

Übungen zu Systemnahe Programmierung in C (SPiC) – Wintersemester 2019/2020

Übung 4

Benedict Herzog
Bernhard Heinloth
Tim Rheinfels

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



Lehrstuhl für Verteilte Systeme
und Betriebssysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Vorstellung Aufgabe 2

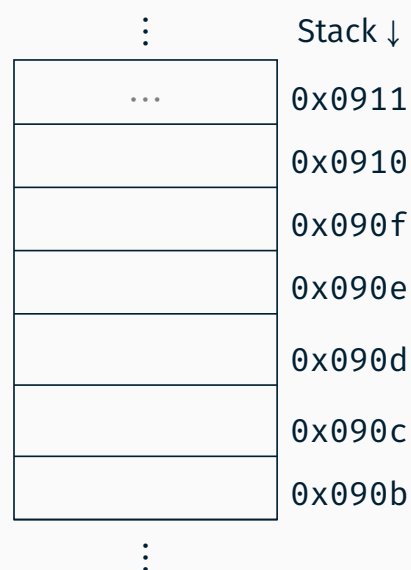
Zeiger & Felder

Vertiefung: Zeiger



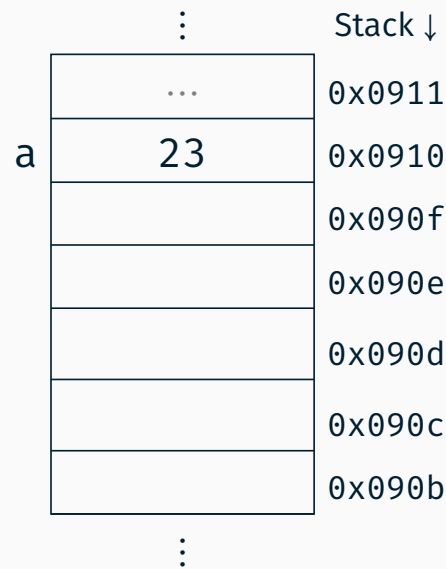
- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;  
02 uint8_t b = 42;  
03 uint8_t * p = &a;  
04 *p = 66;  
05 p = &b;  
06 *p -= 40;  
07 uint8_t c = *p;
```



- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```

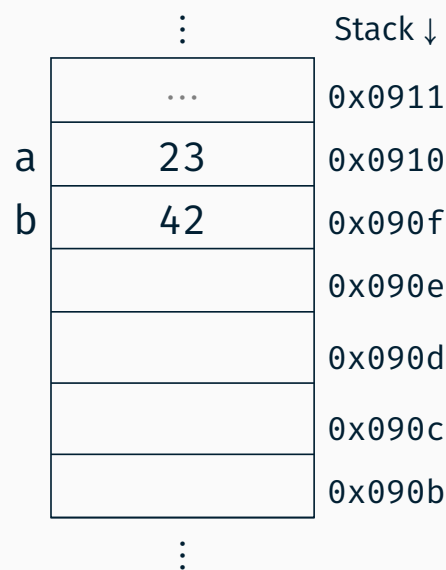


Achtung: Die genaue Anordnung der Variablen auf dem Stack ist abhängig vom Übersetzer und den gewählten Optimierungen!

1

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```

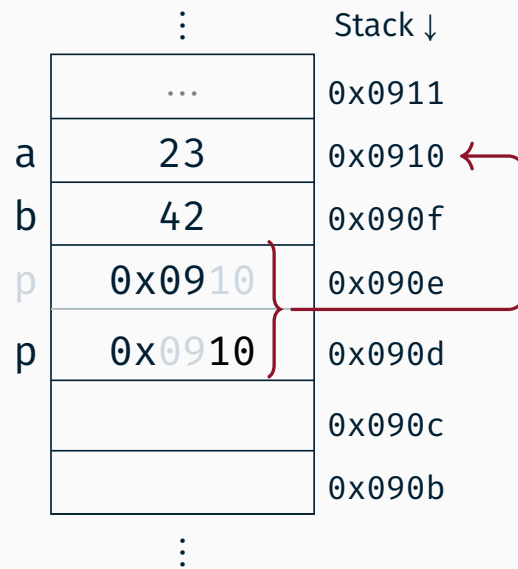


Achtung: Die genaue Anordnung der Variablen auf dem Stack ist abhängig vom Übersetzer und den gewählten Optimierungen!

1

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```

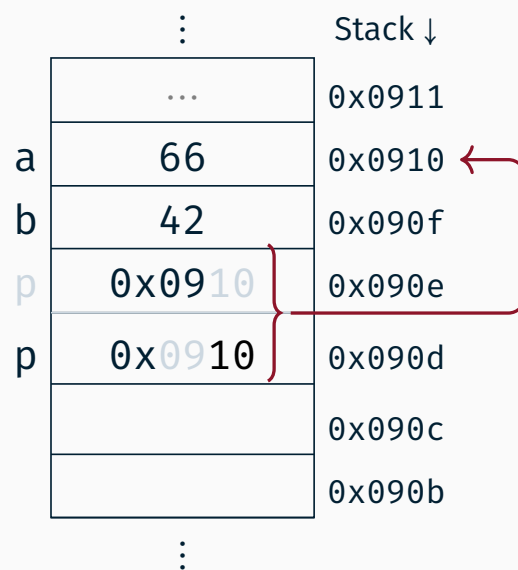


Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

1

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```



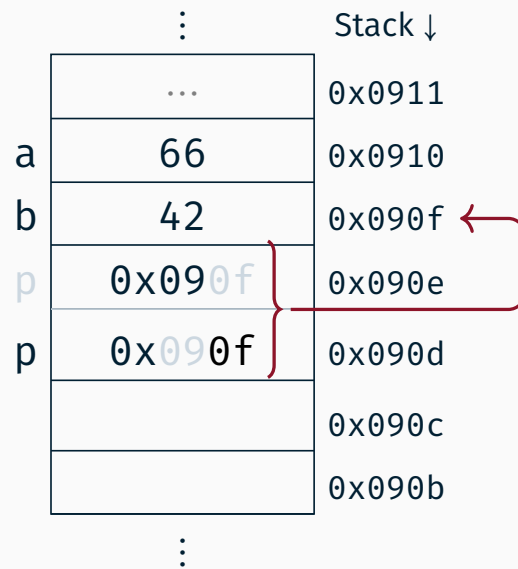
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

1

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```

01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
    
```



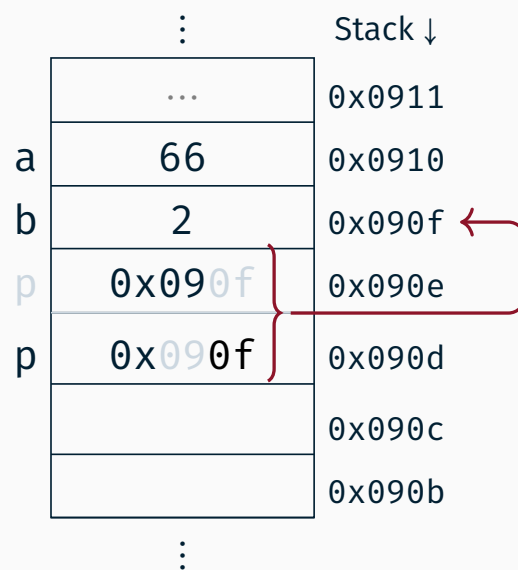
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

1

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```

01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
    
```



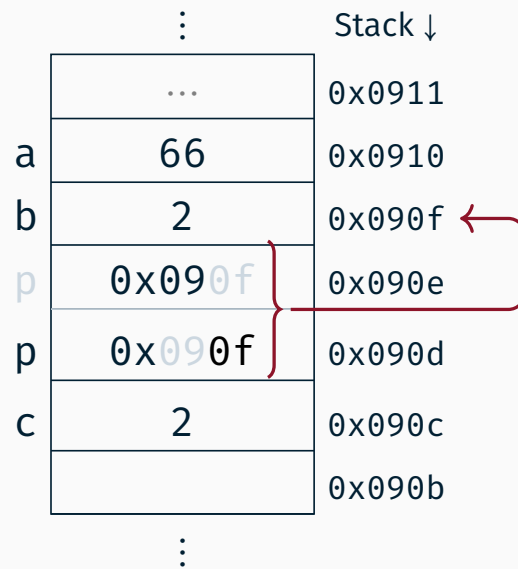
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

1

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```

01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
    
```



Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

1

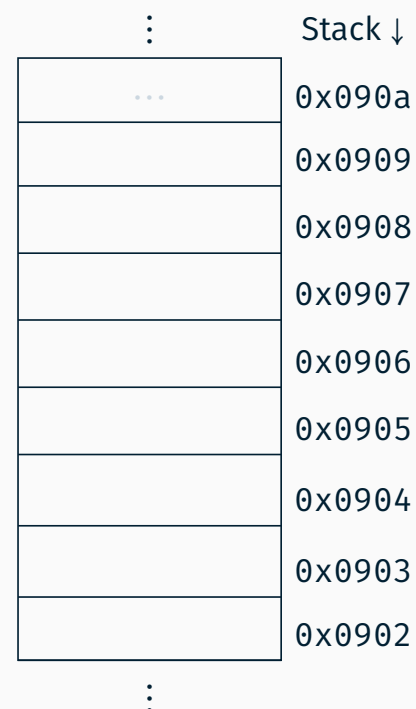
Vertiefung: Felder



- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```

08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
    
```



2

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```

08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
    
```

	⋮	Stack ↓
	...	0x090a
x[3]	16	0x0909
x[2]	8	0x0908
x[1]	4	0x0907
x[0]	2	0x0906
		0x0905
		0x0904
		0x0903
		0x0902
	⋮	

2

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```

08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
    
```

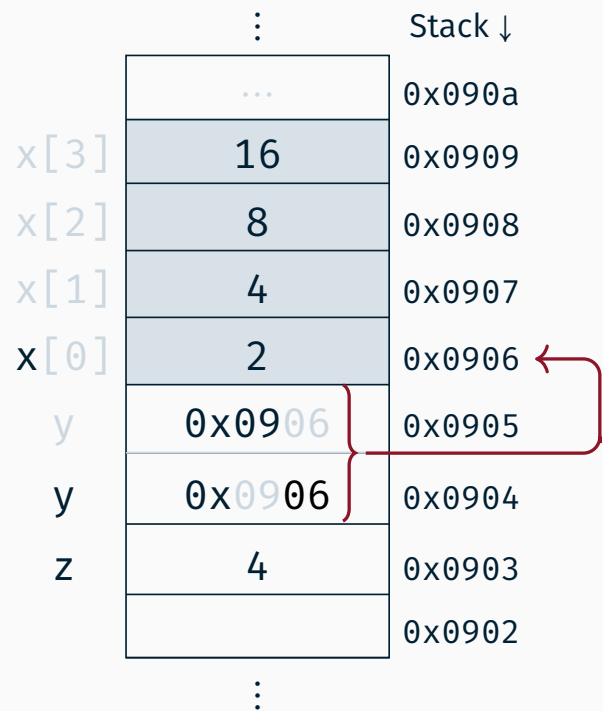
	⋮	Stack ↓
	...	0x090a
x[3]	16	0x0909
x[2]	8	0x0908
x[1]	4	0x0907
x[0]	2	0x0906
y	0x0906	0x0905
y	0x0906	0x0904
		0x0903
		0x0902
	⋮	

2

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```

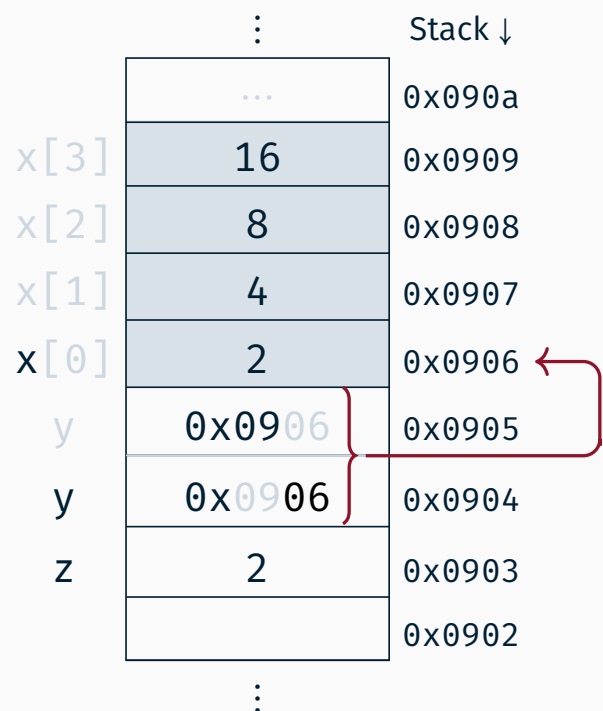
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
    
```



- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```

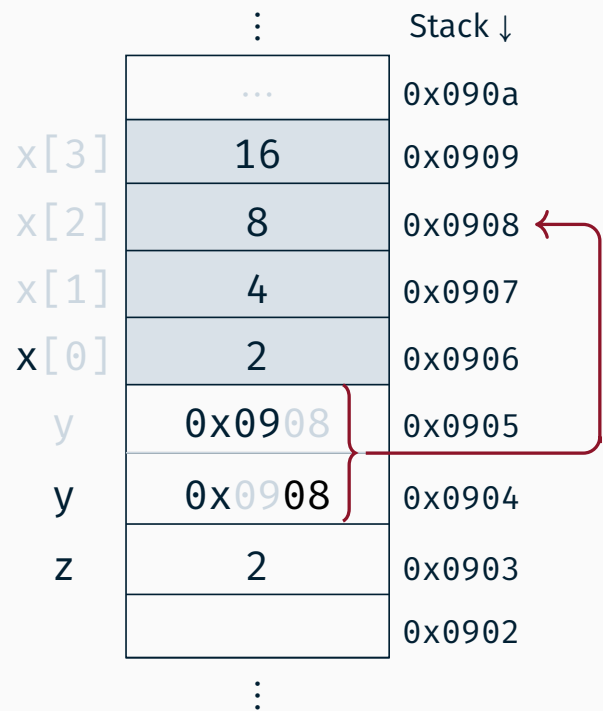
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
    
```



- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```

08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
    
```

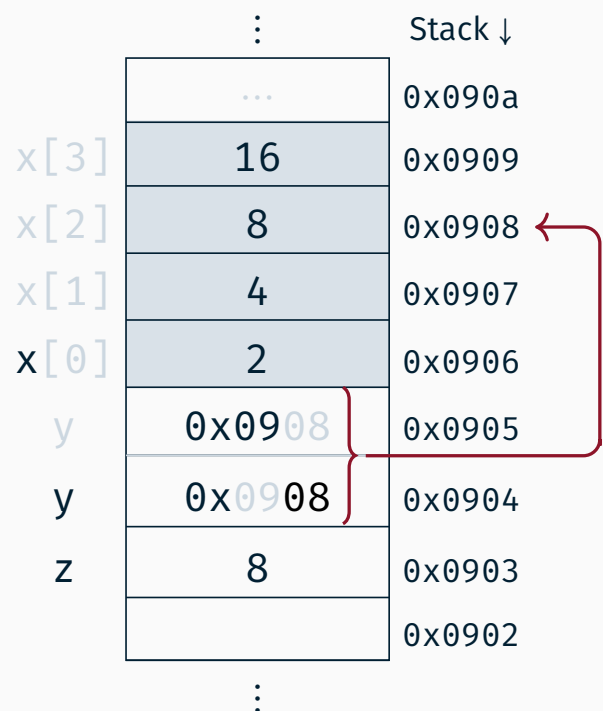


2

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```

08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
    
```

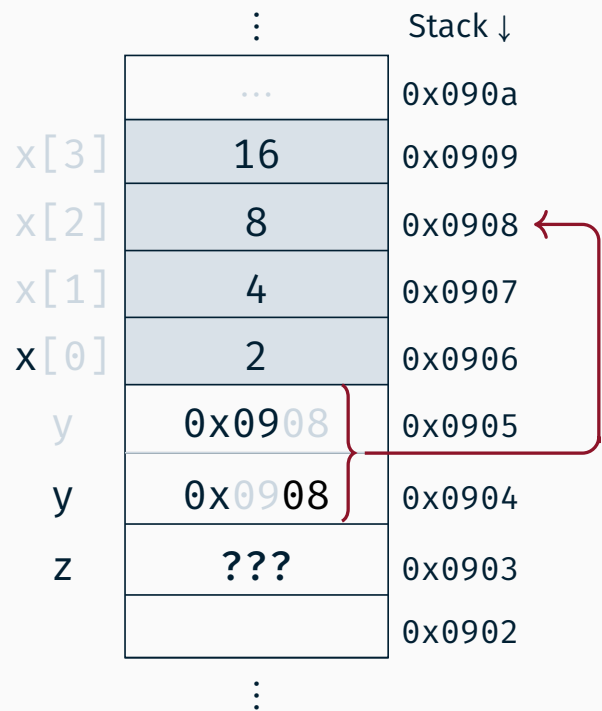


2

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```

08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7]; // ???
    
```



Hands-on: Zeiger

- Call-by-Value vs. Call-by-Reference
- Zeiger und Felder
- Zeigerarithmetik
- `struct` für GPS-Koordinaten
- Feld von GPS-Koordinaten
- Funktionszeiger

Kompilierbar für das SPiCboard (serielle Konsole) oder Linux

Quellcode:

https://www4.cs.fau.de/Lehre/WS19/V_SPIC/Uebungen/material/pointer.c

Hands-on: Laufschrift



- 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";
```

⋮	Stack ↓
...	0x0911
	0x0910
	0x090f
	0x090e
	0x090d
	0x090c
	0x090b
	0x090a
	0x0906
⋮	

4

Vertiefung: Strings



- 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";
```

	:	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
		0x0906
	:	

4

- Funktionsweise:

Schrittweises Anzeigen eines Textes auf der 7-Segment-Anzeige

- Lernziele:

- Alarme & Schlafenlegen
- Zeiger & Zeigerarithmetik
- Zeichenfolgen in C

- Vorgehen:

- Aktivieren eines wiederkehrenden Alarms
(`sb_timer_setAlarm()`)
- Optional: Manuelles Konfigurieren des Timers
- Zusammensetzen des aktuellen Teilstrings
- Ausgabe über 7-Segment-Anzeige
- In Wartephase Mikrocontroller in den Energiesparmodus versetzen