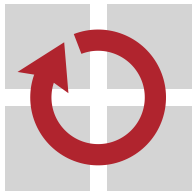
Übungen zu Grundlagen der systemnahen Programmierung in C (GSPiC) im Wintersemester 2018/19

2018-12-05

Alexander von der Haar
Bernhard Heinloth

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



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

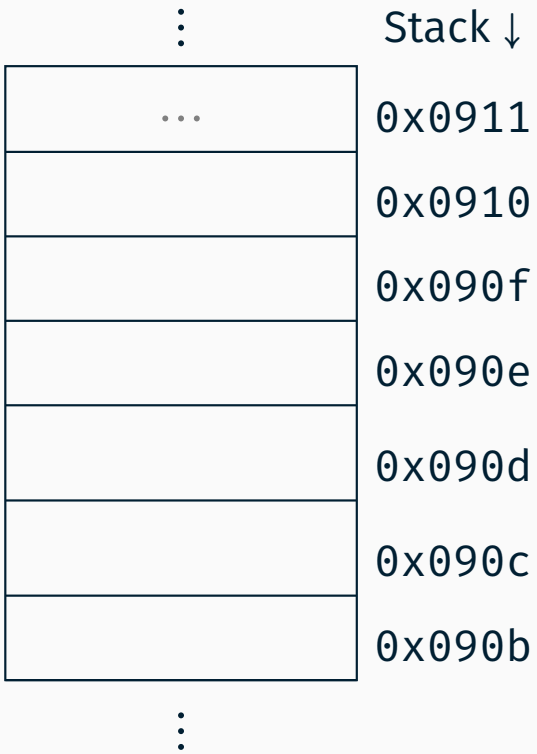
TECHNISCHE FAKULTÄT

Zeiger & Felder

Wiederholung: Zeiger

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

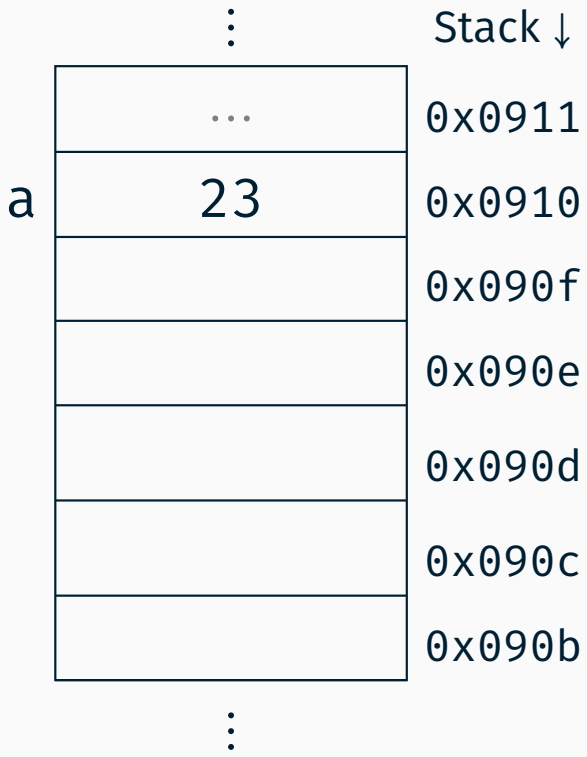
```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```



Wiederholung: 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;
```

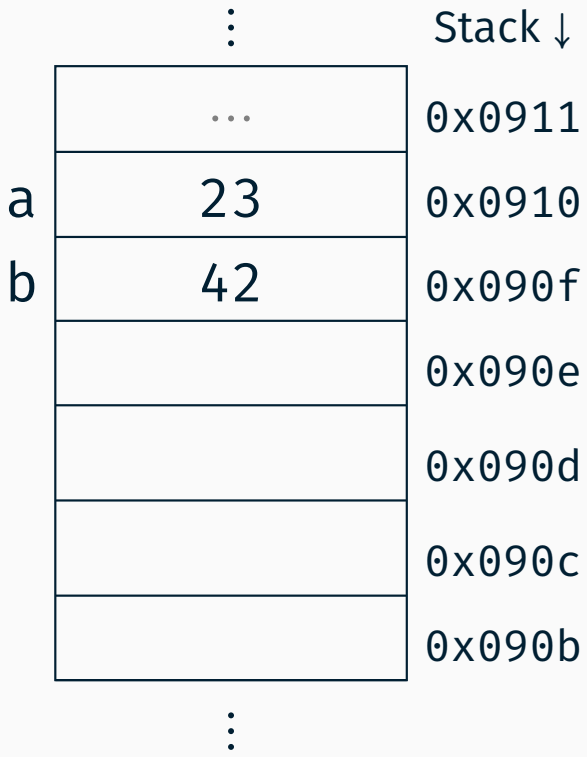


Achtung: Die genaue Anordnung der Variablen auf dem Stack ist abhängig vom Übersetzer und den gewählten Optimierungen!

Wiederholung: 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;
```

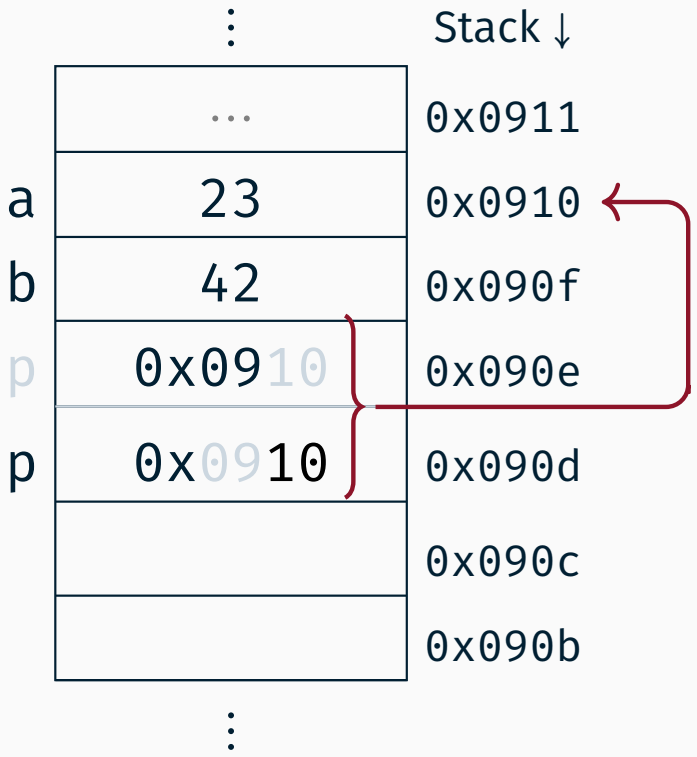


Achtung: Die genaue Anordnung der Variablen auf dem Stack ist abhängig vom Übersetzer und den gewählten Optimierungen!

Wiederholung: 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;
```

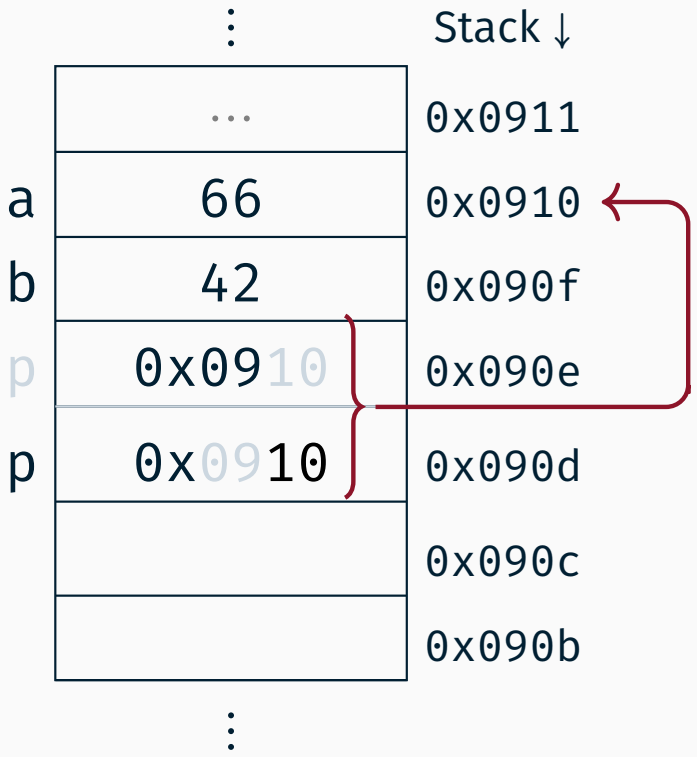


Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: 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;
```

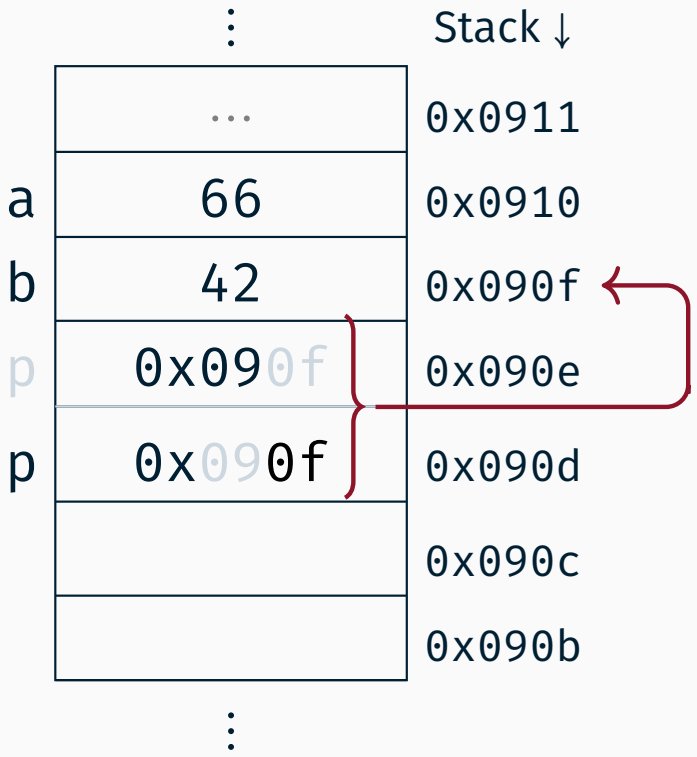


Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: 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;
```

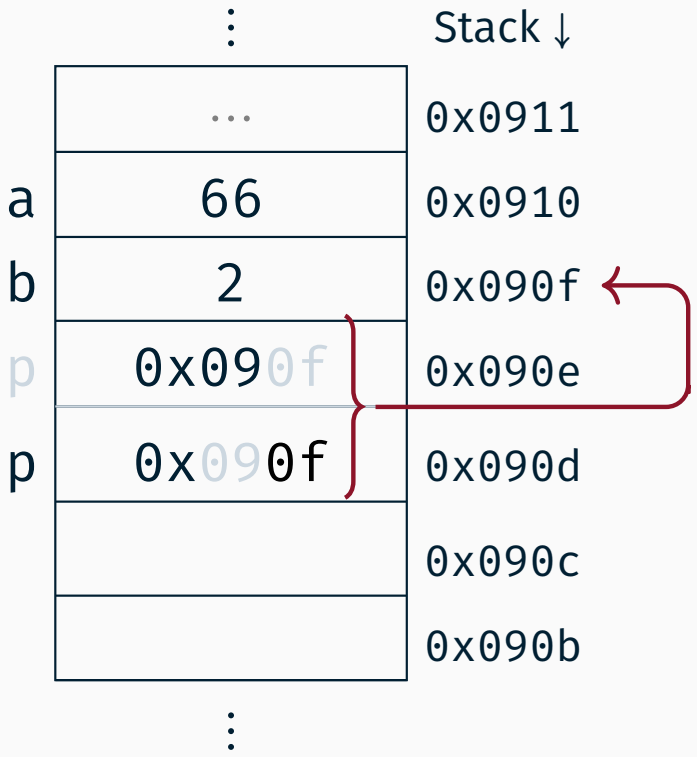


Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: 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;
```

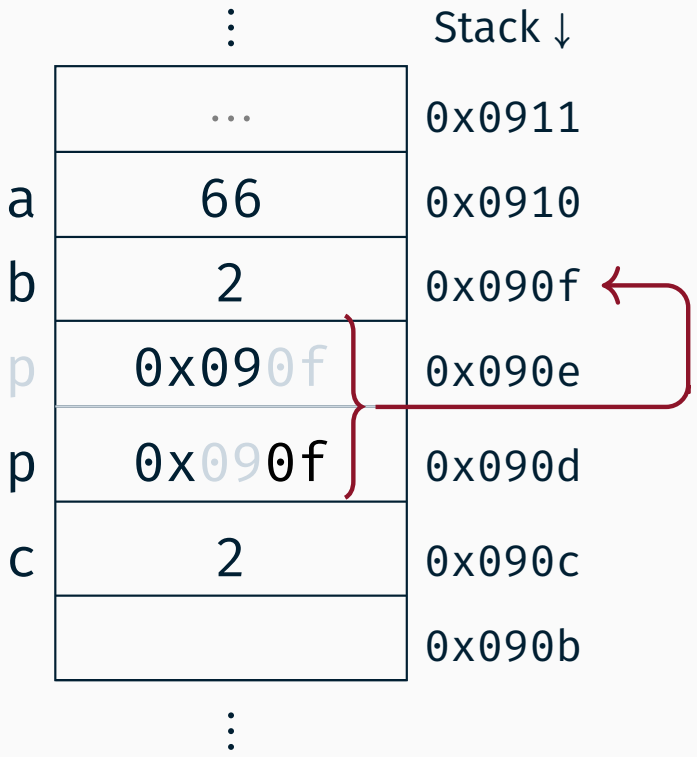


Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: 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;
```

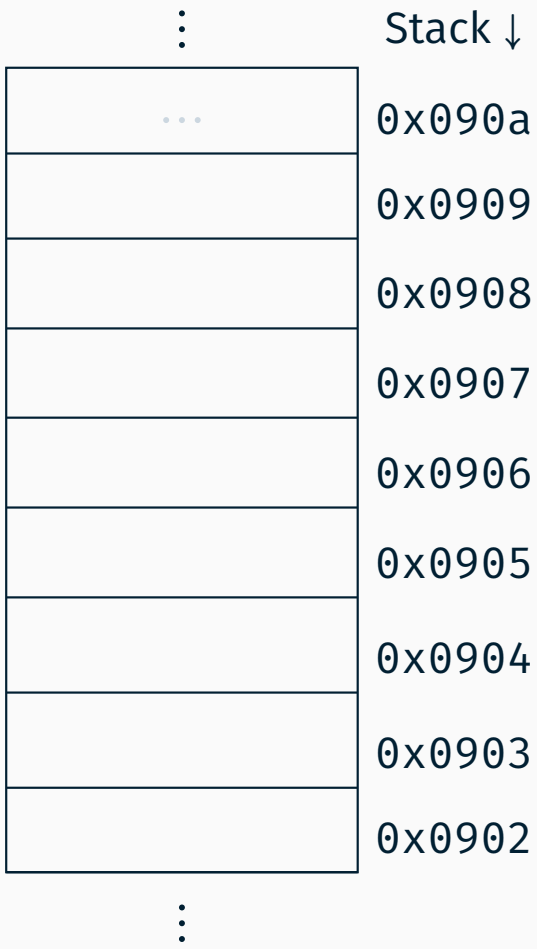


Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Felder

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

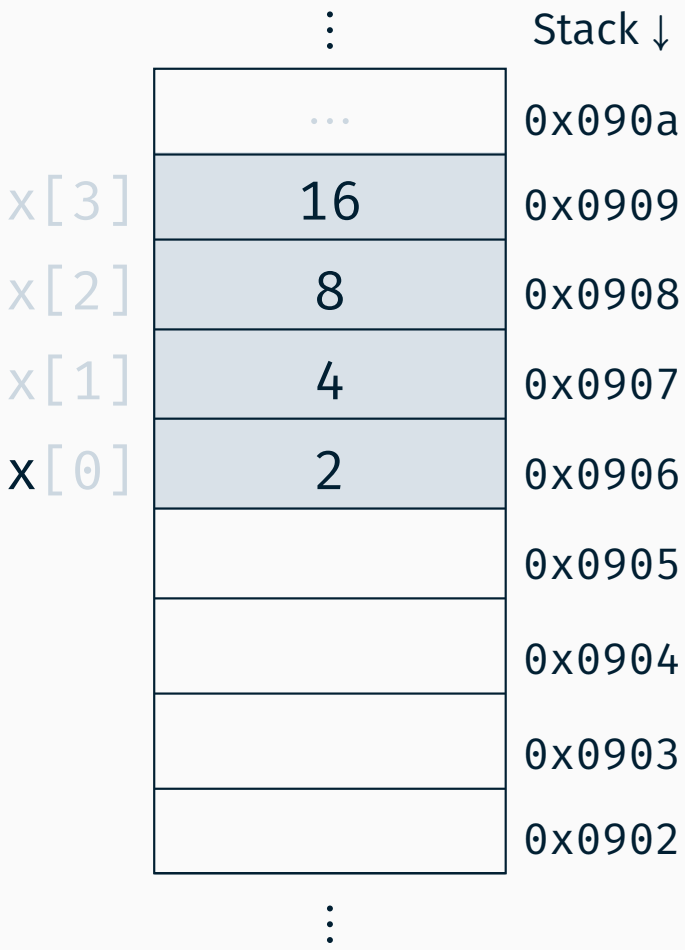
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



Wiederholung: Felder

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

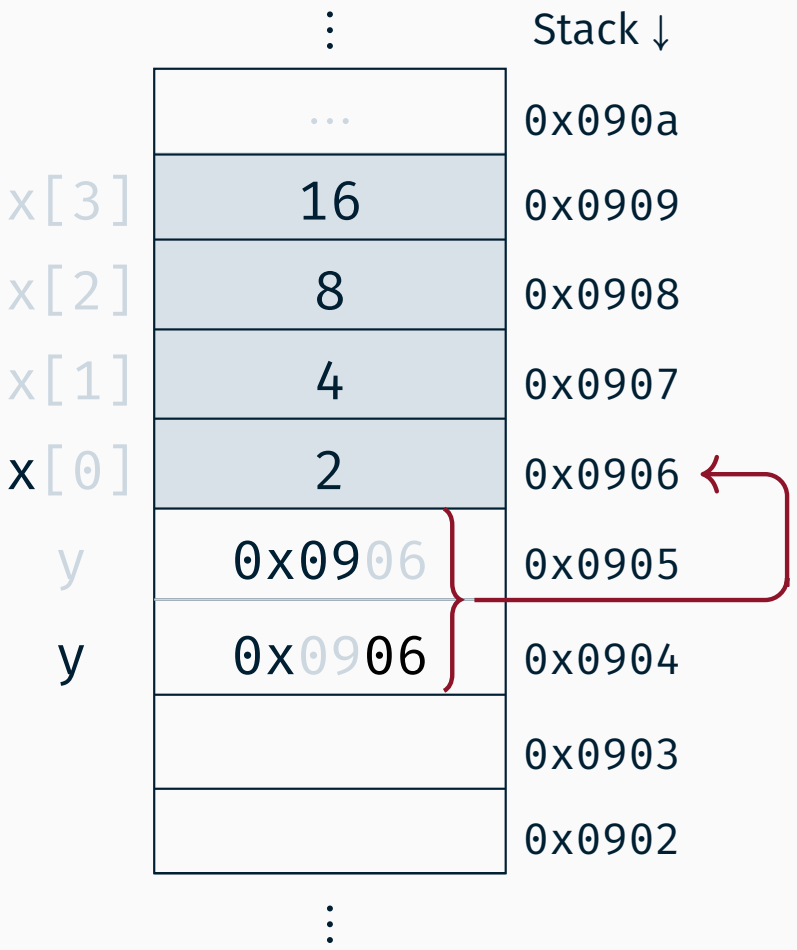
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



Wiederholung: Felder

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

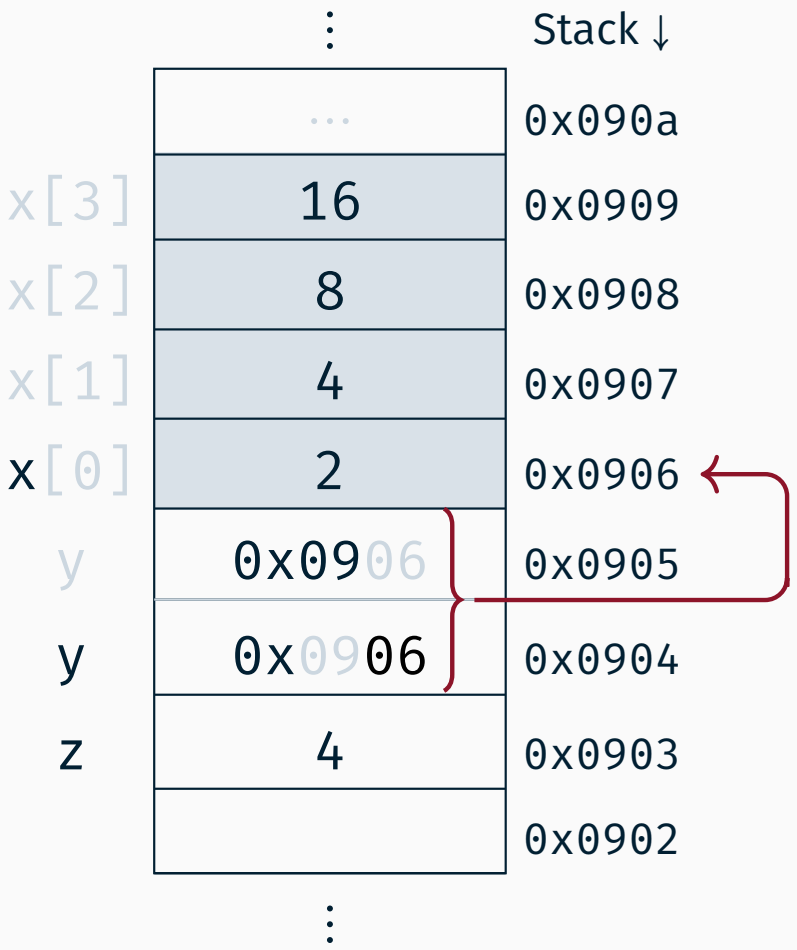
```
08  uint8_t x[] = {2,4,8,16};
09  uint8_t *y = x;
10  uint8_t z = x[1];
11  z = *y;
12  y = y+2;
13  z = *y;
14  z = x[7];
```



Wiederholung: Felder

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

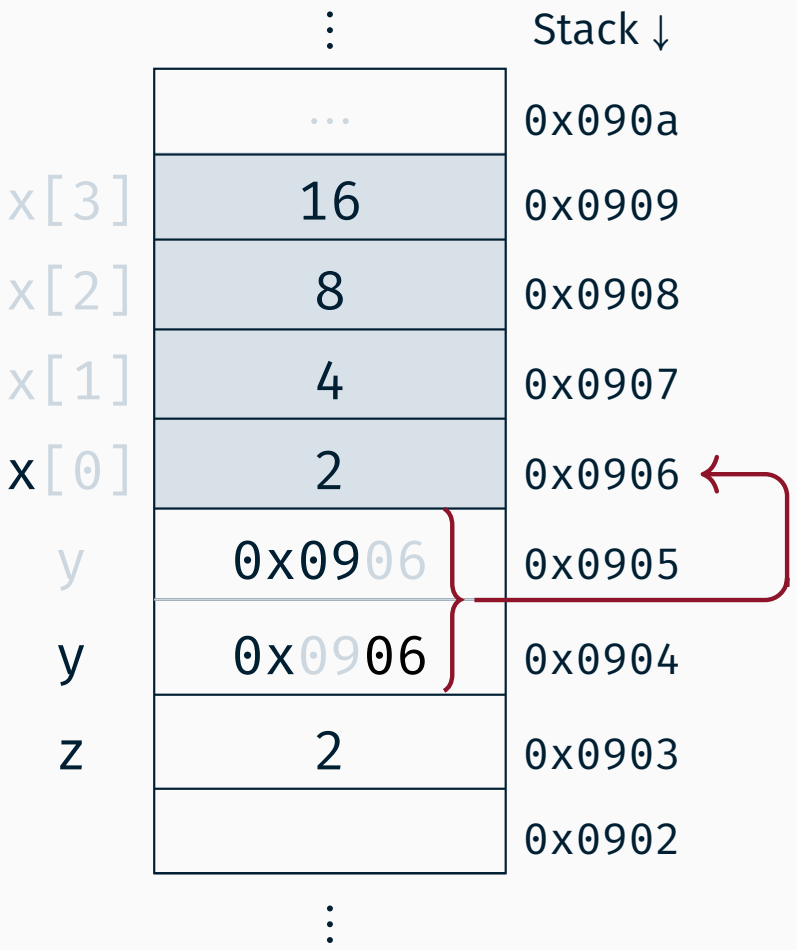
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



Wiederholung: Felder

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

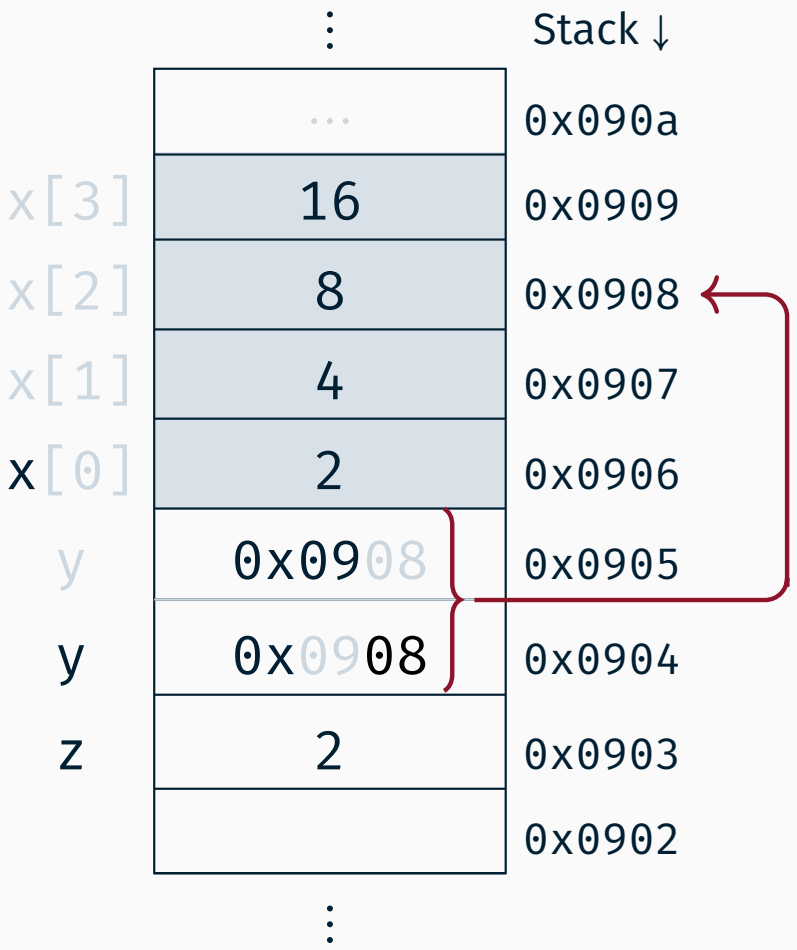
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



Wiederholung: Felder

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

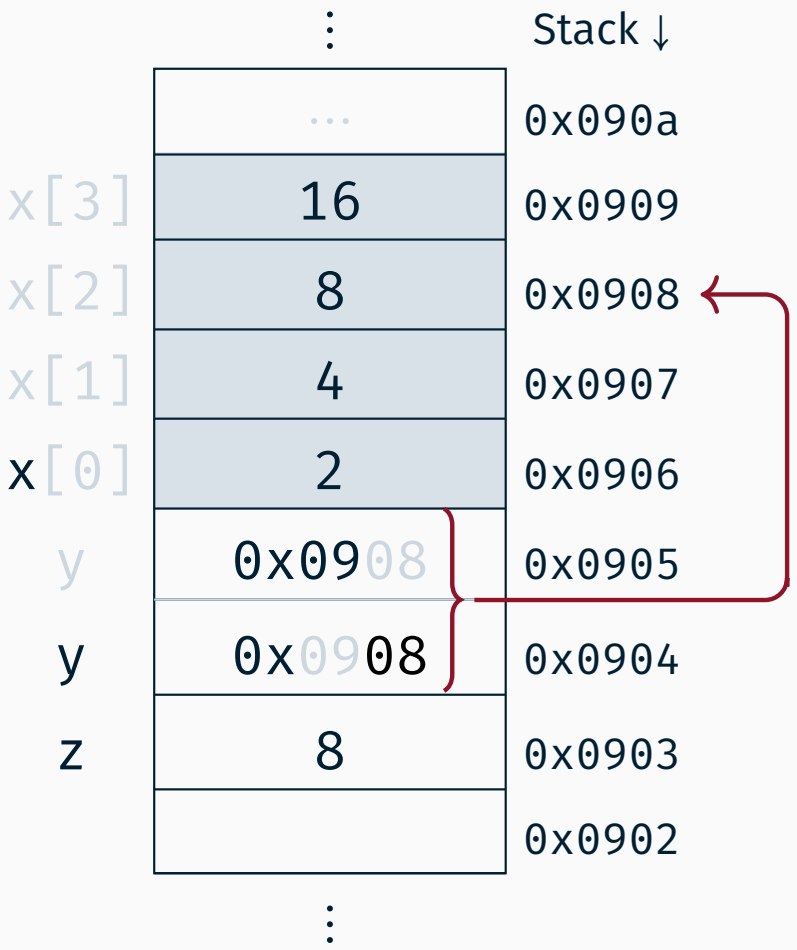
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



Wiederholung: Felder

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

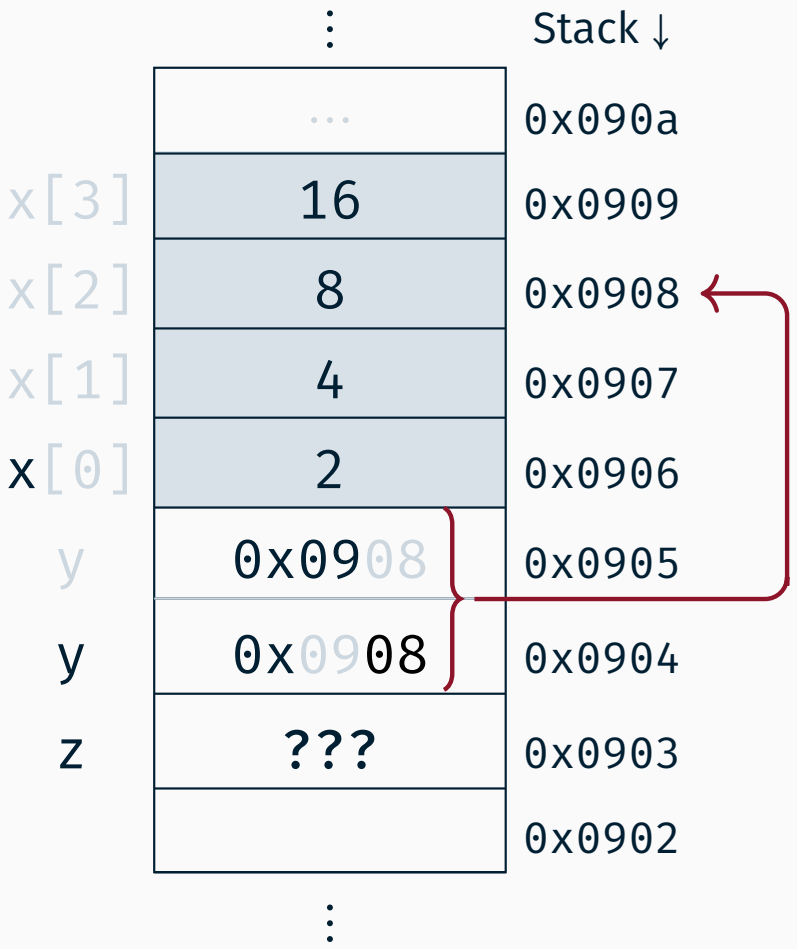
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



Wiederholung: Felder

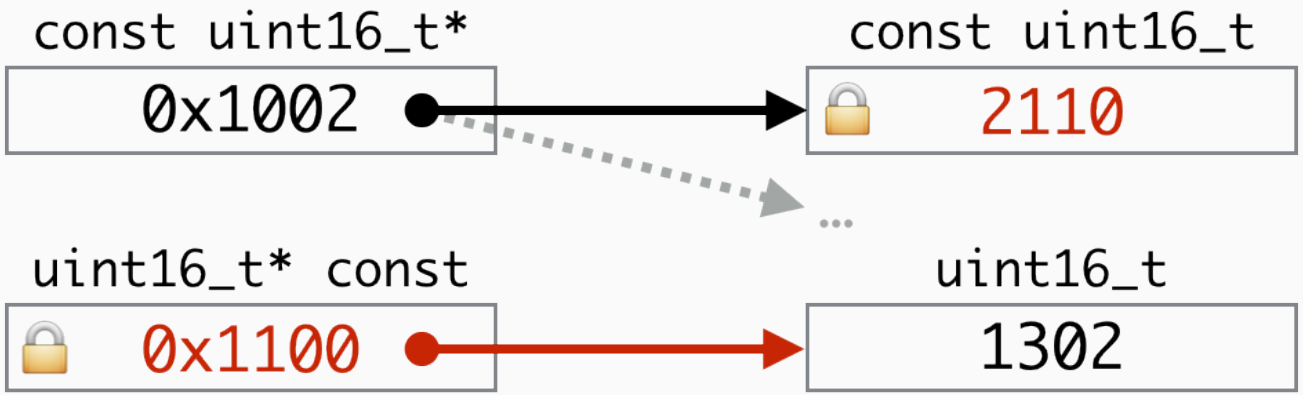
- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```
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]; // ϟϟϟ
```



const uint8_t* vs. uint8_t* const

- `const uint8_t*`
  - ein Pointer auf einen `uint8_t`-**Wert**, der konstant ist
  - Wert nicht über den Pointer veränderbar
- `uint8_t* const`
  - ein **konstanter Pointer** auf einen (beliebigen) `uint8_t`-Wert
  - Pointer darf nicht mehr auf eine andere Speicheradresse zeigen



OLED Display

ASCII-Tabelle

Dez	Hex	Zeichen
32	0x20	SP
33	0x21	!
34	0x22	“
35	0x23	#
36	0x24	$
37	0x25	%
38	0x26	&
39	0x27	’
40	0x28	(
41	0x29	)
42	0x2A	*
43	0x2B	+
44	0x2C	,
45	0x2D	-
46	0x2E	.
47	0x2F	/
48	0x30	0
49	0x31	1
50	0x32	2
51	0x33	3
52	0x34	4
53	0x35	5
54	0x36	6
55	0x37	7
56	0x38	8
57	0x39	9
58	0x3A	:
59	0x3B	;
60	0x3C	≪
61	0x3D	=
62	0x3E	≫
63	0x3F	?

Dez	Hex	Zeichen
64	0x40	@
65	0x41	A
66	0x42	B
67	0x43	C
68	0x44	D
69	0x45	E
70	0x46	F
71	0x47	G
72	0x48	H
73	0x49	I
74	0x4A	J
75	0x4B	K
76	0x4C	L
77	0x4D	M
78	0x4E	N
79	0x4F	O
80	0x50	P
81	0x51	Q
82	0x52	R
83	0x53	S
84	0x54	T
85	0x55	U
86	0x56	V
87	0x57	W
88	0x58	X
89	0x59	Y
90	0x5A	Z
91	0x5B	[
92	0x5C	\
93	0x5D	]
94	0x5E	^
95	0x5F	_

Dez	Hex	Zeichen
96	0x60	‘
97	0x61	a
98	0x62	b
99	0x63	c
100	0x64	d
101	0x65	e
102	0x66	f
103	0x67	g
104	0x68	h
105	0x69	i
106	0x6A	j
107	0x6B	k
108	0x6C	l
109	0x6D	m
110	0x6E	n
111	0x6F	o
112	0x70	p
113	0x71	q
114	0x72	r
115	0x73	s
116	0x74	t
117	0x75	u
118	0x76	v
119	0x77	w
120	0x78	x
121	0x79	y
122	0x7A	z
123	0x7B	{
124	0x7C	|
125	0x7D	}
126	0x7E	-
127	0x7F	DEL

Zeichenketten

- Zeichenkette mit Inhalt *Hallo!*

Zeichenketten

- Zeichenkette mit Inhalt *Hallo!*

```
01 char *str = "Hallo!";
```

Zeichenketten

- Zeichenkette mit Inhalt *Hallo!*

```
01 char *str = "Hallo!";
```

```
01 char str[7] = { 'H', 'a', 'l', 'l', 'o', '!', '\0'};
```

Zeichenketten

- Zeichenkette mit Inhalt *Hallo!*

```
01 char *str = "Hallo!";
```

```
01 char str[7] = { 'H', 'a', 'l', 'l', 'o', '!', '\0'};
```

```
01 char str[7] = { 72, 97, 108, 108, 111, 33, 0};
```

  - Was passiert, wenn die terminierende `0` vergessen wird?

- Variable mit (einstelliger) Zahl als Zeichenkette

Zeichenketten

- Zeichenkette mit Inhalt *Hallo!*

```
01 char *str = "Hallo!";
```

```
01 char str[7] = { 'H', 'a', 'l', 'l', 'o', '!', '\0'};
```

```
01 char str[7] = { 72, 97, 108, 108, 111, 33, 0};
```

  - Was passiert, wenn die terminierende `0` vergessen wird?

- Variable mit (einstelliger) Zahl als Zeichenkette

```
01 uint8_t i = 7;
02 char str[2];
03 str[0] = i + 48; // 48 = '0'
04 str[1] = 0; // oder '\0'
```

Zeichenketten

- Zeichenkette mit Inhalt *Hallo!*

```
01 char *str = "Hallo!";
```

```
01 char str[7] = { 'H', 'a', 'l', 'l', 'o', '!', '\0'};
```

```
01 char str[7] = { 72, 97, 108, 108, 111, 33, 0};
```

  - Was passiert, wenn die terminierende `0` vergessen wird?

- Variable mit (einstelliger) Zahl als Zeichenkette

```
01 uint8_t i = 7;
02 char str[2];
03 str[0] = i + 48; // 48 = '0'
04 str[1] = 0; // oder '\0'
```

  - Wir geht das mit mehrstelligen Zahlen?

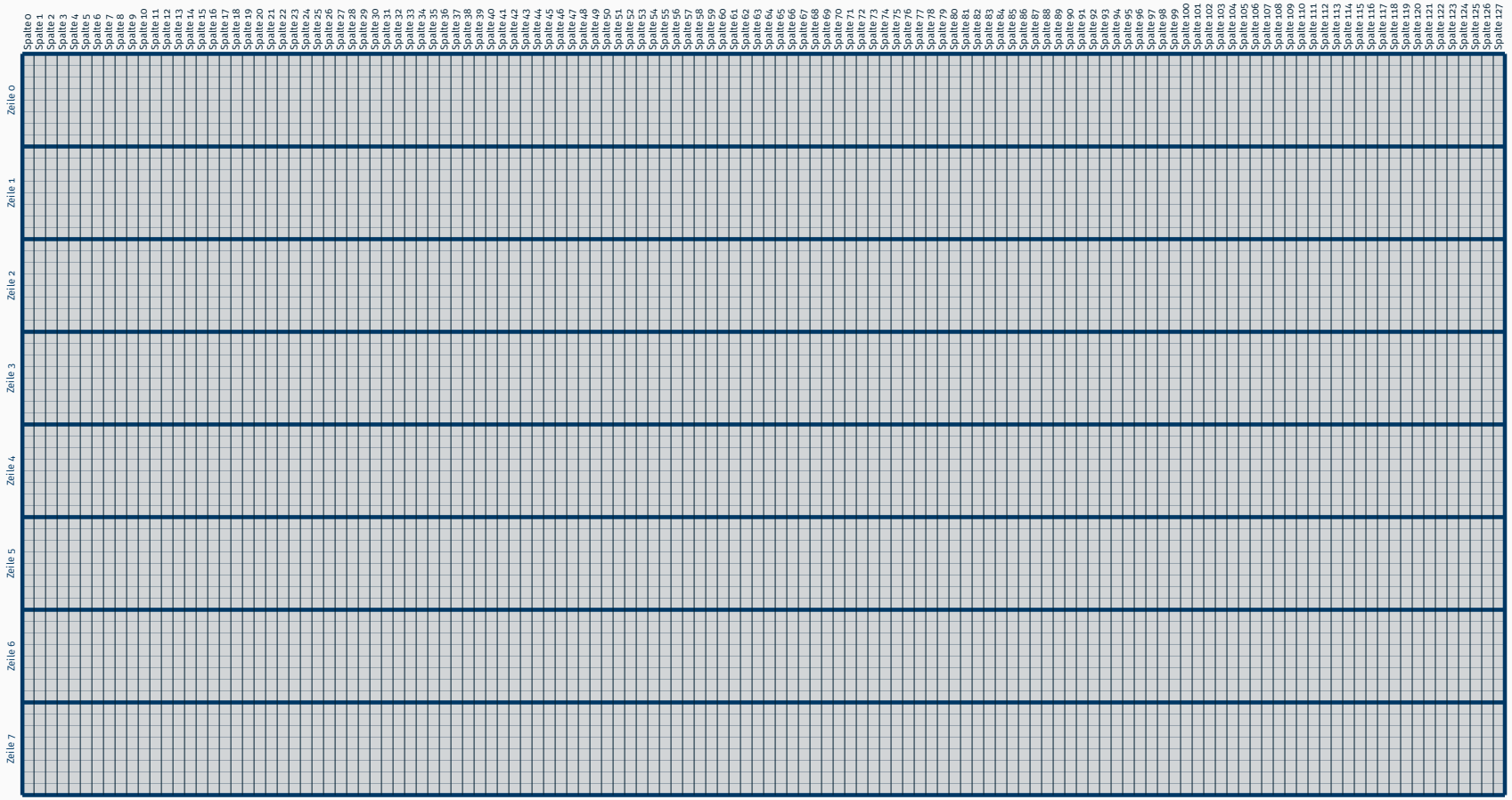
OLED-Display SSD1306

- Monochrom, etwa 2.5 cm Diagonale
- Ansteuerung über I²C
- Auflösung: 128×64 Pixel
  - 128 Spalten und
  - 8 Zeilen (*Pages*) mit je 8 Segmente
  - jedes Segment wird durch ein Byte repräsentiert
- Bibliotheksfunktionen zur Ansteuerung

  `sb_display_available` Prüfen ob Display angeschlossen
  `sb_display_enable` Display aktivieren
  `sb_display_draw` Pixel zeichnen
  `sb_display_showString` Textausgabe

  - Dokumentation lesen!

OLED-Display SSD1306 (Schema)

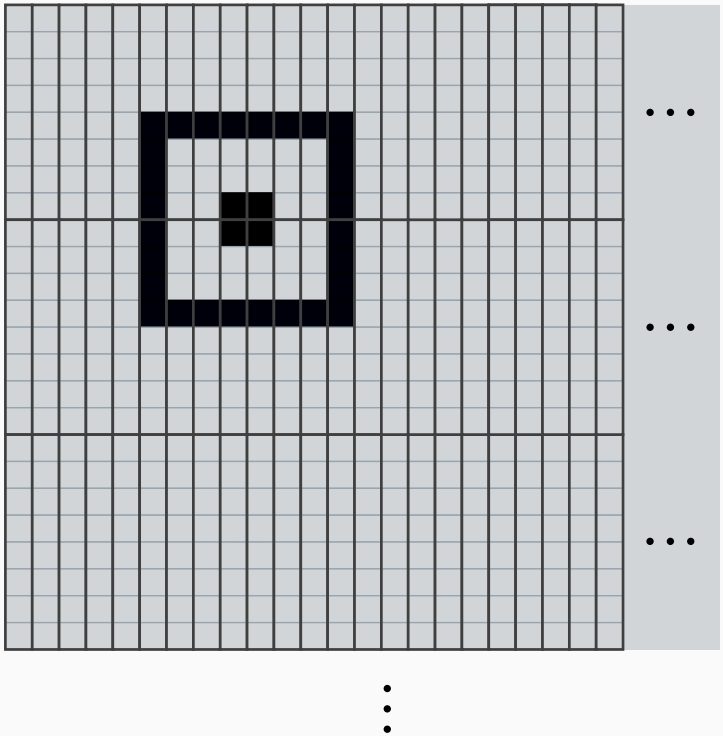


OLED-Display SSD1306 (Beispiel)

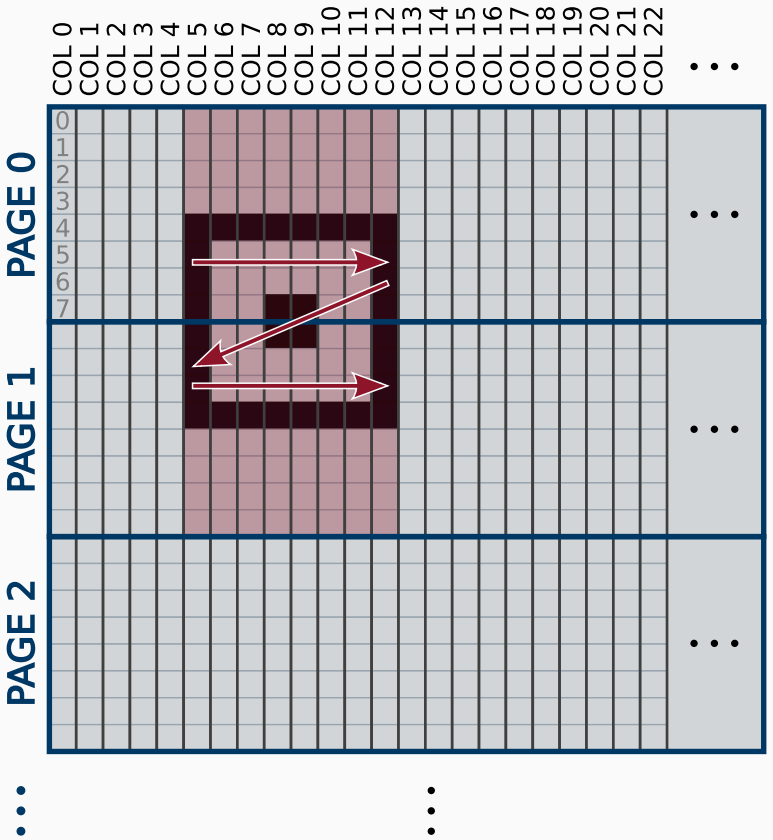
```c
#include <stdint.h>
#include <stddef.h>
#include "display.h"

void main(void){
    // Display vorhanden?
    if (sb_display_enable() == 0){
        // Bildschirm leeren (schwarz)
        sb_display_fillScreen(NULL);
        // Quadrat als Bitmapgrafik
        uint8_t bitmap[] = {
            0xF0, 0x10, 0x10, 0x90, 0x90, 0x10, 0x10, 0xF0,
            0x0F, 0x08, 0x08, 0x09, 0x09, 0x08, 0x08, 0x0F
        };
        // Ab Zeile 0 und Spalte 5 zeichne 2 Zeilen und 8 Spalten
        sb_display_drawBitmap(0, 5, 2, 8, bitmap);
    }
    while(1);
}
```

OLED-Display SSD1306 (Resultat)



OLED-Display SSD1306 (Resultat)



Aufgabe: ADC-Test

Aufgabe: ADC-Test

- Anzeigen der aktuellen Werte von Potentiometer und Fototransistor auf dem Display
  - als Statusbalken und
  - als Zahl
  - mit Beschriftung