# 1. Tafelübung

- Fehlerbehandlung
- Verwendung von qsort
- Dynamische Speicherverwaltung
- Arbeiten mit Zeichen und Zeichenketten
- Variablen: Gültigkeitsbereich und Lebensdauer




# Fehlerbehandlung

- Fast jeder Systemcall kann fehlschlagen
  - ◆ Fehlerbehandlung unumgänglich!
- Vorgehensweise:
  - ◆ Rückgabewerte von Systemcalls abfragen
  - ◆ Im Fehlerfall (meist durch Rückgabewert -1 angezeigt): Fehlercode steht in der globalen Variable **`errno`**
- Fehlermeldung kann mit der Funktion **`perror`** auf die Standardausgabe ausgegeben werden:

```
#include <errno.h>
void perror(const char *s);
```




# Einlesen von Zeichen und Zeichenketten

- Funktionen zum Einlesen von Zeichen und Zeichenketten
  - ◆ Zeichenweise:

```
int getc(FILE *stream);
int getchar(void);
```

  - ◆ Zeilenweise:

```
char *gets(char *s);
char *fgets(char *s, int n, FILE *stream);
```

  - ◆ Allgemein:

```
int scanf(const char *format, ...);
```

  - ◆ Standardkanäle:

```
stdin, stdout, stderr
```




# Sortieren mittels qsort

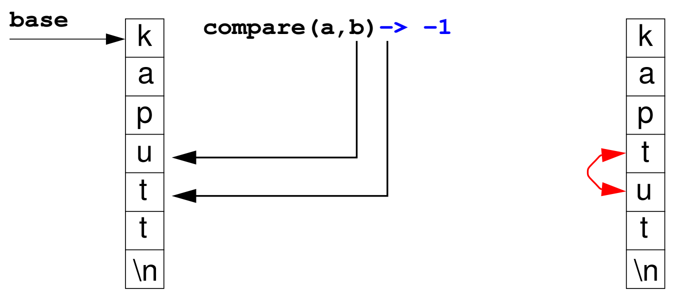
- Prototyp aus **`stdlib.h`**:

```
void qsort(void *base,
       size_t nel,
       size_t width,
       int (*compare) (const void *, const void *));
```

- Bedeutung der Parameter:
  - ◆ **base** : Zeiger auf das erste Element des Feldes, dessen Elemente sortiert werden sollen
  - ◆ **nel** : Anzahl der Elemente im zu sortierenden Feld
  - ◆ **width**: Größe eines Elements
  - ◆ **compare**: Vergleichsfunktion

## Sortieren mittels qsort (2)

- **qsort** vergleicht je zwei Elemente mit Hilfe der Vergleichsfunktion compare
- sind die Elemente zu vertauschen, dann werden die entsprechenden Felder komplett ausgetauscht, z.B.:







## Vergleichsfunktion

- Die Vergleichsfunktion erhält Zeiger auf Feldelemente, d.h. die übergebenen Zeiger haben denselben Typ wie das Feld
- Die Funktion vergleicht die beiden Elemente und liefert:
  - -1, falls Element 1 kleiner bewertet wird als Element 2
  - 0, falls Element 1 und Element 2 gleich gewertet werden
  - 1, falls Element 1 größer bewertet wird als Element 2
- Beispiel:
  - compare('z','a') -> 1
  - compare(1,5) -> -1
  - compare(5,5) -> 0




## Vergleichsfunktion (2)

- Problem: qsort erwartet folgenden Funktionstyp:

```
int (*compar) (const void *, const void *)
```

- Lösung: "casten"
  - innerhalb der Funktion, z.B. (Feld vom Typ char **):

```
int compare(const void *a, const void *b) {
       return strcmp(*((char **)a), *((char **)b));
}
```

  - beim qsort-Aufruf:

```
int compare(char **a, char **b);
...
qsort( field, nel, sizeof(char *),
       (int (*)(const void *, const void *))compare);
```




## Dynamische Speicherverwaltung

- Prototypen aus **stdlib.h**

```
void *malloc(size_t size);
void free(void *ptr);
```

- Beispiel:

```
typedef struct {
    char    *name;
    int     alter;
} person_t;

person_t *person;
person = (person_t *)malloc(sizeof(person_t));
if(person == NULL) {
    perror("malloc of person failed");
    ...
}
```

# Dynamische Speicherverwaltung (2)

- Erzeugen von Feldern der Länge n:
  - mittels: `void *malloc(size_t size)`

```
person_t *personen;
personen = (person_t *)malloc(sizeof(person_t)*n);
if(personen == NULL) ...
```

  - mittels: `void *calloc(size_t nelem, size_t elsize)`

```
person_t *personen;
personen = (person_t *)calloc(n, sizeof(person_t));
if(person == NULL) ...
```

  - `calloc` initialisiert den Speicher mit `0`
  - `malloc` initialisiert den Speicher nicht
  - explizite Initialisierung mit `void *memset(void *s, int c, size_t n)`

```
memset(personen, 0, sizeof(person_t)*n);
```




# Dynamische Speicherverwaltung (3)

- Verlängern von Felder, die durch malloc bzw. realloc erzeugt wurden:

```
void *realloc(void *ptr, size_t size)
```

```
neu = (person_t *)realloc(personen, (n+10) * sizeof(person_t));
if(neu == NULL) ...
```




# Arbeiten mit Zeichenketten

- Initialisierung:

```
char *str         = "Eine Zeichenkette";
char str2[]       = "Noch eine Zeichenkette";
char str3[10]     = "Ein Test";
char str4[4]      = "eindeutig zu lang";
char schlecht[]   = {'a','.','o','u','t'};
char besser[]     = {'a','.','o','u','t','\0'};
char einfacher[]  = "a.out";
```

- Kopieren

```
char *src = "Hello World";
char *dst = malloc(strlen(src) + 1);
strcpy(dst, src);
```

  - oder

```
char *dst = strdup(src);
```




# Variablen - Gültigkeitsbereich und Lebensdauer

- Lokale Variablen

```
void test() {
    int a;
    a = 42;
}
```

- Modul-lokale Variablen

```
static int a;
void test() {
    a = 42;
}
```

- Globale Variablen

test.h

```
extern int a;
```

test.c

```
#include "test.h"
int a;
void test() {
    a = 42;
}
```

hallo.c

```
#include "test.h"

void hallo() {
    a = 13;
}
```