Übungen zu

GdI 2 — Systemnahe Programmierung in C

Sommer 2007

SPiC - Ü

U1 1. Übung

U1-1 UNIX/Linux Benutzerumgebung

- Spezielle Projektverzeichnisse zur Bearbeitung der Übungsaufgaben
  - `/proj/i4gdi/`*login-Name*
  - Wechseln des Arbeitsverzeichnisses dorthin (cd = change directory):
    `cd /proj/i4gdi/`*login-Name*
  - Einrichten eines Verzeichnisses für Aufgabe0:
    `mkdir aufgabe0`
  - Anzeigen der Dateien in einem Verzeichnis:
    aktuelles Verzeichnis anzeigen: `ls` oder `ls -l`
    Verzeichnis `aufgabe1` anzeigen: `ls -l aufgabe1`
- Kommandointerpreter (Shell)
  - Programm, das Kommandos entgegennimmt und ausführt
  - verschiedene Varianten, am häufigsten unter Linux: bash oder tcsh

U1-1 UNIX/Linux Benutzerumgebung (2)

- Sonderzeichen
  - einige Zeichen haben unter UNIX besondere Bedeutung
  - Funktionen:
    - Korrektur von Tippfehlern
    - Steuerung der Bildschirm-Ausgabe
    - Einwirkung auf den Ablauf von Programmen
- Übersicht: (<CTRL> = <STRG>)

<BACKSPACE>	letztes Zeichen löschen (manchmal auch **<DELETE>**)
<CTRL> U	alle Zeichen der Zeile löschen
<CTRL>C	Interrupt - Programm wird abgebrochen
<CTRL>Z	Stop - Programm wird gestoppt (mit `fg` läuft's weiter)
<CTRL>D	End-of-File
<CTRL>S	Ausgabe am Bildschirm wird angehalten
<CTRL>Q	Ausgabe am Bildschirm läuft weiter

U1-1 UNIX/Linux Benutzerumgebung (3)

- Editor
  - verschiedene Editoren unter UNIX verfügbar
    - vi
    - emacs
  - für unerfahrene Benutzer zu empfehlen: **`kate`**
  - Starten
    - durch Eingabe von `kate` in einer Shell
    - oder über Auswahlmenü von KDE

U1-1 UNIX/Linux Benutzerumgebung (4)

- Manual: man-pages
  - ➤ Aufgeteilt nach verschiedenen *Sections*
  - (1) Kommandos
  - (2) Systemaufrufe
  - (3) Bibliotheksfunktionen
  - (5) Dateiformate (spezielle Datenstrukturen, etc.)
  - (7) verschiedenes (z.B. Terminaltreiber, IP, …)
- man-Pages werden normalerweise mit der Section zitiert: **`printf(3)`**
- Aufruf unter Linux

```
man [section] Begriff

z.B. man 3 printf
```

- Suche nach Sections: **`man -f Begriff`**
  Suche von man-Pages zu einem Stichwort: **`man -k Stichwort`**

U1-2 Erläuterung zu Aufgabe 0 und 1

- spezielle Aufrufoptionen des Compilers

  `gcc -pedantic` liefert Warnungen in allen Fällen, die nicht 100% dem ANSI-C-Standard entsprechen

  ... -Wall liefert in vielen evtl. zweifelhaften Situationen (die zwar korrekt sein könnten, aber häufig nicht sind) Warnungen

  diese Optionen führen zwar oft zu nervenden Warnungen, helfen aber auch dabei, Fehler schnell zu erkennen:

  Empfehlung: `gcc -pedantic -Wall -o aufgabe1 aufgabe1.c`

- Beispiel zur Verwendung von getchar() siehe Vorlesungsfolien

- Um getchar verwenden zu können, muss eine spezielle include-Datei eingebunden werden

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

  am Anfang des Programms eingeben

U2 2. Übung

Aufgabe 1

U2-1 Direktory-Struktur für die Aufgaben

- Homedirectory Theo Tester (nach login aktuelles Directory):
  `/home/cip/nf/sithtest`
- Projektdirectory (Theo Tester): `/proj/i4gdi/sithtest`
  - ➤ Umschalten des aktuellen Directory mit Kommando **`cd`** (change Directory)
    `cd /proj/i4gdi/sithtest`
- Anlegen von Unterdirectories für die Aufgaben
  - ➤ mkdir aufgabe1
  - ➤ erzeugt Directory `/proj/i4gdi/sithtest/aufgabe1`
- aktuelles Directory dorthin wechseln
  `cd aufgabe1`

U2-1 Direktory-Struktur für die Aufgaben

- wo bin ich gerade?
  - Kommando **`pwd`** (print working directory)
    - ➤ `pwd`
      `/proj/i4gdi/sithtest/aufgabe1`
  - Promptsymbol in der Shell anpassen (Rechnername:Directory Kommando-Nr)
    - ➤ tcsh
      `set prompt="%S%M:%~ %h%s "`
    - ➤ bash
      `PS1="\h:\w \# "`

U2-2 Erkennen des Dateiendes

- Eingabe von der Tastatur
  - `Ctrl-D` bzw. `Strg-D` am Zeilenanfang
  - in der Zeilenmitte durch zwei mal `Ctrl-D`
  - Zeichen `Ctrl-D` hat ASCII-Code `0x04`
    EOF ist aber nicht 0x04
    ???
  - Ctrl-D wird vom Tastaturtreiber des Betriebssystems erkannt
    - ➤ Betriebssystem schließt daraufhin den Eingabekanal für das Programm
    - ➤ beim nächsten Lesebefehl an das Betriebssystem (innerhalb von `getchar( )`) meldet das BS das nichts mehr kommt
    - ➤ getchar liefert daraufhin den Wert EOF (= -1)

U2-2 Erkennen des Dateiendes

- Lesen aus Datei (von der Platte)
  - Eingabekanal z. B. mit `<Eingabedatei` umgeleitet
  - Betriebssystem erkennt wenn das letzte Zeichen aus der Datei gelesen wurde
    - ➤ Betriebssystem schließt daraufhin den Eingabekanal für das Programm
    - ➤ beim nächsten Lesebefehl an das Betriebssystem (innerhalb von `getchar( )`) meldet das BS das nichts mehr kommt
    - ➤ getchar liefert daraufhin den Wert EOF (= -1)

U2-3 wie zählt man Wörter?

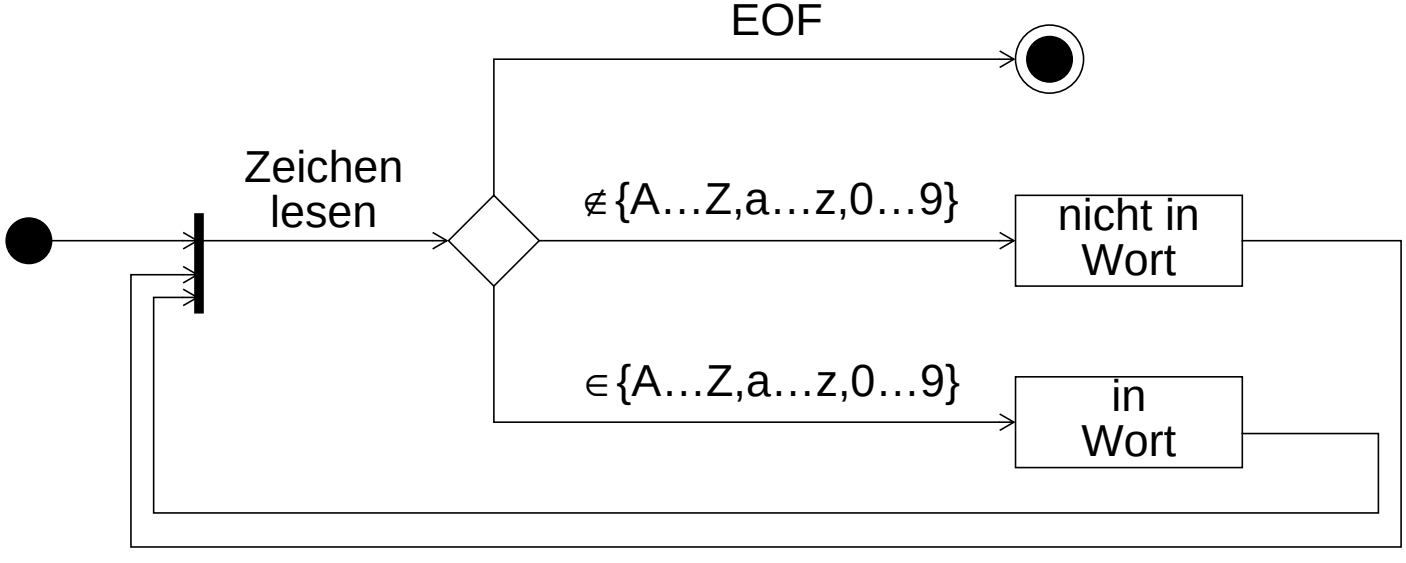
- Wort = Folge von Zeichen aus der Menge {A-Z,a-z,0-9}
- 1. Versuch
  - immer wenn ein anderes Zeichen kommt: worte++
    - Beispiel:
      abc_abc_abc
    - Ergebnis 2 ???
    - Dateiende extra beachten!
    - Ergebnis 3 - ok?
  - was passiert bei der Zeichenfolge
    abc_abc_$%_abc###
    - 8? 9?
    - ????
    - offenbar ein Problem

U2-3 wie zählt man Wörter?

- nur wenn man ein Wort verlässt darf man zählen!

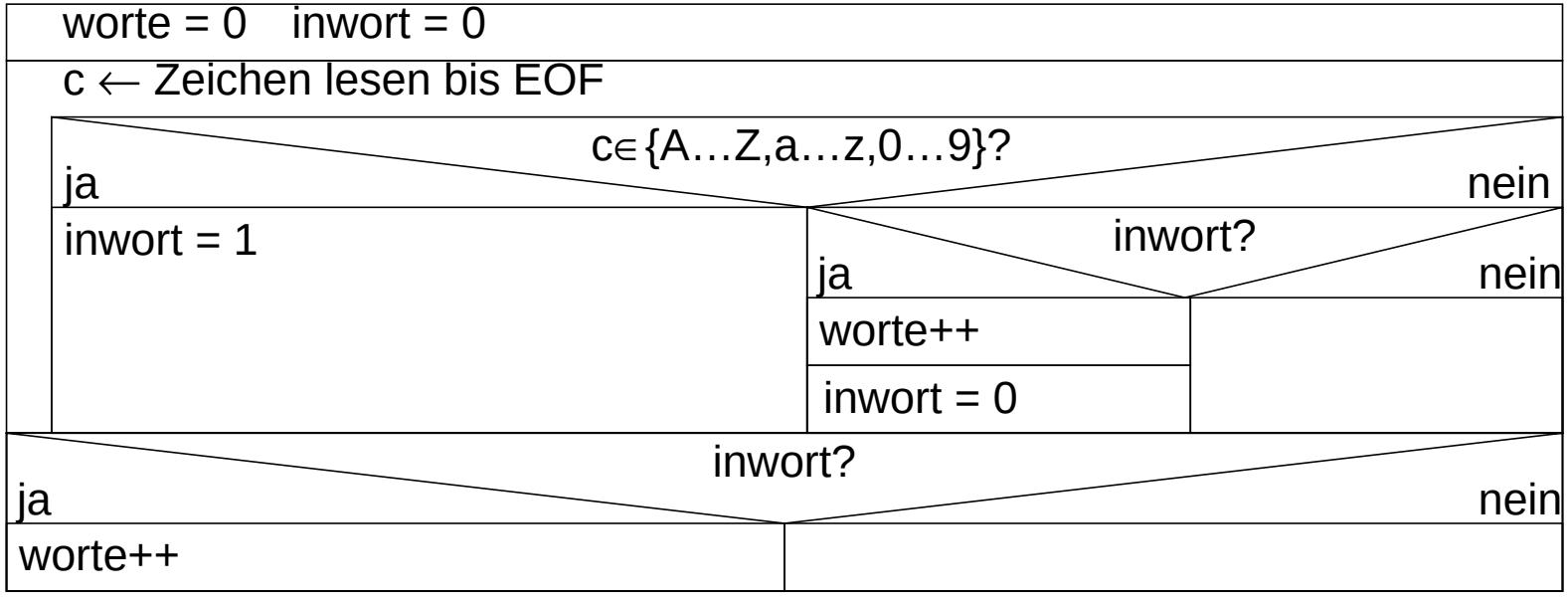
➥ man muss offenbar unterscheiden, ob man gerade in einem Wort war oder nicht

➤ Zustandsautomat



➤ nur beim Übergang "in Wort" -> "nicht in Wort" oder "in Wort" -> Ende darf man zählen!

U2-3 wie zählt man Wörter?



- die Abfrage "c∈{A…Z,a…z,0…9}?" kann man in einem if-Statement formulieren
  - ➤ wie?
  - ➤ man kann Zeichen vergleichen (’A’ < ’B’ !!!)
    (eigentlich werden intern die ASCII-Codes verglichen)
  - ➤ man muss die ASCII-Codes nicht explizit eingeben!

U3 3. Übung

Aufgabe 2

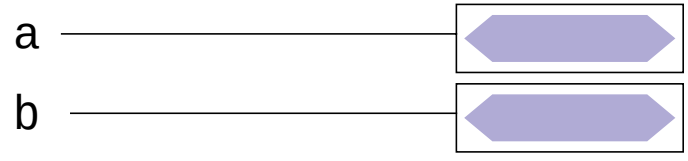
U3-1 einfache swap_double Funktion

- Parameter werden in C *by-value* übergeben
- die aufgerufene Funktion kann den aktuellen Parameter beim Aufrufer nicht verändern
- auch Zeiger werden *by-value* übergeben, d. h. die Funktion erhält lediglich eine Kopie des Adressverweises
- über diesen Verweis kann die Funktion jedoch mit Hilfe des *-Operators auf die zugehörige Variable zugreifen und sie verändern
  - ➥ *call-by-reference*

1 Zeiger als Funktionsargumente

- Beispiel:

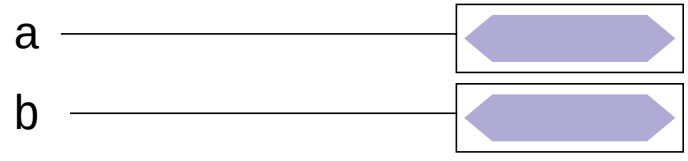
```
void swap (double *, double *);
int main(void) {
double a, b;
...
swap(&a, &b);
}
```



1 Zeiger als Funktionsargumente

- Beispiel:

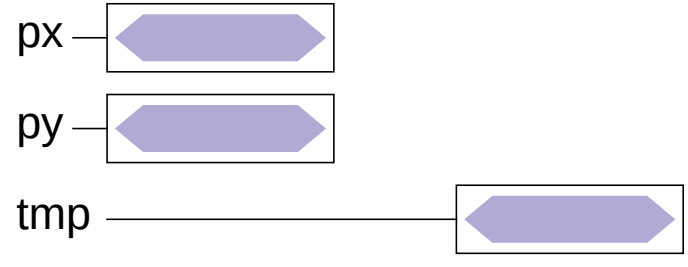
```
void swap (double *, double *);
int main(void) {
double a, b;
...
swap(&a, &b);
}
```




```
void swap (double *px, double *py)
{
     double tmp;

     tmp = *px;
     *px = *py;
     *py = tmp;

}
```



1 Zeiger als Funktionsargumente

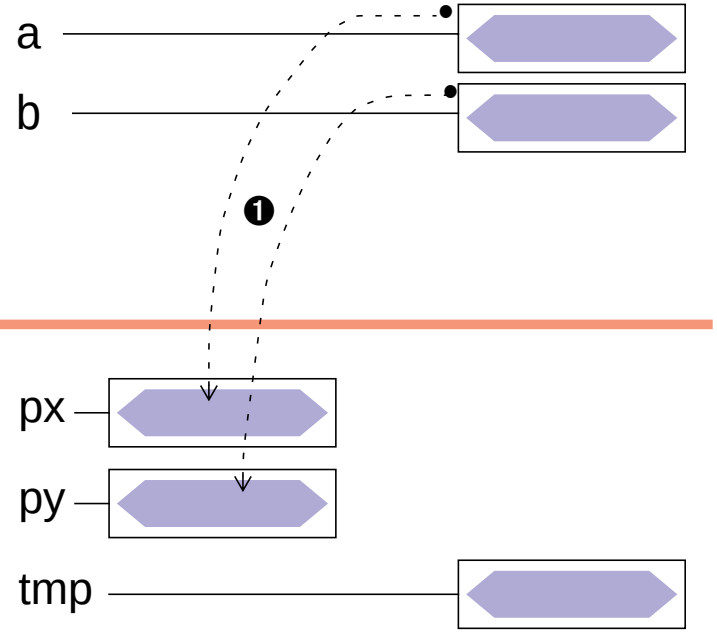
- Beispiel:

```
void swap (double *, double *);
int main(void) {
double a, b;
...
swap(&a, &b); ❶
}
```

```
void swap (double *px, double *py)
{
    double tmp;

    tmp = *px;
    *px = *py;
    *py = tmp;

}
```



1 Zeiger als Funktionsargumente

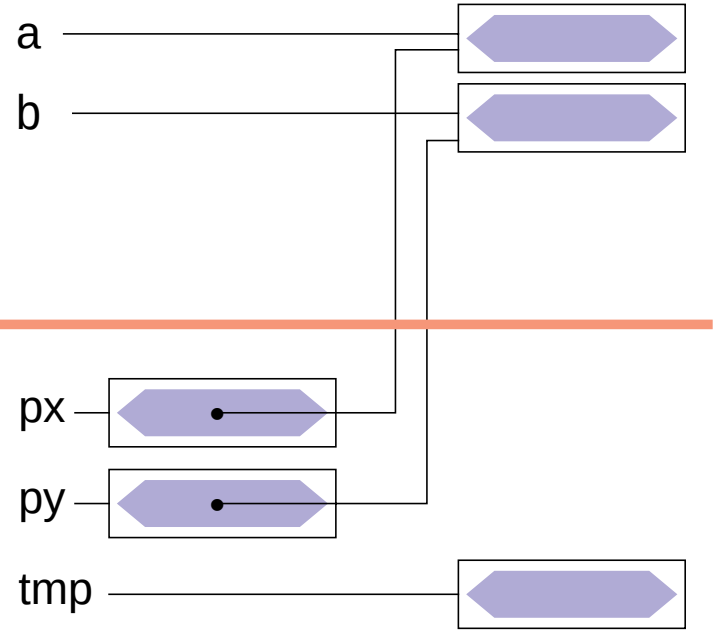
- Beispiel:

```
void swap (double *, double *);
int main(void) {
double a, b;
...
swap(&a, &b);
}
```

```
void swap (double *px, double *py)
{
     double tmp;

     tmp = *px;
     *px = *py;
     *py = tmp;

}
```



1 Zeiger als Funktionsargumente

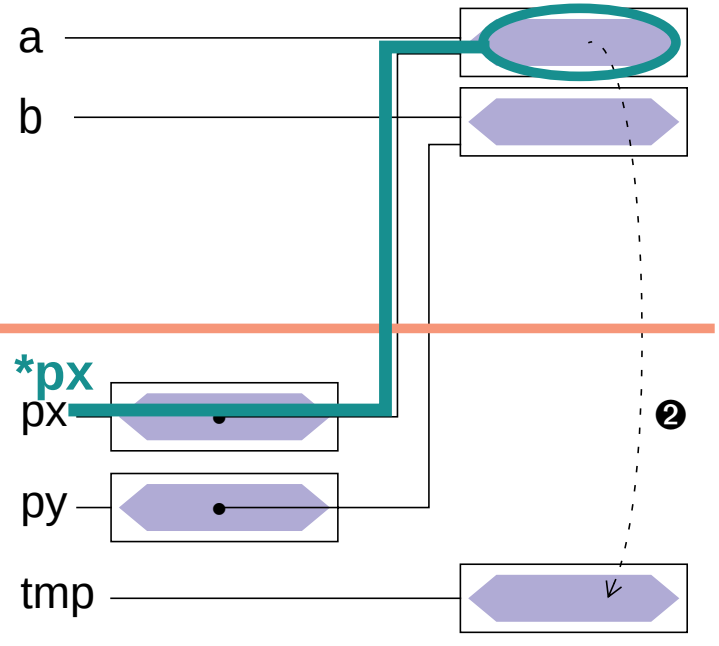
- Beispiel:

```
void swap (double *, double *);
int main(void) {
double a, b;
...
swap(&a, &b); ❶
}
```

```
void swap (double *px, double *py)
{
    double tmp;

    tmp = *px; ❷
    *px = *py;
    *py = tmp;

}
```



1 Zeiger als Funktionsargumente

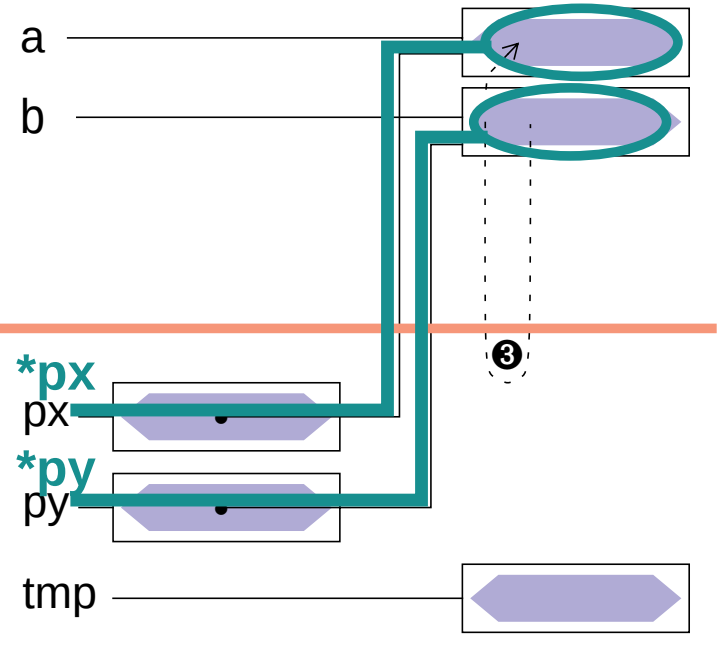
- Beispiel:

```
void swap (double *, double *);
int main(void) {
double a, b;
...
swap(&a, &b); ❶
}
```

```
void swap (double *px, double *py)
{
    double tmp;

    tmp = *px;
    *px = *py; ❸
    *py = tmp;

}
```



1 Zeiger als Funktionsargumente

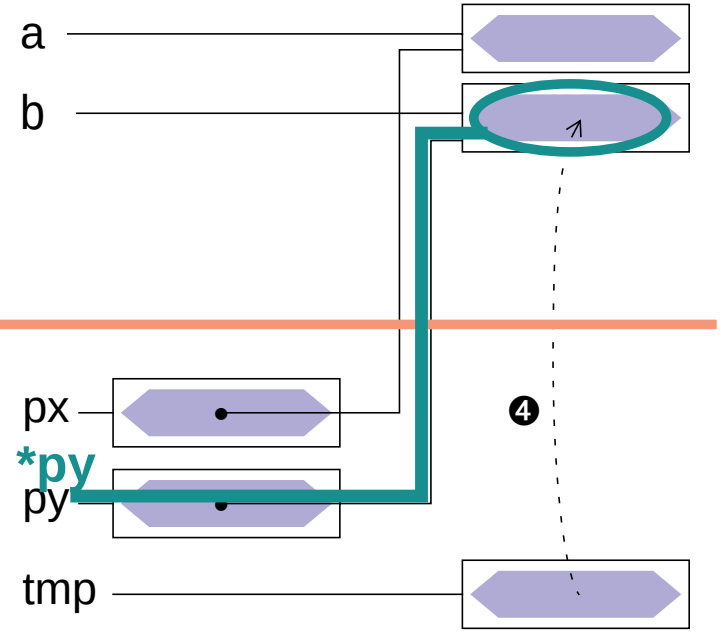
- Beispiel:

```
void swap (double *, double *);
int main(void) {
double a, b;
...
swap(&a, &b); ❶
}
```

```
void swap (double *px, double *py)
{
    double tmp;

    tmp = *px; ❷
    *px = *py; ❸
    *py = tmp; ❹

}
```



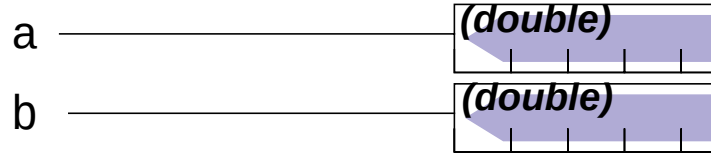
U3-2 generische swap-Funktion

- Funktion soll Zeiger auf beliebigen Datentyp übergeben bekommen
  - ? welchen Typ gibt man dem Parameter
    - ➤ Typ **`(void *)`** = Zeiger auf "irgendetwas"
    - ➤ Schnittstelle der Funktion
      `void swap_generic(void *px, void *py, size_t s)`
  - ? wie benutzt man so einen Zeiger
    - ➤ er kann nicht direkt genutzt werden, weil für das Ergebnis von **`*px`** und **`*py`** der Typ unbekannt ist
      => Programm kann nicht damit umgehen
    - ➤ Lösung: void-Zeiger in einen anderen Zeiger verwandeln
      => cast-Operator
    - ➤ Beispiel: **`char *pa = (char *)px;`**
    - ➤ über **`*pa`** kann nun auf das erste Byte des Speicherbereichs, auf den **`px`** zeigt, zugegriffen werden

1 generische Zeiger als Funktionsargumente

- Beispiel:

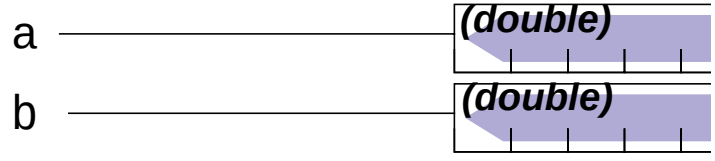
```
main(void) {
double a, b;
void swap_generic(void *, void *, size_t);
...
swap_generic(&a, &b, sizeof(double));
}
```



1 generische Zeiger als Funktionsargumente

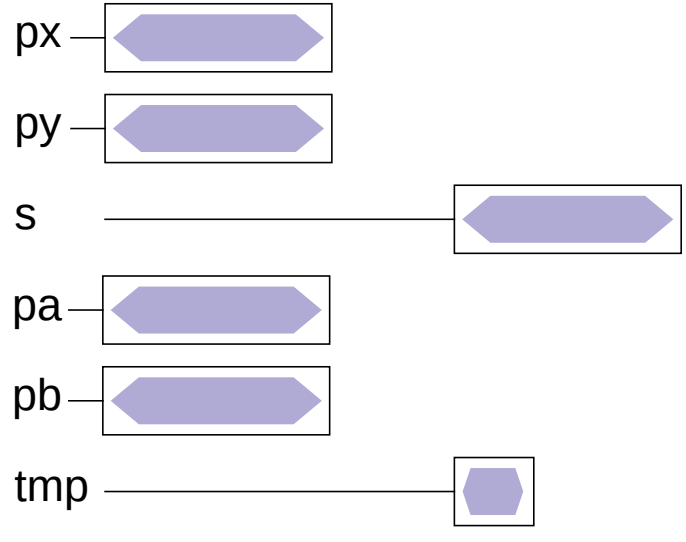
- Beispiel:

```
main(void) {
double a, b;
void swap_generic(void *, void *, size_t);
...
swap_generic(&a, &b, sizeof(double));
}
```




```
void swap_generic(void *px, void *py, size_t s);
{
        char *pa, *pb, tmp;

        pa = (char *)px;
        ...
```



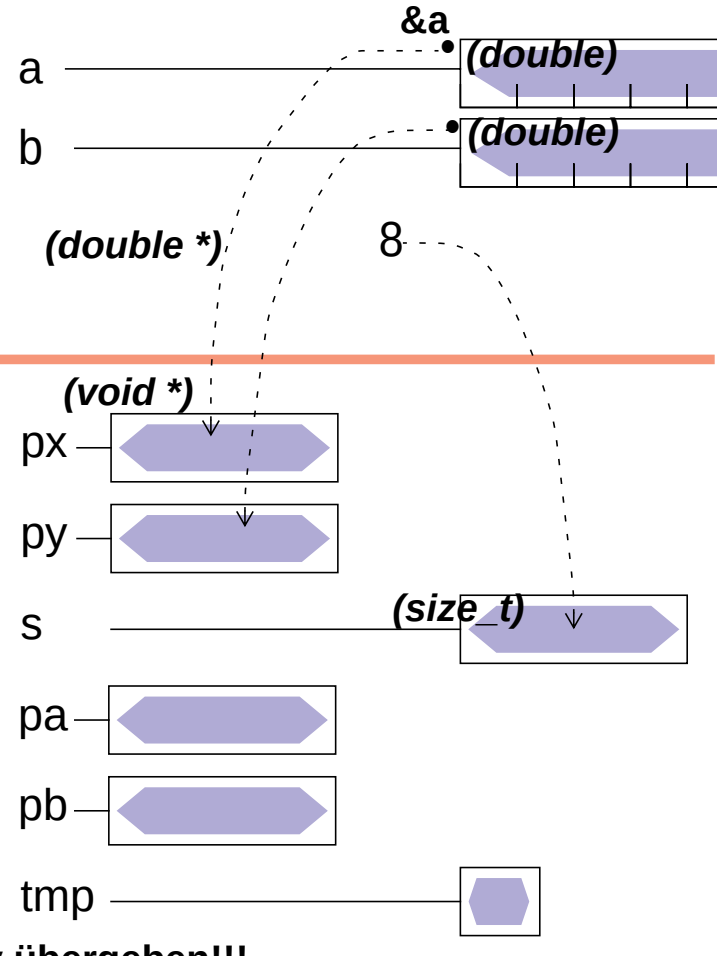
1 generische Zeiger als Funktionsargumente

- Beispiel:

```
main(void) {
double a, b;
void swap_generic(void *, void *, size_t);
...
swap_generic(&a, &b, sizeof(double)); }
```

```
void swap_generic(void *px, void *py, size_t s);
{
        char *pa, *pb, tmp;

        pa = (char *)px;
        ...
```



double-Zeiger &a, &b werden an void-Zeiger px, py übergeben!!!

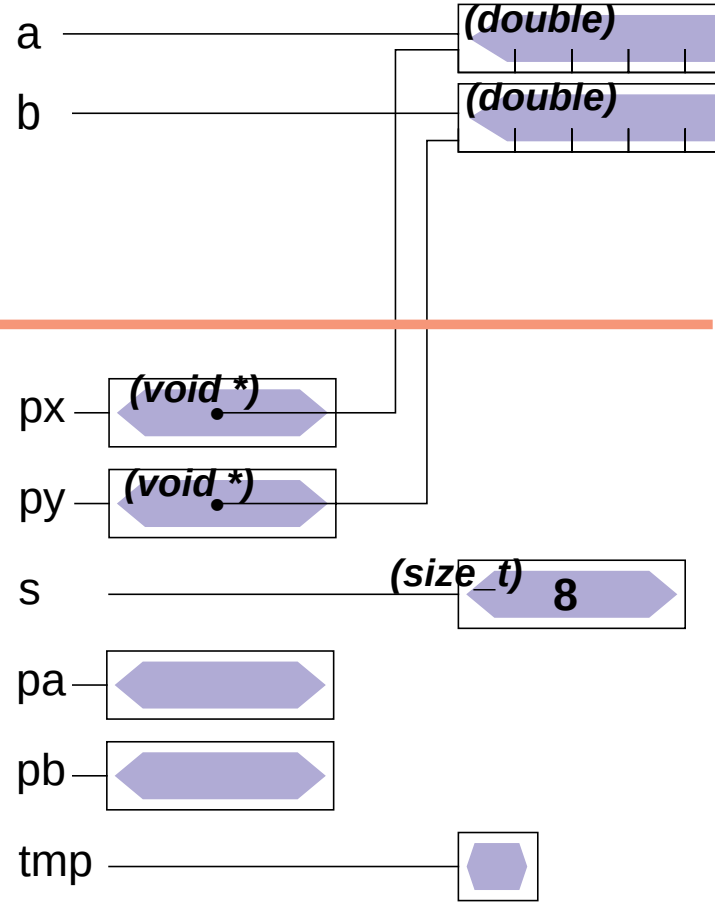
1 generische Zeiger als Funktionsargumente

- Beispiel:

```
main(void) {
double a, b;
void swap_generic(void *, void *, size_t);
...
swap_generic(&a, &b, sizeof(double));
}
```

```
void swap_generic(void *px, void *py, size_t s);
{
        char *pa, *pb, tmp;

        pa = (char *)px;
        ...
```



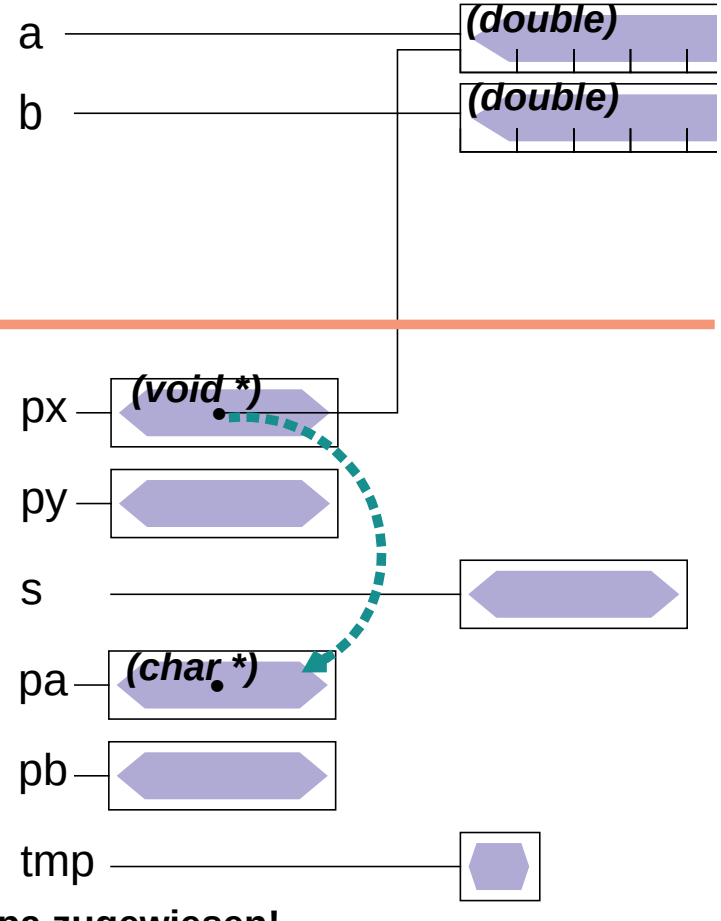
1 generische Zeiger als Funktionsargumente

- Beispiel:

```
main(void) {
double a, b;
void swap_generic(void *, void *, size_t);
...
swap_generic(&a, &b, sizeof(double));
}
```

```
void swap_generic(void *px, void *py, size_t s);
{
        char *pa, *pb, tmp;

        pa = (char *)px;
        ...
```



void-Zeiger px wird in char-Zeiger verwandelt und pa zugewiesen!

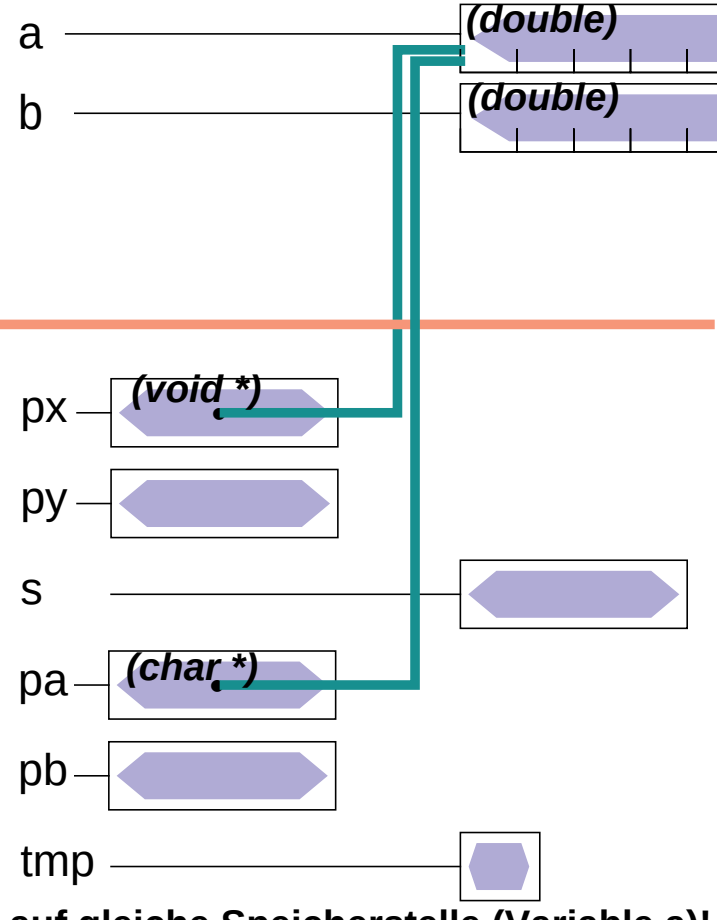
1 generische Zeiger als Funktionsargumente

- Beispiel:

```
main(void) {
double a, b;
void swap_generic(void *, void *, size_t);
...
swap_generic(&a, &b, sizeof(double));
}
```

```
void swap_generic(void *px, void *py, size_t s);
{
        char *pa, *pb, tmp;

        pa = (char *)px;
        ...
```



zwei Zeiger mit unterschiedlichem Typ zeigen jetzt auf gleiche Speicherstelle (Variable a)!

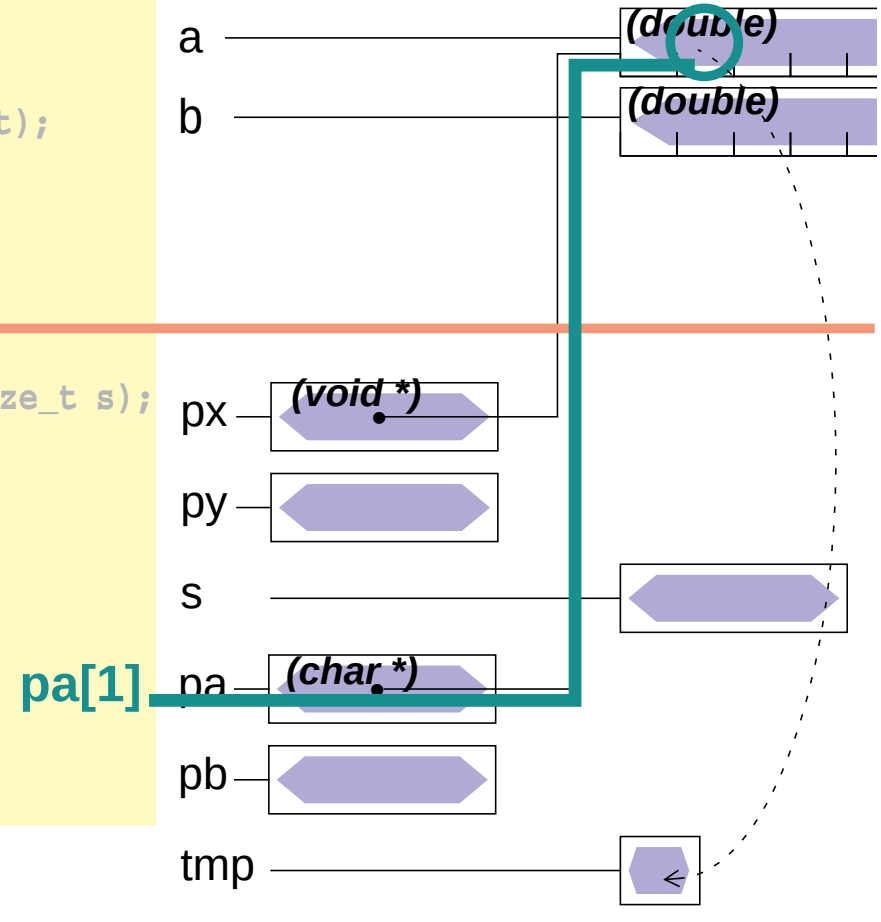
1 generische Zeiger als Funktionsargumente

- Beispiel:

```
main(void) {
double a, b;
void swap_generic(void *, void *, size_t);
...
swap_generic(&a, &b, sizeof(double));
}
```

```
void swap_generic(void *px, void *py, size_t s);
{
        char *pa, *pb, tmp;

        pa = (char *)px;
        ...
        tmp = pa[1];
```



pa wird als char-Array betrachtet!

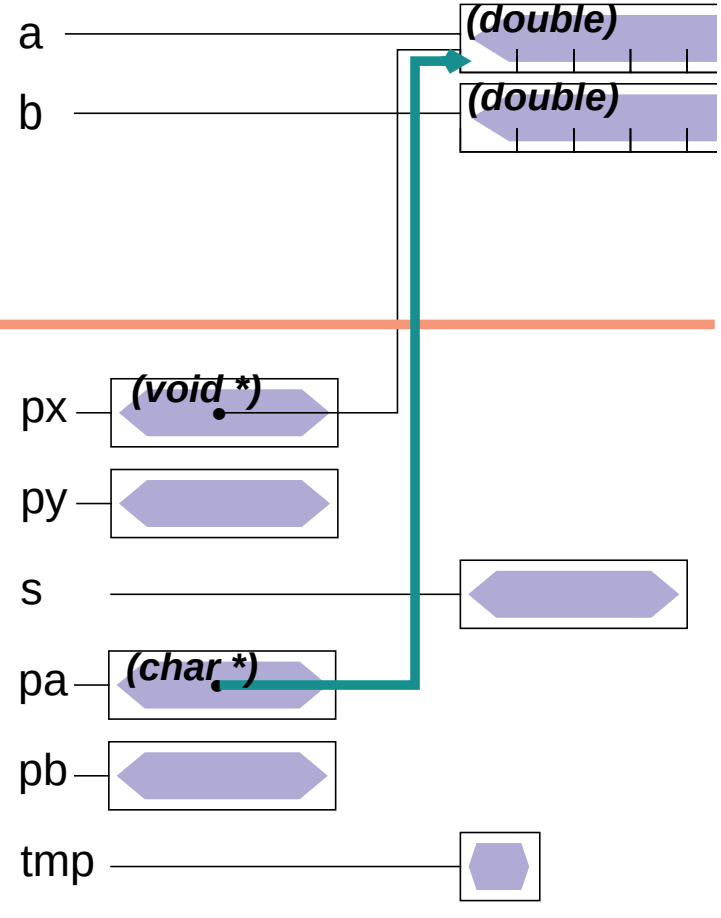
1 generische Zeiger als Funktionsargumente

- Beispiel:

```
main(void) {
double a, b;
void swap_generic(void *, void *, size_t);
...
swap_generic(&a, &b, sizeof(double));
}
```

```
void swap_generic(void *px, void *py, size_t s);
{
        char *pa, *pb, tmp;

        pa = (char *)px;
        ...
```



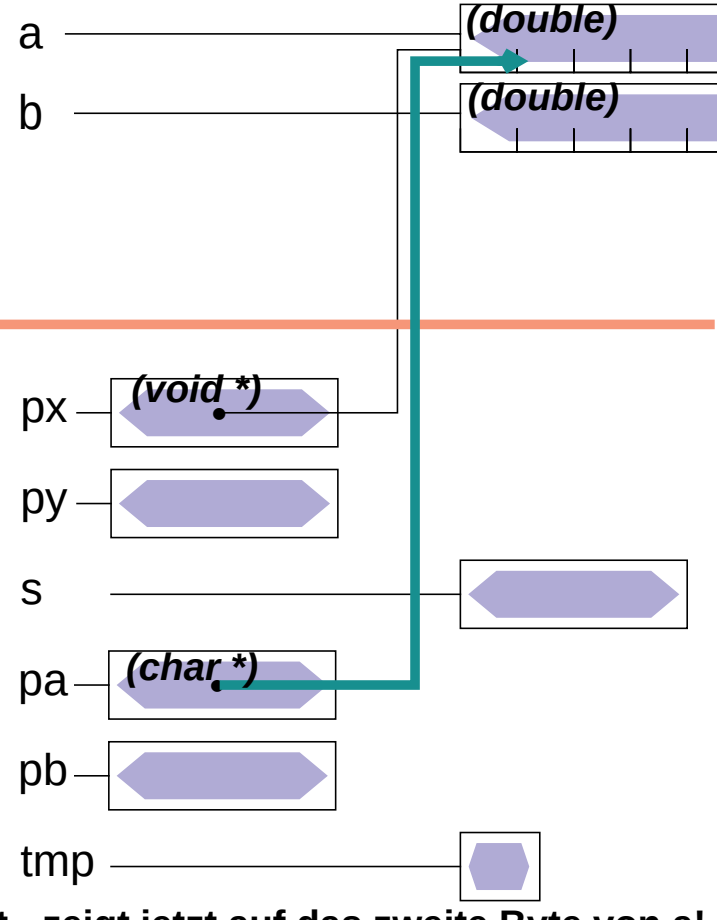
1 generische Zeiger als Funktionsargumente

- Beispiel:

```
main(void) {
double a, b;
void swap_generic(void *, void *, size_t);
...
swap_generic(&a, &b, sizeof(double));
}
```

```
void swap_generic(void *px, void *py, size_t s);
{
        char *pa, *pb, tmp;

        pa = (char *)px;
        ...
        pa++;
```



pa wird als char-Zeiger betrachtet und inkrementiert - zeigt jetzt auf das zweite Byte von a!

1 generische Zeiger als Funktionsargumente

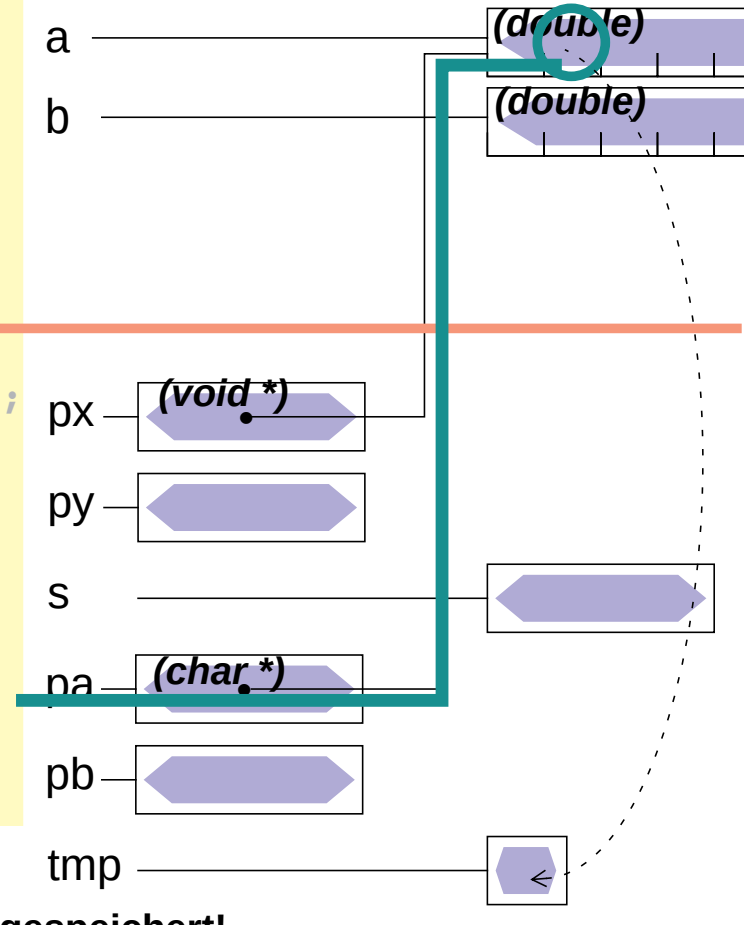
- Beispiel:

```
main(void) {
double a, b;
void swap_generic(void *, void *, size_t);
...
swap_generic(&a, &b, sizeof(double));
}
```

```
void swap_generic(void *px, void *py, size_t s);
{
        char *pa, *pb, tmp;

        pa = (char *)px;
        ...
        pa++;
        tmp = *pa;
```

*pa



zweites Byte der Variablen a wird in tmp zwischengespeichert!

U4 4. Übung

U4-1 Aufgabe 3

- Teilaufgabe a)
  - ➤ Argumente aus der Kommandozeile ausgeben
  - ➤ Vorlesung F.36 ff
- Teilaufgabe b)
  - ➤ Directory öffnen (*opendir(3)*, Vorlesung G.8)
  - ➤ Schleife: Einträge lesen (*readdir(3)*, Vorlesung G.9)
  - ➤ Dateinamen ausgeben (*printf(3)*, `d_name` aus `dirent`-Struktur)

U4-2 Aufgabe 3c

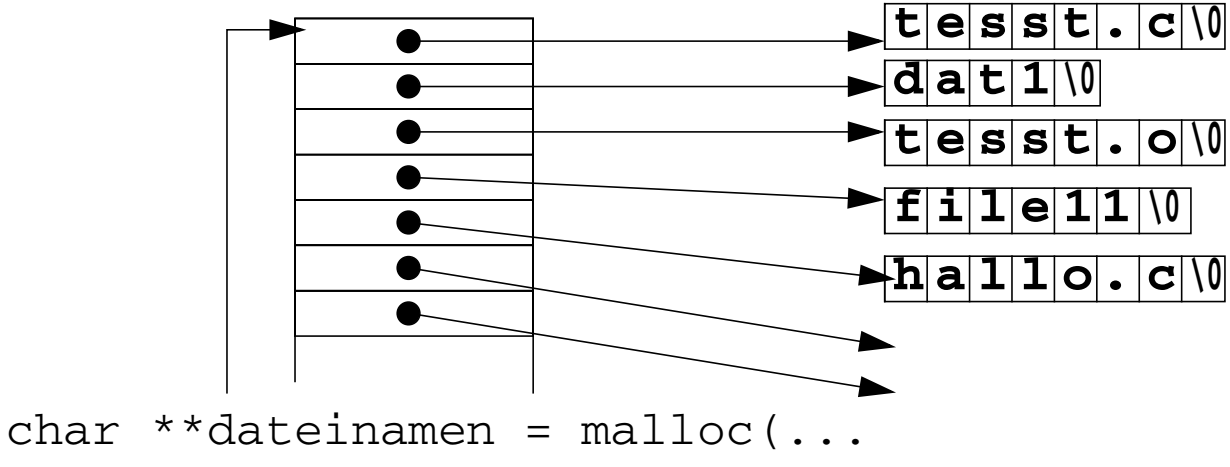
- Problem
  - ➤ Rückgabewert von readdir nur bis zum nächsten Aufruf von readdir gültig
  - ➤ Daten müssen aus dirent-Struktur wegkopiert werden
- Lösung
  - ➤ Speicher für Dateinamen mit *malloc(3)* besorgen
  - ➤ benötigten Speicherplatz mir *strlen(3)* ermitteln
  - ➤ Platz für abschließendes `'\0'`-Zeichen nicht vergessen!
  - ➤ Dateiname in neuen Speicher umkopieren (*strcpy(3)*)
- Problem 2
  - ➤ Einträge in einem Directory sind nicht sortiert
- Lösung
  - ➤ Feld mit Zeigern auf die Dateinamen mit der Funktion *qsort(3)* sortieren

U4-2 Aufgabe 3c (2)

- Zeigerfeld anlegen
  - Annahme: maximal 1000 Einträge in einem Directory (Aufgabenstellung!)
    - Feld fester Größe möglich
      ```
      char dateinamen[1000];
      ```
  - funktioniert nicht, wenn Feld vergrößert werden soll!
    - Feld dynamisch allokieren
      ```
      int eintraege = 1000;
      char **dateinamen = malloc (eintraege * sizeof(char *));
      ```
    - und wenn es dann nicht groß genug ist? — Feld vergößern!
      ```
      eintraege += 1000;
      dateinamen = realloc(dateinamen, eintraege * sizeof(char *));
      ```

U4-2 Aufgabe 3c (3)

- Zeiger im Zeigerfeld auf Dateinamen setzen
  (➔ jeder Zeiger ≙ einem Dateinamen)



U4-2 Aufgabe 3c (4)

- Dateinamen (=Wörter) sortieren
  - Funktion qsort aufrufen
  - Schnittstelle aus **`stdlib.h`**:

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

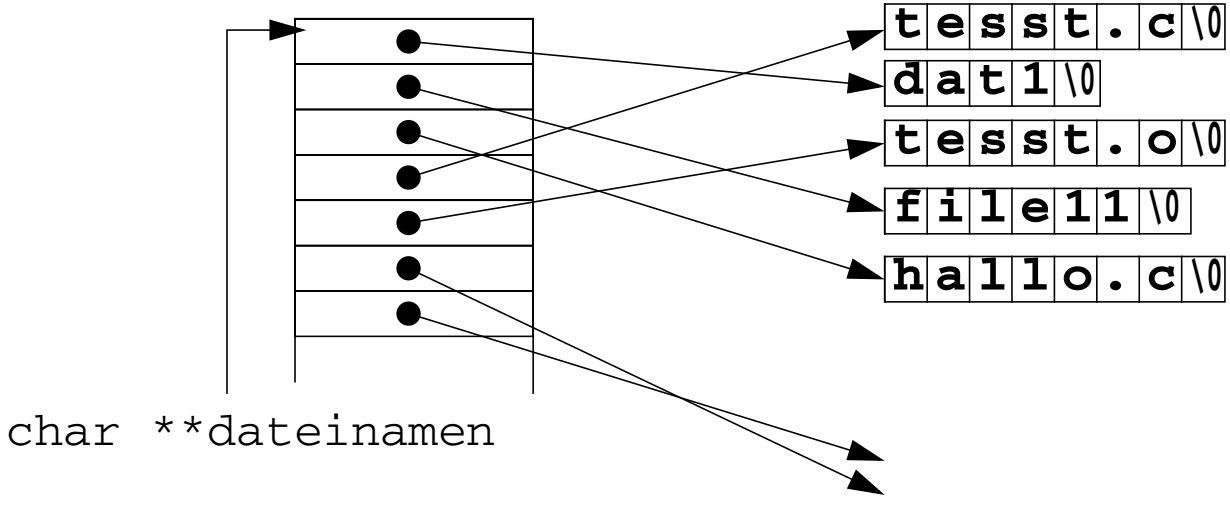
  - Bedeutung der Parameter:
    - **base** : Zeiger auf das erste Element des Feldes, dessen Elemente sortiert werden sollen *(Zeiger auf erstes Wort)*
    - **nel** : Anzahl der Elemente im zu sortierenden Feld *(Zahl der Wörter)*
    - **width**: Größe eines Elements *(Größe eines char-Zeigers - sizeof!)*
    - **compare**: Zeiger auf Vergleichsfunktion

U4-2 Aufgabe 2 (4)

- Lösung für die compare-Funktion:

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

- Resultat



➤ die Wörter (Dateinamen) stehen unverändert, nur die Zeiger auf die Wörter wurden sortiert!

U4-2 Aufgabe 2 (4)

- nach dem Sortieren
  - mit `printf("%s\n", ...)` alle Einträge des Feldes ausgeben

U5 4. Übungsaufgabe

- Grundlegendes zur Übung mit dem AVR-µC
- Register
- I/O Ports
- Interrupts
- AVR-Umgebung

U5-1 Grundlegendes zur Übung mit dem AVR-µC

- Die Übungsaufgaben werden wir mit Hilfe eines Simulators testen und entwickeln
  - ➤ der Simulator simuliert den Mikrocontroller **ATmega128**
- Wer möchte kann sein Programm auch auf einer echten Hardware testen
  - ➤ hier wird ein **ATmega8** eingesetzt
  - ➤ wurde für uns von Studenten der AGEE gebaut (DANKE!)
- Die beiden Plattformen unterscheiden sich geringfügig.
  - ➤ daher sind kleine Anpassungen nötig, um ein Programm welches auf dem Simulator läuft auf der echten Hardware einzusetzen
- Im Folgenden werden immer beide Varianten des AVR-µC angesprochen. Auf Unterschiede wird gelegentlich hingewiesen

U5-2 Register beim AVR-µC

1 Überblick

- Beim AVR µC sind die Register:
  - in den Speicher eingebettet
  - am Anfang des Adressbereichs angeordnet
- Adressen sind der Dokumentation zu entnehmen
- vollständige Dokumentation für "unsere" Mikrocontroller:
  `/proj/i4gdi/tools/doc/`
- Die für die Übungsaufgabe benötigten Register sind auf den Folien erwähnt
- Die Bibliothek (avr-libc), die wir verwenden, definiert bereits sinnvolle Makros für alle Register des AVR µC
  (**`#include <avr/io.h>`**)

2 Makros für Register-Zugriffe

- Makros mit aussagekräftigen Namen können den Umgang mit Registern deutlich vereinfachen
- Beispiel:
  - Makro für Register an Adresse 0x5:

```
#define REG1 (*(volatile unsigned char *)0x5)
```

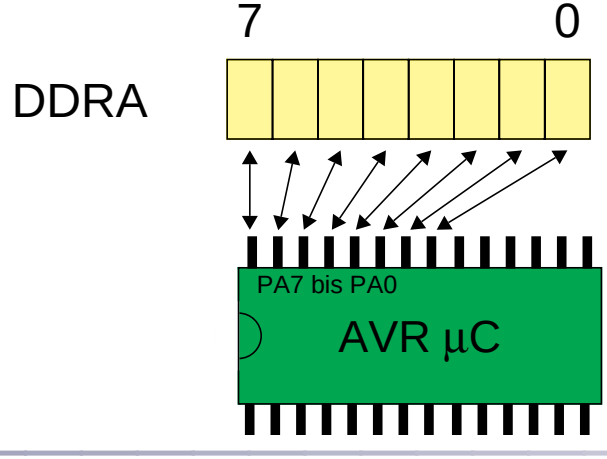
  - Verwenden dieses Registers:

```
REG1 = 0;                // schreibender Zugriff
...
if (REG1 == 0x04)        // lesender Zugriff
    REG1 &= ~4;          // lesender und schreibender Zugriff
```

U5-3 I/O-Ports des AVR µC

1 Überblick

- Jeder I/O-Port des AVR µC wird durch 3 (8-bit) Register gesteuert:
  - Datenrichtungsregister (`DDRx` = data direction register)
  - Datenregister (`PORTx`)
  - Port Eingabe Register (`PINx` = port input register) [nur-lesbar]
- Jedem Anschluss-Pin ist ein Bit in jedem der 3 Register zugeordnet
  - ➤ Beispiel: DDR von Port A:



2 I/O Port Register

- **PINx**: Bit *i* gibt den aktuellen Wert des Pin *i* von Port *x* an (nur lesbar).
- **DDRx**: hier konfiguriert man ob man einen Pin von Port *x* als Ein- oder Ausgang verwenden möchte
  - Bit *i* = 1→ Pin *i* als **Ausgang** verwenden
  - Bit *i* = 0→ Pin *i* als **Eingang** verwenden
- **PORTx**: Auswirkung abhängig von `DDRx`:
  - ist Pin *i* als **Ausgang** konfiguriert, so steuert Bit *i* im `PORTx` Register ob am Pin *i* ein high- oder ein low-Pegel erzeugt werden soll.
    - Bit *i* = 1→ high-Pegel an Pin *i*
    - Bit *i* = 0→ low-Pegel an Pin *i*
  - ist Pin *i* als **Eingang** konfiguriert, so kann man einen internen pull-up Widerstand aktivieren
    - Bit *i* = 1→ pull-up Widerstand an Pin *i* (Pegel wird auf high gezogen)
    - Bit *i* = 0→ Pin *i* als tri-state konfiguriert

3 Beispiel: Aktivieren eines Ports

- Pin 3 von Port B als Ausgang konfigurieren
  und auf V_{cc} schalten:

```
DDRB |= 0x08;  // Pin 3 von Port B als Ausgang nutzen ...

PORTB |= 0x08; // ... und auf 1 (=high) setzen
```

- Pin 0 von Port D als Eingang nutzen,
  pull-up Widerstand aktivieren
  und prüfen ob ein low-Pegel anliegt:

```
DDRD &= ~0x01; // Pin 0 von Port D als Eingang nutzen ...
PORTD |= 0x01; // ... und den pull-up Widerstand aktivieren.

if ( (PIND & 0x01) == 0) { // den Zustand auslesen
    // ja ein low Pegel liegt an...
}
```

U5-4 Externe Interrupts des AVR µC

1 Flanken-/Pegel-Steuerung

- Externe Interrupts durch Pegeländerung an bestimmten I/O-Pins
  - ATmega128: 8 Quellen an den Pins PD0-PD3 und PE4 - PE7
  - ATmega8: 2 Quellen an den Pins PD2 und PD3
- Für jeden Interrupt kann bestimmt werden, ob er Pegel- oder Flanken-gesteuert ist
  - ➤ Steuerung für Interrupt *n* durch Interrupt Sense Control Bits (`ISCn0` und `ISCn1`)

ISC*n*1	ISC*n*0	IRQ bei:
0	0	low-Pegel
0	1	jedem Wechsel
1	0	fallender Flanke
1	1	steigender Flanke

1 Flanken-/Pegel-Steuerung (2)

- Je nach Variante des AVR-µC befinden sich die ISC-Bits in unterschiedlichen Registern:
  - ATmega128: External Interrupt Control Register A bzw. B (`EICRA` für INT 0-3, `EICRB` für INT 4-7).
  - ATmega8: MCU Control Register (`MCUCR`)
- Position der ISC-Bits in den Registern durch Makros definiert `ISC`*n*`0` und `ISC`*n*`1`
- Beispiel: `INT0` bei ATmega128 für fallende Flanke konfigurieren

```
// die ISCs für INT0 befinden sich im EICR A
EICRA &= ~(1<<ISC00);     // ISC00 löschen
EICRA |= (1<<ISC01);      // ISC01 setzen
```

2 Maskieren

- Alle Interruptquellen können separat ausgeschaltet (=maskiert) werden
- Für externe Interrupts sind folgende Register zuständig:
  - ATmega128: External Interrupt Mask Register (`EIMSK`)
  - ATmega8: General Interrupt Control Register (`GICR`)
- Die Bits in diesen Registern sind durch Makros `INT`*n* definiert
- Ein gesetztes Bit aktiviert den jeweiligen Interrupt
- Beispiel: Interrupt 0 aktivieren

```
EIMSK |= (1<<INT0);     // unmask Interrupt 0
```

2 Maskieren (2)

- Alle Interrupts können nochmals bei der CPU direkt abgeschaltet werden
  - ➤ durch speziellen Maschinenbefehl
- Die Bibliothek avr-libc bietet hierfür Funktionen an:
  (`#include <avr/interrupt.h>`)
  - ◆ `sei()` - lässt Interrupts zu
  - ◆ `cli()` - blockiert alle Interrupts
- Beispiel

```
#include <avr/interrupt.h>

sei();                          // IRQs zulassen
```

- Innerhalb eines Interrupt-Handlers sind automatisch alle Interrupts maskiert, beim Verlassen werden sie wieder demaskiert
- Nach dem Starten des µC sind die Interrupts bei der CPU abgeschaltet

3 Interrupt-Handler

- Installieren eines Interrupt-Handlers wird durch die verwendete Bibliothek unterstützt
- Makro `ISR` (Interrupt Service Routine) zur Definition einer Handler-Funktion
  (**`#include <avr/interrupt.h>`**)
- Als Parameter gibt man dem Makro den gewünschten Vektor an, z. B. `INT0_vect` für den externen Interrupt 0
- Beispiel: Handler für Interrupt 0 implementieren

```
#include <avr/interrupt.h>
volatile int zaehler;

ISR (INT0_vect) {
  zaehler++;
}
```

U5-5 AVR Umgebung

1 Compiler

- Um auf einem PC Programme für den AVR-Mikrocontroller zu erstellen, wird ein ***Cross-Compiler*** benötigt
  - AVR-Studio + WinAVR (Windows)
  - GNU gcc (Linux)
- Wir verwenden gcc: Alle benötigten Programme, Werkzeuge und Bibliotheken befinden sich in: `/proj/i4gdi/tools/`
  - ➤ z. B. der Cross-Compiler: `/proj/i4gdi/tools/bin/avr-gcc`
- Wichtiger Parameter: `-mmcu`, um den Werkzeugen die genaue Zielplattform mitzuteilen
  - ➤ Beispiel: ein Programm für den ATmega128 erstellen

```
/proj/i4gdi/tools/bin/avr-gcc -mmcu=atmega128 ampel.c -o ampel
```

2 AVR-Simulator

- Simuliert die AVR-CPU und die integrierte Peripherie auf der Entwicklungsplattform
- Einfaches Testen und Debuggen, ohne das Programm auf der echten Hardware installieren zu müssen
- Bietet Möglichkeiten den Zustand der Pins zu verändern und zu beobachten
- In Verbindung mit einem Debugger ist auch das zeilenweise Ausführen des Programmes möglich
- Mögliche Varianten:
  - ➤ integriert in AVR-Studio (Windows)
  - ➤ `simulavr` bzw. `simulavrxx` (Linux)

3 AVR-Simulator simulavrxx

- Simulator für den ATmega128
- Der Simulator befindet sich in: `/proj/i4gdi/tools/bin/`
- Aufruf des Simulators mit einer kleinen graphischen Anzeige der Peripherie für unsere Übungsaufgabe:
  `/proj/i4gdi/tools/bin/simulavr_gui` Programm-Name
- An den Simulator kann man einen Debugger koppeln
  - Simulator und Debugger starten:
    `/proj/i4gdi/tools/bin/simulavr_gui_debug` Programm-Name

4 Makefile

- Zur Vereinfachung wird unter
  `/proj/i4gdi/pub/aufgabe4/Makefile`
  ein kleines Makefile zur Verfügung gestellt
- Ein Makefile beschreibt wie ein Programm gebaut und ggf. gestartet wird
- Das Programm `make` liest dieses Makefile und baut das Programm entsprechend
- Kopieren sie sich das Makefile in ihr Verzeichnis
  `cp /proj/i4gdi/pub/aufgabe4/Makefile /proj/i4gdi/`loginname`/aufgabe4/`
- Ein Aufruf von `make` erstellt das Programm `ampel` für den ATmega128 falls ein `ampel.c` im selben Verzeichnis liegt
- `make sim` erstellt das Programm und startet es mit Hilfe des Simulators

U5-6 Ein einfaches Beispiel

- `bsp.c`: Die grüne LED einschalten.

```
#include <avr/io.h>

int main(void){

    DDRB |= 0xff;  //alle Pins von Port B als Ausgang nutzen
    PORTB = 0x01;  // ... und Pin 0 auf high setzen

    for (;;);
}
```

- Compilieren:

```
/proj/i4gdi/tools/bin/avr-gcc -mmcu=atmega128 bsp.c -o bsp
```

- Programm im Simulator starten

```
/proj/i4gdi/tools/bin/simulavr_gui bsp
```

U5-7 Übungsaufgabe: Peripherie

- Als externe Peripherie verwenden wir 4 LEDs und einen Taster die wie folgt angeschlossen sind:

	ATmega128 (Simulator)	ATmega8
Taster (Eingang)	Port D Pin 0	Port D Pin 2
LED grün	Port B Pin 0	
LED gelb	Port B Pin 1	
LED rot	Port B Pin 2	
LED blau	Port D Pin 7	

- Adressen der Port-Register im I/O-Adressbereich (identisch für ATmega128 und ATmega8)

Register	Adresse
DDRB	0x37
PORTB	0x38
PINB	0x36

Register	Adresse
DDRD	0x31
PORTD	0x32
PIND	0x30

1 Taster-Details

- Für den Taster muss der pull-up-Widerstand aktiviert werden
- Taster zieht den Pegel auf GND
- Durch den Taster kann Interrupt 0 ausgelöst werden