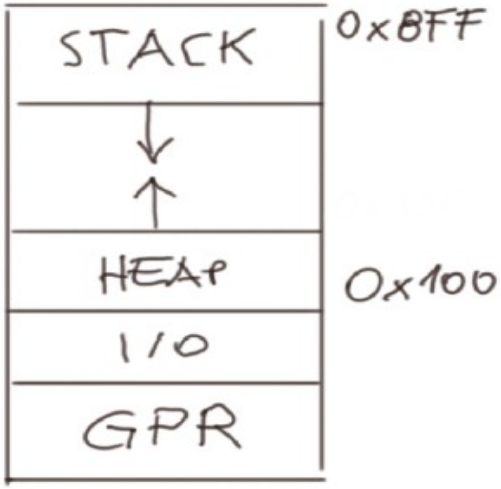
Pointer in C

Inhalt

Speichermodell des ATMEGA328p

Stack: lokale Variable und Rücksprungadressen, nicht initialisiert
Heap: globale Variable, automatisch inititalisiert 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

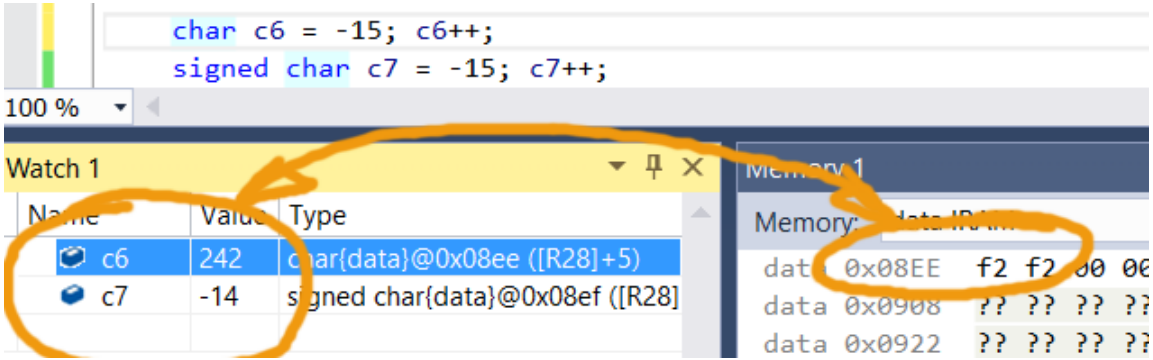
```
PGM_P p = &c2; x = pgm_read_byte(p);
```

Datentyp char

8bit (je nach Compiler mit oder ohne Vorzeichen)

unsigned char, signed char

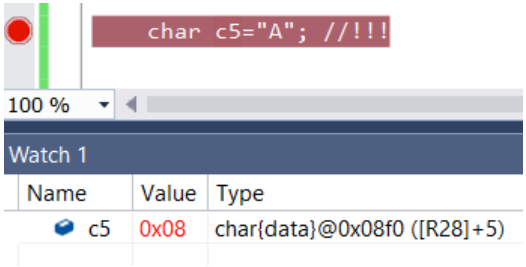
```
char c1 = 'A';char c2 = 0x41;char c3 = 65; char c4 = 0b01000001;
```



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)



Datentyp int

… 16bit

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

100 %

Watch 1

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

```
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

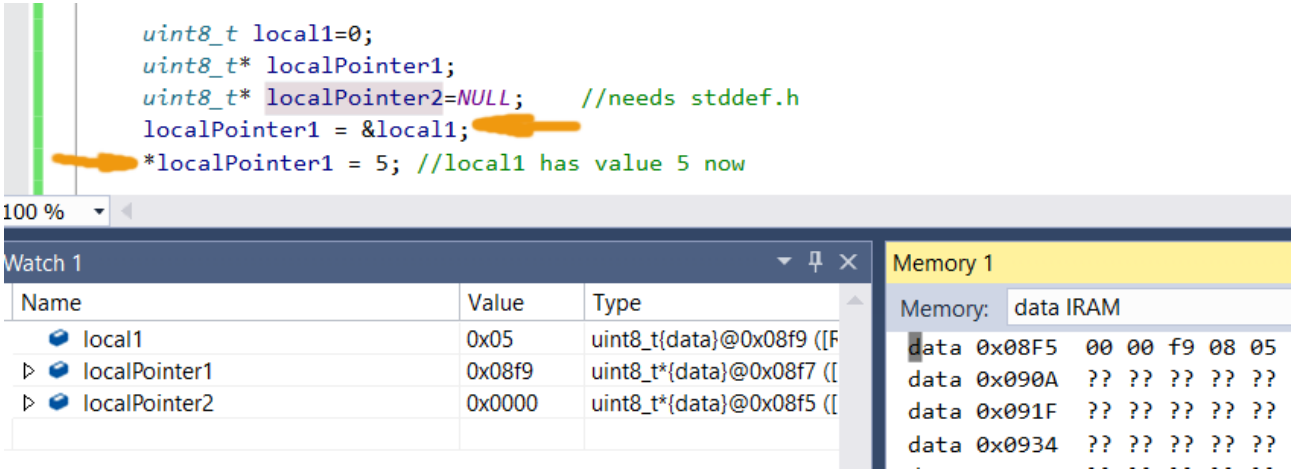
char* p1; // nicht initialisiert, diesem Zeiger nicht folgen!

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

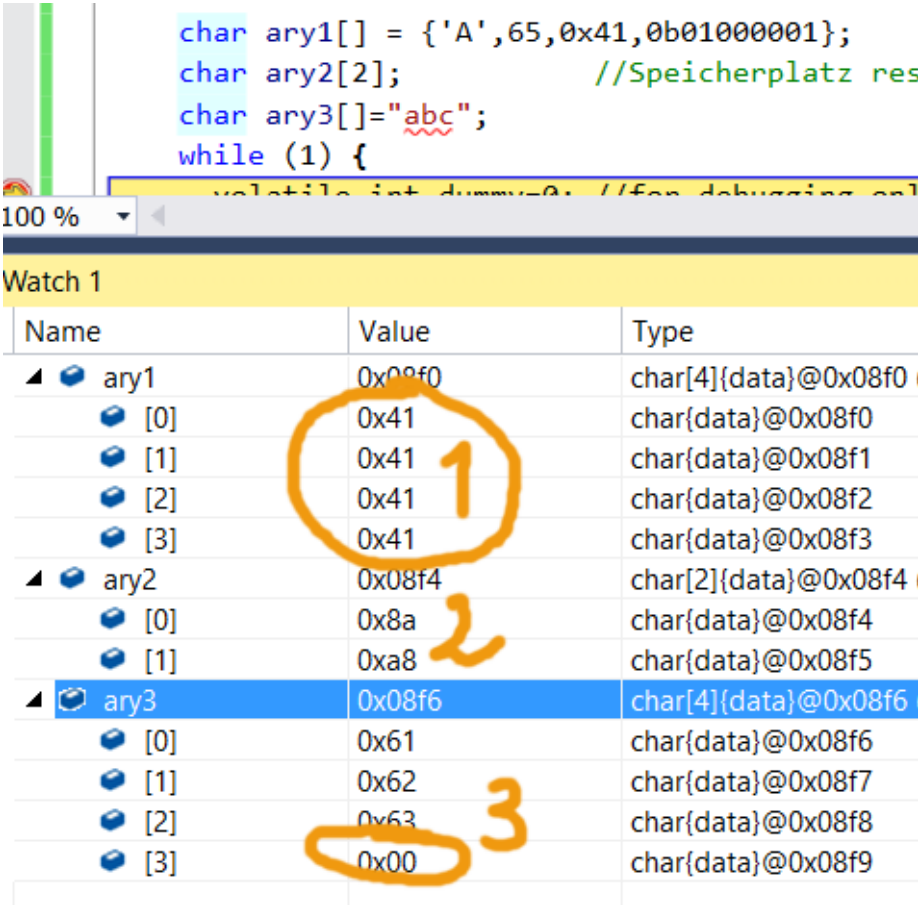
```
char c1='A';
char* pc1 = &c1; //den Pointer auf die Adresse von c1 stellen
*pc1 = 'B'; // dem Pointer folgen
```



Adress-Operator und Dereferenzierung eines Pointers

Arrays in C

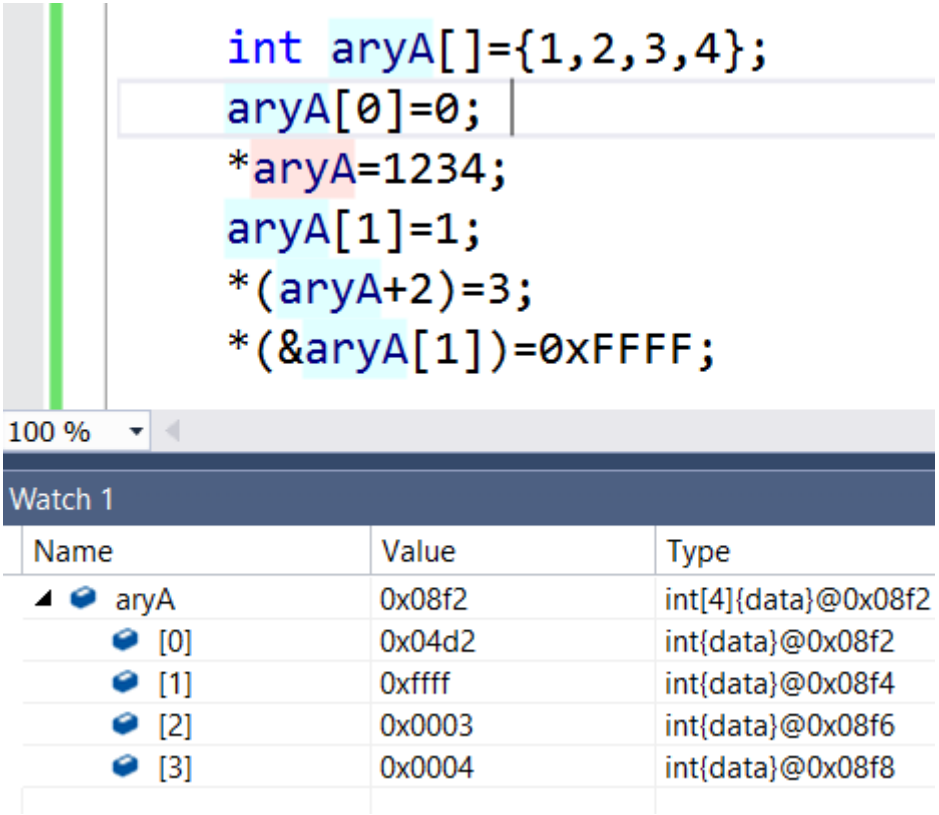
Deklaration



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

Arrays in C

Zugriff auf die Elemente des Arrays



& … 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

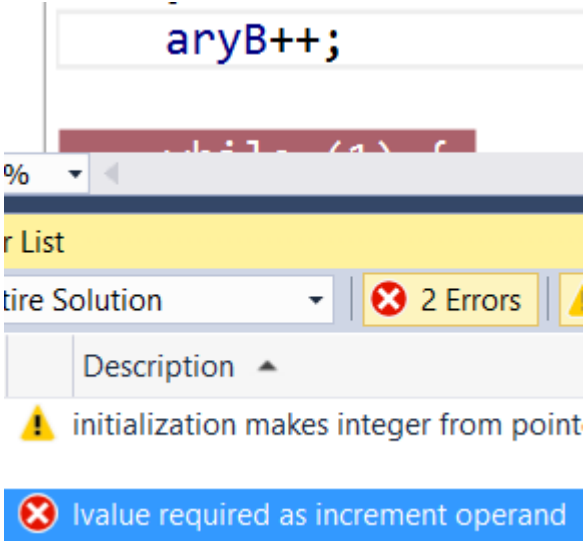
Zugriff über Index

```
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**!



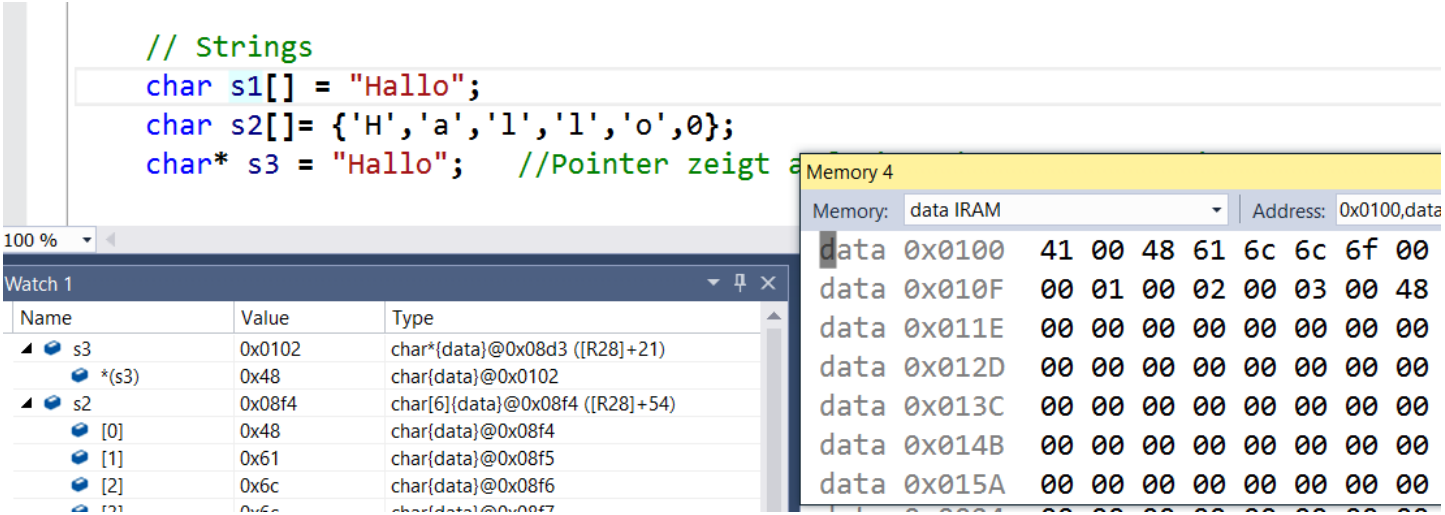
Zeiger auf ein Array

```
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



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(":00000001FF");
}
```

erzeugt folgenden Output ...

```
…
:1008FC000276F6758AA1B657FA91C8847B9986B000
:1008FD00068E48A5A7649B22FB191F5D99F26D9B00
:1008FE00686310F204C749FF581183D4AB0984B100
:1008FF0097CC563F31F1612C0B8089A472F63FDA00
:00000001FF
```

Beispiele

Speicherbereich löschen

```
#include <avr/io.h>
#include  <stddef.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 allokieren 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
}
```