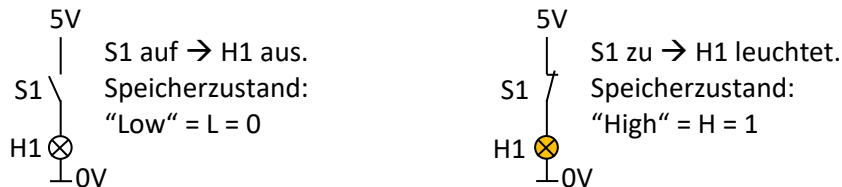


## 2.1.1 Zahlensysteme

In allen datenverarbeitenden Systemen (DV) werden geeignete Zahlensysteme zur Darstellung von Daten, bzw. Zahlen benötigt. Bei der digitalen Datenverarbeitung verwendet man das Dualsystem. Die kleinste Speichereinheit ist **1 Bit** und lässt sich elektronisch sehr einfach realisieren. Eine anschauliche Möglichkeit um 1 Bit zu speichern, stellt ein elektrischer Schalter mit zwei Stellungen dar:



Um größere Zahlen darstellen zu können, müssen mehrere „Bit“ zusammengefasst werden. Wie beim Dezimalsystem (Einer, Zehner, Hunderter, ...) hat dabei jede Stelle im Dualsystem (Einer, Zweier, Vierer, Achter, ...) eine festgelegte Wertigkeit.

Beispiel: Umrechnung der 8 Bit Dualzahl 1001.0110 in das Dezimalsystem

Wertigkeit:	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
Dual:	1	0	0	1	0	1	1	0
Dezimal:	128			+ 16		+ 4	+ 2	= <u>150</u>

Üblicherweise gibt es die folgenden Zusammenfassungen von mehreren Bit zu Dualzahlen mit den entsprechenden Wertebereichen<sup>1</sup>:

Datenbreite	Dezimal	Dual
<b>1 Nibble</b> = 4 Bit	$0 \dots (2^4-1) = 0 \dots 7$	0000 ... 1111
<b>1 Byte</b> = 2 Nibble = 8 Bit	$0 \dots (2^8-1) = 0 \dots 255$	00000000 ... 11111111
<b>1 Wort</b> = 2 Byte = 16 Bit	$0 \dots (2^{16}-1) = 0 \dots 65535$	00000000.00000000 ... 11111111.11111111
<b>1 Doppelwort</b> = 4 Byte = 32 Bit	$0 \dots (2^{32}-1) = 0 \dots 4.294.967.296$	00000000.00000000.000000 00.00000000 ... 11111111.11111111.111111 11.11111111

Da sich sehr große Zahlen nicht mehr anschaulich im Dualsystem darstellen lassen, wird zur Zahlendarstellung im Programmcode auch häufig das **Hexadezimalsystem** verwendet. Zur Konvertierung einer Dualzahl ins Hexadezimalsystem werden jeweils 4 Bit (= 1 Nibble) zu einer Hex-Ziffer zusammengefasst. Da es im Hex-System 16 Ziffern gibt, werden ab  $10_{\text{Dezimal}}$  die Buchstaben A bis F verwendet.

Bsp.: 32-Bit Doppelwort


1010.1110.1000.1101.0011.0010.1101.1111<sub>bin</sub>  
 A E 8 D 3 2 D F = AE8D32DF<sub>hex</sub>

Man beachte, dass ein Byte (= 2 Nibble) immer durch 2 Hexziffern dargestellt wird.

Daneben werden bei größeren Datenmengen in der Speichertechnik Einheitenvorsätze verwendet:

Bsp.: 1 KiB (Kibibyte) =  $2^{10}$  Byte = 1024 Byte      gebräuchlich: 1 kB (kilobyte)  
 1 MiB (Mebibyte) =  $2^{20}$  Byte = 1024 \* 1024 Byte      gebräuchlich: 1 MB (megabyte)

<sup>1</sup> Die Wortbreiten können bei unterschiedlichen Systemen auch von diesen Werten abweichen. Die Aufzählung ist nicht vollständig.

	<b>IT: Hardwarenahes Programmieren</b>	Name: Rahm Datum: 29.09.2022 2_1_Variablen_und_Datentypen_in_C.docx
	Variablen und Datentypen in C	2.1.2

## 2.1.2 Datenspeicher und Datentypen

Für die Verarbeitung von Daten in Computersystemen, werden elektronische Speicher (Register) benötigt. Da es unterschiedliche Arten von Daten gibt und die Speichergrößen endlich sind, muss der vorhandene Speicher organisiert und sinnvoll verwaltet werden.



Man kann einen elektronischen Speicher mit einem Registerschrank mit vielen Schubladen vergleichen. Die Schubladen sind zur Ordnung der Daten durchnummeriert. Man spricht hier von der **Speicheradresse**. In jede Schublade passt genau eine Speichereinheit (meistens 1 Byte = 8 Bit). Im Gegensatz zur „Schublade“ können im elektronischen Speicher die Daten nur komplett aus einem Register genommen oder hineingelegt werden, also normalerweise byteweise.

Daten die größer als 1 Byte sind, werden über mehrere Speicherstellen (Schubladen) verteilt.

Bsp.: Der Text „Hallo“ soll gespeichert werden. Jedes Zeichen wird durch einen 8 Bit-Zeichencode (ASCII-Code) dargestellt. Es werden 5 „Schubladen“ benötigt um den Text abzuspeichern. Die ASCII-Code-Darstellung lautet im Dualsystem: 01110010, 01100001, 01101100, 01101100, 01101111

Die einzelnen „Zeichen“ werden nacheinander im Speicher abgelegt, hier ab Adresse 005. Jedes Zeichen hat damit eine feste aufeinanderfolgende Speicheradresse:

Adresse	005	006	007	008	009
Zeichen	H	a	l	l	o
ASCII-Code (Dual)	01110010	01100001	01101100	01101100	01101111
ASCII-Code (Hex)	72	61	6C	6C	6F


Bei der Programmierung wird für jede Art von Zahlen oder Daten ein möglichst passender Datentyp festgelegt, der genügend aufeinanderfolgende Register (Schubladen) zusammenfasst. Der Zugriff erfolgt im Programm über Namen für **Variablen** oder **Konstanten**, so dass der Programmierer die Speicheradresse selbst nicht kennen muss. In der Programmiersprache C werden Variablen über den Datentyp und Name erzeugt (deklariert). Bei der Deklaration kann auch direkt ein gültiger Wert in den Speicher geschrieben werden (Initialisierung).

Beispiele:

```
int meineZahl;           // 32 Bit (= 4 Byte) Ganzzahl wird deklariert mit
                        // Wertebereich: -2.147.483.648 ... +2.147.483.647

char meinZeichen = 'A'; // 8-Bit (= 1 Byte) ASCII-Zeichen mit Initialisierung.
                        // Gespeichert wird der Zeichencode 41 (hex),
                        // bzw. 01000001 (dual)

float kommaZahl = 31.97; // Deklaration und Initialisierung einer Gleitkommazahl mit
                        // einfacher Genauigkeit
```

	<b>IT: Hardwarenahes Programmieren</b>	Name: Rahm Datum: 29.09.2022 2_1_Variablen_und_Datentypen_in_C.docx
	Variablen und Datentypen in C	2.1.3

Die wichtigsten Datentypen<sup>2</sup> bei der C-Programmierung mit der Arduino-IDE<sup>3</sup> sind:

Datentyp	Typedefs	Erklärung	Wertebereich	Belegter Speicherplatz
<b>bool</b>		false/true (falsch/wahr)	0 oder 1 (bzw. > 0)	1 Byte
<b>byte</b>	uint8_t	Positive ganze Zahlen	0 bis 255	1 Byte
<b>unsigned char</b>	uint8_t		0 bis 65535 ( $=2^{16}$ )	2 Byte
<b>unsigned short</b>	uint16_t		0 bis $2^{32}$	4 Byte
<b>unsigned int</b>	uint32_t		0 bis $2^{64}$	8 Byte
<b>unsigned long</b>	uint32_t	Ganze Zahlen mit Vorzeichen	-128 bis +127	1 Byte
<b>unsigned long64*</b>	uint64_t		-32768 bis +32767	2 Byte
<b>char</b>	int8_t		$-2^{31}$ bis $+(2^{31}-1)$	4 Byte
<b>short</b>	int16_t		$-2^{63}$ bis $+(2^{63}-1)$	8 Byte
<b>int</b>	int32_t	Gleitkommazahlen mit einfacher Genauigkeit	$\pm 1.175494 \cdot 10^{-38}$ bis $\pm 3.4028235 \cdot 10^{38}$	8 Byte
<b>long</b>	int32_t			
<b>long64*</b>	int64_t			
<b>float</b>				
<b>double</b>		Gleitkommazahlen mit doppelter Genauigkeit	$10^{-308}$ bis $10^{+308}$	8 Byte

### 2.1.3 Darstellung von Zahlen mit Beispielen

Format	char/byte	int	float/double
Dezimalzahl	230	3039    -32535	1.143    -1.3e3
Hexadezimalzahl	0xE6    0xe6	0x0BDF    0xFFFF	
Dualzahl (8 Bit)	B11100110		
ASCII-Zeichen	'A', '@', '4'		
Text	"Hallo Welt"		

Die unterschiedlichen Zahlendarstellungen und Datentypen haben zur Folge, dass Variablen-Definitionen oft trotz unterschiedlicher Schreibweise identisch sind.

Bsp.: Im Folgenden wird jeweils ein Byte mit dem gleichen Inhalt erzeugt.

```
unsigned char Zeichen = 'a';
uint8_t Zeichen = 0x61;
byte Zeichen = 97;
```

Da die Definition der Datentypen sehr stark systemabhängig sind, bietet sich die Typedef-Schreibweise an, da hier immer die Information über die Anzahl der Bit und des Vorzeichens enthalten ist und zwar unabhängig, mit welchem Mikroprozessor oder Compiler gearbeitet wird.

### 2.1.4 Definition von Konstanten

Konstanten können wie ganz normale Variablen, aber mit dem Schlüsselwort **const** deklariert werden. Eine mit const deklarierte und initialisierte „Variable“ kann nicht überschrieben werden. Z.B.:

```
const int LEDpin = 32;                    // Die Konstante LEDpin erhält den Wert 32
```

Eine weitere Möglichkeit ist die **#define**-Anweisung. Dabei handelt es sich genommen nicht um eine C-Anweisung, sondern um eine sogenannte Präprozessor-Direktive. Bsp.:


```
#define LEDpin 32                        // Symboldefinition
```

Im Unterschied zu const wird bei #define kein Speicherplatz im RAM belegt. Es wird lediglich ein Symbol definiert, dass der Präprozessor vor dem Compilerlauf im Quellcode ersetzt.

<sup>2</sup> Espressif API-Referenz: <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/preferences.html>

\* Nicht in der Arduino-IDE implementiert.

<sup>3</sup> Arduino Sprachreferenz: <https://www.arduino.cc/reference/de/>

 Friedrich-Ebert-Schule Esslingen FES	<b>IT: Hardwarenahes Programmieren</b>	Name: Rahm Datum: 29.09.2022 2_1_Variablen_und_Datentypen_in_C.docx
	Variablen und Datentypen in C	2.1.4

## 2.1.5 Vordefinierte Arduino-Konstanten

Es gibt einige von Arduino vordefinierte Konstanten. Hier die wichtigsten:

Konstante	Wert	Beschreibung
OUTPUT   INPUT   INPUT_PULLUP	Ausgang   Eingang   Eingang mit Pullup	Datenrichtung des Anschluss-Pin festlegen.
LOW   HIGH	0   1	Pin auf 0V oder 5V (3,3V) festlegen oder abfragen.
true   false	0   1 (>0)	Wahrheitswerte

Bsp.:

```
pinMode(LEDpin, INPUT_PULLUP);    // Pin als Eingang mit Pullup definieren
digitalWrite(LEDpin, HIGH);        // Pin auf 5V (bzw. 3,3V) schalten
```

## 2.1.6 Datenfelder (Arrays)

In Arrays werden mehrere Variablen gleichen Typs zusammengefasst, die mit einem Namen angesprochen werden. Zum Zugriff auf die einzelnen Elemente wird ein Index verwendet.

Bsp.: Definition eines Array aus 4 Byte-Werten (Es werden 4 Byte im RAM reserviert.)

```
uint8_t feld[4];
```

feld[0]	feld[1]	feld[2]	feld[3]
---------	---------	---------	---------

**Häufiger Fehler:** Bei der Definition des Arrays wird die Anzahl der Elemente eingesetzt. Der Index beginnt aber bei 0.

Arrays können bei der Definition initialisiert werden:

```
uint16_t zahl[] = {43,128,23,492};
```

zahl[0] = 43	zahl[1] = 128	zahl[2] = 23	zahl[3] = 492
-----------------	------------------	-----------------	------------------

Tatsächlich werden hier 8 Byte (4 \* 16Bit) reserviert und während der Laufzeit des Programms im RAM angelegt. Damit können Array-Elemente vom Programm verändert werden. Für die Definition konstanter Arrays, werden diese im Code-Segment angelegt.

```
const uint16_t zahl[] = {43,128,23,492};
```

Der Vorteil eines Arrays ist, dass es sich sehr einfach in Zählschleifen verarbeiten lässt.

Bsp.:

```
for(int n=0; n<=3; n++)
{
    feld[n] = n;
}
```

## Zeichenketten (Strings)

Zeichenketten sind in C nichts anderes als Arrays vom Typ **char**. Mit dem Unterschied, dass am Ende einer Zeichenkette das NULL-Zeichen ('\0' = 0) stehen muss, an dem die C-Bibliotheks-Funktionen das Stringende erkennen. Daher muss immer ein Zeichen mehr reserviert werden.

Bsp.: Initialisierung von konstanten Zeichenketten aus ASCII-Zeichen.

```
char MeinText[] = {'H','a','l','l','o','\0'};
```

Da die Schreibweise etwas gewöhnungsbedürftig ist, gibt es eine spezielle Syntax für Zeichenketten:

```
char MeinText[] = "Hallo";           // '\0' wird automatisch angehängt.
```

Das Ergebnis der beiden Variablendeklarationen ist identisch.