

Pointer in C

Inhalt

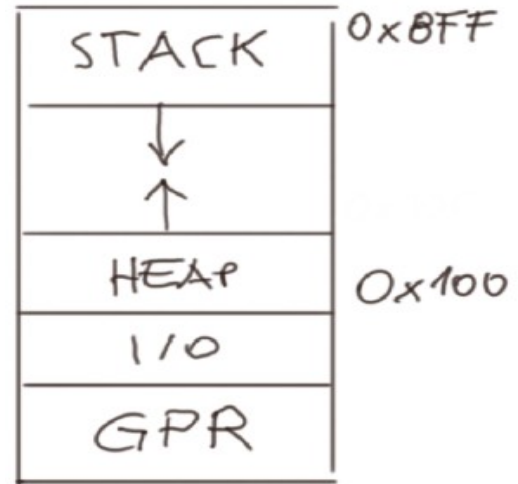
Variable und Pointer in C.....	2
Speichermodell des ATMEGA328p.....	4
Variable und Konstante.....	6
Datentyp char.....	8
Datentyp int.....	12
Datentyp long.....	14
Datentyp long long.....	14
Pointer.....	16
Deklaration eines Pointers und Initialisierung.....	18
Pointer auf Variable Zeigen lassen.....	20
Arrays in C.....	22
Deklaration.....	22
Arrays in C.....	24
Zugriff auf die Elemente des Arrays.....	24
Array = konstanter Zeiger.....	26
Zugriff über Index.....	27
nicht erlaubt:.....	27
Zeiger auf ein Array.....	28
Speicherplatz reservieren: statisch.....	38
Speicherplatz reservieren: dynamisch.....	40
Stimulus File für Zufallszahlen.....	42

Random Data File (Intel Hex Format).....	43
Beispiele.....	44
Speicherbereich löschen.....	44
Beispiele AVR.....	45
Parameterübergabe an Unterprogramme.....	46

Speichermodell des ATMEGA328p

Stack: lokale Variable und Rücksprungadressen, nicht initialisiert

Heap: globale Variable, automatisch initialisiert auf 0



Variable und Konstante

Eine Variable ist ein Speicherplatz im DATA Memory

Der Datentyp gibt an, wie groß dieser Speicher ist und welche Zahlenwerte enthalten sind (wie also das Bitmuster interpretiert werden soll)

uint8_t, int8_t, uint16_t, int16_t usw. benötigen evtl. #include <stdint.h>

Eine Konstante ist ein Speicherplatz, der nicht verändert werden darf. Sie können global am Heap oder lokal am Stack vereinbart werden.

```
const float PI = 3.14;
```

Konstanten werden vom Compiler automatisch in den Datenspeicher kopiert und belasten so den Speicher. Man kann den Compiler zwingen, die Konstante im PROGMEM zu belassen und bei Bedarf von dort zu holen; das funktioniert nur mit statischen und globalen Variablen!

PGM_P ... program memory pointer

```
static const uint8_t c2 PROGMEM = 23;    // oder globale Variable
uint8_t x = pgm_read_byte((PGM_P>(&c2));
oder
```

```
PGM_P p = &c2; x = pgm_read_byte(p);
```

Datentyp char

8bit (je nach Compiler mit oder ohne Vorzeichen)

unsigned char, signed char

```
char c1 = 'A'; char c2 = 0x41; char c3 = 65; char c4 = 0b01000001;
```

The screenshot shows a debugger window with the following code:

```
char c6 = -15; c6++;  
signed char c7 = -15; c7++;
```

The Watch 1 window shows:

Name	Value	Type
c6	242	char(data)@0x08ee ([R28]+5)
c7	-14	signed char(data)@0x08ef ([R28]

The Memory 1 window shows:

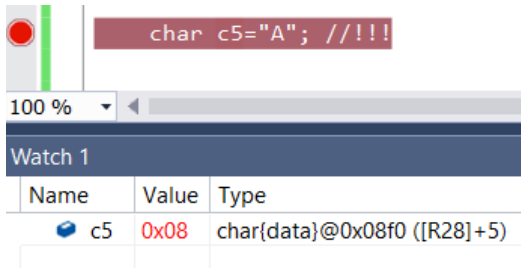
Memory	Address	Value
data	0x08EE	f2 f2 00 00
data	0x0908	?? ?? ?? ??
data	0x0922	?? ?? ?? ??

An orange circle highlights the memory address 0x08EE and the value f2 f2 00 00 in the Memory 1 window, indicating that the same bit pattern is being interpreted differently by the compiler for unsigned and signed char.

man sieht: das gleiche Bitmuster wird unterschiedlich interpretiert.

```
char c1 = "A"; //?????
```

Achtung! Das folgende funktioniert ohne Fehlermeldung, aber macht keinen Sinn ("A" ist eine Stringkonstante)



The screenshot shows a debugger interface. At the top, a code snippet is displayed: `char c5="A"; //!!!`. Below the code, a zoom level of 100 % is indicated. The 'Watch 1' window is open, showing a table with the following data:

Name	Value	Type
 c5	0x08	char[data]@0x08f0 ([R28]+5)

Datentyp int

... 16bit

```
int i1;  
int i2=0x123456;  
int i3=0x123456;
```

100 %

Name	Value	Type
i1	0xed1b	int{data}@0x08ef ([R28]+12)
i2	0x3456	int{data}@0x08eb ([R28]+8)
i3	0x3456	int{data}@0x08ed ([R28]+10)

Memory 1

Memory: data IRAM

Address	Value
data 0x08EB	56 34 56 34 1b ed 00 0
data 0x0905	?? ?? ?? ?? ?? ?? ?? ?
data 0x091F	?? ?? ?? ?? ?? ?? ?? ?

man sieht: i1 enthält einen zufälligen Wert, i2 nimmt die 2 LSB Byte

Datentyp long

Datentyp long long

Datentyp long ... 32bit

Datentyp long long ... 64 bit

Pointer

Ein Pointer ist eine Variable (16 Bit), die eine Adresse enthält.

Deklaration eines Pointers und Initialisierung

char* p1; // nicht initialisiert, diesem Zeiger nicht folgen!

char* p2=NULL; //zeigt ins Nirwana, diesem Zeiger nicht folgen, aber es lässt sich prüfen, ob ein gültiger Zeiger vorliegt, bevor man einem Zeiger folgt

if (p2 == NULL) ... dont follow

Pointer auf Variable Zeigen lassen

```
char c1='A';  
char* pc1 = &c1; //den Pointer auf die Adresse von c1 stellen  
*pc1 = 'B'; // dem Pointer folgen
```

```
uint8_t local1=0;  
uint8_t* localPointer1;  
uint8_t* localPointer2=NULL; //needs stddef.h  
localPointer1 = &local1;  
*localPointer1 = 5; //local1 has value 5 now
```

100 %

Watch 1

Name	Value	Type
local1	0x05	uint8_t{data}@0x08f9 ([F
localPointer1	0x08f9	uint8_t*{data}@0x08f7 ([
localPointer2	0x0000	uint8_t*{data}@0x08f5 ([

Memory 1

Memory: data IRAM

data 0x08F5	00	00	f9	08	05
data 0x090A	??	??	??	??	??
data 0x091F	??	??	??	??	??
data 0x0934	??	??	??	??	??

Adress-Operator und Dereferenzierung eines Pointers

Arrays in C

Deklaration

```

char ary1[] = {'A', 65, 0x41, 0b01000001};
char ary2[2]; //Speicherplatz res
char ary3[] = "abc";
while (1) {
    volatile int dummy=0; //for debugging on
}

```

100 %

Watch 1

Name	Value	Type
▲ ary1	0x08f0	char[4]{data}@0x08f0
[0]	0x41	char{data}@0x08f0
[1]	0x41	char{data}@0x08f1
[2]	0x41	char{data}@0x08f2
[3]	0x41	char{data}@0x08f3
▲ ary2	0x08f4	char[2]{data}@0x08f4
[0]	0x8a	char{data}@0x08f4
[1]	0xa8	char{data}@0x08f5
▲ ary3	0x08f6	char[4]{data}@0x08f6
[0]	0x61	char{data}@0x08f6
[1]	0x62	char{data}@0x08f7
[2]	0x63	char{data}@0x08f8
[3]	0x00	char{data}@0x08f9

1: gleiches Bitmuster, 2: zufällig, weil nicht initialisiert, 3: Stringende

Arrays in C

Zugriff auf die Elemente des Arrays

```
int aryA[]={1,2,3,4};  
aryA[0]=0;  
*aryA=1234;  
aryA[1]=1;  
*(aryA+2)=3;  
*(&aryA[1])=0xFFFF;
```

100 %

Watch 1

Name	Value	Type
▲ aryA	0x08f2	int[4]{data}@0x08f2
[0]	0x04d2	int{data}@0x08f2
[1]	0xffff	int{data}@0x08f4
[2]	0x0003	int{data}@0x08f6
[3]	0x0004	int{data}@0x08f8

& ... Adress-Operator, ermittelt die Adresse eines Speicherplatzes

aryA+2 ... Pointer-Arithmetik; der Compiler weiß, um wie viele Adressen weiter geschaltet werden muss.

Array = konstanter Zeiger

Zugriff über Index

```
uint16_t aryB[]={1,2,3};  
    for (uint8_t i=0; i<sizeof(aryB)/sizeof(uint16_t); i++){  
        aryB[i]=0;  
    }
```

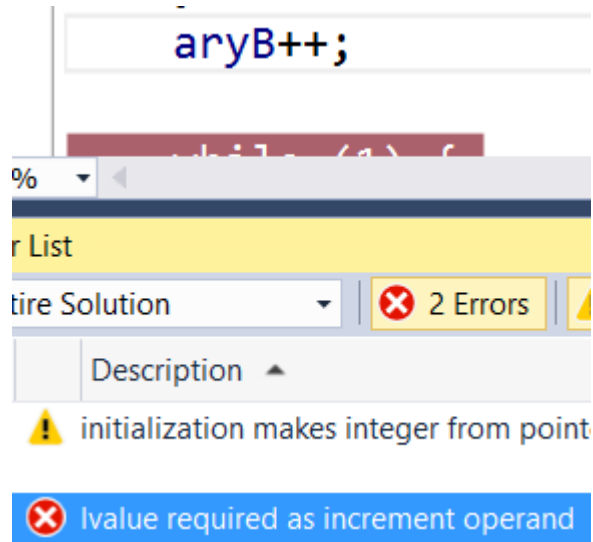
```
sizeof(aryB) == 6  
sizeof(uint16_t) == 2
```

nicht erlaubt:

aryB++

aryB = aryB+2

aryB ist ein **konstanter Zeiger**!



Zeiger auf ein Array

```
uint16_t aryB[]={1,2,3};
```

```
uint16_t* i16 = aryB;
```

```
*i16 = 0x1234;
```

Strings = Spezielles char Array

```
char s1[] = "Hallo";
char s2[] = {'H','a','l','l','o',0};
char* s3 = "Hallo"; //Pointer zeigt auf einen konstanten String
```

End of String: Bitmuster 0x00

The screenshot shows a debugger interface. At the top, the following C code is displayed:

```
// Strings
char s1[] = "Hallo";
char s2[] = {'H','a','l','l','o',0};
char* s3 = "Hallo"; //Pointer zeigt auf einen konstanten String
```

Below the code, the 'Watch 1' window shows the following data:

Name	Value	Type
s3	0x0102	char*(data)@0x08d3 ((R28)+21)
*(s3)	0x48	char(data)@0x0102
s2	0x08f4	char[6](data)@0x08f4 ((R28)+54)
[0]	0x48	char(data)@0x08f4
[1]	0x61	char(data)@0x08f5
[2]	0x6c	char(data)@0x08f6
[3]	0x6c	char(data)@0x08f7
[4]	0x00	char(data)@0x08f8

On the right, the 'Memory 4' window shows a memory dump for 'data IRAM' starting at address 0x0100:

Address	Hex	ASCII
data 0x0100	41 00 48 61 6c 6c 6f 00	A H a l l o
data 0x010F	00 01 00 02 00 03 00 48	
data 0x011E	00 00 00 00 00 00 00 00	
data 0x012D	00 00 00 00 00 00 00 00	
data 0x013C	00 00 00 00 00 00 00 00	
data 0x014B	00 00 00 00 00 00 00 00	
data 0x015A	00 00 00 00 00 00 00 00	

s3 ist ein Pointer (lokale Variable im Stack) auf einen Speicherplatz im Heap!

wie man verhindert, dass die Stringkonstante in den Heap kopiert wird, siehe

[C Programmierung für Fortgeschrittene auf kner.at](#)

String ausgeben

puts(s1)

printf("%s", s1)

Zahl in einen String umwandeln

char **buf1**[20];

uint8_t len1=**sprintf(buf1, "Zahl:%d", 55);**

len1 ... nr of chars converted

%d ... Format string

uint8_t len1=**sprintf(buf1, "Zahl:%f", 1.23);** //float needs special linker settings (ask google)

String in eine Zahl umwandeln

```
char x[10] = "450";  
uint16_t zahl = atoi(x);
```

Achtung!

uint16_t zahl = atoi("450"); funktioniert beim gcc nicht.

Speicherplatz reservieren: statisch

d.h. zur Übersetzungszeit

Statisch: Array

uint8_t buffer[100]; //reserviert 100 Byte am Stack wenn lokal, bzw. am Heap wenn global

Speicherplatz reservieren: dynamisch

d. h. zur Laufzeit des Programms

Speicherleiche = Speicherplatz, auf den kein Pointer mehr zeigt

Dynamisch: malloc(), realloc(), free()

```
char *str = (char *) malloc(100);           // reserviert 100 Byte am Heap
```

```
/* Realloc   nutzt den gleichen Speicherblock */  
str = (char *) realloc(str, 200);
```

```
free(str);
```

calloc() sets allocated memory to **zero**

Stimulus File für Zufallszahlen

Diese Stimulus-Zeile muss in eine Datei stim1.stim geschrieben werden und liest dann die Datei "data", die im selben Verzeichnis liegen muss ein.

1. Debugging starten
2. im Disassembler-Fenster auf der Adresse 0 einen Breakpoint setzen
3. //Debug/Execute Stimulus File
4. //Debug/Step Over

... jetzt ist der Speicher IDATA komplett mit Zufalls-Zahlen gefüllt.

```
//$memload file "data" segment nocheck  
$memload data s nocheck
```

Random Data File (Intel Hex Format)

```
/*  kner 2020 create random intel hex file for atmega328p data space init */
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main(){
    for (uint16_t i=0; i<2048;i++){
        printf("%s", ":10");
        printf("%04X", i+256);
        printf("%s", "00");
        for (uint8_t i=0; i<16;i++){
            uint8_t x=rand();
            printf("%02X", x);
        }

        puts("00");
    }
    puts(":000000001FF");
}
```

erzeugt folgenden Output ...

```
...
:1008FC000276F6758AA1B657FA91C8847B9986B000
:1008FD00068E48A5A7649B22FB191F5D99F26D9B00
:1008FE00686310F204C749FF581183D4AB0984B100
:1008FF0097CC563F31F1612C0B8089A472F63FDA00
:000000001FF
```

Beispiele

Speicherbereich löschen

```
#include <avr/io.h>
#include <stdint.h>
int main(void){
    // Fill start of HEAP with Random Numbers
    for (uint16_t i=0x100; i<0x1FF;i++) {
        char randomData=rand();
        char* pRAM = (char*)i;
        *pRAM = randomData;
    }
```

Beispiele AVR

```
/*  
 * 100 Integer dynamisch allokkieren und auf 0,1,2,3... initialisieren  
 */  
*/  
  
#include <avr/io.h>  
#include <stdlib.h>  
#define MAXNR 100  
int main(void)  
{  
    int* buffer = (int *)malloc(MAXNR*2);  
    for (uint8_t i=0; i<MAXNR; i++){  
        buffer[i]=i;  
    }  
    free(buffer);  
    while (1)  
    {  
        volatile int i; i++; //dummy  
    }  
}
```

Parameterübergabe an Unterprogramme

```
void test (char* x){  
    *x = 5; // auf die Adresse von c1 schreiben  
}  
  
void main(){  
    char c1;  
    test(&c1); //Adresse von c1 übergeben  
}
```