

Übungen zu Systemnahe Programmierung in C (SPiC) – Sommersemester 2025

Übung 7

Maxim Ritter von Onciul
Eva Dengler

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Informatik 4
Systemsoftware

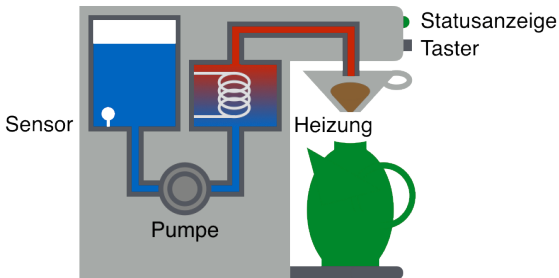


Friedrich-Alexander-Universität
Technische Fakultät

Vorstellung Aufgabe 4

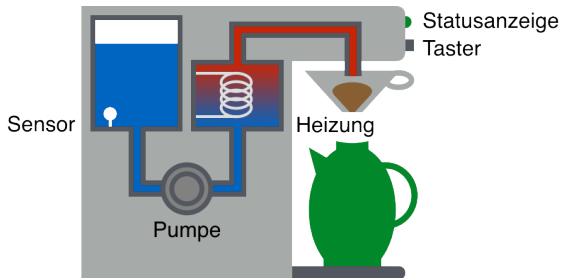
Hands-on: Kaffeemaschine

Screencast: <https://www.video.uni-erlangen.de/clip/id/17647>



■ Lernziele:

- Zustandsautomaten
- Timer bzw. Alarm
- Interrupts & Schlafenlegen



■ Beschaltung:

- Pumpe & Heizung: Port D, Pin 5 (active-low)
- Taster: INT0 an Port D, Pin 2 (active-low)
- Sensor: INT1 an Port D, Pin 3 (Wasser: high; kein Wasser: low)
- Statusanzeige:
 - BLUE0: **STANDBY**
 - GREEN0: **ACTIVE**
 - RED0: **NO_WATER**



STANDBY

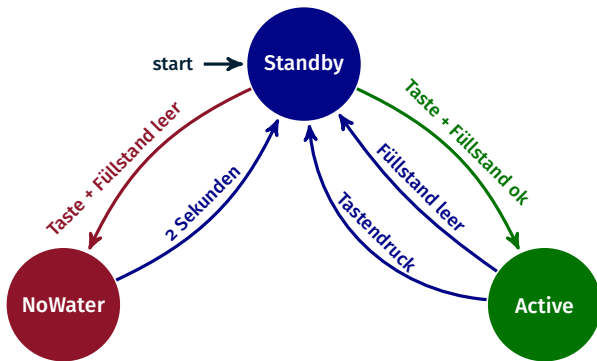
- Kaffeemaschine ist aus
- Pumpe und Heizung sind aus
- Benutzer kann Kaffeezubereitung durch Tastendruck starten
- Anfangszustand

ACTIVE

- Kaffeemaschine ist an
- Pumpe und Heizung sind an
- Wassertank ist nicht leer
- Benutzer kann Kaffeezubereitung durch Tastendruck beenden

NO_WATER

- Kaffeemaschine zeigt an, dass sie nicht genügend Wasser hat
- Pumpe und Heizung sind aus
- Zeitdauer: 2 Sekunden



■ Hinweise:

- Tastendruck & Füllstandsänderung durch Interrupts
- Statusanzeige: `void setLEDState(state_t state)`
- Wartephasen ggf. über SingleShot-Alarm realisieren
- In Wartephasen Mikrocontroller in den Energiesparmodus



DDRx hier konfiguriert man Pin i von Port x als Ein- oder Ausgang

- Bit $i = 1 \rightarrow$ Pin i als Ausgang verwenden
- Bit $i = 0 \rightarrow$ 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 $i = 1 \rightarrow$ high-Pegel an Pin i
 - Bit $i = 0 \rightarrow$ low-Pegel an Pin i
- ist Pin i **als Eingang konfiguriert**, so kann man einen internen pull-up-Widerstand aktivieren
 - Bit $i = 1 \rightarrow$ pull-up-Widerstand an Pin i (Pegel wird auf high gezogen)
 - Bit $i = 0 \rightarrow$ Pin i als tri-state konfiguriert

PINx Bit i gibt aktuellen Wert des Pin i von Port x an (nur lesbar)

- Interrupt Sense Control (ISC) Bits befinden sich beim ATmega328PB im External Interrupt Control Register A (EICRA)
- Position der ISC-Bits im Register durch Makros definiert

Interrupt 0		Interrupt bei	Interrupt 1	
ISC01	ISC00		ISC11	ISC10
0	0	low Pegel	0	0
0	1	beliebiger Flanke	0	1
1	0	fallender Flanke	1	0
1	1	steigender Flanke	1	1

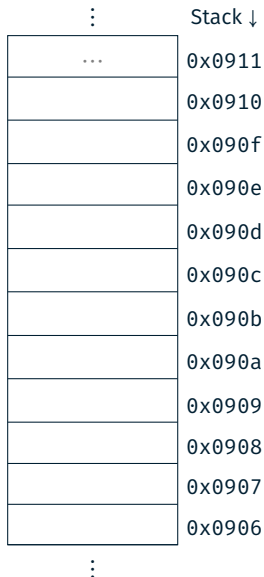
- ATmega328PB: External Interrupt Mask Register (EIMSK)
- Die Bitpositionen in diesem Register sind durch Makros INTn definiert

Hands-on: Laufschrift

Screencast: <https://www.video.uni-erlangen.de/clip/id/18170>

- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: `strlen(s) + 1`

```
01 char s[] = "World\n";  
02 char c = s[0];  
03 c = s[4];  
04 char *s2 = s + 2;  
05 c = s2[1];
```



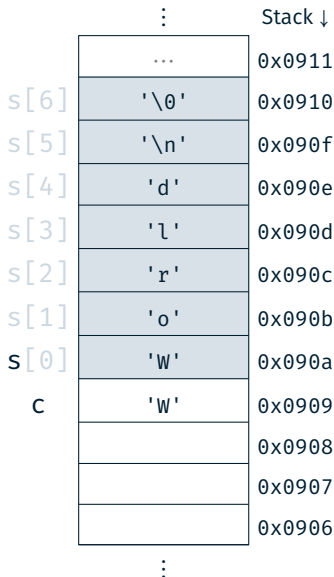
- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: `strlen(s) + 1`

```
01 char s[] = "World\n";  
02 char c = s[0];  
03 c = s[4];  
04 char *s2 = s + 2;  
05 c = s2[1];
```

	:	Stack ↓
	...	0x0911
s[6]	'\0'	0x0910
s[5]	'\n'	0x090f
s[4]	'd'	0x090e
s[3]	'l'	0x090d
s[2]	'r'	0x090c
s[1]	'o'	0x090b
s[0]	'W'	0x090a
		0x0909
		0x0908
		0x0907
		0x0906
	:	

- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: `strlen(s) + 1`

```
01 char s[] = "World\n";  
02 char c = s[0];  
03 c = s[4];  
04 char *s2 = s + 2;  
05 c = s2[1];
```



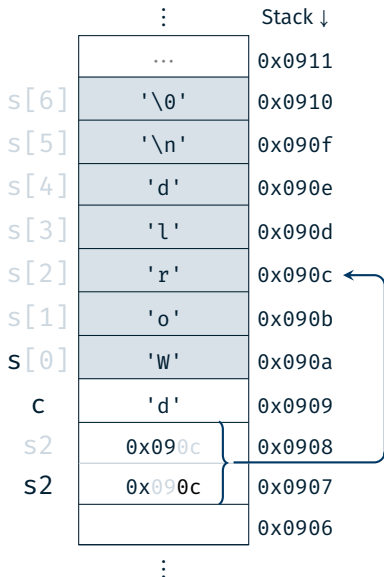
- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: `strlen(s) + 1`

```
01 char s[] = "World\n";  
02 char c = s[0];  
03 c = s[4];  
04 char *s2 = s + 2;  
05 c = s2[1];
```

	⋮	Stack ↓
	...	0x0911
s[6]	'\0'	0x0910
s[5]	'\n'	0x090f
s[4]	'd'	0x090e
s[3]	'l'	0x090d
s[2]	'r'	0x090c
s[1]	'o'	0x090b
s[0]	'W'	0x090a
C	'd'	0x0909
		0x0908
		0x0907
		0x0906
	⋮	

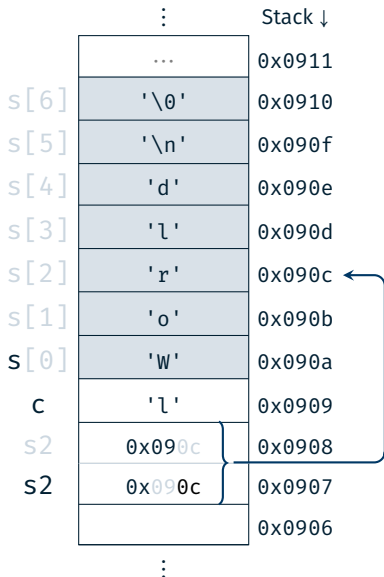
- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: strlen(s) + 1

```
01 char s[] = "World\n";  
02 char c = s[0];  
03 c = s[4];  
04 char *s2 = s + 2;  
05 c = s2[1];
```



- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: strlen(s) + 1

```
01 char s[] = "World\n";  
02 char c = s[0];  
03 c = s[4];  
04 char *s2 = s + 2;  
05 c = s2[1];
```





- Funktionsweise:
Schrittweises Anzeigen eines Textes auf der 7-Segment-Anzeige
- Lernziele:
 - Zeichenketten in C
 - Zeiger & Zeigerarithmetik
 - Alarme & Schlafenlegen
- Vorgehen:
 - Wiederkehrender Alarm mittels TIMER0
 - Zusammensetzen des aktuellen Teilstrings
 - Ausgabe über 7-Segment-Anzeige
 - In Wartephase Mikrocontroller in den Energiesparmodus versetzen (Passives Warten)



```
01 const char *string = "HALLO SPIC";
02 const char *current = string;
03 // current[0] == 'H' && current[1] == 'A'
04 ++current;
05 // current[0] == 'A' && current[1] == 'L'
06 // [...]
07 // current[0] == '\0', current[1] == ??
08 current = string;
```



```
01 const char *string = "HALLO SPIC";  
02 const char *current = string;  
03 // current[0] == 'H' && current[1] == 'A'  
04 ++current;  
05 // current[0] == 'A' && current[1] == 'L'  
06 // [...]  
07 // current[0] == '\0', current[1] == ??  
08 current = string;
```



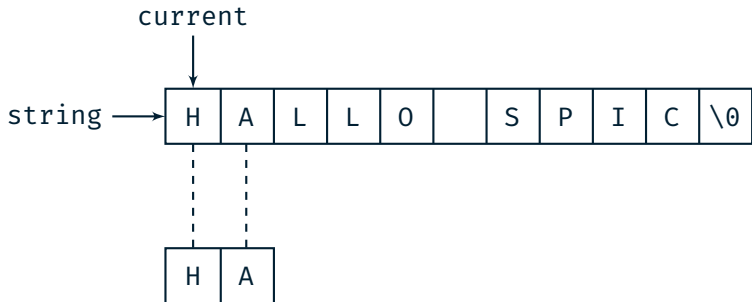


```
01 const char *string = "HALLO SPIC";  
02 const char *current = string;  
03 // current[0] == 'H' && current[1] == 'A'  
04 ++current;  
05 // current[0] == 'A' && current[1] == 'L'  
06 // [...]  
07 // current[0] == '\0', current[1] == ??  
08 current = string;
```



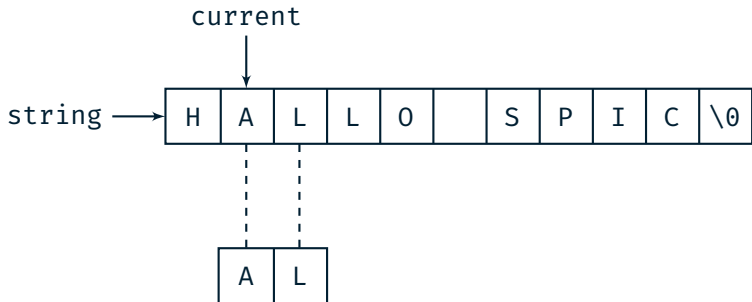


```
01 const char *string = "HALLO SPIC";  
02 const char *current = string;  
03 // current[0] == 'H' && current[1] == 'A'  
04 ++current;  
05 // current[0] == 'A' && current[1] == 'L'  
06 // [...]  
07 // current[0] == '\\0', current[1] == ??  
08 current = string;
```





```
01 const char *string = "HALLO SPIC";  
02 const char *current = string;  
03 // current[0] == 'H' && current[1] == 'A'  
04 ++current;  
05 // current[0] == 'A' && current[1] == 'L'  
06 // [...]  
07 // current[0] == '\\0', current[1] == ??  
08 current = string;
```





```
01 const char *string = "HALLO SPIC";  
02 const char *current = string;  
03 // current[0] == 'H' && current[1] == 'A'  
04 ++current;  
05 // current[0] == 'A' && current[1] == 'L'  
06 // [...]  
07 // current[0] == '\0', current[1] == ??  
08 current = string;
```

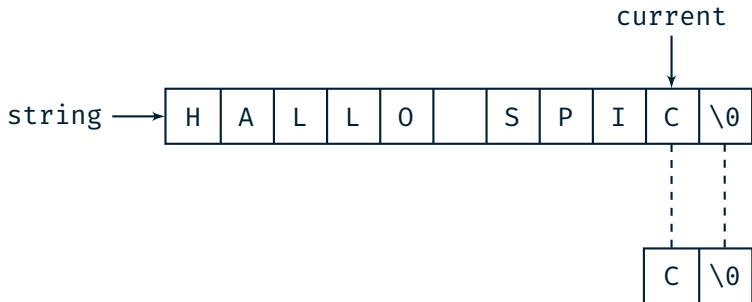
string →

H	A	L	L	O		S	P	I	C	\0
---	---	---	---	---	--	---	---	---	---	----

...

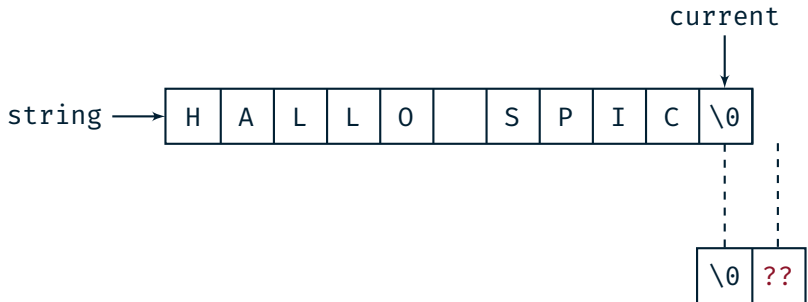


```
01 const char *string = "HALLO SPIC";  
02 const char *current = string;  
03 // current[0] == 'H' && current[1] == 'A'  
04 ++current;  
05 // current[0] == 'A' && current[1] == 'L'  
06 // [...]  
07 // current[0] == '\0', current[1] == ??  
08 current = string;
```



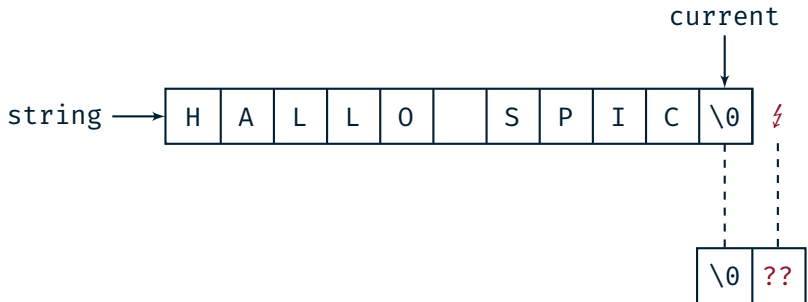


```
01 const char *string = "HALLO SPIC";  
02 const char *current = string;  
03 // current[0] == 'H' && current[1] == 'A'  
04 ++current;  
05 // current[0] == 'A' && current[1] == 'L'  
06 // [...]  
07 // current[0] == '\0', current[1] == ??  
08 current = string;
```





```
01 const char *string = "HALLO SPIC";  
02 const char *current = string;  
03 // current[0] == 'H' && current[1] == 'A'  
04 ++current;  
05 // current[0] == 'A' && current[1] == 'L'  
06 // [...]  
07 // current[0] == '\0', current[1] == ?? ⚡  
08 current = string;
```





```
01 const char *string = "HALLO SPIC";  
02 const char *current = string;  
03 // current[0] == 'H' && current[1] == 'A'  
04 ++current;  
05 // current[0] == 'A' && current[1] == 'L'  
06 // [...]  
07 // current[0] == '\0', current[1] == ?? ⚡  
08 current = string;
```

