Sprachelemente Arduino

06.10.2020 15:29:00

1 Inhaltsverzeichnis

2 Inhalt

3 Datentypen

3.1 strings

Beschreibung:

Textzeichenfolgen können auf zwei Arten dargestellt werden. Du kannst den `String`-Datentyp verwenden, der ab Version 0019 Teil des Kerns ist oder du kannst einen `String` aus einem Array des Typs char erstellen und ihn mit einem Nullterminator abschließen. Diese Seite beschreibt die letztere Methode. Für weitere Informationen zum `String`-Objekt, das mehr Funktionalität auf Kosten von mehr Arbeitsspeicher bietet, siehe die string object-Seite.

Syntax:

Alle folgenden Angaben sind gültige Deklarationen für Strings.

String variable;
String inhalt = ‘‘Hallo“;

```
char Str1[15];
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
char Str4[ ] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino";
```

Möglichkeiten zur Deklaration von Strings:

- Deklariere ein Array von Zeichen, ohne es wie in Str1 zu initialisieren
- Deklariere ein Array von chars (mit einem zusätzlichen char) und der Compiler fügt das erforderliche Nullzeichen wie in Str2 hinzu
- Füge das Nullzeichen explizit hinzu, Str3
- Initialisiere mit einer `String`-Konstante in Anführungszeichen. Der Compiler passt die Größe des Arrays an die `String`-Konstante und fügt ein abschließendes Nullzeichen an, Str4
- Initialisiere das Array mit einer expliziten Größen- und Zeichenfolgekonstante, Str5
- Initialisiere das Array und lasse zusätzlichen Platz für eine größere Zeichenfolge, Str6

Null-Termination:

Im Allgemeinen werden Zeichenfolgen mit einem Nullzeichen (ASCII-Code 0) abgeschlossen. Dadurch können Funktionen (wie `Serial.print()`) erkennen, wo das Ende eines Strings ist. Andernfalls würden sie nachfolgende Speicherbytes lesen, die eigentlich nicht Teil der Zeichenfolge sind.

Das bedeutet, dass die Zeichenfolge Platz für ein Zeichen mehr als den gewünschten Text haben muss. Aus diesem Grund müssen Str2 und Str5 acht Zeichen haben, obwohl "Arduino" nur sieben hat - die letzte Position wird automatisch mit einem Nullzeichen gefüllt. Str4 wird automatisch in acht Zeichen umgewandelt, eines für die zusätzliche Null. In Str3 haben wir das Nullzeichen (geschrieben als '\0') selbst explizit eingefügt.

Beachte, dass es möglich ist, eine Zeichenfolge ohne ein abschließendes Nullzeichen zu verwenden (z. B. wenn du die Länge von Str2 mit sieben statt mit acht angegeben hättest). Dadurch werden die meisten Funktionen, die Zeichenfolgen verwenden, nicht mehr richtig funktionieren. Wenn du feststellst, dass sich etwas merkwürdig verhält (mit Zeichen, die sich nicht in der Zeichenfolge befinden), kann dies jedoch das Problem sein.

Einfache oder doppelte Anführungszeichen?

Strings werden immer in doppelten Anführungszeichen ("Abc") und chars in einfachen Anführungszeichen (`'A'`) definiert.

Lange strings umbrechen

Du kannst lange Zeichenfolgen wie folgt umbrechen:

```
char myString[] = "This is the first line"
" this is the second line"
" etcetera";
```

Arrays von Strings

Wenn du mit großen Textmengen arbeitest, z. B. bei einem Projekt mit LCD-Display, ist es oft praktisch, ein `String`-Array zu erstellen. Da Strings selbst Arrays sind, handelt es sich eigentlich um ein zweidimensionales Array.

In dem folgenden Code zeigt das Sternchen hinter dem Datentyp `char` “char*” an, dass dies ein Array von “Zeigern” ist. Alle Array-Namen sind eigentlich Zeiger, daher ist dies zum Erstellen eines Arrays aus Arrays erforderlich. Zeiger sind eine der eher esoterischen Komponenten von C ++, die für Anfänger schwer verständlich sind, aber es ist nicht notwendig,Zeiger im Detail zu verstehen, um sie hier effektiv zu verwenden.

Beispielcode

```
char* myStrings[]={ "This is string 1", "This is string 2", "This is string
3",
"This is string 4", "This is string 5","This is string 6"};

void setup() {
  Serial.begin(9600);
}

void loop() {
  for (int i = 0; i < 6; i++) {
    Serial.println(myStrings[i]);
    delay(500);
  }
}
```

3.2 byte

byte

Byte speichert einen 8-bit numerischen, ganzzahligen Wert ohne Dezimalkomma. Der Wert kann zwischen 0 und 255 sein.
Der Datentyp 'byte' ist identisch mit „unsigned char'.

```
byte someVariable = 180;        // deklariert 'someVariable'
                                // als einen 'byte' Datentyp
```

3.3 int

Ganzzahl

-32.767 bis 32.768.

belegt 2 Byte

3.4 long

long
Datentyp für ganze Zahlen mit erweiterer Größe, ohne Dezimalkomma, gespeichert in einem 32-bit Wert in einem Spektrum von -2,147,483,648 bis +2,147,483,647.

```
long someVariable = 90000;      // deklariert 'someVariable'
                                // als einen 'long' Datentyp
```

belegt 4 Byte

3.5 float

float
Ein Datentyp für Fliesskomma Werte oder Zahlen mit Nachkommastelle. Fliesskomma Zahlen haben eine bessere Auflösung als Integer und werden als 32-bit Wert gespeichert; Der Zhalenbereich geht von -3.4028235E+38 bis 3.4028235E+38

```
float someVariable = 3.14;      // deklariert 'someVariable'
                                // als einen 'float' Datentyp
```

Bemerkung: Flieskommazahlen sind nicht immer genau und führen manchmal zu merkwürdigen Resultaten, wenn sie verglichen werden. Ein Vergleich wie z.B.
if (someVariable == 1.0)...
sollte vermieden werden, da der vermeintliche Wert auch intern als 1.0000001 dargestellt werden kann.

Berechnungen mit 'float' sind sehr viel langsamer als mit Integer Datentypen. Dieser Datentyp sollte deshalb nur selten verwendet werden.

belegt 4 Byte

3.6 double

Fließkommazahl mit doppelter Genauigkeit

belegt 8 Byte

3.7 Array

Ein Array ist eine Sammlung von Variablen, auf die mit einer Indexnummer zugegriffen wird. Arrays in der Programmiersprache C ++, in der Arduino-Skizzen geschrieben sind, sind zwar kompliziert, aber die Verwendung einfacher Arrays ist relativ unkompliziert.

Array erstellen / deklarieren

Alle folgenden Methoden sind gültige Methoden zum Erstellen (Deklarieren) eines Arrays.

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hello";
String inhalt[20]
```

Du kannst ein Array deklarieren, ohne es wie in `myInts` initialisieren zu müssen.
In `myPins` deklarieren wir ein Array, ohne explizit eine Größe zu wählen. Der Compiler zählt die Elemente und erstellt ein Array der entsprechenden Größe.
Schließlich können wir das Array wie in `mySensVals` sowohl initialisieren als auch skalieren.
Beachte, dass beim Deklarieren eines Arrays vom Typ `char` ein weiteres Element als Initialisierung erforderlich ist, um das erforderliche Nullzeichen zu enthalten.

Zugriff auf ein Array

Arrays sind nullindiziert, das heißt, bezogen auf die Array-Initialisierung oben, befindet sich das erste Element des Arrays am Index 0.

`mySensVals[0] == 2, mySensVals[1] == 4`, and so forth.

Das bedeutet auch, dass in einem Array mit zehn Elementen der Index neun das letzte Element ist. Daher:

```
int myArray[10]={9, 3, 2, 4, 3, 2, 7, 8, 9, 11};
// myArray[9] enthält 11
// myArray[10] ist ungültig und enthält zufällige Informationen (andere
Speicheradresse)
```

Aus diesem Grund solltet du beim Zugriff auf Arrays vorsichtig sein. Beim Zugriff auf das Ende eines Arrays (mit einer Indexnummer, die größer ist als die von dir deklarierte Arraygröße - 1), wird aus dem Speicher gelesen, der für andere Zwecke verwendet wird. Das Lesen an diesen Orten wird wahrscheinlich nicht viel bewirken, außer ungültige Daten liefern. Das Schreiben in zufällige Speicherorte ist definitiv eine schlechte Idee und kann oft zu unglücklichen Ergebnissen wie Abstürzen oder Programmstörungen führen. Dies kann auch ein schwieriger Fehler sein.

int meineWerte[5] = {10,12,32,46,50};

Im Beispiel wird als erstes ein Array vom Typ int angelegt. Die 5 in eckigen Klammern hinter dem Variablennamen bestimmen die Anzahl der Speicherplätze, die das Array bereit stellt. Man nennt die Anzahl der Speicherplätze auch die Länge des Arrays.
Im Programm kann man durch indizierte Abfragen auf die Speicherplätze des Arrays zugreifen.

Die erste Stelle im Array ist die Stelle 0: meineWerte[0], der Wert ist 10 usw.

So weist du einem Array einen Wert zu:

```
mySensVals[0] = 10;
```

So rufst du einen Wert aus einem Array ab:

```
x = mySensVals[4];
```

Arrays und FOR-Schleifen

Arrays werden oft innerhalb von Schleifen bearbeitet, wobei der Schleifenzähler als Index für jedes Array-Element verwendet wird. Wenn du beispielsweise die Elemente eines Arrays über den seriellen Anschluss ausgeben möchtest, kannst du Folgendes tun:

```
for (byte i = 0; i < 5; i = i + 1) {
  Serial.println(myPins[i]);
}
```

3.8 char Datentyp

Ein Datentyp zum Speichern eines Zeichenwerts. Zeichenliterale werden in einfache Anführungszeichen geschrieben: 'A' (für mehrere Zeichen - Zeichenfolgen - verwende doppelte Anführungszeichen: "ABC").
Zeichen werden jedoch als Zahlen gespeichert. Du kannst die spezifische Kodierung im ASCII-Chart sehen. Dies bedeutet, dass es möglich ist, Zeichen zu berechnen, bei denen der ASCII-Wert des Zeichens verwendet wird (z. B. hat 'A' + 1 den Wert 66, da der ASCII-Wert des Großbuchstaben A 65 ist). Siehe *Serial*.println für weitere Informationen, wie Zeichen in Zahlen übersetzt werden.
Die Größe des char-Datentyps beträgt mindestens 8 Bit. Es wird empfohlen, char nur zum Speichern von Zeichen zu verwenden. Verwende für einen vorzeichenlosen 1-Byte-Datentyp (8 Bit) den byte-Datentyp.

Werte werden in einfachen Anführungszeichen (Minutenstrich) übergeben.

```
char myChar = 'A';
char myChar = 65; // Beide sind gleichwertig
```

Die Deklaration eines char-Arrays ist identisch mit der bisher bekannten Form der Array-Deklaration:

```
char string_array[100];

int meineWerte[5] = {10,12,32,46,50};


const char hallo[] = { 'H', 'a', 'l', 'l', 'o', ' ', 'W', 'e', 'l', 't', '\n', '\0' };
oder
const char hallo[] = { "Hallo Welt\n" };
```

3.9 static Variable

Beschreibung

Das `static`-Keywort ermöglicht es, dass Variablen nur für eine Funktion sichtbar sind. Anders als lokale Variablen, die bei jedem Funktionsaufruf neu belegt werden, behalten statische Variablen deren Wert über das Funktionsende hinaus bei.

Statische Variablen werden nur einmal erstellt, wenn die Funktion das erste Mal aufgerufen wird.

Beispielcode

Der Code wandert zufällig zwischen 2 Endpunkten umher.

```
/* RandomWalk
* Paul Badger 2007
* RandomWalk wandert zufällig zwischen 2 Endpunkten umher.
* Die maximale Bewegug pro Schritt wird über "stepsize" festgelegt.
* Eine statische Variable wird dazu um einen zufälligen Wert hoch- bzw.
* heruntergezählt.
* Das ist auch bekannt als "pink noise" und "drunken walk".
*/

// Definiere die maximale und minimale Beschränkung der Funktion
#define randomWalkLowRange -20
#define randomWalkHighRange 20
// Variable für die Schrittweite pro Durchlauf
int stepsize;
// Variable zur Speicherung des aktuellen Ergebnisses der RandomWalk-
Funktion
int thisTime;
int total;

void setup() {
  // Initialisiere die serielle Verbindung
  Serial.begin(9600);
}

void loop() {
  // Initialisiere die Schrittweite
  stepsize = 5;
  // Rufe die randomWalk-Funktion auf und speichere das Ergebnis
  // in der Variable "thisTime"
  thisTime = randomWalk(stepsize);
  // Gib den Wert der Variable "thisTime" aus
  Serial.println(thisTime);
  // Warte 10 ms
  delay(10);
}

int randomWalk(int moveSize) {
  // Variable, um den Wert in der randomWalk-Funktion zu speichern
  // Nur diese Funktion kann den Wert ändern, der Wert ist aber von
  // Aufruf zu Aufruf wieder verfügbar
  static int  place;
  // Zufällige Bewegung
  place = place + (random(-moveSize, moveSize + 1));
```

```
  // Check, ob die Limits verletzt wurden
  if (place < randomWalkLowRange) {
        // Korrigiere den aktuellen Schritt in die positive Richtung
    place = randomWalkLowRange + (randomWalkLowRange - place);
  }
  else if(place > randomWalkHighRange) {
        // Korrigiere den aktuellen Schritt in die negative Richtung
    place = randomWalkHighRange - (place - randomWalkHighRange);
  }
  // Gib die neue Position zurück
  return place;
}
```

3.10 Zeiger

Beschreibung:

Referenzierung ist eines der Features, bei denen Zeiger benötigt werden. Der Operator & wird dafür verwendet. Wenn x eine Variable ist, dann ist &x die Adresse der Variable x.

Beispielcode:

Der Code zeigt ein kleines Rechenbeispiel zur Verwendung von Zeigern.

```
int *p;              // Definiere einen Zeiger als Integerdatentyp
int i = 5;
int result = 0;  // Initialisiere 'i' mit 5 und 'result' mit 0

p = &i;          // 'p' enthält nun die Adress von 'i'

result = *p;     // 'result' erhält nun den Wert, der an der Adress von 'i' steht
                        // 'result' ist also 5
```

Die Anweisungen ' **char s [] = “geeksquiz”** ' **erstellen** ein Zeichenarray, das wie jedes andere Array ist, und wir können alle Arrayoperationen ausführen. Das einzig Besondere an diesem Array ist, dass es, obwohl wir es mit 9 Elementen initialisiert haben, eine Größe von 10 hat (der Compiler fügt automatisch '\ 0' hinzu).

Die Anweisung ' **char * s = "geeksquiz"** ' erstellt ein String-Literal. Das String-Literal wird von den meisten Compilern im schreibgeschützten Teil des Speichers gespeichert. Die C- und C ++ - Standards besagen, dass String-Literale eine statische Speicherdauer haben. Jeder Versuch, sie zu ändern, führt zu undefiniertem Verhalten.
s ist nur ein Zeiger und speichert wie jeder andere Zeiger die Adresse des String-Literal.
Vorteile:

1. Es ist nur ein Zeiger erforderlich, um auf die gesamte Zeichenfolge zu verweisen. Das zeigt, dass dies speichereffizient ist.
2. Die Größe der Zeichenfolge muss nicht im Voraus angegeben werden.

4 String Bearbeitung

4.1 indexOf() (String im String suchen)

Beschreibung

Sucht ein Zeichen oder eine Zeichenfolge in einer anderen Zeichenfolge. Sucht standardmäßig vom Anfang des Strings, kann aber auch von einem bestimmten Index aus beginnen. Ermöglicht das Auffinden aller Instanzen des Zeichens oder der Zeichenfolge.

Syntax

```
myString.indexOf(val)
myString.indexOf(val, from)

pos=Quellstring.indexOf(“Suchstring“);
```

Parameter

`myString`: Eine Variable vom Typ `String`. Erlaubte Datentypen: `String`.+ `val`: Der Wert, nach dem gesucht werden soll. Erlaubte Datentypen: `char` oder `String`.
`from`: Der Index, von dem aus die Suche gestartet werden soll.

Rückgabewert

Der Index des Werts innerhalb des Strings oder -1, falls nicht gefunden.
Die Position des Suchstrings beginnt bei 0

4.2 size_t Größe eines Datentyps in Bytes

Beschreibung
`size_t` ist ein Datentyp, der die Größe eines Objekts in Bytes darstellen kann. Beispiele für die Verwendung von `size_t` sind der Rückgabetyp von `sizeof()` and `Serial.print()`.
Syntax
`size_t var = val`
Parameter
`var`: Variablenname.
`val`: Der Wert, der der Variablen zugewiesen wird.

4.3 sizeof() Anzahl der Bytes

Beschreibung

`sizeof` gibt die Anzahl an Bytes in einem Variablentyp oder die Anzahl an Bytes eines Arrays zurück.

Syntax

`sizeof(variable)`

Parameter

`variable`: Erlaubte Datentypen: Beliebiger Datentyp.

Rückgabewert

Die Anzahl der Bytes in einer Variablen oder die Bytes, die in einem Array belegt sind. Datentyp: `size_t`.

Beispielcode

Der `sizeof`-Operator ist nützlich, um mit Arrays (wie z.B. Strings) umzugehen. Dort ist es Standard, dass die Größe des Arrays sich ändert, ohne dass das ganze restliche Programm umgeschrieben werden muss.

Das Programm gibt einen Text zeichenweise aus. Versuche, den Text zu ändern, das Programm wird weiter funktionieren.

```
// Definiere Text und Zählervariable
char myStr[] = "this is a test";

void setup() {
  // Initialisiere seriellen Port
  Serial.begin(9600);
}

void loop() {
```

```
    // Laufe über das Array (den String), egal, wie lange dieser ist
    for (byte i = 0; i < sizeof(myStr) - 1; i++) {
        // Gib die Zeichennummer als Dezimalzahl aus
        Serial.print(i, DEC);
        // Gib ein Gleichheitszeichen aus
        Serial.print(" = ");
        // Gib das Zeichen des Strings an Stelle "i" aus
        Serial.write(myStr[i]);
        // Schreibe eine neue Zeile
        Serial.println();
    }
    delay(5000);  // Warte 5 Sekunden
}
```

Anmerkungen und Warnungen

`sizeof` gibt die Anzahl an Bytes zurück. Für Arrays größerer Variablentypen, wie z.B. `int`, sieht die Schleife wie folgt aus:

```
int myValues[] = {123, 456, 789};
// Diese for-Schleife funktioniert ordnungsgemäß mit einem Array eines beliebigen Typs und jeder Größe

for (byte i = 0; i < (sizeof(myValues)/sizeof(myValues[0])); i++) {
    // Tue etwas mit myValues[i]
}
```

Beachte zusätzlich, dass ein richtig formattierter `String` mit dem `NULL`-Symbol beendet wird, also dem ASCII-Zeichen mit Nummer 0.

4.4 Länge eines Strings

laenge = variable.length();

Gibt die Länge eines Strings aus.

4.5 Substring Teilstring aus String

Teilstring = Stringvariable.substring(Startpos, Endpos);

Startpos und Endpos beginnen bei 0 (Index)

4.6 char zurücksetzen

telnr[20] = NULL //wird zurückgesetzt, leer gemacht

char text[10] = "abcdefg";
text[0] = '\0'
oder
strcpy(text, "");

4.7 Verketten von char Strings mit strcat in char

Mit strcat (stringcatenate) Strings verketten:

char* strcat (char* destination, const char* source);

Funktion: Verbindet source mit destination, d.h. source wird an destination angehängt (verkettet)

char text1[10] = "12345678";
char text2[10] = "absdefgh";

strcat(text1, text2);

4.8 Zeichen anhängen char

Ein einzelnes Zeichen anhängen:
char text1[10] = "abc";
char c = 'x';
strncat(text1, &c, 1);

Die ersten 2 Zeichen eines Character-Arrays anhängen:

char text1[10] = "abcd";
char text2[] = "ABCD";

strncat(text1, text2, 2);

4.9 Anführungszeichen in Anführungszeichen

\“Text in Anführungszeichen\“

oder

‘Text‘
oder mit Chr(34) einfügen

4.10 Escape Zeichen

\a	akustisches Signal (von englisch *alert*)
\b	Rückschritt (von englisch *backspace*)
\e od. \E	ANSI Escape, hexadezimal 0x1B Ein Escapezeichen für eine höhere Interpretationsebene, siehe oben. *Nicht* Bestandteil von ISO C und ISO C++!
\f	Seitenvorschub (von englisch *form feed*)
\n	Zeilenvorschub (von englisch *new line*)
\r	Wagenrücklauf (von englisch *carriage return*)
\t	Horizontal-Tabulatorzeichen (von englisch *horizontal tabulator*)
\v	Vertikal-Tabulatorzeichen (von englisch *vertical tabulator*)

\'	Das Zeichen ', einfaches Anführungszeichen
\"	Das Zeichen ", doppeltes Anführungszeichen
\?	Das Fragezeichen ?
\\	Das Zeichen \, Backslash (umgekehrter Schrägstrich)

4.11 Splitten von Strings mit strtok

Splitten von Strings

Mit strtok Strings zerteilen und splitten
Mit **strtok** können wir einen String anhand von Trennzeichen zerteilen und die einzelnen Abschnitte herauslesen. Die Trennzeichen werden im Parameter übergeben.
Bei Folgeaufrufen wird statt **string** der **NULL** Wert übergeben, da **strtok** bereits initialisiert ist und intern einen Zeiger auf **string** gespeichert hat. Es sollten beim Verwenden von **strtok** immer nur eine Kopie eines Strings übergeben werden.

Teilstring = strtok(Quellstring,Trennzeichen)

```
//sind ,, Stellen im String, so werden sie nicht als Leerstrings dargestellt, es verschiebt sich alles

char  StartString[10];    //Größe egal
char  buf[]  =  "aaa,y,bb,ccc,xx,vvvv";

void  setup()  {    //.....................................................................
   Serial.begin(9600);

  for  (int  i  =  0;  i  <  30;  i++)  {        // i< mindestens so groß wie Gesamtstring buf
       StartString[i]  =  char(buf[i]);
   }

   char*  Part0  =  strtok(StartString,  ",");  //1. Teilstring
   char*  Part1  =  strtok(NULL,  ",");
   char*  Part2  =  strtok(NULL,  ",");
   char*  Part3  =  strtok(NULL,  ",");
   char*  Part4  =  strtok(NULL,  ",");
   char*  Part5  =  strtok(NULL,  ",");
   //char* Part6 = strtok(NULL, ",");  //weitere Parts führen zu keinem Fehler, obwohl es sie nicht gibt!!!

   Serial.println();
   Serial.println(Part0);
   Serial.println(Part1);
   Serial.println(Part2);
   Serial.println(Part3);
   Serial.println(Part4);
   Serial.println(Part5);
   //Serial.println(Part6);


}  //.....................................................................

void  loop()  {      //.....................

}    //.....................
```

andere Variante:

```
char  text[]=  "aaa,bbb,ggg,dd,eeeeee,,gggg";      //zu splittender Text
char  *einzel;      //Teilstring von , zu ,
char*  variable[10];
int  a=1;

void  setup()  {//.....................................................
Serial.begin(9600);

einzel  =  strtok(text,",");

while(einzel  !=  NULL){

    variable[a]=einzel;
    a=a+1;

    einzel  =  strtok(NULL,",");

}
//leere Felder werden nicht angezeigt es verschiebt sich alles

Serial.println();
Serial.println(variable[1]);
Serial.println(variable[2]);
Serial.println(variable[3]);
Serial.println(variable[4]);
Serial.println(variable[5]);
Serial.println(variable[6]);
Serial.println(variable[7]);

}//.....................................................................

void  loop()  {    //.....................................................

}    //.....................................................................
```

Beispiel:

```
char  StartString[40];
char  buf[]  =  "54.1254,N,11.2783,E,255,1,13:01:12,H";

void  setup()  {
   Serial.begin(9600);
}

void  loop()  {
   for  (int  i  =  0;  i  <  50;  i++)  {        //i<50 min. wie Gesamtstring buf
      StartString[i]  =  char(buf[i]);
   }

   char*  Part0  =  strtok(StartString,  ",");
   char*  Part1  =  strtok(NULL,  ",");
   char*  Part2  =  strtok(NULL,  ",");       //es können auch Konvertierungen vorgenommen werden
   char*  Part3  =  strtok(NULL,  ",");       // z.B. int Part2 = atoi(strtok(NULL, ","));
   char*  Part4  =  strtok(NULL,  ",");
   char*  Part5  =  strtok(NULL,  ",");
   char*  Part6  =  strtok(NULL,  ",");
   char*  Part7  =  strtok(NULL,  ",");
   //wenn zu viele Part verwendet werden als tatsächlich kein Fehler sondern Leerstrings

   Serial.println(Part0);
   Serial.println(Part1);
   Serial.println(Part2);
   Serial.println(Part3);
   Serial.println(Part4);
   Serial.println(Part5);
   Serial.println(Part6);
   Serial.println(Part7);


   Serial.println();

   delay(1000);
}
```

Beispiel GPS.cpp

ab ***Breite*** wiederholt sich der Code abwechsend mit ***N S*** !!!

evt. Funktionsaufruf ändern

```
char GPSGSM::getPar(char *str_zeit, char *str_lat, char *str_lat1, char *str_lon, char *str_lon1, char *str_qual,
char *str_satanz, char *str_abw , char *str_hoehe)
{
    char ret_val=0;
    char *p_char;
    char *p_char1;
    gsm.SimpleWriteln("AT+CGPSINF=2");
    gsm.WaitResp(5000, 100, "OK");
    if(gsm.IsStringReceived("OK"))
        ret_val=1;

    //Zeit  1. Block
    p_char = strchr((char *)(gsm.comm_buf),',');
    p_char1 = p_char+1;  //we are on the first char of longitude
    p_char = strchr((char *)(p_char1), ',');
    if (p_char != NULL) {
        *p_char = 0;
    }
    strcpy(str_zeit, (char *)(p_char1));  // Ergebnis in String kopieren


    // Breite 2. Block
    p_char++;
    p_char1 = strchr((char *)(p_char), ',');
    if (p_char1 != NULL) {
        *p_char1 = 0;
    }
    strcpy(str_lat, (char *)(p_char));

    // N S 3.Block
    p_char1++;
    p_char = strchr((char *)(p_char1), ',');
    if (p_char != NULL) {
        *p_char = 0;
    }
    strcpy(str_lat1, (char *)(p_char1));

    // Länge  wie 2.Block
    p_char++;
    p_char1 = strchr((char *)(p_char), ',');
    if (p_char1 != NULL) {
        *p_char1 = 0;
    }
    strcpy(str_lon, (char *)(p_char));

     // E W  wie 3. Block
    p_char1++;
    p_char = strchr((char *)(p_char1), ',');
    if (p_char != NULL) {
        *p_char = 0;
    }
    strcpy(str_lon1, (char *)(p_char1));

    // Qualität  wie 2. Block
    p_char++;
    p_char1 = strchr((char *)(p_char), ',');
```

```
    if (p_char1 != NULL) {
        *p_char1 = 0;
    }
    strcpy(str_qual, (char *)(p_char));

     // Sat Anz  wie 3. Block
    p_char1++;
    p_char = strchr((char *)(p_char1), ',');
    if (p_char != NULL) {
        *p_char = 0;
    }
    strcpy(str_satanz, (char *)(p_char1));

            // horiz. Abweichung wie 2.Block
    p_char++;
    p_char1 = strchr((char *)(p_char), ',');
    if (p_char1 != NULL) {
        *p_char1 = 0;
    }
    strcpy(str_abw, (char *)(p_char));

            // Höhe  wie 3. Block usw.
    p_char1++;
    p_char = strchr((char *)(p_char1), ',');
    if (p_char != NULL) {
        *p_char = 0;
    }
    strcpy(str_hoehe, (char *)(p_char1));



    return ret_val;
}
```

4.12 Splitten von Strings über Position des Trennzeichens (indexOf)

Bei der Variante mit ***strtok*** werden leere Strings nicht angezeigt. Bei dieser Variante werden sie angezeigt. Es wird über die Positionen des Trennzeichens gearbeitet.

Beispiel:

```
//Leerstrings werden mit angezeigt!!!
String buf = "aaa,1,,3,,5,6,,,qqqqq,ggggg,54321";
String teil[20];    //mind so viel wie Teilstrings da sind
int pos;
int pos1[20];    //mind so viel wie Kommas
int i;

void setup() {    //.....................................................................
   Serial.begin(9600);
   Serial.println();

   for (i = 0; i < 20; i++) {              //i< mind. Anzahl der Kommas
      if (pos < 0) {
         break;
      }

      pos = buf.indexOf(",", pos + 1);                  //Positionen der Kommas suchen
      pos1[i] = pos;      //Positionen der Kommas in Array speichern

      //Serial.println(pos1[i]);
   }

   //Teilstrings bilden
   teil[0] = buf.substring(0, pos1[0]);        //1. Teilstring
   for (int y = 1; y < (i + 1); y++) {          //2. bis vorletzter Teilstring
      teil[y] = buf.substring(pos1[y - 1] + 1, pos1[y]);
   }
   teil[i] = buf.substring(pos1[i - 2], buf.length());    //letzter Teilstring

   //Ausgabe
   Serial.println(teil[0]);
   Serial.println(teil[1]);
   Serial.println(teil[2]);
   Serial.println(teil[3]);
   Serial.println(teil[4]);
   Serial.println(teil[5]);
   Serial.println(teil[6]);
   Serial.println(teil[7] + "rrr");
   Serial.println(teil[8]);
   Serial.println(teil[9]);
   Serial.println(teil[10]);
   Serial.println(teil[11]);

} //.....................................................................

void loop() {      //.....................
```

4.13 Splitten von Strings mit strchr

Mit **strchr** können wir ein Zeichen in einem String suchen. Das zu suchende Zeichen wird mit dem Parameter übergeben.
Diese Funktion gibt einen String zurück von gefundener 1. Position des Trennzeichens bis Ende Quellstring.

Rückgabewerte:

Es gibt den Zeiger auf das erste Vorkommen des Zeichens in der angegebenen Zeichenfolge zurück. Wenn wir also den Zeichenfolgenwert des Zeigers anzeigen, sollte der Teil der Eingabezeichenfolge ab dem ersten Vorkommen des angegebenen Zeichens angezeigt werden.

NULL, wenn das Zeichen nicht gefunden wurde

Syntax:

char *teistring = strchr(Quellstring,‘,‘);
oder
char* teistring = strchr(Quellstring,‘,‘);
oder
String teistring = strchr(Quellstring,‘,‘);

Beispiel:

```
        Serial.begin(9600);
//String teilstring;
char  quelle[]  =  "Ein String mit Worten";

        char*  teilstring=strchr(quelle,  'W');
                        //oder
      //char* teilstring=strchr(quelle, 'W');
                        //oder
    //String teilstring=strchr(quelle, 'W');

        Serial.println();
        Serial.println(teilstring);

//Ergebnis:  "Worten"
```

Hiermit wird ab dem Auftreten des Buchstabens 'W' der komplette String ausgegeben.

Beispiel:

```
char  quelle[]  =  "aaXaa";
if(strchr(quelle,  'X'))  {
      printf("String enthaelt ein X\n");
}
```

Beispiel:

```
//Leerstrings werden mit angezeigt aber am Ende muss ein Komma sein !!!!
char  quelle[]  =  "aaa,,d,ddd,qq,123,";
String  teil[20];
int  i  =  0;

void  setup()  {    //.....................................................

   Serial.begin(9600);

   char  *start  =  quelle;
   char  *ptr  =  strchr(start,  ',');  //ptr ist 1. Teilstring

   while  (ptr  !=  nullptr)  {
      *ptr  =  '\0';                  //Char Abschließen
      teil[i]  =  start;
      start  =  ptr  +  1;
      ptr  =  strchr(start,  ',');  //Teilstring
      i  =  i  +  1;         //Array Nummern für Variable teil
   }

   //Ausgabe
   Serial.println();
   Serial.println("Ausgabe");
   Serial.println(teil[0]);
   Serial.println(teil[1]);
   Serial.println(teil[2]);
   Serial.println(teil[3]);
   Serial.println(teil[4]);
   Serial.println(teil[5]);

}

void  loop()  {}
```

5 Datentyp Umwandlung

5.1 String() Umwandlung in String

Beschreibung:

Erstellt eine Instanz der `String`-Klasse. Es gibt mehrere Versionen, die Strings aus verschiedenen Datentypen erstellen (d. h. als Zeichenfolgen formatieren), darunter:

- Eine konstante Zeichenfolge in Anführungszeichen (d. h. ein Char-Array)
- Ein einzelnes konstantes Zeichen in einfachen Anführungszeichen
- Eine andere Instanz des `String`-Objekts
- Ein konstanter `int` oder `long int`
- Ein konstanter `int` oder `long int`, der eine angegebene Basis verwendet
- Eine konstante `int` oder `long int` Variable
- Eine konstante `int` oder `long int` Variable, die eine angegebene Basis verwendet
- Ein `float` oder `double` mit einer bestimmten Anzahl Dezimalstellen

Beim Konstruieren einer Zeichenfolge aus einer Zahl wird eine Zeichenfolge erstellt, die die ASCII-Darstellung dieser Zahl enthält. Der Standard ist also Basis zehn:

```
String thisString = String(13);
```

Gibt den `String` "13" zurück. Du kannst jedoch auch andere Basen verwenden. Zum **Beispiel:**

```
String thisString = String(13, HEX);
```

Gibt den String "D" zurück. Dies ist die hexadezimale Darstellung des Dezimalwerts 13. Oder, wenn du das Binärformat bevorzugst:

```
String thisString = String(13, BIN);
```

Gibt den String "1101" zurück. Das ist die binäre Darstellung von 13.

Syntax:

```
String(val)
String(val, base)
String(val, decimalPlaces)
```

Parameter:

`val`: Eine Variable, die als `String` formatiert werden soll. Erlaubte Datentypen: `string`, `char`, `byte`, `int`, `long`, `unsigned int`, `unsigned long`, `float`, `double`.
`base` (Optional): Die Basis, in der ein ganzzahliger Wert formatiert werden soll.
`decimalPlaces` (**Nur, wenn val `float` oder `double` ist**): Die gewünschten Dezimalstellen.

5.2 int() String nach Int

Beschreibung

Konvertiert einen Wert in den `int`-Datentyp.

Syntax

```
int(x)
(int)x (Typ-Konvertierung wie in C)
```

Parameter

`x`: Ein Wert beliebigen Typs. Erlaubte Datentypen: Beliebiger Datentyp.

Rückgabewert

Datentyp: `int`.

5.3 toInt String nach Int

IntVariable = Stringvariable.toInt();

5.4 Zahl nach String

Stringvariable = String(zahl);

5.5 (unsigned int)

Beschreibung

Konvertiert einen Wert in den `unsigned int`-Datentyp.

Syntax

```
(unsigned int)x
```

Parameter

`x`: Ein Wert beliebigen Typs. Erlaubte Datentypen: Beliebiger Datentyp.

Rückgabewert

Datentyp: `unsigned int`.

5.6 String nach Char

```
char var[20] ;              //char Variable deklarieren
String varstring            //Stringvariable deklarieren

varstring.toCharArrey(var,20);    //Umwandlung
```

5.7 Char nach String

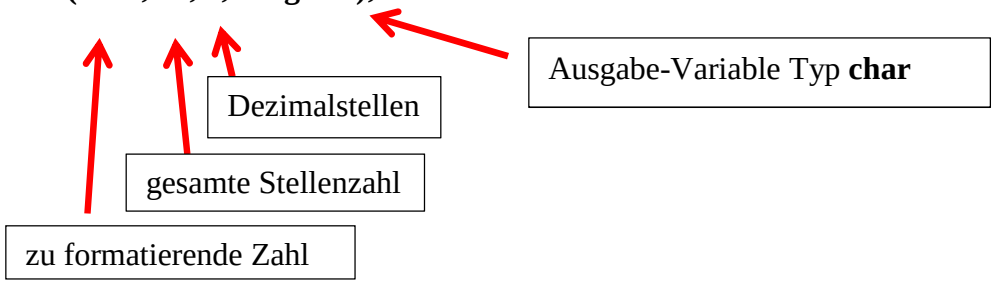
String stringvariable(charvariable);

6 Formatierte Ausgabe

6.1 Formatierung von Zahlen mit dtostrf

double zahl = 2.0;
char ausgabe[10];
char einheit[] = " Meter \0"; //evt mit 0 Terminierung nicht unbedingt

dtostrf(zahl, 10, 2, ausgabe);



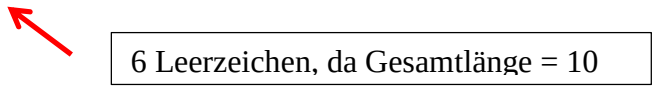
zahl ist Variable die Formatiert werden soll (double,int,float,long

10 ist Anzahl aller Stellen
(vor zahl, die Zahl selbst und Dezimalzeichen und Nachkommastellen)
ist der Wert zu klein wird Zahl trotzdem dargestellt

2 ist Anzahl der Dezimalstellen

Variable ausgabe muß vom Typ: **signed char, unsigned char, char*oder char** sein.
nicht String !!!

Ergebnis: 2.00



mit Maßeinheit dahinter:

strcat(ausgabe, einheit); //Anhängen an den Ursprungstext

Ergebnis: .....2.00 Meter

6.2 Formatieren von Zahlen mit Serial.print()

```
float gleitkommaZahl = 3.123456;

Serial.println(gleitkommaZahl, 4);          //liefert 3.1234
Serial.println(gleitkommaZahl, 6);          //liefert 3.123456
```

Anzahl der Dez. Stellen

6.3 String bilden mit Umwandlung

```
String wochentag = "Freitag";
int tag = 14;
int monat = 8;
int jahr = 2020;
int stunde = 13;
int minute = 37;
int sekunde = 45;

String zeile = "Heute ist " + wochentag + " der " + String(tag,DEC) + "." +
String(monat,DEC) + "." + String(jahr,DEC) + " " + String(stunde,DEC) + ":" +
String(minute,DEC) + ":" + String(sekunde,DEC);
Serial.println(zeile);
```

Ergebnis:

Heute ist Freitag der 14.8.2020 13:37:45

6.4 Formatieren mit sprintf oder snprintf

sprintf("Formatierung", Var1, Var2, Var3,...)

Format	**Bedeutung**
%d %i	Decimal signed Integer
%o	Octal Integer
%x %X	Hex Integer
%u	Unsigned Integer
%c	Character
%s	String
%f	Double
%e %E	Double
%g %G	Double
%p	Zeiger
%n	Number of characters written by this printf. No argument expected
%% %	No argument expected.
Flag	**Bedeutung**
-	linksbündig
0	Felder mit 0 ausfüllen (an Stelle von Leerzeichen).
+	Vorzeichen einer Zahl immer ausgeben.
blank	positive Zahlen mit Leerzeichen beginnen.
#	verschiedene Bedeutung:
%#o	(Oktal) 0 Präfix wird eingefügt.
%#x	(Hex) 0x Präfix wird bei Werten ungleich Null eingefügt.
%#X	(Hex) 0X Präfix wird bei Werten ungleich Null eingefügt.
%#e	Dezimalpunkt immer anzeigen.
%#E	Dezimalpunkt immer anzeigen.
%#f	Dezimalpunkt immer anzeigen.
%#g	Dezimalpunkt immer anzeigen. Nullen nach dem Dezimalpunkt werden nicht beseitigt.
%#G	Dezimalpunkt immer anzeigen. Nullen nach dem Dezimalpunkt werden nicht beseitigt.

Mit der Funktion **sprintf** kann man eine Zeichenkette in einem char Array formatiert ablegen

Variable ausgabe muss vom Typ **char** sein !!!

Beim Arduino Uno werden nur Datentypen int und char unterstützt –
Speichermangel!!! ESP32 keine Einschränkungen !!!

Beispiel1:

```
int tag;
int monat;
int jahr;
char s[20];

sprintf(s,"%02d.%02d.%04d",tag, monat, jahr);
Serial.println(s);
```

Ausgabe erfolgt mit führender 0 also 02.04.2020

Beispiel2:

```
String text = "Stefan";          //geht nicht mit Arduino Uno
char text[20] = "Stefan";        //geht mit ESP u.Arduino

//reservieren eines char Arrays mit maximal 20 Zeichen
char buffer[20];

//formatieren des Textes und ablegen in dem Array
sprintf(buffer, "mein Name ist %s", text);

//ausgebn des Textes auf der seriellen Schnittstelle
Serial.println(buffer);   // mein Name ist Stefan
```

Beispiel3:

```
int zahl = 32;
char name[20] = "Stefan";

//reservieren eines char Arrays mit maximal 20 Zeichen
char buffer[50];

sprintf(buffer, "mein Name ist %s die Zahl ist %04d", name,zahl);
```

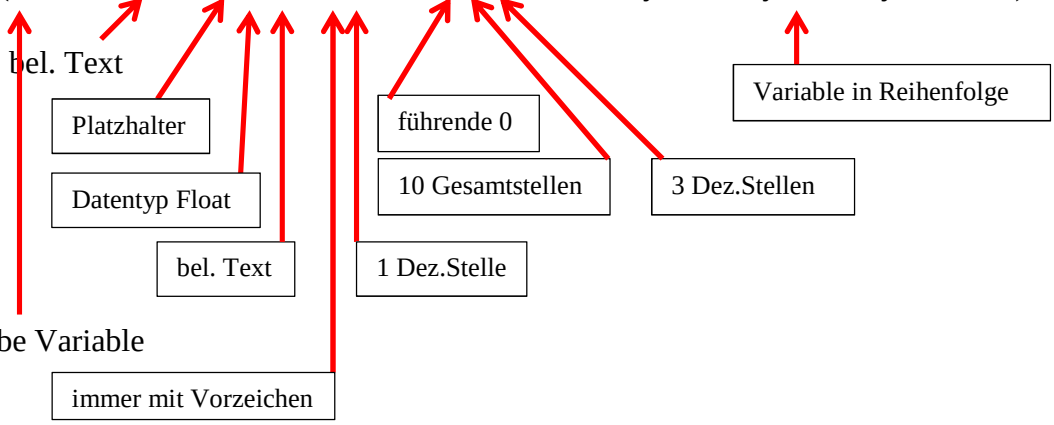
04 bedeutet Anzahl der Gesamtstellen mit führender 0 aufgefüllt
Ergebnis: mein Name ist Stefan die Zahl ist 32.0000

Beispiel4:

float myFloat = 32.4512;
//char text[]="WW"; //oder
String text="WW";
//reservieren eines char Arrays mit maximal 50 Zeichen
char buffer[50];

//formatieren des Textes und ablegen in dem Array

sprintf(buffer, "Inhalt: %f xx %+.1f xx %010.3f - %s", myFloat, myFloat, myFloat,text);



Serial.println(buffer);

Ergebnis:

Inhalt: 32.4512 xx +32.4 xx 32.451 - WW

4 Leerzeichen, da 10 Gesamtlänge

Ähnlich ist **snprintf**
Der Funktion **snprintf** wird zusätzlich die Länge des Ziel Arrays übergeben

snprintf(buffer,50, "mein Name ist %s die Zahl ist %04d", name,zahl);

7 Operatoren

x ++; // identisch mit x = x + 1, oder Erhöhung von x um +1
x –-; // identisch mit x = x - 1, oder Verminderung von x um -1
x += y; // identisch mit x = x + y, oder Erhöhung von x um +y
x -= y; // identisch mit x = x - y, oder Verminderung von x um -y
x *= y; // identisch mit x = x * y, oder Multiplikation von x mit y
x /= y; // identisch mit x = x / y, oder Division von x mit y

x == y // x ist gleich y
x != y // x ist ungleich y
x < y // x ist kleiner als y
x > y // x ist größer als y
x <= y // x ist kleiner oder gleich y
x >= y // x ist größer oder gleich y

Logisch AND:

if (x > 0 && x < 5) // nur WAHR wenn beide Ausdrücke WAHR sind

Logisch OR:

if (x > 0 || y > 0) // WAHR wenn einer der Ausdrücke WAHR ist

Logisch NOT:

if (!x > 0) // nur WAHR wenn der Ausdruck FALSCH ist

8 void

Beschreibung

Das Schlüsselwort `void` wird nur in Funktionsdeklarationen verwendet. Es zeigt an, dass die Funktion voraussichtlich keine Informationen an die Funktion zurückgibt, von der sie aufgerufen wurde.

Beispielcode

Der Code zeigt, wie `void` verwendet wird.

```
// Aktionen werden in den Funktionen `setup()` und `loop()` ausgeführt.
// Es werden jedoch keine Informationen an das größere Programm gemeldet.

void setup() {
  // ...
}

void loop() {
  // ...
}
```

9 Strukturen

9.1 If Verzweigung

```
if (someVariable >= value)
{
doSomething;
}
```

Wenn man die geschweiften klammern weg läßt dann wird „nur“ die nächste Zeile zur Bedingung hinzugezogen, die übernächste Zeile wird somit immer ausgeführt.

9.2 If Else Verzweigung

```
if (inputPin == HIGH)
{
      doThingA;
}
else
{
      doThingB;
}
```

9.3 For Schleife

```
for (Initialisierung; Bedingung; Ausdruck)
{
        doSomething;
}
```

Beispiel:

```
for (int i=0; i<20; i++)          // deklariert 'i', teste ob weniger
{  // als 20, Erhoehung um 1

digitalWrite(13, HIGH);           // schaltet Pin 13 ein
delay(250);                       // Pause fuer 1/4 Sekunde
digitalWrite(13, LOW);            // schaltet Pin 13 aus
delay(250);
                        // Pause fuer 1/4 Sekunde
}
```

9.4 break z.B. Sprung aus Schleifen

Beschreibung

`break` wird benutzt, um aus `for`, `while` oder `do…while`-Schleifen zu springen, wobei die normalen `condition` übersprungen werden. Es wird auch benutzt, um aus `switch case`-Statements zu springen.

Syntax

```
while (condition) {
   //Statement(s)
   break;
   //Statement(s)
}
```

9.5 return z.B. Beendet Funktionen

Beschreibung

Beendet eine Funktion und gibt einen Wert an die aufrufende Funktion zurück, wenn gewünscht.

Syntax

```
return
return value; // Beide Formen davon sind gültig
```

Parameter

`value`: Beliebige Variable oder Parameter. Erlaubte Datentypen: Beliebiger Datentyp.

Rückgabewert

Den Datentyp des Parameters `value`.

Beispielcode

Vergleicht einen Sensorwert mit einem Thresholdwert.

```
int checkSensor() {
  // Lies Wer von Sensor ein und vergleiche diesen mit der Konstante 400
  if (analogRead(0) > 400) {
    // Wenn Sensorwert größer als 400: Gib 1 zurück
    return 1;
  }
  else {
    // Sonst gib 0 zurück
    return 0;
  }
}
```

Das `return`-Keywort kann dazu dienen, Codeabschnitte zu testen ohne viel Code auskommentieren zu müssen.

```
void loop() {
  // Codeidee, die getestet werden soll, hier.

  return;

  // Der Rest des Sketches hier.
  // Wird niemals ausgeführt
}
```

9.6 switch...case

Beschreibung

Wie auch if-Statements, erlaubt es auch `switch case`, dass abhängig von der Bedingung in verschiedenen Situationen unterschiedlicher Code ausgeführt wird. Im Detail vergleicht `switch case` die Variablenwerte mit denen in den `case`-Statements. Wenn ein passendes `case`-Statement gefunden wird, so wird der Code in diesem `case`-Statement ausgeführt.

Das break-Keywort beendet das `switch case`-Statement und wird üblicherweise am Ende jedes `case`-Statements verwendet. Wenn kein break-Keywort
verwendet wird, führt `switch case` alle Statements aus, bis ein break-Keywort auftaucht oder das `switch case` zu Ende ist.

Syntax

```
switch (var) {
  case label1:
    // Statement(s)
    break;
  case label2:
    // Statement(s)
    break;
  default:
    // Statement(s)
    break; // Wird nicht benötigt, wenn Statement(s) vorhanden sind
}
```

Parameter

`var`: Eine Variable, die mit den eizelnen Fällen verglichen werden sollen. Erlaubte Datentypen: `int`, `char`.
`label1`, `label2`: Konstanten. Erlaubte Datentypen: `int`, `char`.

Rückgabewert **Nichts.**

Beispielcode

Das Code-Beispiel zeigt die Verwendung des switch-case-Statements.

```
switch (var) {

    case 1:
        // Tue etwas, wenn "var" 1 ist
        break;
    case 2:
        // Tue etwas, wenn "var" 2 ist
        break;
    default:
        // Tue etwas, im Defaultfall
        // Dieser Fall ist optional
        break;  // Wird nicht benötigt, wenn Statement(s) vorhanden sind
}
```

9.7 while

Beschreibung

Eine `while`-Schleife läuft solange weiter (Eventuell auch unendlich), bis die Bedingung in den Klammern `()` `false` wird. Wenn die Variable in der Schleife
sich nie ändert, läuft die Schleife unendlich. Dies kann z.B. durch das Hochzählen einer Variable oder das Lesen eines Sensorwertes erfolgen.

Syntax

```
while(condition) {
  // Statement(s)
}
```

Parameter

`condition`: Ein Ausdruck, der auf `true` oder `false` evaluiert.

Rückgabewert

Nichts.

Beispielcode

Dieser Code zählt eine Variable 200-mal um 1 hoch.

```
var = 0;
// Wiederhole etwas 200 Mal
while(var < 200) {
   // Zähle Variable x um 1 hoch
   var++;
}
```

9.8 do...while

Beschreibung

`do…while` funktioniert genauso wie eine `while`-Schleife. Der einzige Unterschied ist, dass `do…while` immer mindestens einmal ausgeführt wird,
da die Bedingung erst am Schluss der Schleife getestet wird.

Syntax

```
do {
   //Statement(s)
} while (condition);
```

Parameter

`condition`: Ein Ausdruck, der auf `true` oder `false` evaluiert.

Rückgabewert

Nichts.

Beispielcode

Liest in do-while-Schleife 100 Mal einen Sensorwert aus.

```
// Initialisiere x mit Wert 0
int  x  =  0;

do  {
   delay(50);                  // Warte 50 Millieskunden, bis der Sensor wieder einen stabilen Wert liefert.
   x  =  readSensor();       // Lies den Sensorwert
   x++;                              //Inkrementiere die Zählervariable
}  while  (x  <  100);        // Wiederhole das ganze 100 Mal
```

10 PROGMEM

Beschreibung

Speichere Daten im Flash-/Programm-Speicher statt im SRAM. Eine Beschreibung der unterschiedlichen Arten von Speicher des Arduinos findest du hier.

`PROGMEM` ist ein Variablenmodifikator, welcher nur mit den Datentypen in pgmspace.h verwendet werden sollte. Es ist eine Anweisung an den Compiler, um die Daten im Flash-/Programm-Speicher statt im SRAM zu speichern.

`PROGMEM` gehört zur pgmspace.h-Softwarebibliothek. Diese ist in allen modernen IDE-Versionen standardmäßig enthalten. Solltest du eine IDE-Version niedriger als 1.0 (von 2011) besitzen, musst du die Bibliothek erst händisch einbinden. Dies funktioniert wie folgt:

```
#include <avr/pgmspace.h>
```

Syntax

const dataType variableName[] PROGMEM = {data0, data1, data3…};`

`dataType` - Beliebiger Variablentyp `variableName` - Der Variablenname

`PROGMEM` ist ein variabler Modifikator, weshalb die Arduino-IDE alle folgenden synonymen Versionen der Syntax akzeptiert. Durch Experimente wurde allerdings festgestellt, dass in einigen Versionen der Arduino-IDE (Durch die unterschiedlichen GCC-Versionen) an einigen Stellen funktioniert und an anderen nicht. Die Beispiele unten funktionieren mit der Arduino IDE Version 13. Frühere Versionen der IDE arbeiten besser, wenn `PROGMEM` nach dem Variablennamen eingefügt wird.

```
const dataType variableName[] PROGMEM = {}; // Benutze dies
const PROGMEM dataType variableName[] = {}; // oder das,+ `const dataType
```
PROGMEM variableName[] = {}; // aber nicht diese Version!

`PROGMEM` kann auch für einzelne Variablen benutzt werden, dies ist aber nur sinnvoll, wenn ein großer Block von Daten gespeichert werden soll. Normalerweise ist das bei Arrays, Strings (die auch Arrays sind) und weiteren komplizierteren Datenstrukturen der Fall.

Die Benutzung von `PROGMEM` erfolgt in 2 Schritten. Nachdem die Daten in den Flash-/Programm-Speicher geladen wurden, müssen spezielle Methoden verwendet werden, um diese Variablen wieder auszulesen und zu bearbeiten. Diese sind auch in der pgmspace.h-Softwarebibliothek definiert.

Beispielcode

Das folgende Code-Fragment zeigt, wie ein unsigned chars (bytes) und ints (2 bytes) in `PROGMEM` geschrieben und wieder ausgelesen werden können.

```
// Speichere einige unsigned ints
const PROGMEM uint16_t charSet[] = { 65000, 32796, 16843, 10, 11234};
```

```
// Speichere einen String
const char signMessage[] PROGMEM = {"I AM PREDATOR, UNSEEN COMBATANT. CREATED BY THE UNITED STATES DEPART"};

unsigned int displayInt; // Rückgabewert der Funktion zum Auslesen der Daten
char myChar; // Definiere einen Char, um diesen zu bearbeiten

void setup() {
  Serial.begin(9600); // Initialisiere den seriellen Port
  while (!Serial);    // Warte, bis der serielle Port verbunden ist

  // Setup code:
  // Lies einen 2-Byte-Int
  for (byte k = 0; k < 5; k++) {
    displayInt = pgm_read_word_near(charSet + k); // Lies den charSet-Wert wieder aus dem Flash-/Programm-Speicher
    Serial.println(displayInt); // Gib den gelesenen Wert aus
  }

  Serial.println(); // Schreibe eine neue Zeile

  // Lies einen Char
  for (byte k = 0; k < strlen_P(signMessage); k++) {
    myChar = pgm_read_byte_near(signMessage + k); // Lies den signMessage-Wert wieder aus dem Flash-/Programm-Speicher
    Serial.print(myChar); // Gib den gelesenen Wert aus
  }

  Serial.println(); // Schreibe eine neue Zeile
}

void loop() {
  // ...
}
```

Array von Strings

Wenn mit einer großen Menge an Text gearbeitet wird (Beispiel: Projekt mit LCD-Display), ist es häufig nötig, Texte in ein Array von Strings zu packen. Dadurch, dass Strings bereits selbst Arrays sind, entsteht dadurch ein 2-dimensionales Array.

Diese großen Datenstrukturen können in den Flash-/Programm-Speicher geladen werden. Das Beispiel unten zeigt das.

```
/*
 PROGMEM string demo
 Wie man ein Stringarray in den Flash-/Programm-Speicher lädt und wieder liest.

 Für weitere Informationen siehe:
 http://www.nongnu.org/avr-libc/user-manual/pgmspace.html

 Stringarrays in den Flash-/Programm-Speicher zu laden, ist etwas komplizierter,
 das Beispiel ziegt dies jedoch.
*/

#include <avr/pgmspace.h>
const char string_0[] PROGMEM = "String 0"; // Definiere deine Strings "String 0" ist hier nur ein Beispiel
const char string_1[] PROGMEM = "String 1"; // Definiere deine Strings "String 1" ist hier nur ein Beispiel
```

```
const char string_2[] PROGMEM = "String 2"; // Definiere deine Strings "String 2" ist hier nur ein Beispiel
const char string_3[] PROGMEM = "String 3"; // Definiere deine Strings "String 3" ist hier nur ein Beispiel
const char string_4[] PROGMEM = "String 4"; // Definiere deine Strings "String 4" ist hier nur ein Beispiel
const char string_5[] PROGMEM = "String 5"; // Definiere deine Strings "String 5" ist hier nur ein Beispiel

// Initialisiere die Tabelle von Strings
const char* const string_table[] PROGMEM = { string_0, string_1, string_2, string_3, string_4,
string_5 };

char buffer[30]; // Der Buffer zum Lesen der Daten muss so lang sein wie der längste String!

void setup() {
   Serial.begin(9600); // Initialisiere den seriellen Port
   while(!Serial); // Warten, bis der serielle Port verfügbar ist
   Serial.println("OK"); // Schreibe "OK" auf den seriellen Port
}

void loop() {
   /* Spezielle Funktionen zum Lesen der Daten sind nötig. "strcpy_P" kopiert den String vom
   Flash-/Programm-Speicher in dem SRAM/ den Buffer. Stelle sicher, dass der empfangende
   String lang genug ist für den String.
 */

   // Iteriere über das Array: 6-mal, da 6 Strings im Array sind
   for (int i = 0; i < 6; i++) {
      strcpy_P(buffer, (char*)pgm_read_word(&(string_table[i])));  // Casts und Dereferenzierung des
Speichers
      Serial.println(buffer); // Gib den gelesenen Wert aus
      delay(500); // Warte eine halbe Sekunde
   }
}
```

Anmerkungen und Warnungen

Bitte beachte, dass die Variablen entweder global definiert oder als `static` definiert sein müssen, um mit `PROGMEM` zu funktionieren.

Der folgende Code wird nicht funktionieren, wenn er in einer Funktion ausgeführt wird:

```
const char long_str[] PROGMEM = "Hi, I would like to tell you a bit about
myself.\n";
```

Der folgende Code wird funktionieren, wenn er in einer Funktion ausgeführt wird:

```
const static char long_str[] PROGMEM = "Hi, I would like to tell you a bit
about myself.\n"
```

11 Das `F()`-Makro

Wenn eine Instruktion wie

```
Serial.print("Write something on  the Serial Monitor");
```

benutzt wird, wird der `String` bevor er auf den seriellen Ausgang geschrieben wird, normalerweise in den RAM gespeichert. Der RAM kann aber sehr leicht volllaufen, wenn der

Code sehr viel auf den seriellen Port schreibt. Wenn du noch freien Flash-/Programm-Speicher hast, kannst du dem Compiler sagen, dass er die Werte in den Flash-/Programm-Speicher schreiben soll:

```
Serial.print(F("Write something on the Serial Monitor that is stored in
FLASH"));
```

12 tone()

Beschreibung

Generiert einen Ton mit der angegebenen Frequenz (und 50% duty cycle) auf einem Pin. Eine Dauer des Tons kann festgelegt werden; Sonst wird der Ton abgespielt, bis ein noTone() aufgerufen wird. Der Pin kann an einen Piezo oder an einen Lautsprecher angeschlossen werden, um Töne abzuspielen.

Nur ein Ton kann gleichzeitig abgespielt werden. Wenn bereits ein Ton auf einem anderen Pin abgespielt wird, hat `tone()` auf einem 2. Pin keinen Effekt. Wenn der Ton auf dem gleichen Pin gespielt wird, setzt der Aufruf der Funktion die Frequenz des Tons.

Die Nutzung der `tone()` Funktion interferiert mit PWM outputs auf Pin 3 und 11 (Auf Boards außer dem Arduino Mega).

Es ist nicht möglich, Töne mit einer Frequenz niedriger als 31Hz zu erstellen. Für technische Details, siehe Brett Hagman's post.

Syntax

```
tone(pin, frequency)
tone(pin, frequency, duration)
```

Parameter

`pin`: Der Arduino-Pin, auf dem der Ton generiert werden soll.
`frequency`: Die Frequenz des Tons in Hertz. Erlaubte Datentypen: `unsigned int`.
`duration`: Die Dauer des Tons in Millisekunden (optional). Erlaubte Datentypen: `unsigned long`.

Rückgabewert

Nichts.

13 noTone()

Beschreibung

Stoppt die Tonerzeugung, die von `tone()` erzeugt wurde. Macht nichts, wenn kein Ton erzeugt wurde.

Syntax

```
noTone(pin)
```

Parameter

`pin`: Der Arduino-Pin, auf dem die Tonerzeugung gestoppt werden soll.

Rückgabewert

Nichts.

Anmerkungen und Warnungen

Wenn du verschiedene Töne auf mehreren Pins spielen willst, musst du zunächst mit `noTone()` auf einem Pin den Ton stoppen und anschließend mit ../tone[tone()] auf dem nächsten Pin einen neuen Ton generieren.

14 pulseIn()

Beschreibung

Liest einen Wert von einem vorgegebenen Digitalpin ein, entweder `HIGH` oder `LOW`. Wenn **value** z.B. HIGH ist, wartet `pulseIn()` darauf,dass der Pin auf den Wert HIGH wechselt, startet einen Timer und wartet anschließend darauf, dass der Pin wieder auf LOW wechselt. Daraufhin stoppt `pulseIn()` den Timer. Gibt die Länge des Impulses in Mikrosekunden zurück. Stoppt und gibt 0 zurück, wenn ein bestimmter Timeout erreicht wird.

Das Timing der Funktion wurde empirisch getestet und wird vermutlich bei einem längeren Impuls falsch messen. Die Funktion funktioniert mit Impulsen der Länge von 10 Mikrosekunden bis 3 Minuten.

Syntax

```
pulseIn(pin, value)
pulseIn(pin, value, timeout)
```

Parameter

`pin`: Die Arduino-Pinnummer, an der der Impuls gelesen werden soll. Erlaubte Datentypen: `int`.
`value`: Welche Art von Impuls gelesen werden soll: Entweder HIGH oder LOW. Erlaubte Datentypen: `int`.
`timeout` (optional): Die Anzahl an Mikrosekunden, die gewartet werden soll, bis ein Impuls gemessen wurde; Default-Wert ist eine Sekunde. Erlaubte Datentypen: `unsigned long`.

Rückgabewert

Die Länge des Impulses (in Mikrosekunden) oder 0, wenn kein Impuls vor dem Timeout gemessen wird. Datentyp: `unsigned long`.

Beispielcode

Das Beispiel gibt den Zeitraum eines Impulses auf Pin 7 aus.

```
int pin = 7;
unsigned long duration;

void setup() {
  Serial.begin(9600);
  pinMode(pin, INPUT);
}

void loop() {
  duration = pulseIn(pin, HIGH);
  Serial.println(duration);
}
```

15 pulseInLong()

Beschreibung

Liest einen Wert von einem vorgegebenen Digitalpin ein, entweder `HIGH` oder `LOW`. Wenn der **Wert** z.B. **HIGH** ist, wartet `pulseInLong()` darauf, dass der Pin auf den Wert **HIGH** wechselt, startet einen Timer und wartet anschließend darauf, dass der Pin wieder auf **LOW** wechselt. Daraufhin stoppt `pulseInLong()` den Timer. Gibt die Länge des Impulses in Mikrosekunden zurück. Stoppt und gibt 0 zurück, wenn ein bestimmter Timeout erreicht wird.

Das Timing der Funktion wurde empirisch getestet und wird vermutlich bei einem längeren Impuls falsch messen. Die Funktion funtioniert mit einer Stromstößen der Länge von 10 Mikrosekunden bis 3 Minuten. Bitte beachte auch, dass, wenn ein Pin, der bereits **HIGH** ist, wenn die Funktion aufgerufen wird, zunächst auf **LOW** und dann wieder auf **HIGH** wechselt, bevor die Funktion startet. Diese Routine kann nur benutzt werden, wenn Interrupts aktiviert werden. Die beste Auflösung wird erreicht, wenn große Intervalle verwendet werden.

Syntax

```
pulseInLong(pin, value)
pulseInLong(pin, value, timeout)
```

Parameter

`pin`: Die Arduino-Pinnummer, an der der Impuls gelesen werden soll. Erlaubte Datentypen: `int`.
`value`: Welche Art von Impuls gelesen werden soll: Entweder HIGH oder LOW. Erlaubte Datentypen: `int`.
`timeout` (optional): Die Anzahl an Mikrosekunden, die gewartet werden soll, bis ein Impuls gemessen wurde; Default-Wert ist eine Sekunde. Erlaubte Datentypen: `unsigned long`.

Rückgabewert

Die Länge des Impulses (in Mikrosekunden) oder 0, wenn kein Impuls vor dem Timeout gemessen wird. Datentyp: `unsigned long`.

Beispielcode

Das Beispiel gibt den Zeitraum eines Impulses auf Pin 7 aus.

```
int pin = 7;
unsigned long duration;

void setup() {
  Serial.begin(9600);
  pinMode(pin, INPUT);
}

void loop() {
  duration = pulseInLong(pin, HIGH);
  Serial.println(duration);
}
```

Anmerkungen und Warnungen

Diese Funktion verlässt sich auf `micros()` und kann somit in noInterrupts() Kontexten nicht verwendet werden.

16 ESP32 Neustart

ESP.restart();

17 Datum Zeit

Datum und Zeit von RTC-Chip lesen und formatiert anzeigen

```
void loop(){    //-------------------void loop Beginn----------------------
-------------------

Time t = rtc.time();     // Zeit vom Chip abrufen
char zeit[50];
snprintf(zeit,sizeof(zeit),"%02d.%02d.%04d
%02d:%02d:%02d",t.date,t.mon,t.yr,t.hr,t.min,t.sec);  //Formatierung Datum
und Zeit
//lcd.clear();   //LCD löschen
lcd.setCursor(0,0);  //LCD 2.Stelle 1. Zeile
lcd.print(zeit);      //Ausgabe auf LCD akt Datum unmd Zeit formatiert

switch(t.day){
  case 1:
  wochentag = "So";
  break;
  case 2:
  wochentag = "Mo";
  break;
  case 3:
  wochentag = "Di";
  break;
  case 4:
  wochentag = "Mi";
  break;
  case 5:
  wochentag = "Do";
  break;
  case 6:
  wochentag = "Fr";
  break;
  case 7:
  wochentag = "Sa";
  break;
}

Serial.println(wochentag);
```

18 Dateizugriff (ESP32)

SPIFFS.h muss geladen werden
FS.h

Es wird der SPIFFS Speicher benutzt. Größe kann in IDE geändert werden!

Wird für den Pfad nichts weiter als der Dateiname angegeben, so wird im Ordner /data gespeichert. Der kann auch von der Arduino IDE hochgeladen werden.

SPIFFS.open(**Pfad,**"Modus"**);**
Modus: r =read w= write

SPIFFS.format	formatiert Speicher
SPIFFS.exists(Pfad)	prüft ob Pfad existiert Rückgabe: true oder false
SPIFFS.remove(Pfad) l	öscht die angegebene Datei
SPIFFS.rename(alterName,neuer Name)	Umbenennen von Dateien

Infos zum Speicher:

FSInfo fs_info;
SPIFFS.info(fs_info);

18.1 Schreiben in Datei

```
//speichern in Datei

if(SPIFFS.begin()){
  //Serial.println("SPIFFS initialisierung OK");
}
else{
  //Serial.println("SPIFFS initialisierung Fehler");
}

File x = SPIFFS.open("/uhr1.txt","w");
x.print(uhr1ges1);
x.close();   //Datei schließen
```

18.2 Lesen aus Datei

```
if(SPIFFS.begin()){
  //Serial.println("SPIFFS Initialisierung OK");
}
else{
  //Serial.println("SPIFFS Initialisierung Fehler");
}
//Lesen der Datei
File f = SPIFFS.open("/uhr1.txt","r");
for(i=0;i<f.size();i++){
  uhr1ges= uhr1ges+(char)f.read();
}
f.close();  //Datei schließen
```

19 Serielle Kommunikation

Serial.begin(9600);

Der Serial.begin()- Befehl aktiviert die serielle Schnittstelle mit der wir über den Computer kommunizieren können. Der Wert "9600" ist die Übertragungsrate. Man sagt auch "9600 Baud".

Serial.println("Das ist der serielle Monitor");

Dieser Befehl gibt einen Text auf den seriellen Monitor aus und springt direkt in die nächste Zeile. "println" ist voll ausgesprochen "print line".

Serial.print("Das ist der serielle Monitor");

ohne Zeileenumbruch

```
if (Serial.available() > 0)
  {
                Anweisungen z.B. nummer =Serial.read();
  }
```

Prüft ob Serielle Schnittstelle verfügbar ist (>0)
wenn ja dann die Anweisungen

nummer =Serial.read();

Hier wartet dein Arduino auf eine Eingabe. Diese Eingabe wird dann in der Variable "nummer" gespeichert.

GANZ WICHTIG: Wir übergeben keine "reine" Zahl. Wir übergeben ein Zeichen oder ein Character (char). Der entsprechende Wert für dieses Zeichen wird dann in die Variable abgespeichert. Wenn du also eine 0 abschickst, wird nicht die Zahl 0 abgespeichert, sondern der ASCII- Code für '0'.

Serial.flush(); //seriellen Puffer löschen

Serial.readString()

Beschreibung
Serial.readString() liest Zeichen aus dem seriellen Puffer in einen String. Die Funktion wird abgebrochen, wenn eine Zeitüberschreitung auftritt (siehe setTimeout()).
Serial.readString() erbt von der Stream-Dienstklasse.

Beispiel1:

```
int incomingByte = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {

  if (Serial.available() > 0) {

      incomingByte = Serial.read();

      Serial.print("Ausgabe:");

      Serial.println(incomingByte, DEC);
  }
}
```

Beispiel2:

- Sie können Serial.readString () und Serial.readStringUntil () verwenden, um -Strings aus Serial on arduino zu analysieren
- Sie können Serial.parseInt () auch verwenden, um Ganzzahlwerte aus seriell zu lesen

Code-Beispiel

```
int x;
String str;

void loop()
{
    if(Serial.available() > 0)
    {
        str = Serial.readStringUntil('\n');
        x = Serial.parseInt();
    }
}
```

Der seriell zu sendende Wert wäre "my string\n5" und das Ergebnis wäre str = "my string" und x = 5

Beispiel warten auf Eingabe:

```
void loop() {

char eingang;
String gesamt;

  while (Serial.available() == 0) {};   //Wartet auf eine Eingabe im
Seriellen Monitor
  while (Serial.available()){           // Zeichenweise Eingabe wird zu
einem Gesamtstring

    eingang = Serial.read();
    gesamt = gesamt + (String)eingang;
  }
  Serial.println(gesamt);
delay(100);
if (gesamt.indexOf("www")>=0){     //bei Eingabe von www wird gewählt

      Serial.println("wählen");
      sim808.callUp(PHONE_NUMBER);    //wählen der Nummer

}

}
```

mehrere Strings ausgeben: **gleicher VariableTyp !!!**

Serial.println("Breite: " + String(Part2) + " " + String(Part3));

20 Input Output der Pins

```
pinMode(pin, OUTPUT);              // setzt ‘pin’ als Ausgang

pinMode(pin, INPUT_PULLUP );       // setzt ‘pin’ als Eingang und
                                   // schaltet den 'Pullup' Widerstand ein

value = digitalRead(Pin);          //liest den Zustand des Pins aus

digitalWrite(pin, HIGH);           // setzt 'pin' auf high (an)


value = analogRead(pin);           // setzt 'value' gleich mit dem Analogwert von 'pin'

analogWrite(pin, value);           // schreibt 'value' auf den analogen 'pin'
```

21 Timer

21.1 Timer mit millis();

millis(); gibt Zeit in mSek nach letztem Start an

```
/* Timer mit millis();
 * 2 LEDs blinken mit unterschiedlichen Zeiten mit Timer millis()
 */

  long frequenz1 = 8000;  //in mSek
  long dauer1 =1000;    //in mSek
  long blinkzeit1;

  long frequenz2 = 2000;  //in mSek
  long dauer2 =1000;    //in mSek
  long blinkzeit2;


void setup() {    //**********************Setup************************
 Serial.begin(115200);

 pinMode(25,OUTPUT);
 pinMode(26,OUTPUT);

}    //***********************************Setup
Ende*************************


void loop() {    //*************************Loop*********************


//LED 1 GPIO 25-----------------------------------------
if (millis() >= blinkzeit1 + frequenz1){
  digitalWrite(25,HIGH);
  Serial.println(millis());
}

if (millis() >= blinkzeit1 + frequenz1 + dauer1) {
  digitalWrite(25,LOW);
  blinkzeit1 = millis() - dauer1;
}
//-------------------------------------------------

// LED 2 GPIO 26 ----------------------------------------
if (millis() >= blinkzeit2 + frequenz2){
  digitalWrite(26,HIGH);
 }

if (millis() >= blinkzeit2 + frequenz2 + dauer2) {
  digitalWrite(26,LOW);
  blinkzeit2 = millis() - dauer2;
}
//-------------------------------------------------

if (millis() >= 60000){  //evt. bei 60000 mSek Neustart oder etwas anderes
   //ESP.restart();
}

}  //*********************************Loop Ende**********************
```

21.2 mit Library

21.2.1 mit Lib TimerOne.h

Verwendung des Timer1
Als erstes muss die folgende Zeile an den Programmanfang eingefügt werden:
#include <TimerOne.h> //Library einbinden
Dann muss im Setup() der Timer1 konfiguriert werden:
Timer1.initialize(100000); // Timerlänge 100.000µsek (0,1 sek)
Timer1.attachInterrupt(timerIsr); // ISR aufrufen
Hinter Timer1.initialize muss die Zeit in Mikrosekunden angegeben werden. Das ist das Intervall,
indem die Interrupt-Service-Routine (ISR) aufgerufen wird. Diese wird dann als Funktion aufgerufen und muss den unter Timer1.attachInterrupt angegebenen Namen haben.

Beispiel:

```
#include <TimerOne.h>
void setup()
{
pinMode(13, OUTPUT); // LED auf dem Board
Timer1.initialize(100000); // Timerlänge 0,1sek
Timer1.attachInterrupt( timerIsr ); // ISR aufrufen
}
void loop()
{
// Haupprogrammschleife
// TODO: Eigenes Programm hier rein
}
/// -------------------------------------
/// Eigene ISR Timer Routine
/// -------------------------------------
void timerIsr()
{
digitalWrite( 13, digitalRead( 13 ) ^ 1 ); // Toggle LED
}
```

21.2.2 mit Lib Ticker.h

```
/*Timer mit Ticker.h
*
* wird immer in Schleife ausgeführt
*
*/

#include <Ticker.h>
Ticker ticker;
Ticker ticker2;

void setup() {
  pinMode(25,OUTPUT);
  ticker.attach(20,anFunc);    //Tickerobjekt(Zeit in Sekunden,Funktion die
aufgerufen wird)
  ticker2.attach(22,ausFunc);

}

void loop() {}


void anFunc(){
  digitalWrite(25,HIGH);
  //ticker.detach();  //Schaltet Tricker aus ansonsten immer wieder
}

void ausFunc(){
  digitalWrite(25,LOW);
  ticker2.detach();  //Schaltet Tricker2 aus ansonsten immer wieder
}
```

21.3 Hardware Timer

```
/*
Timer

*/
hw_timer_t * timer = NULL;  //Timer erstellen

volatile byte state = LOW;

void IRAM_ATTR onTimer(){  //++Funktion+++Wenn Timerzeit erreicht ist+++

  state = !state;
  digitalWrite(25, state);
  Serial.println(state);  //Rückgabe 0 und 1
}  //++++++++++++++++++++++++Funktion Ende++++++++++++++++++++

void setup() {    //*********Setup Beginn*****************************
  Serial.begin(115200);

  pinMode(25, OUTPUT);

  // 1. Timer von 4
  // 1 tick take 1/(80MHZ/80) = 1us so we set divider 80 and count up auf
1µs Stellen
  timer = timerBegin(0, 80, true);

  /* Attach onTimer function to our timer */
  timerAttachInterrupt(timer, &onTimer, true);

  /* Set alarm to call onTimer function every second 1 tick is 1us
  => 1 second is 1000000us */
  /* Repeat the alarm (third parameter) */
  timerAlarmWrite(timer, 100000, true);  // Timerzeit in µSek 1Sekunde

  timerAlarmEnable(timer);   //Timer starten

  Serial.println("start timer");

}    //********************Setup Ende*****************

void loop() {

delay (10000);

}
```