

0x24

`A`

0b10101100

Der Datentyp char*

255



char array
char x[16];



Der Datentyp char*

Das Problem:

- int read (int address, **char * data**, int length,
bool repeated = false)
- int write(int address, const **char * data**, int length,
bool repeated = false)

Die i2c-Befehle benötigen Parameter des Typs **char***?!



Der Datentyp char*

```
char daten[5]={1,2,3,4,5};  
i2c.write(device, daten,5,false)
```

Die Lösung:



Der Datentyp char*

Die Lösung:

```
int X=123456;
```

```
i2c.write(device, (char*)&x, sizeof(x), false)
```



Der Datentyp char*

Die Lösung:

```
float x=123456;
```

```
i2c.write(device, (char*)&x, sizeof(x), false)
```



Der Datentyp char*

Die Lösung:

struct s

```
{ int i=123456;  
  float j=0.8765;
```

```
} x;
```

```
i2c.write(device, (char*)&x, sizeof(x), false)
```



Der Datentyp char*

char ist der kleinste Datentyp mit lediglich 8 Bit können Zahlen von 0 bis 255 oder ASCII-Zeichen gespeichert werden

Grunddatentypen in C STM32:

char x; // 1 Byte, Wertebereich 0..255 oder ASCII-Zeichen 'a', 'b'

int y; // 4 Byte, Wertebereich -2147483648..2147483647

float z; // 4 Byte, Wertebereich $\pm 2^{127}$



Der Datentyp char*

Zwei Arrays:
char t[5]="Hallo";
char w[4]={1,0b10101100,0x34,17};

Die beiden Arrays liegen hier im RAM
des Mikrocontrollers



t: 0x20000000
0x20000001
0x20000002
0x20000003
0x20000004
w: 0x20000005
0x20000006
0x20000007
0x20000008
0x20000009
0x0000000A
0x2000000B
0x2000000C

'H'
'a'
'l'
'l'
'o'
1
0b10101100
0x34
17



Der Datentyp char*

Zwei Arrays:

```
char t[5]="Hallo";  
char w[4]={1,0b10101100,0x34,17};
```

Die beiden Arrays liegen hier im RAM
des Mikrocontrollers



t:	0x20000000
	0x20000001
	0x20000002
	0x20000003
	0x20000004
w:	0x20000005
	0x20000006
	0x20000007
	0x20000008
	0x20000009
	0x0000000A
	0x2000000B
	0x2000000C

'H'
'a'
'l'
'l'
'o'
1
0b10101100
0x34
17



Der Datentyp char*

Zwei Arrays:
char t[5]="Hallo";
char w[4]={1,0b10101100,0x34,17};
char* z;

Die nächste Variable im Speicher ist char* z.
Aber welche Daten beinhaltet sie?



t:	0x20000000	'H'
	0x20000001	'a'
	0x20000002	'l'
	0x20000003	'l'
	0x20000004	'o'
w:	0x20000005	1
	0x20000006	0b10101100
	0x20000007	0x34
	0x20000008	17
z:	0x20000009	
	0x0000000A	
	0x2000000B	
	0x2000000C	



Der Datentyp char*

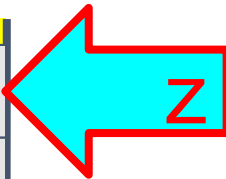
```
Zwei Arrays:  
char t[5]="Hallo";  
char w[4]={1,0b10101100,0x34,17};  
char* z;  
z=t;
```

Jetzt speichert z die Speicheradresse des Arrays t



t:	0x20000000
	0x20000001
	0x20000002
	0x20000003
	0x20000004
w:	0x20000005
	0x20000006
	0x20000007
	0x20000008
z:	0x20000009
	0x0000000A
	0x2000000B
	0x2000000C

	'H'
	'a'
	'l'
	'l'
	'o'
	1
	0b10101100
	0x34
	17
	0x00
	0x00
	0x00
	0x20



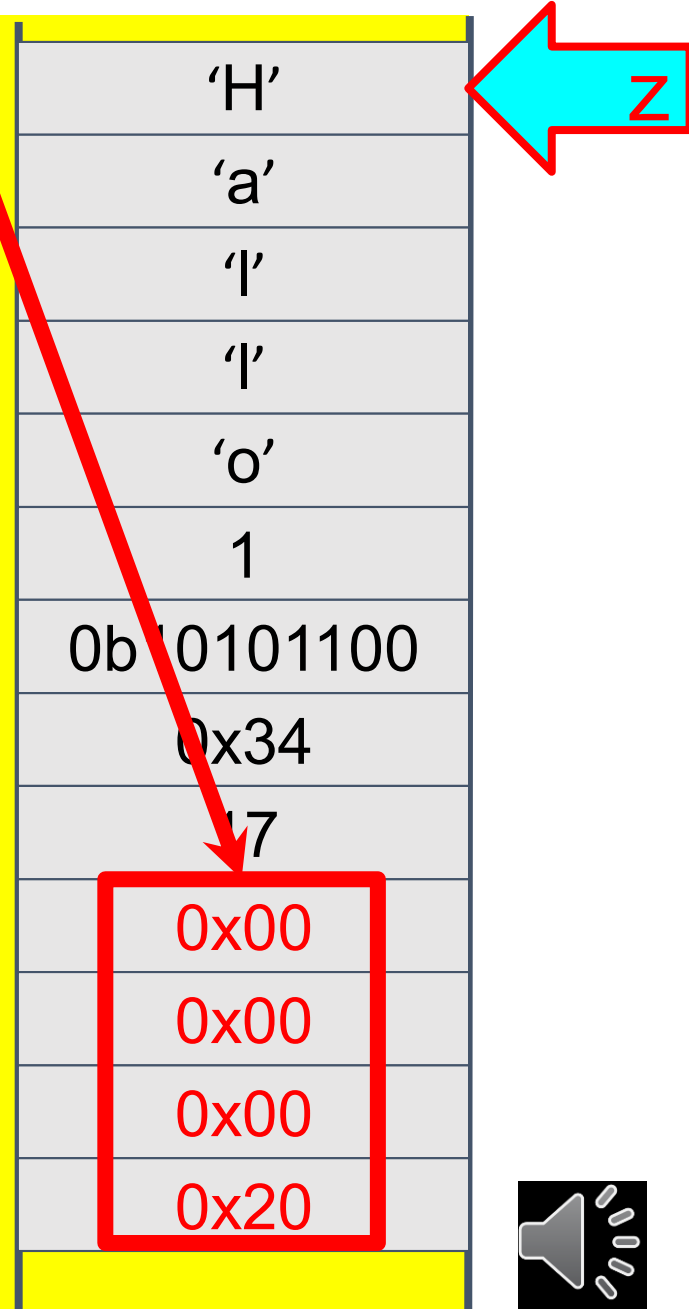
Der Datentyp char*

Zwei Arrays:
char t[5]="Hallo";
char w[4]={1,0b10101100,0x34,17};
char* z;
z=t;

Jetzt speichert z die Speicheradresse des Arrays t



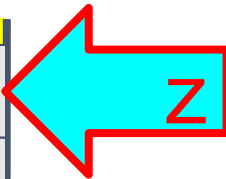
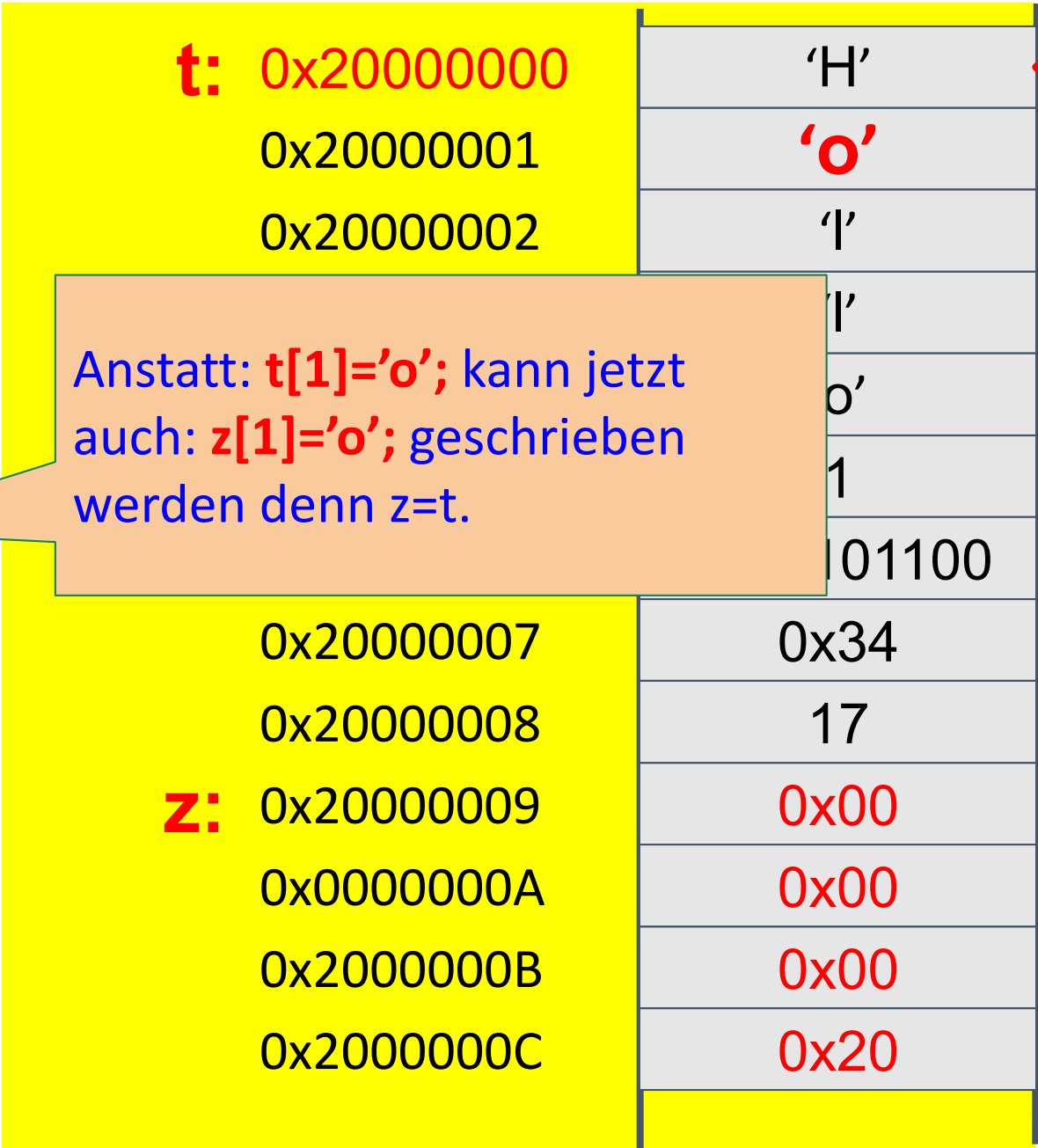
t:	0x20000000
	0x20000001
	0x20000002
	0x20000003
	0x20000004
w:	0x20000005
	0x20000006
	0x20000007
	0x20000008
z:	0x20000009
	0x0000000A
	0x2000000B
	0x2000000C



Der Datentyp char*

```
Zwei Arrays:  
char t[5]="Hallo";  
char w[4]={1,0b10101100,0x34,17};
```

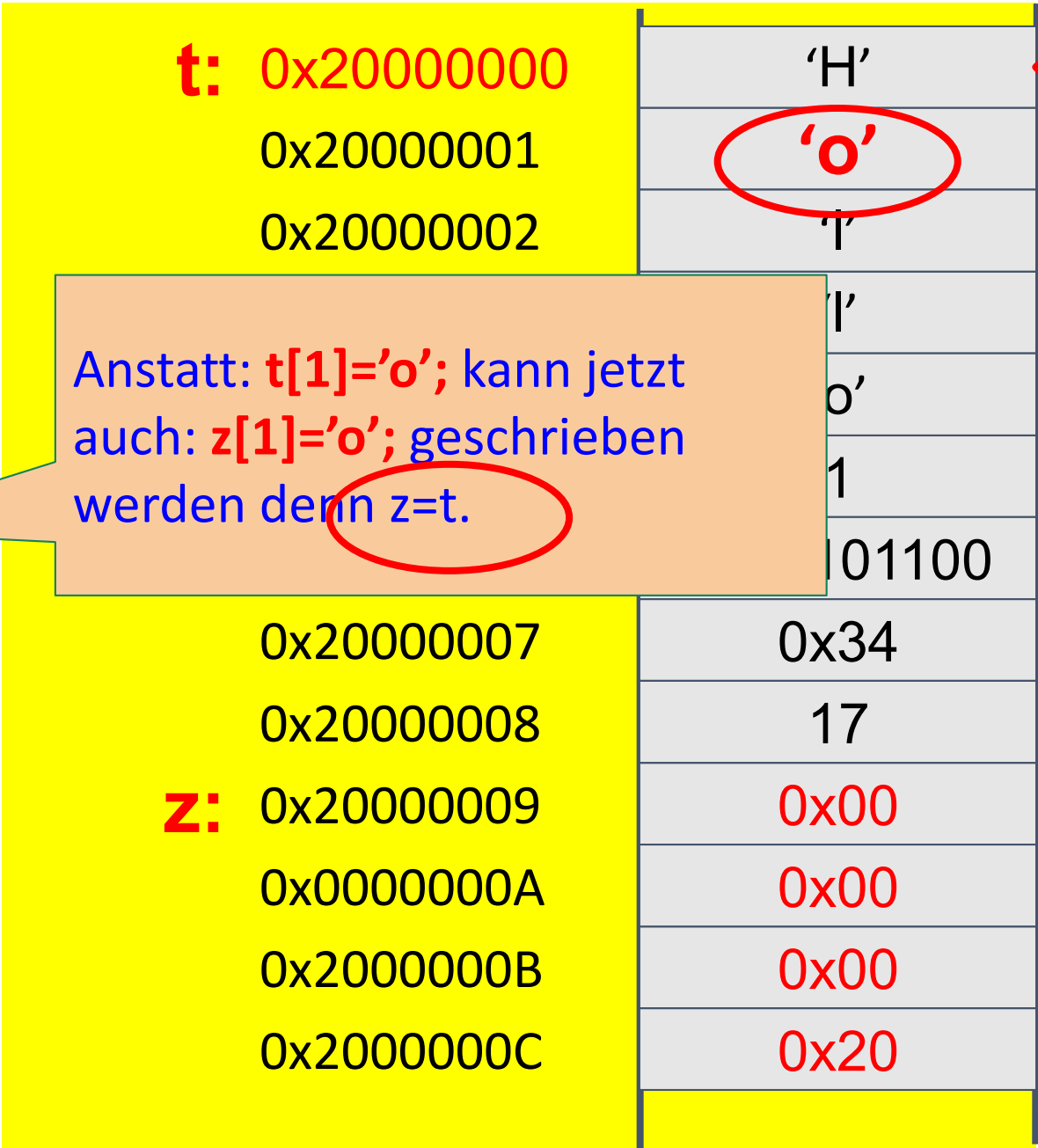
```
char* z;  
z=t;  
z[1]='o';
```



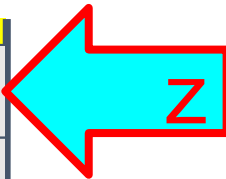
Der Datentyp char*

Zwei Arrays:
char t[5]="Hallo";
char w[4]={1,0b10101100,0x34,17};

char* z;
z=t;
z[1]='o';



Anstatt: **t[1]='o'**; kann jetzt auch: **z[1]='o'**; geschrieben werden denn **z=t**.



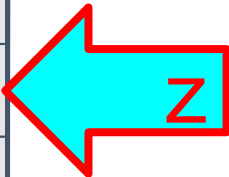
Der Datentyp char*

```
Zwei Arrays:  
char t[5]="Hallo";  
char w[4]={1,0b10101100,0x34,17};  
char* z;  
z=w;
```

Jetzt speichert z die Speicheradresse des Arrays w, bzw. des chars mit dem Wert 1



t:	0x20000000	'H'
	0x20000001	'a'
	0x20000002	'l'
	0x20000003	'l'
	0x20000004	'o'
w:	0x20000005	1
	0x20000006	0b10101100
	0x20000007	0x34
	0x20000008	17
z:	0x20000009	0x05
	0x0000000A	0x00
	0x2000000B	0x00
	0x2000000C	0x20



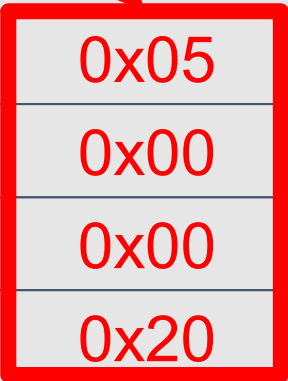
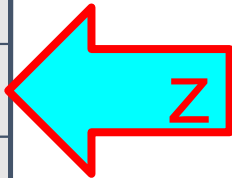
Der Datentyp char*

Zwei Arrays:
char t[5]="Hallo";
char w[4]={1,0b10101100,0x34,17};
char* z;
z=w;

Jetzt speichert z die Speicheradresse des Arrays w, bzw. des chars mit dem Wert 1



t:	0x20000000	'H'
	0x20000001	'a'
	0x20000002	'l'
	0x20000003	'l'
	0x20000004	'o'
w:	0x20000005	1
	0x20000006	0b10101100
	0x20000007	0x34
	0x20000008	17
z:	0x20000009	0x05
	0x0000000A	0x00
	0x2000000B	0x00
	0x2000000C	0x20



Der Datentyp char*

Zwei Arrays:

```
char t[5]="Hallo";
```

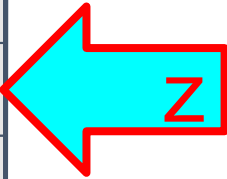
```
char w[4]={1,0b10101100,0x34,17};
```

```
char* z;
```

```
z=w;
```

t: 0x20000000
0x20000001
0x20000002
0x20000003
0x20000004
w: 0x20000005
0x20000006
0x20000007
0x20000008
z: 0x20000009
0x0000000A
0x2000000B
0x2000000C

z ist ein Zeiger auf
Speicherplätze mit char



0x05

0x00

0x00

0x20



Der Datentyp char*

Zwei Arrays:

```
char t[5]="Hallo";
```

```
char w[4]={1,0b10101100,0x34,17};
```

```
char* z;
```

```
z=w;
```

```
z[2]=123;
```

z[2]=123; ist äquivalent
zu

```
w[2]=123;
```

da **z=w**



t: 0x20000000

0x20000001

0x20000002

0x20000003

0x20000004

w: 0x20000005

0x20000006

0x20000007

0x20000008

z: 0x20000009

0x0000000A

0x2000000B

0x2000000C

'H'

'a'

'l'

'l'

'o'

1

0b10101100

123

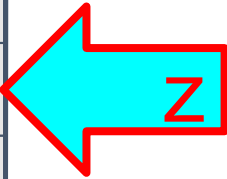
17

0x05

0x00

0x00

0x20



Der Datentyp char*

Zwei Arrays:

```
char t[5]="Hallo";
```

```
char w[4]={1,0b10101100,0x34
```

```
char* z;
```

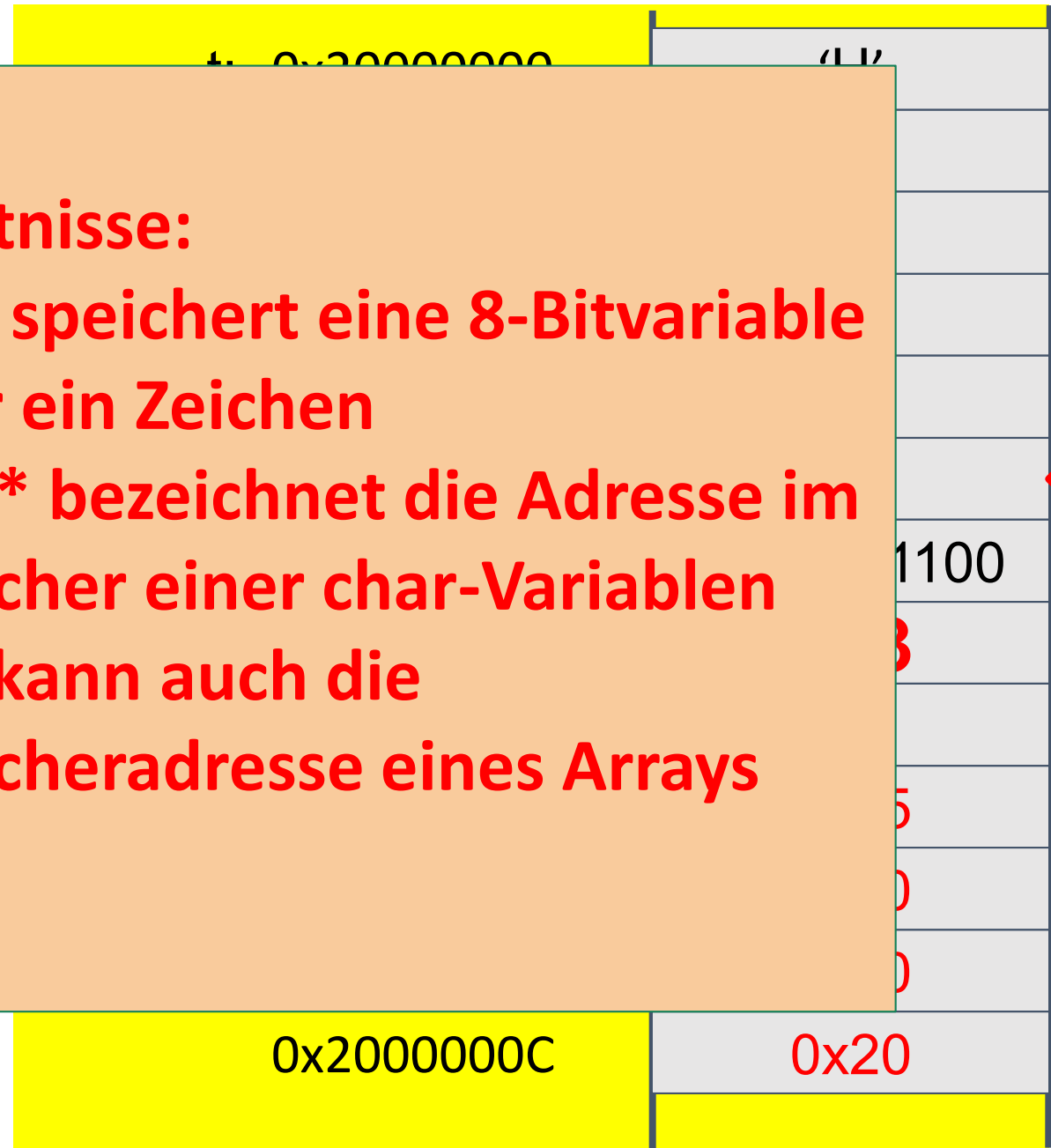
```
z=w;
```

```
z[2]=123;
```



Erkenntnisse:

- char speichert eine 8-Bitvariable oder ein Zeichen
- char* bezeichnet die Adresse im Speicher einer char-Variablen
- Das kann auch die Speicheradresse eines Arrays sein



Der Datentyp char*

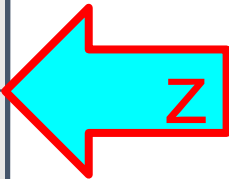
```
void VtoA(char* z, int anz)
{ for(int i=0;i<anz;i++)
  { if(z[i]=='e' || z[i]=='i' |
    { z[i]='a';
    }
  }
}

void main(void)
{
char t[5]="Hallo";
VtoA(t, 5);
}
```



z[i]='u')

t: 0x20000000		'H'
0x20000001		'a'
0x20000002		'l'
char* als Parameter		'l'
		'o'
		1
0x20000006		0b10101100
0x20000007		123
0x20000008		17
z: 0x20000009		0x00
0x0000000A		0x00
0x2000000B		0x00
0x2000000C		0x20



Der Datentyp char*

```
void VtoA(char* z, int anz)
{ for(int i=0;i<anz;i++)
  {
    if(z[i]=='e' || z[i]=='i' || z[i]=='o' || z[i]=='u')
      { z[i]='a';
      }
  }
}

void main(void)
{
  char t[5]="Hallo";
  VtoA(t, 5);
}
```



t: 0x20000000

0x20000001

20000002

20000003

20000004

20000005

20000006

20000007

20000008

'H'

'a'

'l'

'l'

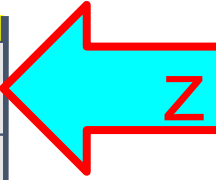
'o'

1

0b10101100

123

17



z: 0x2

0x0

0x2

0x2

**char* z wird wie ein
normales Array
verwendet**



Der Datentyp char*

```
void VtoA(char* z, int anz)
{ for(int i=0;i<anz;i++)
  {
    if(z[i]=='e' || z[i]=='i' || z[i]=='o' || z[i]=='u')
      { z[i]='a';
      }
  }
}

void main(void)
{
  char t[5]="Hall
  VtoA(t, 5)
}
```



Die Anzahl anz der
Arrayfelder wird als
Wert übergeben

t: 0x20000000
0x20000001
0x20000002
0x20000003
0x20000004
w: 0x20000005
0x20000006
0x20000007

0x2000000C

'H'

'a'

'l'

'l'

'a'

1

0b10101100

123

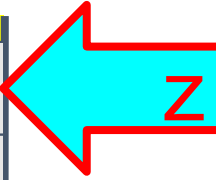
17

0x00

0x00

0x00

0x20



Der Datentyp char*

```
void VtoA(char* z, int anz)
{ for(int i=0;i<anz;i++)
  {
    if(z[i]=='e' || z[i]=='i' || z[i]=='o' || z[i]=='u')
      { z[i]='a';
      }
  }
}

void main(void)
{
  char t[5]="Hall
  VtoA(t, 5)
}
```



t: 0x20000000
0x20000001
0x20000002
0x20000003
0x20000004
w: 0x20000005
0x20000006
0x20000007

Vom Array selbst
wird nur die Adresse
übergeben

0x2000000C

'H'

'a'

'l'

'l'

'a'

1

0b10101100

123

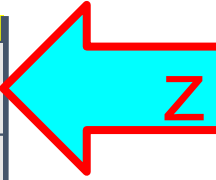
17

0x00

0x00

0x00

0x20



Der Datentyp char*

```
void VtoA(char* z, int anz)
{
    for(int i=0;i<anz;i++)
    {
        if(z[i]=='e' || z[i]=='i' || z[i]=='o' || z[i]=='u')
        {
            z[i]='a';
        }
    }
}

void main(void)
{
    char t[5]="Hallo";
    VtoA(t, 5);
}
```



t: 0x20000000

0x20000001

0x20000002

0x20000003

0x2

0x2

0x2

0x2

0x2

z: 0x2

0x0

0x2

0x2

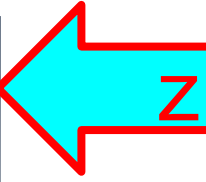
'H'

'a'

'l'

'l'

Die Anweisungen
der Operation
arbeiten direkt mit
dem Original!
Eine Return-
Anweisung ist
überflüssig!



Der Datentyp char*

xc: 0x20000000
0x20000001
0x20000002
0x20000003

123
0
0
0

char xc[4]={123, 0, 0, 0}

Ein char-Array



x: 0x20000000
0x20000001
0x20000002
0x20000003

123
0
0
0

int x=123;



Der Datentyp char*

xc: 0x20000000
0x20000001
0x20000002
0x20000003

123
0
0
0

char xc[4]={123, 0, 0, 0};

Eine int-Variable



x: 0x20000000
0x20000001
0x20000002
0x20000003

123
0
0
0

int x=123;



Der Datentyp char*

xc: 0x20000000
0x20000001
0x20000002
0x20000003

123
0
0
0

char xc[4]={123, 0, 0, 0};



x: 0x20000000
0x20000001
0x20000002
0x20000003

123
0
0
0

int x=123;

**Ergibt im Speicher die
identisch Bytefolge**

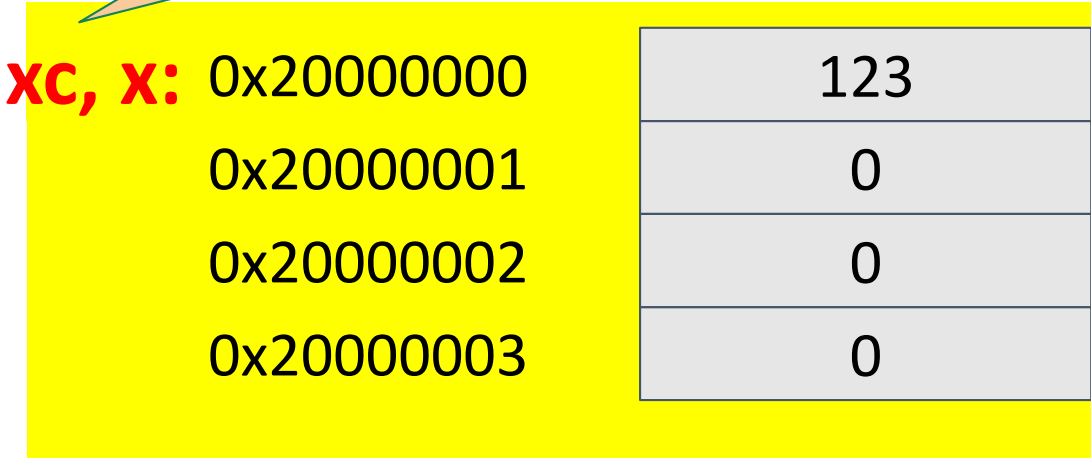


Der Datentyp char*



```
char xc[4]={123, 0, 0, 0};
```

```
int x=123;
```



```
int x=123;  
char* xc=(char *)&x;
```



Der Datentyp char*

xc, x: 0x20000000
0x20000001
0x20000002
0x20000003

123
0
0
0

```
int x=123;
```

```
char* xc=(char*)&x;
```

Was bedeutet das
denn?



Der Datentyp char*

x=123

xc, x: 0x20000000

0x20000001

0x20000002

0x20000003

123

0

0

0

```
int x=123;
```

```
char* xc=(char*)&x;
```



x=123 Wert der
Variablen



Der Datentyp char*

x=123

&x= 0x20000000

xc, x: 0x20000000

0x20000001

0x20000002

0x20000003

123

0

0

0

int x=123;

char* xc=(char*)&x;



&x=0x20000000
Speicheradresse der
Variablen x



Der Datentyp char*

x=123

&x= 0x20000000

(char*)&x=0x20000000

**Adresse des char-
Arrays**

xc, x: 0x20000000

0x20000001

0x20000002

0x20000003

123

0

0

0

int x=123;

char* xc=(char*)&x;



(char*)&x=0x20000000
**Speicheradresse des char-
Arrays**



Der Datentyp char*

x=123

&x= 0x20000000

(char*)&x=0x20000000

**Adresse des char-
Arrays**

xc, x: 0x20000000

0x20000001

0x20000002

0x20000003

123

0

0

0

float x=123.345;

char* xc=(char*)&x;



**Das geht auch mit:
float x;
und allen anderen
Datentypen, auch
mit structs**



Der Datentyp char*

x=123

&x= 0x20000000

(char*)&x=0x20000000

**Adresse des char-
Arrays**

**xc[0]=200; bewirkt
dass x=200 wird!!!!**

xc, x: 0x20000000

0x20000001

0x20000002

0x20000003

123

0

0

0

int x=123;

char* xc=(char*)&x;



**Char-Array xc und int
Variable x belegen den
selben Speicherplatz.
Änderungen wirken sich
deshalb doppelt aus**



Der Datentyp char*

xc, x: 0x20000000
0x20000001
0x20000002
0x20000003

123
0
0
0

```
int x=123;
```

```
char* xc=(char *)&x;
```



**Ganz schön, aber
wozu ist das gut?**



Der Datentyp char*

```
int write (    int  
            address, char *  
            data,  
            int  length,  
            bool repeated = false  
            )
```



xc, x: 0x20000000
0x20000001
0x20000002
0x20000003

123
0
0
0

```
int x=123;  
char* xc=(char *)&x;
```

**Die I2C-write-
Operation möchte
als Daten ein char-
Array!**



Der Datentyp char*

```
int write (    int  
            address, char *  
            data,  
            int  length,  
            bool repeated = false  
            )
```

Übertragen werden
soll aber die int-
Variable int x;

xc, x: 0x20000000
 0x20000001
 0x20000002
 0x20000003

123
0
0
0



```
int x=123;  
char* xc=(char *)&x;
```



Der Datentyp char*

```
int write (    int  
            address, char *  
            data,  
            int  length,  
            bool repeated = false  
            )
```

Die Lösung: x einfach
als char-Array
übertragen

xc, x: 0x20000000
 0x20000001
 0x20000002
 0x20000003

123
0
0
0

```
int x=123;  
char* xc=(char *)&x;
```

```
2c.write(deviceadr,  
         (char*)&x,  
         sizeof(x),  
         false);
```



Der Datentyp char*

```
int write (    int
            address, char *
            data,
            int length,
            bool repeated = false
)
```

Die Anzahl der Bytes (length) kann mittels der Operation sizeof() ermittelt werden.

xc, x: 0x20000000
0x20000001
0x20000002
0x20000003

123
0
0
0

```
int x=123;
char* xc=(char *)&x;
```

```
write(deviceadr,
      (char*)&x,
      sizeof(x),
      false);
```



Der Datentyp char*

```
int write ( int  
            address, char *  
            data,  
            int length,  
            bool repeated = false  
        )
```



float ändert nichts!!!
Der Ausgangsdattentyp ist
egal

xc, x: 0x20000000
 0x20000001
 0x20000002
 0x20000003

123
0
0
0

float x=123.23e-6;

char* xc=(char *)&x;

i2c.write(deviceadr,
 (char*)&x,
 sizeof(x),
 false);



Der Datentyp char*

```
int read (    int
            address, char *
            data,
            int length,
            bool repeated = false
            )
```

Beim read-Befehl geht
es genauso



xc, x: 0x20000000
0x20000001
0x20000002
0x20000003

123
0
0
0

float x=123.23e-6;

char* xc=(char *)&x;

i2c.read(deviceadr,
(char*)&x,
sizeof(x),
false);



Der Datentyp char*

Zwei Arrays:

```
char t[5]="Ha
```

```
char w[4]={1
```

```
char* z;
```

```
z=w;
```

```
z[2]=123
```

Erkenntnisse:

- **char speichert eine 8-Bitvariable oder ein Zeichen**
- **char* bezeichnet die Adresse im Speicher einer char-Variablen**
- **Das kann auch die Speicheradresse eines Arrays sein**



Der Datentyp char*

Zwei Arrays:

```
char t[5]="Ha
```

```
char w[4]={1
```

```
char* z;
```

```
z=w;
```

```
z[2]=123
```

Erkenntnisse:

- char speichert eine 8-Bitvariable oder ein Zeichen
- char* bezeichnet die Adresse im Speicher einer char-Variablen
- Das kann auch die Speicheradresse eines Arrays sein
- char* als Formalparameter: void op(char* x) benötigt als Aktualparameter ein char-Array:
char t[5]="Hallo"; op(t);



Der Datentyp char*

Zwei Arrays:

```
char t[5]="Ha
```

```
char w[4]={1
```

```
char* z;
```

```
z=w;
```

```
z[2]=123
```

Erkenntnisse:

- **char speichert eine 8-Bitvariable oder ein Zeichen**
- **char* bezeichnet die Adresse im Speicher einer char-Variablen**
- **Das kann auch die Speicheradresse eines Arrays sein**
- **char* als Formalparameter: void op(char* x) benötigt als Aktualparameter ein char-Array:**
char t[5]="Hallo"; op(t);
- **Alle Variablen belegen Bytes und sind deshalb auch char-Arrays: (char*)&Variable**



Der Datentyp char*

Zwei Arrays:

```
char t[5]="Hallo"
```

```
char w[4]={1,0b
```

```
char* z;
```

```
z=w;
```

```
z[2]=123;
```

Erkenntnisse:

- char speichert eine 8-Bitvariable oder ein Zeichen
- char* bezeichnet die Adresse im Speicher einer char-Variablen
- Das kann auch die Speicheradresse eines Arrays sein
- char* als Formalparameter: void op(char* x) benötigt als Aktualparameter ein char-Array:
char t[5]="Hallo"; op(t);
- Alle Variablen belegen Bytes und sind deshalb auch char Arrays: (char*)&Variable
- Deshalb auch möglich: int x; op((char*)&x);

