc, const char *argv[]) {
    date_t date;
    if (sscanf(argv[1], "%d-%d-%d", &date.year, &date.month, &date.day) != 3) {
        // invalid
    }
}
```

User Input with getChar

```c
while ((pressedKey = getchar()) != '\n') {
    ...
}
```

Clear Input Buffer

```c
int c;
while((c = getchar()) != '\n' && c != EOF) {}
```

Continue Loop as long as User wants

```c
do {
    int c;
    while((c = getchar()) != '\n' && c != EOF) {}
    printf("Continue? Y/N");
} while(getchar() == 'Y');
```

User Input Integer with validation

```c
int getIntegerValue(const char *text, int min, int max) {
    int wert;
    char eingabe[max];

    while (1) {
        (void) printf("Bitte geben Sie %s ein (%d-%d): ", text, min, max);
        fgets(eingabe, sizeof(eingabe), stdin);

        // Umwandlung in Integer
        wert = atoi(eingabe);

        // Bereichsprüfung
        if (wert >= min && wert <= max) {
            return wert;
        }
        (void) printf("Ungültige Eingabe. Bitte erneut versuchen.\n");
    }
}

int getIntegerValue(char * str, int min, int max) {
    int value;
    scanf("%s: %d", str, &value);

    if (min < value) {
        min = value;
    }

    if (value > max) {
        max = value;
    }

    return value;
}
```

Print color

```
#define RED   "\x1B[31m"
#define GRN   "\x1B[32m"
#define YEL   "\x1B[33m"
#define WHT   "\x1B[37m"
#define RESET "\x1B[0m"
printf("%s*" RESET, color);
```

Read File with error handling and fgetc

```
#include <stdio.h>
#include <stdlib.h>

void perror_and exit (const char *context) { perror (context); exit(EXIT_FAILURE) ; }

int main (int argc, char *argv[]) {
    if (argc > 1) {
        FILE *f = fopen(argv[1], "rb");
        if (!f) perror_and_exit(argv [1]);
        int line_no = 1;
        int print_line_no = 1;
        int c;
        while((c =fgetc(f)) >= 0) {
            if (print_line_no && printf("%6d ", line_no++) < 0) perror_and_exit ("printf");
            print_line_no = c == '\n';
            if (putchar(c) < 0) perror_and_exit ("putchar");
        }
        if (!feof(f)) PERROR_AND_EXIT("fgetc");
        if (fclose(f) != 0) perror_and_exit("fclose");
        return EXIT_SUCCESS;
    }
    return EXIT_FAILURE;
}
```

Read File with fgets

```
char[] * fgets(buff, n, stream);
FILE *fp;
char filename[] = "person_list.csv";
fp = fopen(filename, "r");
if (fp == NULL) {
    return;
}
char s[128];
list_init();
while(fgets(s, 128, fp) != NULL) {
    person_t* person = malloc(sizeof(person_t));
    person_from_csv_string(person, s);
    list_insert(person);
}
fclose(fp);
```

Write File

```
FILE *fp;
char filename[] = "person_list.csv";
fp = fopen(filename, "w");
if (fp == NULL) {
    perror_and_exit("fopen");
}

person_t* person = list_getFirst();
for(int i = 0; i < list_size(); i++) {
    char s[128];
    if(person_to_csv_string(person, s)){
        fprintf(fp, "%s\n", s); // <-- write
    }
    person = list_getNext();
}

fclose(fp);
```

Serialize with snprintf

```
int snprintf(char * buffer, size_t max_len, const char * format, ... );

snprintf(s,  max_len, "%s,%s,%d", person->name, person->first_name, person->age);
```

Deserialize

```
sscanf(s, "%[^,],%[^,],%d", person->name, person->first_name, &(person->age));
```

Arrays

address of element with index 1 = start address + 1 * sizeof(Element) Bytes

Declarations

```
int data[100] = {0};
data[7] = 20;

int data[5];
data = {1, 2, 3, 4, 5};

int data[] = {1, 2, 3, 4, 5}; // length 5

int data[5] = {1, 2, 3, 4}; // data[4] = 0

int b = data[200]; // unchecked
data [300] = 1; // no error on runtime

size_t c = 100;
? s = sizeof(data); // amount of bytes of array
sizeof(*data); // amount of bytes of first element
sizeof(data)/sizeof(*data); // amount of elements

const data[] = {0,1,2,3,4};
data[0] = 1; // compile error
```

Array explicit size

```
#define N_ENTRIES 100
int array[N_ENTRIES] = {0};
for (size_t i = 0; i < N_ENTRIES; i++) {
    array[i] = ...
}

// END MARKER
#define DATA_SENTRY (-1) // cannot be used elsewhere in array
int array[] = {1, 2, 3, DATA_SENTRY};
for(size_t i = 0; array[i] != DATA_SENTRY; i++) {
    array[i] = ...
}
```

Pass array size

```
void access(int array[], size_t n) {
    for(size_t i = 0; i < n; i++) {
        array[i] = ...
    }
}
//or
void access(int *array, size_t n) {
    for(size_t i = 0; i < n; i++) {
        array[i] = ...
    }
}

int a[100] = {0};
size_t n = sizeof(a) / sizeof(a[0]); // or n = 100
access(a, n);
```

Immutable start address

```
int a[3] = {1,2,3};
int b[3] = {1,2,3};
a = b; // compile error
if(a == b) // checks if start address is the same

int *p;
p = a;
p = b;
```

Jagged Array

```
char *jagged = {"two", "three"};
// 0 until jagged[0][2];
// 1 until jagged[1][4];
```

Return array reference

```
int *create_copy(const int array[], int n) { // not int[]
    const int bytes = n * sizeof(int);
    int *cp = malloc(bytes); // not int cp[], would be local variable
    if (cp) memcpy(cp, array, byes);
    return cp;
}
int a[] = {1, 2, 3, 4};
int *copy = create_copy(a, 4);
```

Iterate multidimensional Array

```
for(int row = 0; row < 2; row++) {
    for(int col = 0; col < 3; col++) {
        array[row][col] = ...
    }
}
```

Copy String into Array

```
(void) strcpy(array[i], string);
```

Copy array (b = a)

```
(void) memcpy(b, a, sizeof(a));

for (int i = 0; i < 5; ++i) {
    b[i] = a[i];
}
```

Sort Array of Strings

```
void sortWords(char array[MAX_ROWS+1][MAX_COLUMNS+1], int count) {
    for (int i = 0; i < count - 1; i++) {
        for (int j = 0; j < count - i - 1; j++) {
            if (strcmp(array[j], array[j + 1]) > 0) {
                char temp[MAX_LENGTH + 1];
                (void) strcpy(temp, array[j]);
                (void) strcpy(array[j], array[j + 1]);
                (void) strcpy(array[j + 1], temp);
            }
        }
    }
}
```

Find String in Array

```
int isDuplicate(char array[MAX_WORDS][MAX_LENGTH + 1], size_t n, char *mystring) {
    for (int i = 0; i < n; i++) {
        if (strcmp(array[i], mystring) == 0) {
            return EXIT_SUCCESS;
        }
    }
    return EXIT_FAILURE;
}
```

Pointers

Pointer declarations

```
int *p;
int* p;
char **ppc; // pointer to pointer to char
char *(*ppc);

char *d[20]; // array of 20 pointers to chars
char **d;
char (*e)[20]; // pointer to array of 20 chars

int * p, q; // p is pointer to int, q is int
```

```c
int i;
int *ip;
ip = &i; // ip is now address of i
*ip = 3; // i is now 3

int *ip;
*ip = 25; // 25 is written somewhere unknown
```

Void Pointer

```c
int i = 1;
int *ip = &i;
void *vp = ip; // a void pointer can point at anything
int *ip2 = vp; // no casting is needed
```

Const behavior

```c
int i;
const int ci;

int * const ip = &i; // only pointer is const
*ip = 20; // ok
ip = &k; // error

const int * ip = &ci; // only ci ist const
*ip = 20; // error
ip = &k; // ok

cont int * const ip = &ci; // both are const
*ip = 20; // error
ip = &k; // error
```

NULL

```c
int *p1 = 0; //ok
int *p2 = NULL; // better
if(p2 == NULL || p2 == 0) ...
```

Pointer on Struct

```c
struct student {
char name[30];
};
struct student *sp, s;
(void)strcpy(s.name, "hans");
sp = &s;
(void)printf("%s", (*sp).name);
(void)printf("%s", sp->name);

struct student *sp = malloc(sizeof(struct student));
```

Pointer operations

```c
int a[5] = {1, 2, 3, 4, 5}
int *p;

a[3] = 1;
*(a + 3) = 1;
*(p + 3) = 1;
p[3] = 1;

p = a + 3;
p = &a[3];

// p = a+3
*(p + 1) = 17;
p[1] = 17;
a[4] = 17;

p[-1] = 13;
a[2] = 13;

*(p++) = 19;
*p = 19; p++;
a[3] = 19; p = &p[1];

// p = a+4
```

```
p -= 2;
p = &p[-2];
p = &a[2];

if (a < a + 1) // true
if (a > p) // false

p = a; // ok
a = p; // error
```

Iterate array using pointer

```
int a[2];

int *it = a;
for(size_t i = 0; i > 5; i++) {
    *(it + i) = ...
}

for(int *it = a; it != a+5; ++it) {
    *it = ...
}

int *pe = &a[4];
for(int *it = a; it <= pe; it++) {
    *it = ...
}
```

Pass function pointer

```
double sin(double x);
double return_func(int (*func)(double arg), double x) {
    return func(x);
}
double result = return_func(sin, 90.0);
```

Initialize pointer with string / char array

```
char a[] = "hi" // a contains hi
char *pa = "hello"; // pointer to hello on code segment not stack
a = pa; // error
pa = a; // ok, pointer now to a
pa[1] = 'A'; // error, "hello" is immutable
a[1] = 'A'; // ok

int a[] = {1, 2, 3}; // ok
int *pa = {1, 2, 3}; // error
```

Multidimensional array pointer

```
int a[5] = {1, 2, 3, 4, 5};
int *p = a;

int a[2][3] = {{1, 2, 3},{4, 5, 6}};
int (*p)[3] = a;

q[2][3] = 1;
*(*(q + 2) + 3) = 1;
```

Call by reference

```
void swap_int(int *a, int *b) {
    int saved_a = *a;
    *a = *b;
    *b = *a;
}
```

Malloc

Array

```
int *p = malloc(3 * sizeof(int));
if(p == NULL) {
    // error handling
}
p[0] = 1;
```

```
...
free(p); // only once! do not reassign to other space before freeing
```

Dynamic data

```
size_t number_of_bytes = get_file_size(my_file);
char *data = malloc(number_of_bytes);
if(!data || read_data(data, my_file) == -1) {
    // error handling
}
process_data(data);
free(data);
```

Linked list

Implementation Link

```
typedef struct node { struct node *next; void *payload; } node_t;

node_t linked_list_append(node_t *root, void *payload) {
    assert(root);
    node_t *p = root;
    while(p->next) p = p->next;
    p->next = malloc(sizeof(node_t));
    if(p->next) {
        *(p->next) = (node_t){ NULL, payload };
    }
    return p->next;
}
```

Threads

Fork

```
pid_t pid;
int status;
pid = fork();
switch (pid) {
  case -1:
    perror("Could not fork");
    break;
  case 0:
    printf("Child with pid", getpid());
    printf("has Parent    ", getppid());
    break;
  default:
    printf("Parent with pid ", getpid());
    printf("knows child pid", pid);
    wait(&status);
    break;
}
```

Process management

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

#define PERROR_AND_EXIT(M) do { perror(M); exit (EXIT_FAILURE); } while(0)

int main () {
    pid_t cpid = fork();
    if (cpid == -1) PERROR_AND_EXIT("fork");
    if (cpid > 0) {
        // still in parent process
        printf ("Parent: %d forked child %d\n", getpid() , cpid);
        int wstatus;
        // wait blocking for child to terminate
        pid_t wpid = waitpid(cpid, &wstatus, 0);
        if (wpid == -1) PERROR_AND_EXIT ("waitpid");
        printf("Parent: child %d exited with %d (status=0x%x) \n", cpid, WEXITSTATUS(wstatus), wstatus);
        exit(EXIT_SUCCESS);
    } else {
        // in child process
        printf("Child: %d forked by parent %d\n", getpid(), getppid()) ;
        sleep(3);
        exit(123);
```

```
    }
}
```

Load Image

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

#define PERROR_AND_EXIT(M) do { perror(M); exit(EXIT_FAILURE); } while(0)

int main () {
    pid_t cpid = fork();
    if (cpid == -1) PERROR_AND_EXIT("fork");
    if (cpid > 0) {
        // still in parent process
        printf("Parent: %d forked child %d\n", getpid() , cpid);
        int wstatus;
        // wait blocking for child to terminate
        pid_t wpid = waitpid(cpid, &wstatus, 0);
        if (wpid == -1) PERROR_AND_EXIT ("waitpid");
        printf("Parent: child %d exited with %d (status=0x%x) \n", cpid, WEXITSTATUS (wstatus) , wstatus) ;
        exit (EXIT_SUCCESS) ;
    } else {
        // in child process: replace current image by new image
        // argv of the execv image below
        static char *eargv[] = { "ls", "-l", NULL } ;
        if (execv("/bin/ls", eargv) == -1) {
            PERROR_AND_EXIT ("execv: /bin/ls");
        }
        // this line is never reached
    }
}
```

execl / execv

execl(path, ... char arguments)

execv(path, char arguments[])

```
pid_t pid;
int status;
pid = fork();
switch (pid) {
  case -1:
    perror("Could not fork");
    break;
  case 0:
    retval = execl("./ChildProc.e", "ChildProc.e", "argument", NULL);
    if (retval < 0) perror("execl not successful");
    // child process is now replaced
    break;
  default:
    printf("Parent with pid ", getpid());
    printf("knows child pid", pid);
    wait(&status);
    break;
}

// ChildProc.c
int main(int argc, char *argv[]) {
    if (argv[1] == NULL) {
        printf("argument missing\n");
        exit(-1);
    }
}
```

System

Runs program and waits for termination. Returns exit code.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>

#define PERROR_AND_EXIT(M) do { perror(M); exit (EXIT_FAILURE); } while(0)

int main () {
    int ret = system("/bin/ls -1");
```

```
    printf("Exited with %d (status=0x%x) \n", WEXITSTATUS (ret), ret);
    return EXIT_SUCCESS;
}
```

Popen

Allows to read/write stdin and stdout of process.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

#define PERROR_AND_EXIT(M) do { perror(M); exit(EXIT_FAILURE); } while(0)

int main () {
    FILE *df = popen("df -k --output=pcent . 2>/dev/null", "r") ;
    if (!df) PERROR_AND_EXIT("popen: df -k .");
    char line [BUFSIZ], *end = NULL;
    long int used = -1;
    while (fgets (line, BUFSIZ, df) ) {
        used = strtol(line, &end, 10);
        // line is spaces-number%-newline
        if (end && end != line && *end == '%') break;
        used = -1;
    }
    if (pclose(df)) PERROR_AND_EXIT ("pclose () ");
    if (used < 0 || used > 100) {
        errno = ERANGE;
        PERROR_AND_EXIT ("df -k .");
    }
    char *msg
        = used < 60 ? "Plenty of disk space (%d%% available) \n"
        : used < 80 ? "Maybe some future disk space problems (%d%% available) \n"
        : used < 90 ? "Need to clear out files (%d%% available) \n"
        : "You may face soon some severe disk space problems (%d%% available) \n";
    printf(msg, 100-used);
    return EXIT_SUCCESS;
}
```

Pthread with argument

```
#include <pthread.h>

void *ThreadF(void *argument) {
    charArg =  *(char *)argument;
    ...
    fflush(stdout);
    pthread_exit(0);
}

pthread_t thread1;
char argument;

pthr = pthread_create(&thread1, NULL, ThreadF, (void *)&argument);

if (pthr != 0) perror("Could not create thread");

pthread_join(thread1, NULL);
pthread_join(thread2, NULL);
```

Pthread with return value

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <pthread.h>

#define PERROR_AND_EXIT(M) do { perror(M); exit(EXIT_FAILURE) ; } while(0)
#define CHECKED_PTHREAD(C) do { int ret = (C); if (ret) { errno = ret; PERROR_AND_EXIT(#C) ; } } while(0)

void *worker (void *arg) {
    printf("worker\n");
    sleep(3);
    static int ret_value = 123;
    return &ret_value;
}

int main () {
    pthread_t thread;
```

```
    CHECKED_PTHREAD(pthread_create(&thread, NULL, worker, NULL));
    printf("main\n");

    static void *retval;
    CHECKED_PTHREAD(pthread_join(thread, &retval));
    printf("worker retval = %d\n", *((int*)retval));

    exit(EXIT_SUCCESS);
}
```

Detach

pthread_detach = Thread resources are freed after it terminates, not after pthread_join. This means there is no return value and you cannot join anymore.

always either join or detach!

Cancel

```
pthread_cancel(pthread_t thread)
```

Thread ID

```
thread_t id = pthread_self()
```

Wait

- In parent process before child terminates: blocked until child terminates
- In parent process after child terminates: Child is Zombie until wait was called.
- Parent terminates without waiting for child: Child is orphaned and will be adopted by the first started process.

Signal graceful termination

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

if (kill(child_pid, SIGTERM) == -1) PERROR_AND_EXIT("kill (SIGTERM)");
```

Signal Handling

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#define PERROR_AND_EXIT(M) do { perror(M); exit(EXIT_FAILURE) ; } while(0)
static pid_t start_child(int wait_for_signal) {
    pid_t cpid = fork();
    if (cpid == -1) PERROR_AND_EXIT("fork");
     // the parent returns the child pid
    if (cpid > 0) return cpid;
    // one child waits for a signal (pause() until signal)
    if (wait_for_signal && pause() == -1) PERROR_AND_EXIT("pause");
    // the child exits normally
    exit(123);
}
static void wait_for_child () {
    int wsts;
    pid t wpid = wait(&wsts) ; // wait blocking for any child to terminate
    if (wpid == -1) PERROR_AND_EXIT("wait");
    // WIFEXITED gets exit code
    if (WIFEXITED(wsts)) printf("Child %d: exit=%d (status=0x%04X)\n", wpid, WTERMSIG(wsts) , wsts);
    // WIFSIGNALED gets signal value
    if (WIFSIGNALED(wsts)) printf("Child %d: signal=%d (status=0x%04X)\n", wpid, WTERMSIG(wsts) , wsts);
}
int main () {
    // start child that exits with exit code
    pid_t cpid1 = start_child(0);
    // start child that waits for signal to terminate
    pid_t cpid2 = start_child(1);
    printf ("Children started: %d (term with exit), %d (term with signal) \n", cpid1, cpid2);
    // let the children start so that both child processes exist
    sleep (1);
    // tell the child to terminate gracefully
    if (kill(cpid2, SIGTERM) == -1) PERROR_AND_EXIT("kill");
    wait_for_child(); // waits blocking until some child terminates
    wait_for_child(); // waits blocking until some child terminates
}
```

```
printf ("Child %d: exit=%d (status=0x%04X) \n", wpid, WEXITSTATUS (wsts) , wsts) ;



}
```

Custom Signal Handler

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <signal.h>
#define PERROR_AND_EXIT (M) do { perror (M); exit (EXIT_FAILURE) ; } while (0)
static void handler(int sig, siginfo_t *siginfo, void *context) {
    printf("caught (%d): source=%d, this=%d\n", sig, siginfo->si_pid, getpid()) ;
    // raise = send SIGTERM to itself (identical to kill(getpid(), SIGTERM))
    raise(SIGTERM);
}
static void set_handler(int sig, void (*handler) (int, siginfo_t *, void *)) {
    struct sigaction a = { 0 };
    // handler variant with additional signal info signature
    a.sa_flags = SA_SIGINFO;
    // the handler to be registered
    a.sa_sigaction = handler;
    // block all Signals
    if (sigfillset(&a.sa_mask) == -1) PERROR_AND_EXIT ("sigfillset");
    // register handler
    if (sigaction (sig, &a, NULL) == -1) PERROR_AND_EXIT ("sigaction");
}

static void set_default(int sig) {
    struct sigaction a = { 0 };
    a.sa_flags = 0;
    a.sa_handler = SIG_DFL;
    if (sigfillset(&a.sa_mask) == -1) PERROR_AND_EXIT ("sigfillset");
    if (sigaction (sig, &a, NULL) == -1) PERROR_AND_EXIT ("sigaction");
}

static void ignore(int sig) {
    struct sigaction a = { 0 };
    a.sa_flags = 0;
    a.sa_handler = SIG_IGN;
    if (sigfillset(&a.sa_mask) == -1) PERROR_AND_EXIT ("sigfillset");
    if (sigaction (sig, &a, NULL) == -1) PERROR_AND_EXIT ("sigaction");
}

static pid_t start_child() {
    pid_t cpid = fork();
    if (cpid == -1) PERROR_AND_EXIT("fork");
    if (cpid > 0) return cpid; // parent returns cpid
    set_handler(SIGUSR1, handler) ; // child ...
    if (pause() ==- 1) PERROR_AND_EXIT ("pause");
    exit(EXIT_FAILURE);
}

int main () {
    // start child that waits for signal to terminate
    pid_t cpid = start_child();
    printf("parent=%d, child=%d\n", getpid(), cpid);
    // give time to the child to start
    sleep(1);
    // send signal to child
    if (kill(cpid, SIGUSR1) == -1) PERROR_AND_EXIT ("kill");

    int ws;
    if (wait (&ws) == -1) PERROR_AND_EXIT ("wait");
    if (WIFEXITED (ws)) {
        printf("child exit=%d (status=0x%04X)\n", WEXITSTATUS(ws), ws);
    }
    if (WIFSIGNALED (ws)) {
        printf("child signal=%d (status=0x%04X)\n", WTERMSIG(ws), ws);
    }
}
```

Pipes

Both ends of the FIFO pipe have their own file descriptor.

- Anonymous: `pipe(int[2]); fork()
- Named: `mkfifo()`

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

int fd[2];
pipe(fd);
pid_t cpid = fork();

if (cpid > 0) {
    // still in parent process: read from pipe
    close (fd[1]) ;
    read(fd [0], ...);
} else {
    // in child process: write to pipe
    close (fd[0]);
    write(fd[1], ...);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

#define PERROR_AND_EXIT(M) do { perror(M); exit (EXIT_FAILURE) ; } while(0)
#define MESSAGE "blocking pipe example message ... \n"
#define MSIZE 10

int main () {
    int fd[2];
    if (pipe(fd) ==- 1) PERROR_AND_EXIT ("pipe");
    pid_t cpid = fork();
    if (cpid == -1) PERROR_AND_EXIT("fork");
    if (cpid > 0) {
        // still in parent process: read from pipe
        if (close(fd[1]) ==- 1) PERROR_AND_EXIT ("close");
        int n;
        do {
            char msg [MSIZE];
            n = read(fd[0], msg, MSIZE);
            if (n == -1) PERROR_AND_EXIT("read") ;
            write (STDOUT_FILENO, msg, n) ;
        } while (n>0);
        if (close(fd[0]) == -1) PERROR_AND_EXIT ("close");
        if (wait(NULL) ==- 1) PERROR_AND_EXIT("wait");
    } else {
        // in child process: write to pipe
        if (close(fd[0]) == -1) {
            PERROR_AND_EXIT ("close");
        }
        if (write(fd[1], MESSAGE, sizeof(MESSAGE)) == -1) {
            PERROR_AND_EXIT ("write");
        }
    }
}
```

Non-Blocking Pipe

```c
void set_nonblocking(int fd) {
    int flags = fcntl(fd, F_GETFL, 0) ;
    if (flags == -1) PERROR_AND_EXIT ("fcntl");
    if (fcntl(fd, F_SETFL, flags | O_NONBLOCK) == -1) PERROR_AND_EXIT("fcntl");
}
int pfd[2];
if (pipe(pfd) ==- 1) PERROR_AND_EXIT("pipe");
set_nonblocking(pfd[0]); // set reading file descriptor to non-blocking

...
int n = read(pfd[0], buf, MSGSIZE);
if (n > 0) // use received data
else if (n == 0) // close and stop polling
else if (errno == EAGAIN) // has to wait for poll interval - avoid spin-lock
else // real error
```

Named Pipe

```c
#include <sys/types.h
#include <sys/stat.h>
int mkfifo(const char *pathname, mode_t mode);
```

Multiple Readers/Writers

```c
#include <sys/types.h>
#include <unistd.h>
#define MSIZE 100

char msg [MSIZE];
struct mq_attr a = { .mq_maxmsg = 10, .mq_msgsize = MSIZE };
q = mq_open("/my-queue", O_CREAT | O_RDWR, 0666, &a) ; // queue name must start on a
pid_t cpid = fork();

if (cpid > 0) {
    // still in parent process: read from queue
    ...
    mq_receive(q, msg, MSIZE, NULL) ;
    ...
    mq_close (q)
} else {
    // in child process: write to queue
    mq_send(q, msg, MSIZE, prio);
    ...
}
```

Posix Queue

```c
#include <mqueue.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <sys/wait.h>
#define PERROR_AND_EXIT(M) do { perror(M); exit(EXIT_FAILURE); } while(0)
#define QNAME "/demo" // the name must start on a slash
#define MSIZE 10
int main () {
    int q = 0, cpid = 0, n = 0, wpid = 0;
    struct mq_attr a = { .mq_maxmsg = 10, .mq_msgsize = MSIZE };
    if ((q = mq_open(QNAME, O_CREAT|O_RDWR|O_NONBLOCK|O_EXCL, 0666, &a) ) == -1) {
        PERROR_AND_EXIT("mq_open");
    }
    if ((cpid =fork()) ==- 1) {
        PERROR_AND_EXIT("fork");
    }
    if (cpid > 0) { // parent: shares queue descriptor with child
        if (mq_unlink (QNAME) ==- 1) {
            PERROR_AND_EXIT("mq_unlink"); // remove it from the filesystem again
        }
        // buffer allows for final '\0' to allow interpretation as string
        char msg [MSIZE+1];
        while (wpid == 0) {
            // read messages while the child process has not yet terminated
            // poll interval for non-blocking mq_receive() and non-blocking waitpid()
            sleep(1)
            // read while there are messages
            while ((n = mq_receive(q, msg, MSIZE, NULL) ) > 0) {
                msg[n] ='\0'; // prepare for printf
                printf("Message: '%s'\n", msg);
            }
            if (n == -1 && errno != EAGAIN) {
                PERROR_AND_EXIT ("mq_receive"); // non-blocking read handling
            }
            if ((wpid = waitpid(cpid, NULL, WNOHANG)) == -1) {
                PERROR_AND_EXIT("waitpid"); // non-blocking
            }
        }
        if (mq_close(q) ==- 1) PERROR_AND_EXIT("mq_close"); // close when completed
    } else {
        // child: shares queue descriptor with parent
        if (mq_send(q, "Hello", sizeof("Hello"), 1) ==- 1) {
            PERROR_AND_EXIT("mq_send");
        }
        sleep(2);
        if (mq send(q, "Queue", sizeof ("Queue"), 1) ==- 1) {
            PERROR_AND_EXIT("mq_send");
        }
    }
}
```

Sockets

Unix Domain Socket

```c
#ifndef _DEFS_H_
#define _DEFS_H_
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#define SRV_SOCK "/tmp/my-server-socket. sock"
#define CLI_SOCK "/tmp/my-client-socket.sock"
#define BUFFER_SIZE 8192
#define ERR() do { perror(""); exit(1); } while (0)
#endif // _DEFS_H_

// server.c
#include "defs.h"
int main (int argc, const char* argv[]) {
    // server: connection-less UNIX socket
    int fd = socket(AF_UNIX, SOCK_DGRAM, 0);
    if (fd < 0) ERR();
    struct sockaddr_un a = { AF_UNIX, SRV_SOCK };
    unlink(SRV_SOCK) ;
    if (bind(fd, (void*) &a, sizeof(a)) < 0) ERR();
    while (1) {
        char buf[BUFFER_SIZE] = { 0 };
        struct sockaddr_un client = { 0 };
        socklen_t client_len = sizeof(client);
        int n = recvfrom(fd, buf, sizeof(buf), 0,
            (void*) &client, &client_len);

        if (n > 0) printf("%s\n", buf);
        if (n < 0) ERR();
    }
}

}

// client.c
#include "defs.h"
int main(int argc, const char* argv[]) {
    // client: connection-less UNIX socket
    int fd = socket(AF_UNIX, SOCK_DGRAM, 0);
    if (fd < 0) ERR();
    struct sockaddr_un a = { AF_UNIX, CLI_SOCK } ;
    unlink(CLI_SOCK) ;
    if (bind(fd, (void*) &a, sizeof(a))<0) ERR();
    for(int i = 1; i < argc; i++) {
        const char *p = argv[i];
        int len = strlen (p)+  1;
        struct sockaddr_un s = { AF_UNIX, SRV_SOCK } ;
        int n = sendto (fd, p, len, 0,
            (void*) &s, sizeof(s));

        if (n < 0) ERR();
    }
    close(fd);
    unlink(CLIENT);
}
```

TCP Client

```c
int client_connect(const char* ServerName, const char* PortNumber) {
    struct addrinfo hints, *server_info, *p;
    memset(&hints, 0, sizeof(hints));

    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;

    int status = getaddrinfo(ServerName, PortNumber, &hints, &server_info);

    if (status != 0) {
      ExitOnError(status, "getaddrinfo");
    }

    int sockfd;
    for (p = server_info; p != NULL; p = p->ai_next) {
        if ((sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1)
        {
             ExitOnError(-2, "socket error");
             continue;
        }
    if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
```

```
        close(sockfd);
        ExitOnError(-3, "connect error");
        continue;
    }
      break; // exit loop if connected successfully
    }

    if (p == NULL) {
        fprintf(stderr, "Could not connect to server\n");
        return -1;
    }

    return sockfd;
}

int receiveResponse(int communicationSocket, char* buffer, int len) {
    int totalReceived = 0;

    while (totalReceived < len) {
        int bytesReceived = recv(
            communicationSocket,
            buffer + totalReceived,
            len - totalReceived,
            RECEIVE_TIMEOUT
        );
        if (bytesReceived == -1) {
            perror("error in recv");
            break;
        }
        totalReceived += bytesReceived;
    }

    return totalReceived;
}

void sendRequest(int communicationSocket, char* buffer, int len) {
    int totalSent = 0;

    while (totalSent < len) {
        int bytesSent = send(
            communicationSocket,
            buffer + totalSent,
            len - totalSent,
            SEND_TIMEOUT
        );

        if (bytesSent == -1) {
            perror ("Error in send");
            break;
        }
        totalSent += bytesSent;
    }
}
```

TCP Server

```
void server_init(char * portNumber) {
    struct addrinfo hints, *server_info, *p;

    memset(&hints, 0, sizeof(hints));

    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;

    int status = getaddrinfo(NULL, portNumber, &hints, &server_info);
    if(status != 0){
        ExitOnError(status, "getaddrinfo");
    }

    int sockfd;
    for (p = server_info; p != NULL; p = p->ai_next) {
        if ((sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1)
        {
             perror("socket error");
             continue;
        }

        if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
             close(sockfd);
             perror("connect error");
             continue;
        }
```

```
        break; // exit when connection successful
    }

    ListeningSocket = sockfd;

    listen(ListeningSocket, MAX_QUEUE);
}

int getRequest(char* requestBuffer, int max_len)
{
    int bytesReceived = recv(
        connectedSocket,
        requestBuffer,
        max_len -1,
        RECEIVE_TIMEOUT
    );
    requestBuffer[bytesReceived] = '\0';
    return bytesReceived;
}

void sendResponse(char* response, int resp_len) {
    send(connectedSocket, response, resp_len, 0);
}

void server_close_connection(void) {
    close(connectedSocket);
}
```

Synchronisation

Race Condition = The behavior depends on the order that two tasks are done. This can be prevented with

- Blocking
- Signaling
- Waiting
- Temporal exclusive access to a critical section

Mutex

10_SNP_Sys_Thread_Synchronisation-mit-Notizen

```
#include <sys/types.h>
#include <pthread. h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#define FATAL(M) do { perror(M); exit(EXIT_FAILURE) ; } while (0)
// for the demo: increases the chance that a context switch happens at an undesired place
#define N 100000000

volatile int value = 0; // shared variable: read and written in both threads
pthread mutex_t mutex;
void * count(void *arg) {
    int delta = *(int*)arg; // increment or decrement (arg is +1 or -1)
    for(int i = 0; i < N; i++) {
        if (pthread_mutex_lock(&mutex) != 0) FATAL("lock");
            int a = value; // read shared variable
            a += delta; // modify
            value = a; // write shared variable
            if (pthread mutex_unlock(&mutex) != 0) FATAL("unlock");
        }
    return NULL;
}

int main (void) {
    if (pthread mutex_init(&mutex, NULL) != 0) FATAL("mutex_init");
    pthread_t th_inc;
    pthread_t th_dec;

    int inc = +1;
    int dec =- 1;
    if (pthread_create(&th_inc, NULL, count, &inc) != 0) FATAL("create") ;
    if (pthread_create(&th_dec, NULL, count, &dec) != 0) FATAL("create");

    if (pthread_join(th_inc, NULL) != 0) FATAL("join");
    if (pthread_join(th_dec, NULL) != 0) FATAL("join");

    // N times increment and N tmies decrement is expected to result in value 0 again
```

```
    if (value != 0) fprintf(stderr, "ERROR: exp-$d, act-%d\n", 0, value);
}
```

Rekursive Locks

```
#include <sys/types.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#define FATAL(M) do { perror(M); exit(EXIT_FAILURE); } while (0)
#define N 10000000
#define MUTEXATTR PTHREAD_MUTEX_RECURSIVE
pthread_mutexattr_t mutex_attr;
pthread mutex_t mutex;
int value = 0;
void calc(int step) {
    if (pthread_mutex_lock(&mutex) != 0) FATAL("inner lock");
    value += step;
    if (pthread_mutex_unlock (&mutex) != 0) FATAL("inner unlock");
}
void *count(void *p) {
    for(int i= 0; i < N; i++) {
        if (pthread_mutex_lock(&mutex) != 0) FATAL("lock");
        calc(*(int*)p);
        if (pthread_mutex_unlock(&mutex) !- 0) FATAL("unlock");
}
int main (void) {
    if (pthread mutexattr_init(&mutex_attr) != 0) {
        FATAL("mutexattr_init");
    }
    if (pthread mutexattr_settype(&mutex_attr, MUTEXATTR) != 0) {
        FATAL("mutexattr_set");
    }
    if (pthread_mutex_init (&mutex, &mutex_attr) != 0) {
        FATAL("mutex_init");
    }
    pthread_t th_inc;
    pthread_t th_dec;
    int inc = +1;
    int dec =- 1;
    if (pthread_create(&th_inc, NULL, count, &inc) != 0) FATAL ("create");
    if (pthread_create(&th_dec, NULL, count, &dec) != 0) FATAL("create");
    if (pthread_join(th_inc, NULL) != 0) FATAL("join");
    if (pthread_join (th_dec, NULL) != 0) FATAL("join");
    if (value != 0) fprintf(stderr, "ERROR: exp=%d, act=%d\n", 0, value);
}
```

Signalization with Semaphores

Unnamed, Between Threads

sem_t sem;	**semaphore instance**
sem_init (&sem, 0, initial_value);	init memory semaphore with initial count
sem_wait (&sem);	wait opertation
sem_post (&sem);	signal operation
sem destroy (&sem);	cleanup

Named, Between Processes

sem_t *sem;	**semaphore instance address**
sem = sem_open("/name", O_CREAT, mode=0700, value);	init IPC semaphore
sem_wait(sem);	wait opertation
sem_post(sem);	signal operation
sem_close(sem);	cleanup
sem_unlink("/name");	remove file

- 0 = tasks wait from beginning
- 1 = 1 tasks can pass
- post = +1
- wait = -1 when available

Unnamed Sephamore

```c
#include <sys/types.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <limits.h>
#include <semaphore.h>
#define FATAL (M) do { perror (M); exit(EXIT_FAILURE); } while(0)
#define CHECK(E,A,M) if((E) == (A));else fprintf(stderr,"ERROR: "M": exp=%d, act=%d\n",E????
#define N 10000

// shared variable: init in one thread, then use in both
volatile int array [N] = { 0 };
sem_t sem;
// initialize the data and calculate min value of all
void *min(void *arg) {
    for(int i = 0; i < N; i++) {
        // init the shared data -N/2. . .N-1-N
        array[i] = i - N/2;
    }
    if (sem_post(&sem) == -1) FATAL("post");
    int value = INT_MAX;
    for(int i = 0; i < N; i++) {
        if (value > array[i]) value = array[i];
    }
    CHECK(-N/2, value, "wrong min value");
    return NULL;
}
// calculate max value of already initialized data
void *max(void *arg) {
    if (sem_wait(&sem) == -1) FATAL("wait");
    int value = INT_MIN;
    for(int i = 0; i < N; i++) {
        if (value < array[i]) value = array[i];
    }
    CHECK(N-1-N/2, value, "wrong max value");
    return NULL;
}
int main (void) {
    if (sem_init(&sem, 0, 0) == -1) FATAL("sem");
    pthread_t th_max;
    pthread_t th_min;
    if (pthread_create(&th_max, NULL, max, NULL) != 0) FATAL("create");
    if (pthread_create(&th_min, NULL, min, NULL) != 0) FATAL("create");
    if (pthread_join(th_max, NULL) != 0) FATAL("join");
    if (pthread_join(th_min, NULL) !- 0) FATAL("join");
}
```

Named Sephamore

P08 → basicSequence

Action needed 3 Times → Wait 3 Times

Avoid Deadlock with Fixed Order

```c
int first = fromB < toB ? fromB : toB;
int second = fromB > toB ? fromB : toB;
```

Producer-Consumer Problem

Synchronized

```c
// Producer
while (1) {
    item = produce_item() ;
    // blocks until space left for item
    sync_insert(sync_fifo, item) ;
}

// Consumer
while (1) {
// blocks until one item available
item = sync_get (sync_fifo) ;
consume_item(item) ;

}
```

Unsynchronized

```
Semaphore space_left = capacity(fifo);
Semaphore space_used = 0;
Mutex mutex;
// Producer
while (1) {
    item = produce_item();
    // blocks until space left for item
    wait(space_left); //block or dec
    lock(mutex);
    insert(fifo, item);
    unlock(mutex);
    post(space_used); // signal or inc
}
// Consumer
while (1) {
    // blocks until one item available
    wait(space_used); // block or dec
    lock(mutex);
    item = get(fifo);
    unlock(mutex);
    post(space_left); // signal or inc
    consume_item(item);
}
```

Read-Write Problem

Multiple readers allowed but only one writer. Use seperate ReadLock and WriteLock. pthread_rwlock_t

GCC Snippets

gcc -o name name.c → ./name

gcc name.c -o name

gcc -o name main.c other.c

-lm → link libraries like <math.h>

Make Snippets

- `$<` means "the first dependency" (i.e., the `.c` file)
- `$@` would mean "the target" (i.e., the `.o` file)

```
# Default target: builds 'rechner'
all: rechner

# Linking step: combine all .o files into executable
rechner: add.o sub.o mul.o div.o main.o
    gcc -o rechner add.o sub.o mul.o div.o main.o

# Cleanup: remove all .o files and the executable
clean:
    rm -f *.o rechner

# --- Explicit compilation rules for each file ---
# Note: Defines dependencies AND commands for each .o file.
# 'Makefile' is included as a dependency to force recompile
# if the Makefile itself changes (e.g., flags are modified).

add.o: add.c def.h Makefile
    gcc -c add.c

sub.o: sub.c def.h Makefile
    gcc -c sub.c

mul.o: mul.c def.h Makefile
    gcc -c mul.c

div.o: div.c def.h Makefile
    gcc -c div.c

main.o: main.c def.h Makefile
    gcc -c main.c
```

```
CC = gcc
CFLAGS = -Wall -g
TARGET = program
SOURCES = main.c functions.c
```

```
OBJECTS = $(SOURCES:.c=.o)

# Rules
$(TARGET): $(OBJECTS)
    $(CC) -o $(TARGET) $(OBJECTS)

%.o: %.c
    $(CC) $(CFLAGS) -c $<

clean:
    rm -f *.o $(TARGET)
```

PHONY

```
.PHONY default all clean test mytarget
```

Basic Variables

```
TARGET     := bin/triangle
SOURCES    := src/triangle.c src/read.c src/rectang.c
TSTSOURCES := tests/tests.c
LIBS       := -lm
```

Modules

```
TARGET     := bin/tic-tac-toe
MODULES    := src/model.c src/view.c src/control.c
SOURCES    := src/main.c $(MODULES)
TSTSOURCES := tests/tests.c $(MODULES)
```

List from other List (replace a with b)

```
MYFILES := $(SOURCES:%a=%b)
```

Suffix Rule: for .a, generate .b

```
%.b: %.a
    action...
```

Target depending on Lists

```
mytarget: $(MYLIST1) $(MYLIST2)
    echo the target $@ is done
```

Shell Snippets

IO

```
... < file
... > new_file
... 1> new_file
... >> append_to_file
... 2> new_error_file
... >& new_combi_file
... | ... // out to next in
... && ... // only if prev successful
... & // background process
```

Path shortcuts

```
~/ current useres home dir
~name/ names home dir
./ current dir
../ parent dir
* any amount of characters
? one character
[...] one of ...
```

Soft link b referenziert a:

- Verweist auf Pfad der Datei
- Wenn die Zieldatei gelöscht wurde, zeigt der Link ins leere

```
ln -s a b
```

Hard Link

- Verweist direkt auf denselben inode.
- Beide Dateinamen sind gleichwertig (es gibt keine "Originaldatei" mehr)
- Wenn du eine der Dateien löschst, bleibt die andere bestehen, solange noch mindestens ein Hard Link existiert.

```
ln a b
```