

Übungen zu Systemnahe Programmierung in C (SPiC) – Wintersemester 2022

Übung 4

Phillip Raffeck
Maximilian Ott

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

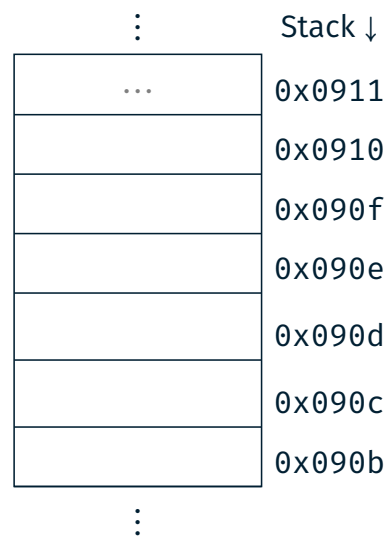
Zeiger & Felder

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;
```

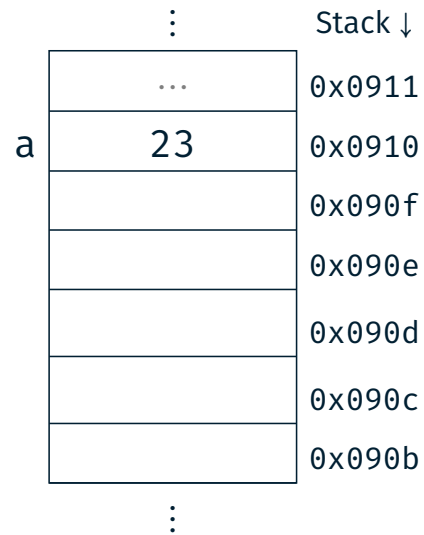




- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```

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;
    
```



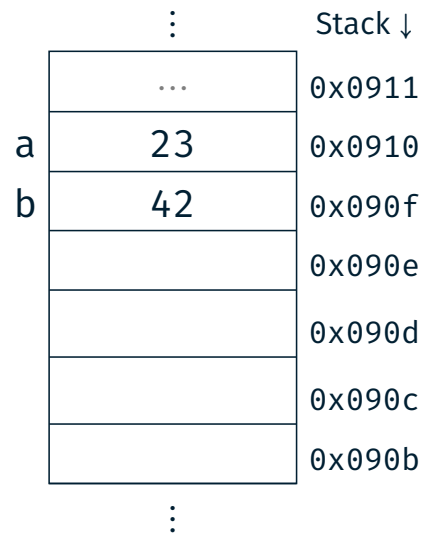
Achtung: Die genaue Anordnung der Variablen auf dem Stack ist abhängig vom Übersetzer und den gewählten Optimierungen!



- 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;
    
```



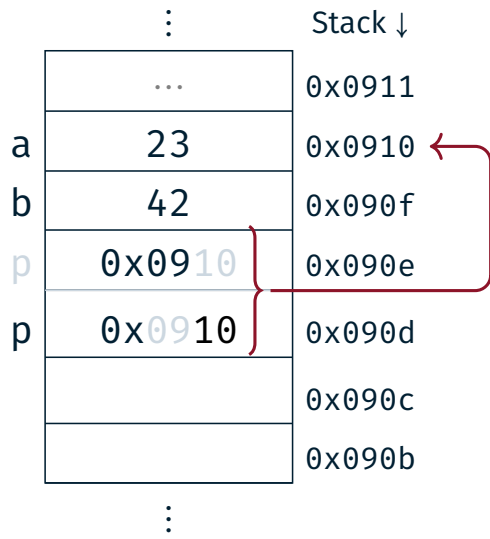
Achtung: Die genaue Anordnung der Variablen auf dem Stack ist abhängig vom Übersetzer und den gewählten Optimierungen!



- 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;
    
```



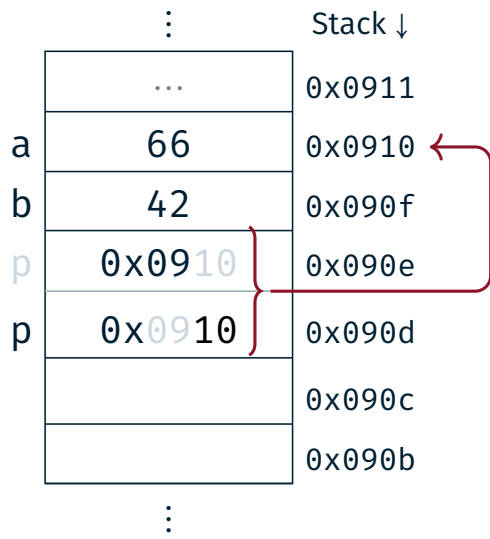
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen



- 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;
    
```



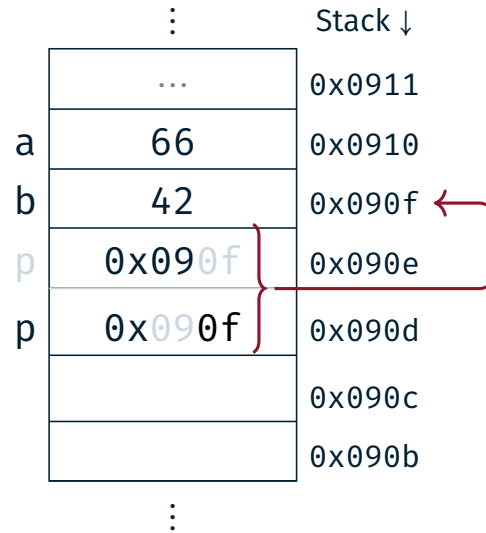
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen



- 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;
    
```



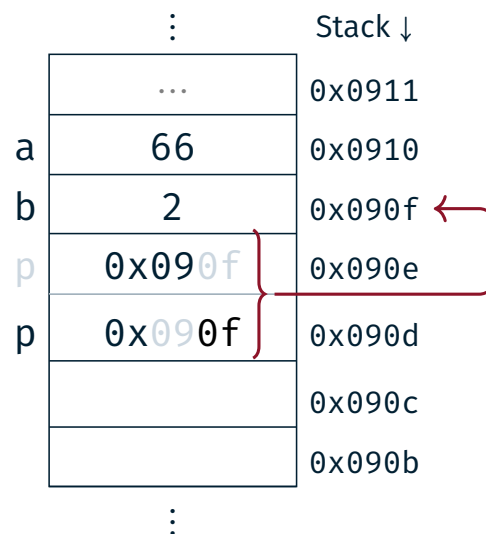
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen



- 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;
    
```



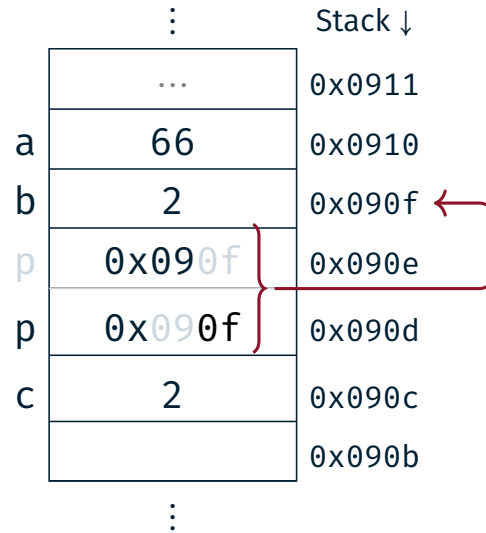
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen



- 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;
    
```



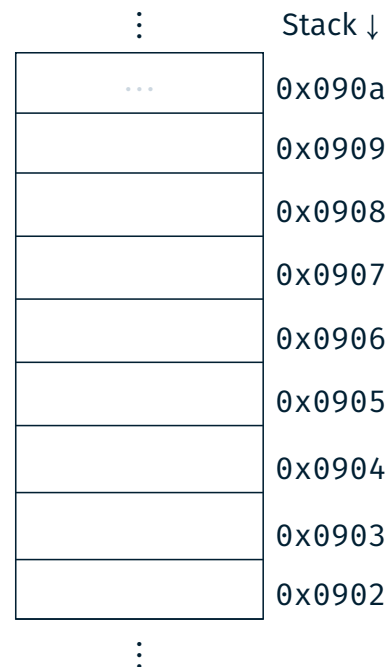
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen



- 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];
    
```

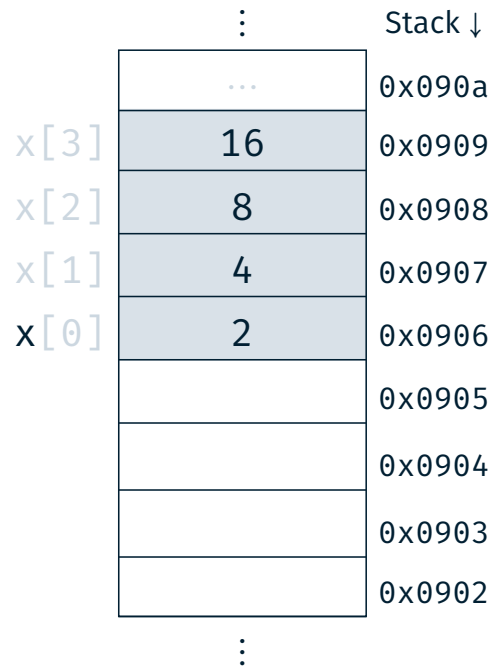




- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```

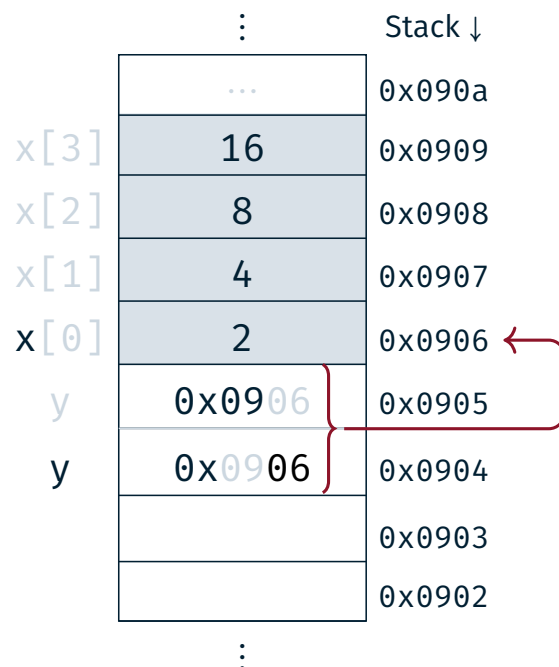
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
    
```



- 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];
    
```

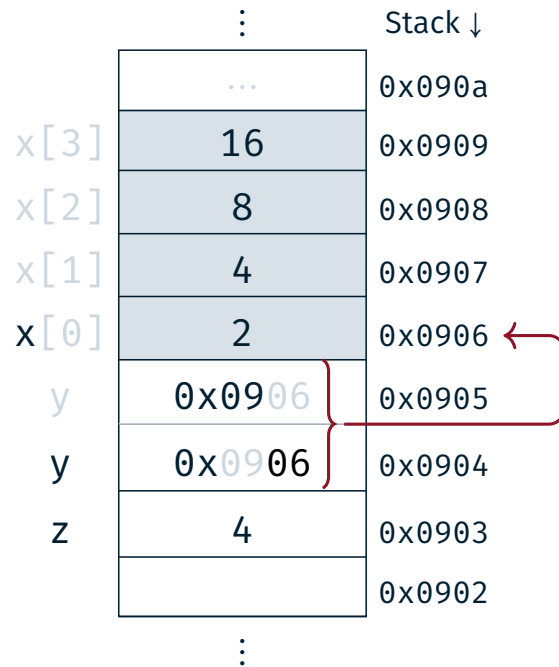




- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```

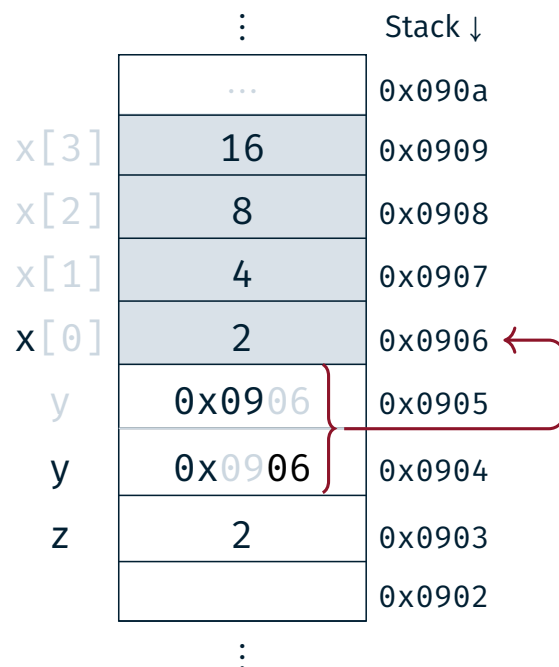
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
    
```



- 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];
    
```

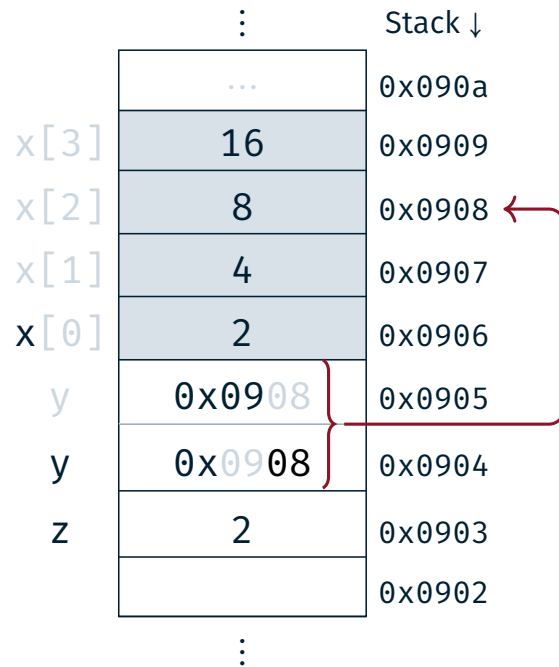




- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```

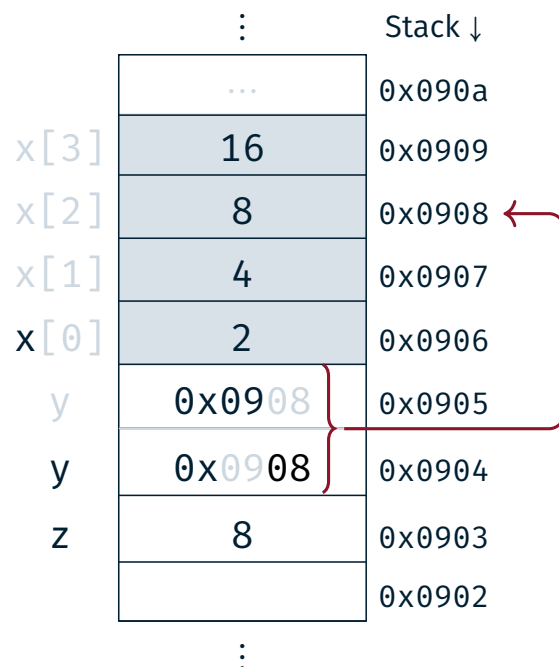
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
    
```



- 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];
    
```

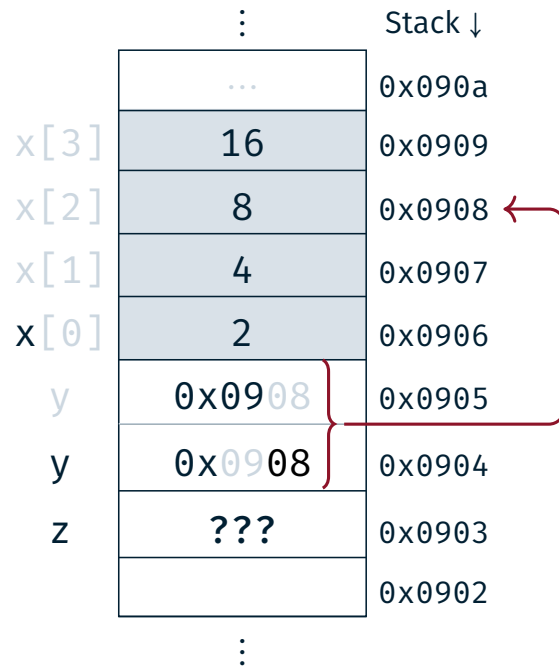




- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```

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]; // ???
    
```



Hands-on: Zeiger

Kein Screencast



- 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), den SPiCsim oder Linux

Quellcode:

<https://sys.cs.fau.de/lehre/WS22/spic/uebung/material/pointer.c>

Hands-on: Laufschrift

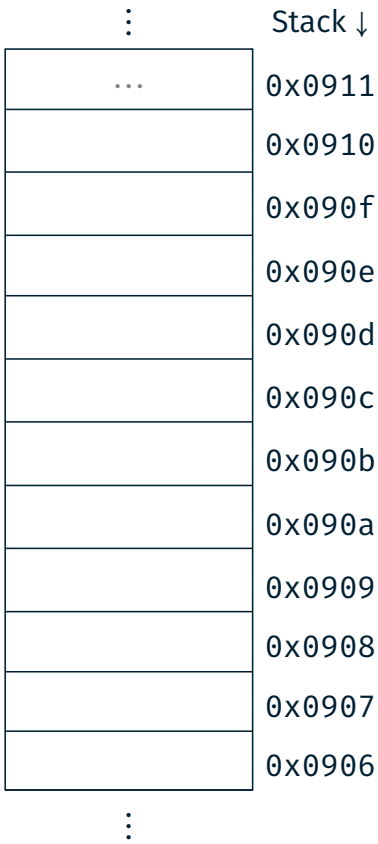
Screencast: <https://www.video.uni-erlangen.de/clip/id/18170>



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

```

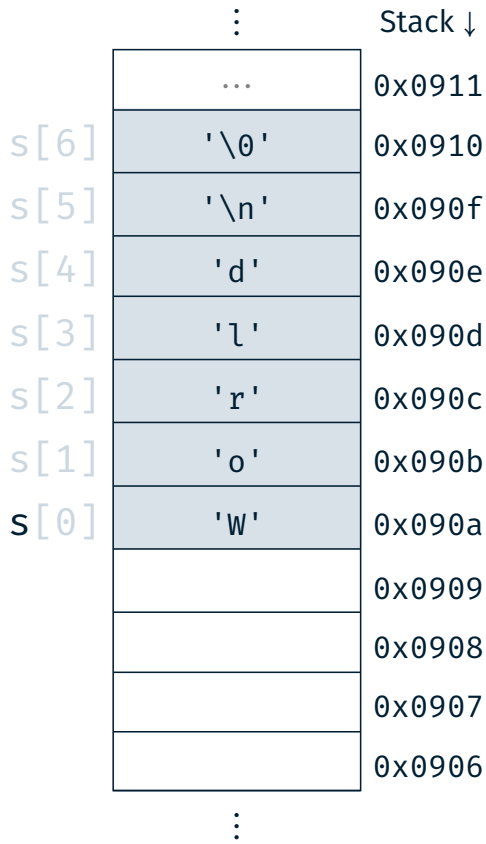
01 char s[] = "World\n";
02 char c = s[0];
03 c = s[4];
04 char *s2 = s + 2;
05 c = s2[1];
    
```



- 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];
    
```

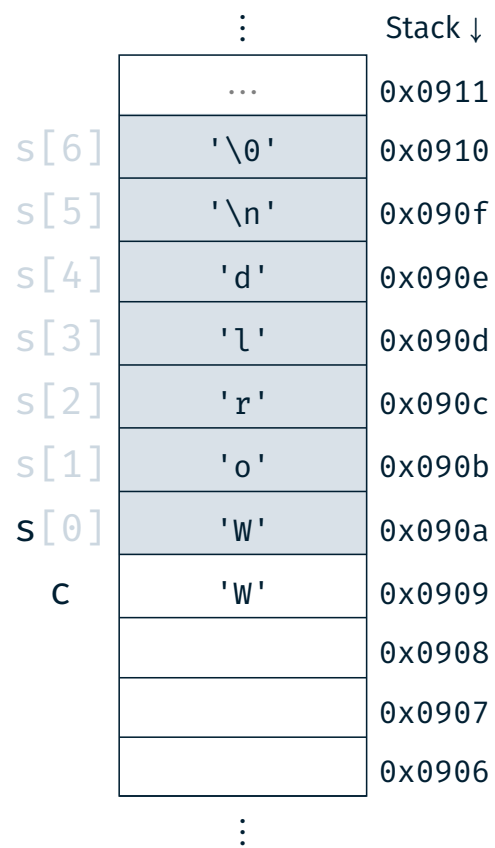




- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: strlen(s) + 1

```

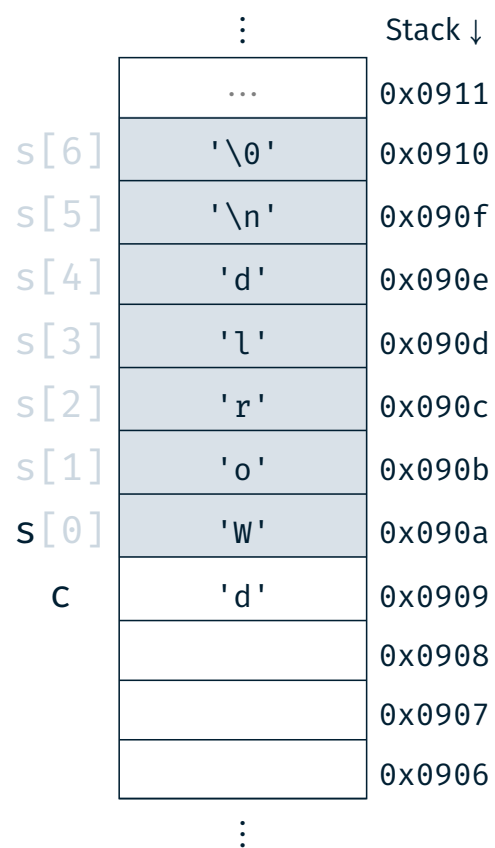
01 char s[] = "World\n";
02 char c = s[0];
03 c = s[4];
04 char *s2 = s + 2;
05 c = s2[1];
    
```



- 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];
    
```

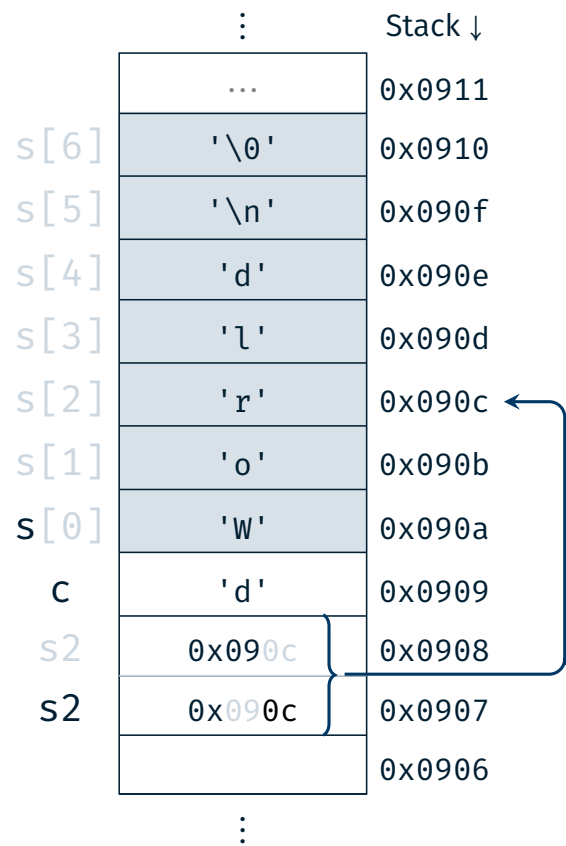




- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: strlen(s) + 1

```

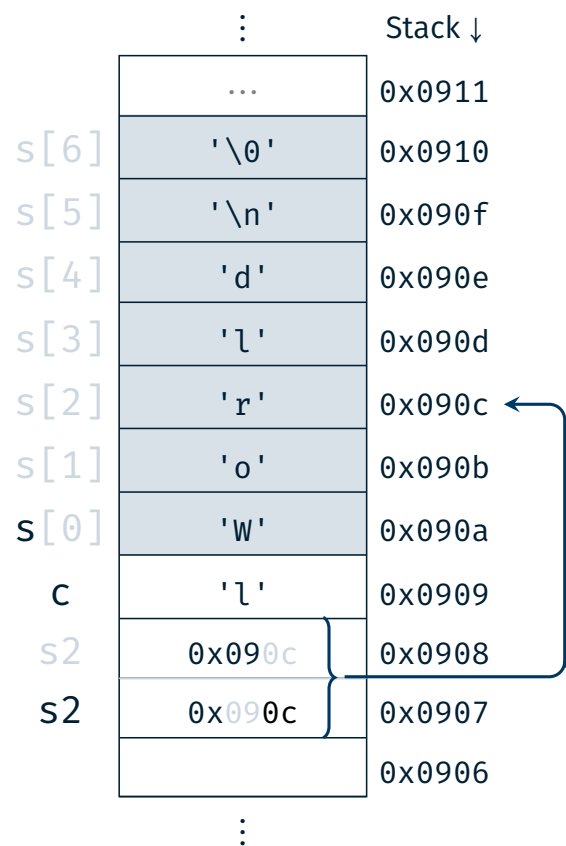
01 char s[] = "World\n";
02 char c = s[0];
03 c = s[4];
04 char *s2 = s + 2;
05 c = s2[1];
    
```



- 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
 - Alarmer & Schlafenlegen
- Vorgehen:
 - Wiederkehrender Alarm mittels TIMER0
 - Zusammensetzen des aktuellen Teilstrings
 - Ausgabe über 7-Segment-Anzeige
 - In Wartephase Mikrocontroller in den Energiesparmodus versetzen (Passives Warten)

7

Hands-on: Laufschrift – Bestimmung des Teilstrings



```
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;
```

8



```
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;
```



8



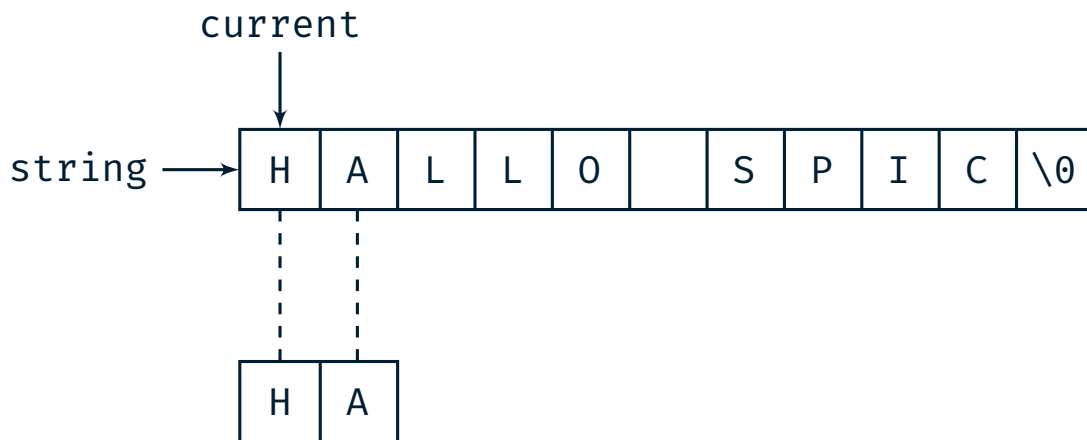
```
01 const char *string = "HALLO SPIC";
02 const char *current = string;
03 // current[0] == 'H' && current[1] == 'A'
04 ++current;
05 // current[0] == 'A' && current[1] == 'L'
06 // [...]
07 // current[0] == '\0', current[1] == ??
08 current = string;
```



8



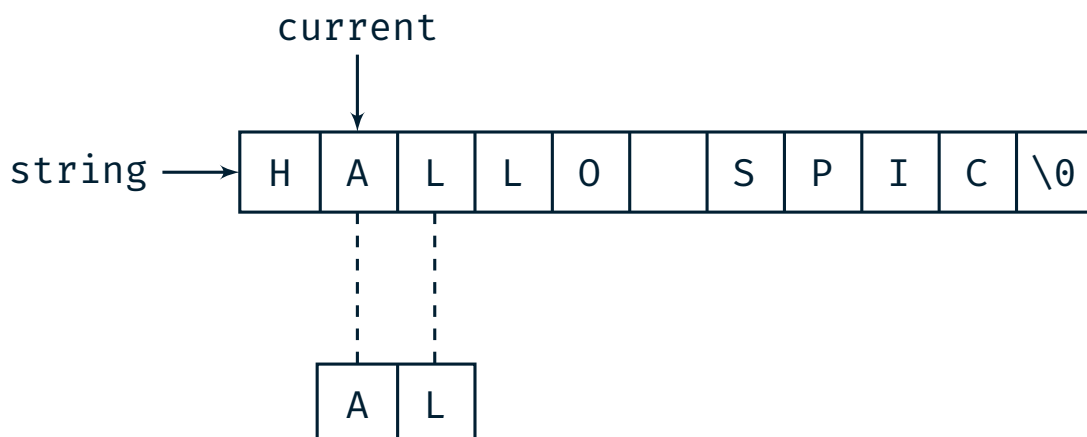
```
01 const char *string = "HALLO SPIC";
02 const char *current = string;
03 // current[0] == 'H' && current[1] == 'A'
04 ++current;
05 // current[0] == 'A' && current[1] == 'L'
06 // [...]
07 // current[0] == '\0', current[1] == ??
08 current = string;
```



8



```
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;
```



8



```
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;
```

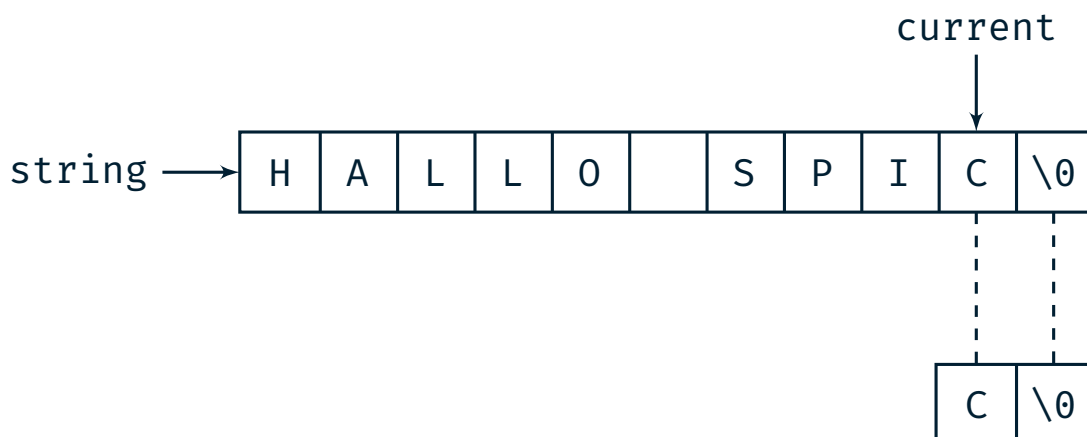


...

8



```
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;
```

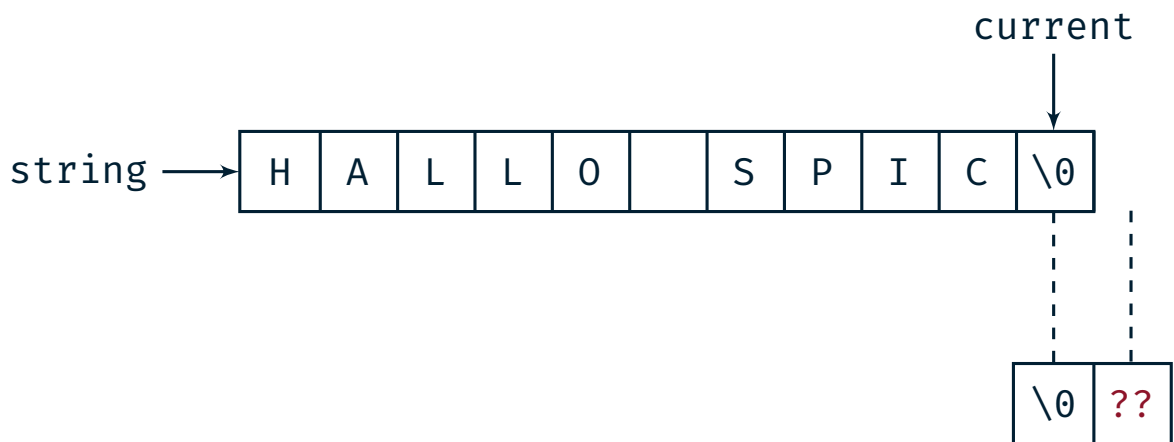


8



```

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;
    
```

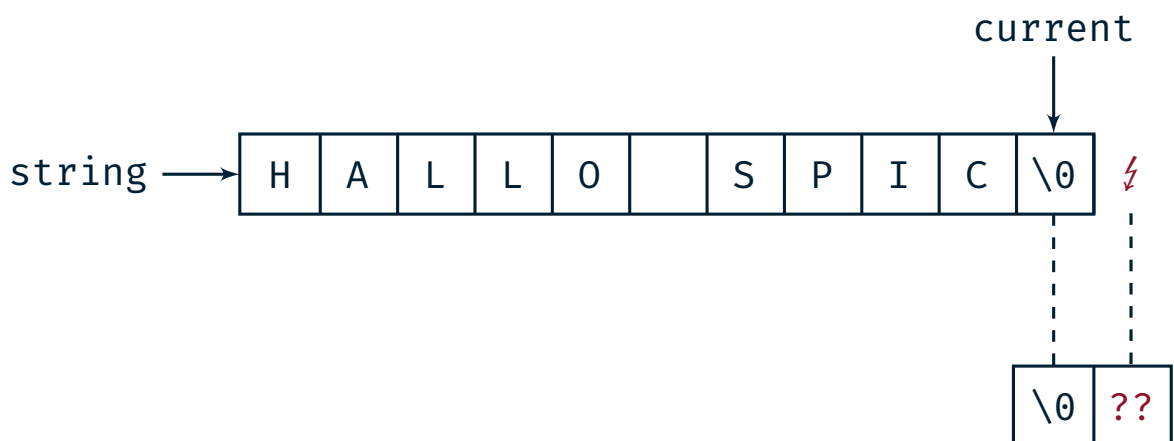


8



```

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;
    
```



8



```
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;
```

