7.0. Arduino Programmeren voor Beginners – Deel 7: Strings

In deze les, deel 7 van Arduino Programmeren voor Beginners, gaan we kijken naar Strings of te wel tekst.

We hebben tot dusver met nummers en booleans gewerkt, maar een beetje kunnen spelen en werken met tekst zou de interactie met de gebruiker stukken beter maken natuurlijk. We hebben al wel wat met tekst gedaan, maar nog niet echt wetende wat we precies met tekst kunnen doen.

Inhoudsopgave

7.1. Werken met Tekst of te wel: Strings!

Het is natuurlijk leuk om met nummers en booleaans te werken, maar tekst is toch iets wat we missen en tekst speelt een belangrijke rol als we met mensen gaan communiceren. Ook al moet er natuurlijk wel gezegd worden dat Arduino er misschien niet veel aan heeft omdat het geen toetsenbord en scherm heeft. Toch is het werken met tekst ook voor de Arduino van belang.

Eigenlijk hebben we tekst, of te wel strings, al een paar keer gebruikt. Elke keer als we “Serial.print() ” of “Serial.println() ” gebruiken, sturen we beiden een string. Vergeet niet dat in de taal C, een tekst (of: string) ingesloten is met dubbele haakjes!

Om nu te kunnen gaan werken met strings (tekst) moeten we een beetje meer inzicht hebben in wat nu werkelijk een string is.

We kunnen als eerste beginnen met het zogenaamde “string” (allemaal kleine letters!) datatype – hierbij is een string een zogenaamde “array” van karakters (char), of te wel een “array of char”. In het Engels klinkt dit allemaal logischer, lees het als “een lijst met karakters”.

De twee manier om met strings te gaan werken is met het “String” object (let goed op de hoofdletter “S”!).

We zullen beiden bespreken in de volgende paragrafen …

7.2. Array of Characters (string)

Een “array” (Array Data Type, Array Data Structuur) kan gezien worden als een verzameling elementen welke allemaal van hetzelfde data type zijn. Hierbij kan ieder element van de “lijst” (array) met een index nummer benaderd worden. In de meest eenvoudige vorm is een array dus gewoon een lijst, en in het geval van een string (allemaal kleine letters!) is dat ook het geval. Een lijst van karakters.

Een string (kleine letters !!!) is een ARRAY VAN KARAKTERS (een lijst van letters)

De variabele die we als array gaan definiëren wijst ook weer naar de geheugenlocatie waar de waarden zijn opgeslagen. In het geval van een array echter, omdat er meerder waarden kunnen zijn, wijst het naar het eerste element in de lijst. Even goed onthouden!

Een string, met kleine letters, heeft nog een kleine eigenaardigheid die we ook even moeten onthouden; een string eindigt altijd met een zogenaamd NULL-karakter, zodat de Arduino weet dat het aan het einde van de tekst is aangekomen.

Ehm, what zeg je nou toch?
Ja, een NULL-karakter, of te well een “niks”. Dit is een speciaal karakter, en het eerste karakter, in de lijst van karakters die de Arduino kent, en heeft eigenlijk geen waarde – het kan niet worden weergegeven. Voor normale tekst hebben we dus geen toepassing voor dit karakter, wat het dus uitstekend maakt voor het einde aangeven van een string (zie ook onze karakter lijst).

Je hoeft hier nog niet veel mee te doen, omdat de meeste functies hier al rekening mee houden, maar je moet het wel even onthouden.

Een ander belangrijk aspect van arrays is dat ze beginnen te tellen bij nul, dus de index voor het eerste element in een array is “0”, voor het tweede element dus “1”, etc. Dit noemt men zero indexed (nul index) of zero based numbering (nul gebaseerd nummeren) – dus het starten met tellen bij nul.

Een string (Array of Characters), wordt afgesloten met een NULL character.

Arrays zijn ZERO INDEXED, wat wil zeggen dat men start te tellen met nul.

Een voorbeeld van een string declaratie (of definitie) had dit kunnen zijn: string Naam = “Hans”;

Echter, dat werkt niet … je krijgt een melding dat “string” niet gedefinieerd is (was not defined).

De correct manier is: char Naam[] = "Hans";

Deze instructie definieert (declareert) een array van char (karakters of letters). De vierkante haken geven vervolgens aan de compiler door dat de compiler maar moet bepalen hoeveel elementen er in de array gaan zitten. We kunnen tussen de vierkante haken ook een nummer zetten, wat dan aangeeft hoeveel elementen er in de array moeten gaan zitten – maar dan moet je dus wel in de gaten houden dat de lengte van de array gelijk moet zijn aan het aantal letters in de tekst PLUS het NULL karakter aan het einde. Het is daarom niet ongebruikelijk om dat nummer weg te laten als we toch al meteen de tekst weten.

In het voorbeeld heeft de array dus 5 elementen: De letters “H”, “a”, “n” en “s” en het NULL-karakter. De compiler heeft dat dan voor jou bepaald.

We kunnen de array ook een lengte opgeven, en voor ons voorbeeld zou dat dus het volgende zijn: char Name[5] = "Hans";

Het is goed om te weten dat als je een groter nummer gebruikt dan wat nodig is, dat dit geen kwaad kan. Het neemt alleen wat meer geheugenruimte in beslag, wat handig kan zijn als de tekst groter wordt als het programma draait. Als er echter niet meer tekst in de array geplaatst gaat worden, dan is dat natuurlijk wel zonde voor de verkwisting van geheugen.

Als het nummer echter te laag is, dan zul je een foutmelding zien, of rare bijverschijnselen zien als het programma draait.

Als we de lengte van een string array niet opgeven dan bepaald de compiler de benodigde lengte – maar dat gebeurt slechts eenmalig!

Het aanspreken van een element in de array, buiten de gedefinieerde lengte van de array, kan ongewenste resultaten leveren of programma crashes veroorzaken!

In beide voorbeelden ziet de array er zo uit:

String Array	
positie (index)	inhoud van geheugen locatie
0	H
1	a
2	n
3	s
4	NULL

De variabele “Naam” wijst dus naar de geheugenlocatie van de letter “H” van de string, welke positie 0 (nul) heeft en het index nummer is dus ook nul.

We kunnen deze letter zelfs individueel ophalen, bijvoorbeeld het weergeven (print) van “Naam[2]” resulteert in “n”, “Name[0]” levert een “H”, etc.
Als we de hele variabele “Naam” naar “print” sturen, dan wordt het begin adres doorgegeven, maar omdat we geen specifieke positie hebben opgegeven, zla het programma alle karakters weer gaan geven tot het NULL karakter wordt gevonden. Dus de hele inhoud wordt weergegeven.

```
char Naam[] = "Hans";

Serial.println(Naam[2]); // print "n"
Serial.println(Naam[0]); // print "H"

Serial.println(Naam);    // print "Hans"
```

We kunnen echter ook een nieuwe tekst aan de variabele geven of een enkel karakter wijzigen.

Waarden toewijzen aan een string

We zijn al bekend met variabelen, en een beetje bekend met strings (tekst) definitie en initiële inhoud. Bijvoorbeeld als we de tekst “Hans” willen veranderen naar “Bram”?

Vreemd genoeg werkt het volgende helaas niet: Name = "Bram";

De Arduino compiler geeft dan een foutmelding “invalid array assignment”. Of te wel “verkeerde array toewijzing”. Maar een string is toch een string?
Nou eigenlijk niet. Kun je jezelf nog herinneren dat we zeiden dat een variabele naar de geheugenlocatie van de waarde wijst? Dus bij een array, naar het eerste element van de “lijst”? Het is dus een geheugenlocatie van een char (karakter) en dus niet een string. Geloof me, dit probleem

zul je nog wel een paar keer tegen komen, en het is één van de redenen waarom ik geen fan ben van de programmeertaal C. Ik ben meer een Pascal fan – en er zullen legio mensen zijn die het daar niet mee eens zijn. Maar voor iedere taal geldt: welke taal wordt ondersteund door het apparaat waar je mee aan de slag gaat, en welke programmeertaal ligt jou persoonlijk het beste.

In ieder geval, door dit probleempje wordt het toewijzen van een nieuwe waarde dus wat lastiger voor strings. Je zou natuurlijk letter voor letter de oude string kunnen gaan aanpassen, maar dat is niet echt handig he? Gelukkig is er een functie die dit voor ons doet: strcpy() .

Deze functie moet je lezen als “String Copy” of te wel “String Kopiëren”, welke je 2 parameters moet meegeven.
Als eerste moeten we de bestemming opgeven (in het Engels: destination) waar de tekst naartoe gekopieerd dient te worden.
Als tweede parameter geven we de bron (Engels: source) op, dus waar de tekst vandaan gaat komen.

De bestemming (destination) moet natuurlijk wel een variabele zijn, van het juiste datatype, zodat het de informatie ook werkelijk kan ontvangen. De bron daarentegen mag een variabele, constante of gewoon tekst zijn (tussen dubbele haakjes).

Kijk in het volgende voorbeeld naar regel 12 waar we strcpy hebben gebruikt:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    char Naam[] = "Hans";
6
7    Serial.println(Naam[2]); // print "n"
8    Serial.println(Naam[0]); // print "H"
9
10   Serial.println(Naam);    // print "Hans"
11
12   strcpy(Naam, "Bram");
13
14   Serial.println(Naam[2]); // print "a"
15   Serial.println(Naam[0]); // print "B"
16
17   Serial.println(Naam);    // print "Bram"
18 }
19
20 void loop() {
21   // leave empty for now
22 }
```

Heel erg speciale tekens … of te wel “Escape Characters”

Soms willen we tekst opbouwen met tekens die een string zouden “breken” of onmogelijk maken om de betreffende tekens in te voeren. Er zijn een aantal van dit soort speciale tekens, bijvoorbeeld de dubbele haakjes die we gebruiken om rond een string tekst te zetten. De compiler denkt immers bij de tweede dubbele haakjes dat de string klaar is, en alles wat daar achter staat wordt dus een onduidelijke bedoeling.

Een voorbeeld waarbij dit dus fout gaat is: Serial.println("Hallo "gast", welkom bij Arduino");

Overigens zul je dit in de code zien, omdat de tekst kleuren in de Arduino IDE ineens gek gaan doen … deze tekst kleuren noemt men “code highlighting”.

De code highlighting van de Arduino IDE tekst editor, laat je zien dat een string “breekt” omdat de kleuren van de tekst van jouw code ineens raar gaat doen.

Onderstaand voorbeeld illustreert dat.
De eerste regel is onze foute regel. Je ziet dat trefwoorden die van belang zijn oranje worden weergegeven, en dat een string een beetje groenachtig is. Echter, in de eerste regel zien we ineens dat “gast” zwart is en dus geen onderdeel uitmaakt van de string.

De tweede regel laat zien hoe de string er uit zou zien als we de dubbele haakjes rond “gast” weg laten – de hele tekst is groenachtig en dus een gehele string.

De derde regel laat ons de zogenaamde “escape” methode zien. Om namelijk dit soort bijzonder tekens in te kunnen voeren, moeten we een zogenaamde backslash (\) voor het bijzondere teken zetten zodat de compiler weet dat het teken direct na de backslash een bijzonder teken is.

```
Serial.println("Hallo "gast", welkom bij Arduino");
Serial.println("Hallo gast, welkom bij Arduino");
Serial.println("Hallo \"gast\", welkom bij Arduino");
```

Arduino Code Highlighting Example

De correcte manier is dus: Serial.println("Hallo \"gast\", welkom bij Arduino");

Let wel op, want als je meerdere speciale karakters achter elkaar zet dan moet je ze ieder voorzien van een backslash (\), dus als bijvoorbeeld als we meerdere dubbele haakjes rond het woord “gast” gaan zetten: Serial.println("Hallo \"\"gast\"\", welkom bij Arduino");

Deze truc is ook nodig voor een bepaald aantal andere speciale teken (kijk in de ASCII HTML tabel, in de “Esc” kolom). Als de dus een nieuwe regel willen beginnen in een string, dan moeten we ASCII karakter 10 gebruiken (lf = line feed = nieuwe regel), wat we schrijven als “\n”.

De meest gebruikte speciale karakters zijn:

Meest gebruikte speciale karakters		
Escape “code”	ASCII	Purpose
\n	10	LF (Line Feed) – Start een nieuwe regel.
\r	13	CR (Carriage Return) – Ga naar het begin van een regel
\t	9	Tab – Ga naar de volgende tab-stop (tab)
\”	34	Dubbele aanhalingstekens
\’	39	Enkele aanhalinstekens
\\	92	Backslash – zodat we die dus ook kunnen typen

Je hoeft ze echt niet uit je hoofd te leren hoor, je zult ze vanzelf nodig hebben en na een paar keer blijven ze vanzelf in je hoofd plakken.

Character versus String of te wel Enkele versus Dubbele Aanhalingstekens

We hebben net al aangegeven dat we individuele karakters kunnen veranderen. Dus de vraag: hoe doen we dat dan?

We hebben al kennis gemaakt met de dubbele aanhalingstekens (“) om een reeks karakters in te sluiten zodat het een string wordt.
Hiermee kunnen we ook een string van slechts 1 karakter maken (“A”) of zelfs een string van geen karakters (“”).
Maar dit zijn allemaal strings.

Dubbele aanhalingstekens worden gebruikt voor strings,
en een string kan nul of meer karakters lang zijn.

De volgende code zal niet werken, wanneer we slechts een enkel karakter willen veranderen van een string (array of char):

```
Naam[1] = "H"; // dit zal een foutmelding geven
```

De foutmelding (in het Engels: error of error message) is: invalid conversion from 'const char*' to 'char' wat ons zegt dat we een verkeerd datatype proberen toe te wijzen aan een van onze array elementen. In andere woorden: we proberen een string toe te wijzen aan karakter (char). Om het makkelijk te onthouden: zoals we weten, bestaat een string uit de tekst gevolgd door een NULL-karakter, terwijl de char slechts een enkele karakter is.

Om dit toch te kunnen doen, moeten we dus een char gebruiken in plaats van een string. Een enkele char (karakter) geven we daarom aan met enkele aanhlaningstekens (‘). Als ezelsbruggetje: enkel aanhalingsteken = enkel karakter, meerdere aanhalingstekens = meerdere karakters.

Enkele aanhalingsteken worden gebruikt voor char(acters),
Wat slechts een enkel karakter (char) is.

Het volgende voorbeeld zal dus wel werken:

```
Name[1] = 'H'; // this works!
```

Laten we het voorbeeld aanpassen waar we van “Hans” de naam “Bram” maken.

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    char Naam[] = "Hans";
6
7    Serial.println(Naam[2]); // print "n"
8    Serial.println(Naam[0]); // print "H"
9
10   Serial.println(Naam);    // print "Hans"
11
12   Naam[0] = 'B';
13   Naam[1] = 'r';
14   Naam[2] = 'a';
15   Naam[3] = 'm';
16
17   Serial.println(Naam[2]); // print "a"
18   Serial.println(Naam[0]); // print "B"
19
20   Serial.println(Naam);    // print "Bram"
21 }
22
23 void loop() {
24   // leave empty for now
25 }
```

Zie je hoe we ieder individueel element van de array hebben gewijzigd?
Omdat “Hans” en “Bram” alle twee even lang zijn, kunnen we “Naam[4]” overslaan. Deze had het NULL-karakter moete krijgen, maar dat had dit element al als overblijfsel van “Hans”.

Echter, mocht de nieuwe tekst korter worden, dus als we bijvoorbeeld de tekst “Hans” naar “Max” (mijn andere neefje) willen omzetten, dan MOETEN we de NULL charakter wel gaan invullen:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    char Naam[] = "Hans";
6
7    Serial.println(Naam[2]); // print "n"
8    Serial.println(Naam[0]); // print "H"
9
10   Serial.println(Naam);    // print "Hans"
11
12   Naam[0] = 'M';
13   Naam[1] = 'a';
14   Naam[2] = 'x';
15   Naam[3] = 0;
16
17   Serial.println(Naam[2]); // print "x"
18   Serial.println(Naam[0]); // print "M"
19
20   Serial.println(Naam);    // print "Max"
21 }
22
23 void loop() {
24   // leave empty for now
25 }
```

Nu zijn dus, als je opgelet hebt, bevatten zowel “Naam[3]” als “Naam[4]” een NULL character (wat hetzelfde is als nul).

Zie je hoe we Naam[3] een nummer toewijzen? Dus een nummer toewijzen aan een char?
Dit kan omdat we een nummer kunnen toewijzen zolang dat nummer tussen 0 en 256 blijft (een byte) omdat dit overeenkomt met de lengte van een char. Zie ook de ASCII tabel waar je dit ook kunt zien – kijk maar eens in de kolom “DEC”.

We kunnen ons voorbeeld zelfs omschrijven door alleen maar ASCII codes te gebruiken:

```
// Met karakters
Naam[0] = 'B';
Naam[1] = 'r';
Naam[2] = 'a';
Naam[3] = 'm';

// Met ASCII codes
Naam[0] = 66;  // = 'B'
Naam[1] = 114; // = 'r'
Naam[2] = 97;  // = 'a'
Naam[3] = 109; // = 'm'
```

Uiteraard is het gebruik maken van ASCII tekens niet altijd even voor de hand liggend als we de tekst van een string willen veranderen. Er zijn echter situaties waarbij dit juist handig is.

Als een snel voorbeeld, gaan we alle letters van het alfabet laten zien en daarvoor zullen we een karakter veranderen in een “for” loop, waarbij we de letters ‘A’ t/m ‘Z’ toewijzen en weergeven. Bedenk dat in ASCII ‘A’ = 65 en ‘Z’ = 90.

De “for” loop gebruiken we dus om te tellen van 65 tot en met 90, als byte, en wijzen de waarde toe aan Letter[x] en geven dan vervolgens deze weer:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    char Letter[] = " ";
6
7    for(byte A=65; A<=90; A++) {
8      Letter[0] = A;
9      Serial.println(Letter[0]);
10   }
11 }
12
13 void loop() {
14   // leave empty for now
15 }
```

Nu dat we weten dat we tekens ook met enkele quotes kunnen opgeven, en dat een “for” loop met zogenaamde “enumerated” data types kan werken, kunnen we de loop verassend makkelijk maken door “Char” te gaan tellen in plaats van een integer of een byte.

Vergeet dus niet dat een char ook enumerated is en dus geteld kan worden. Een char is in principe een byte en dus een geheel nummer.

N.b. : Omdat we char gaan “tellen” dus niet vergeten dat je enkelvoudige aanhalingstekens moet gebruiken!

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    char Letter[] = " ";
6
7    for(char A='A'; A<='Z'; A++) {
8      Letter[0] = A;
9      Serial.println(Letter[0]);
10   }
11 }
12
13 void loop() {
14   // leave empty for now
15 }
```

Beide voorbeelden leveren dus hetzelfde: een lijst van letters in het alfabet, in hoofdletters weergegeven.

7.3. Andere Array Functies: strlen, sizeof, strcat

We hebben de functie “strcpy()” (lees als “String Copy”) al gezien, maar er zijn nog een aantal handige functies voor array’s die we even snel gaan bekijken.

strlen()

“strlen()” (lees als: string Length – of te wel: string lengte) heeft maar 1 parameter nodig, de string (array) en geeft terug hoe veel tekens er in een string staan. De NULL-karakter wordt daarbij NIET meegeteld.

sizeof()

“sizeof()” doet hetzelfde als “strlen()”, maar deze neemt WEL de NULL karakter mee in de telling. Het geeft de lengte van de gehele array terug, ook al zou deze helemaal met NULL-karakters gevuld zijn.

strcat()

“strcat()” is een vreemde eend, het plakt twee array’s (strings) aan elkaar. Je leest het als “String Concatenate” wat Engels is voor “String Samenvoegen” of “String aaneenschakelen”.

Voorbeelden voor strlen, sizeof, strcat

Laten we deze functies eens uitproberen.

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    char Naam[45] = "Hans";
6    int  count = 0;
7
8    Serial.print("Naam = ");
9    Serial.println(Naam);
10
11   // determine length
12   count = strlen(Naam);
13   Serial.print("De lengte van Naam met strlen: ");
14   Serial.println(count);
15
16   // determine array size
17   count = sizeof(Naam);
18   Serial.print("De lengte van Naam met sizeof: ");
19   Serial.println(count);
20
21   strcat(Naam, " heeft 2 neefjes, genaamd Bram en Max!");
22   Serial.print("Naam = ");
23   Serial.println(Naam);
24
25   // determine length
26   count = strlen(Naam);
27   Serial.print("De lengte van Naam met strlen: ");
28   Serial.println(count);
29
30   // determine array size
31   count = sizeof(Naam);
32   Serial.print("De lengte van Naam met sizeof: ");
```

33	Serial.println(count);
34	}
35	
36	void loop() {
37	// leave empty for now
38	}

De output ziet er als volgt uit:

Naam = Hans
De lengte van Naam met strlen: 4
De lengte van Naam met sizeof: 45
Naam = Hans heeft 2 neefjes, genaamd Bram en Max!
De lengte van Naam met strlen: 42
De lengte van Naam met sizeof: 45

Even ter herhaling van wat we eerder zeiden:

Als we de lengte van een string niet opgeven dan zal de compiler dit eenmalig voor jou bepalen!

Elementen buiten het bereik van de array aanspreken kan foutmeldingen en onverwachte situaties leveren.

Laten we dit eens opzettelijk fout laten gaan door regel 5 als volgt te doen char Name[] = "Hans";

Dus we zeggen nu dat de compiler maar moet uitzoeken hoe groot de array moet worden, en de compiler baseert dat op de initiële waarde toewijzing. Dus de lengt van de array voor de string wordt “5”, de 4 letters en het NULL-karakter.

Als we dit nu starten en naar de Arduino sturen dan kunnen we rare dingen gaan zien, zoals bijvoorbeeld:

Name = Hans
Length of Name with strlen: 4
Length of Name with sizeof: 5
phews, called Bram and Max!Hans has two nephews, called Bram����

Zie je de rare tekens na “Bram”?

Dat komt omdat de array maar 5 tekens lang is en we duidelijk voorbij “5” gaan als we het stukje ” heeft 2 neefjes, genaamd Bram en Max!” en “Naam” aan elkaar gaan plakken. De Arduino gaat lekker onbenullig voorbij de 5, en dus kijken naar geheugen dat helemaal niet meer bij deze array hoort, en dus “rommel” oplevert. Het kan zelfs zijn dat de output eindeloos door lijkt te gaan omdat de Arduino geen NULL-karakter kan vinden.

Niet iets wat we willen he? En dat is precies de reden waarom we op moeten letten als we de lengte van een array definiëren. Meer hierover in het hoofdstuk waar we arrays gaan bespreken.

7.4. String Object (String met hoofdletter “S”)

We hebben net gekeken naar de “string” als een Array of Char. Met een kleine letter “s”.
De Arduino IDE biedt echter nog een andere string vorm aan: De “String” (hoofdletter “S”), wat een object is.

De onhandige dingen die we deden met de oude “string” (kleine letter “s” dus een Array van Char), zijn met het “String” (hoofdletter “S”, dus het object) object veel eenvoudiger. Maar wat is nou een?

Objecten

We hebben nog niks geleerd over objecten, ook al hebben we een van die objecten al heel veel gebruikt, het Serial object.

De korte uitleg: een “object” is een “ding” of “item” met eigenschappen (properties) en methoden (methods). Methoden moet je zien als “functies”.

Een Object is een “ding” met eigenschappen (properties) en Methoden (methods).

Zo een “ding” kan van alles zijn, b.v. een “auto“.

Om het eenvoudig te houden, zouden we kunnen zeggen dat de auto een aantal eigenschappen (Engels: properties!) heeft zoals: kleur, werkende moter, wat benzine, aantal deuren, een achterbak, en wielen. Dit noemen we dus “properties” (eigenschappen). Properties doen opzichzelf niets, maar geven eigenlijk alleen maar feiten door met betrekking tot het object “auto”.

Een property kan een waarde hebben of de waarde kan ontbreken. Zo kan de auto een “bestuurder” hebben, of geen “bestuurder” hebben. In dat laatste geval is “bestuurder” gelijk aan NULL. Als “bestuurder!=NULL” dan zit er dus een bestuurder in de auto.

Nu kan een auto ook bepaalde dingen doen: Rijden, parkeren, toeteren, open of sluit achterback, open of sluit deur, etc.
Dit zijn functies van het object “auto” en in “object georiënteerd programmeren” noemt men deze functies “methods” (methodes).
Methods kunnen dus iets “doen”; dus acties die de auto kan uitvoeren.

Er zijn verschillende redenen waarom objecten handig zijn.

Als eerst staat alles lekker bij elkaar – de eigenschappen (properties) en de functies (methods) hangen vast aan het object Er zal dus zelden onduidelijkheid zijn naar welk object we wijzen als we een property wijzigen of een methode aanroepen.

Een andere reden, welke misschien belangrijker is, is dat een object zich als een soort datatype kan gedragen, waarbij elke variabele van dat type zich op precies dezelfde manier gedraagt en precies dezelfde soort properties heeft. We kunnen dus een variabele “MijnAuto” van het object type “auto” definiëren. Kleine moeite en ineens zijn alle methoden en properties beschikbaar voor “MijnAuto”. Maar als we nu een garage vol met auto’s hebben dan kunnen we dit zelfs een array maken van dit object type of meerder variabelen definieren: MijnAuto, JouwAuto, MammasAuto, PappasAuto, of als array: Autos[0], Autos[1], Autos[2],….

We zullen niet te diep op objecten in gaan, maar het is belangrijk dat je de basis snapt omdat we toch wel wat objecten gebruiken, ook al hebben we dat niet in de gaten.
De “String” (met hoofdletter “S”) en “Serial” zijn dat soort objecten.

We hebben met “Serial” al een aantal methods (functies of methoden) zoals “begin”, “print” en “println”. We gebruiken deze methoden om ons object (Serial) iets te laten doen. Zie je nu dat we een dergelijk functie altijd aanroepen in dit formaat: OBJECT.METHODE of OBJECT.PROPERTY ….? Een punt tussen object en property of methode.

Het “String” Object

We hebben al gezien dat werken met de string met een kleine “s” (array van char) soms best lastig kan zijn. En de mensen van Arduino zagen dat ook en hebben daarom het “String” (met hoofdletter “S”) in het leven geroepen. Deze “String” maakt het stukken eenvoudiger om met tekst te werken.

We zullen snel zien hoeveel handiger een “String” is in vergelijk met een “string”.

Laten we eerst eens kijken hoe we een variabele definiëren voor een “String”: String Naam = "Hans";

Dit begint al goed, makkelijker dan de string met een kleine “s”, en geen gezeur over de lengte van de tekst – de vierkant haakjes worden hier dus niet gebruikt.

Net als met “string” kunnen we “String” ook gewoon gebruiken als we “Serial.print()” aanroepen. Dus een klein voorbeeld:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    String Naam = "Hans";
6
7    Serial.print("Naam = ");
8    Serial.println(Naam);
9  }
10
11 void loop() {
12   // leave empty for now
13 }
```

Dit werkt dus hetzelfde als bij de oude “string” en de Serieel Monitor laat dat zien: Naam = Hans

Nog steeds eigenlijk weinig nieuws en ook niks spannend of ingewikkeld.

Laten we nu een de tekst “Hans” naar “Bram” veranderen door de “String” een nieuwe “String” te geven:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    String Naam = "Hans";
6
7    Serial.print("Naam = ");
8    Serial.println(Naam);
9
10   Naam = "Bram";
11
12   Serial.print("Naam = ");
13   Serial.println(Naam);
14 }
15
16 void loop() {
17   // leave empty for now
18 }
```

Regel 10 kan overigens ook geschreven worden als Naam = String("Bram"); , wat dus ook wertk. Het verschil is echter dat we in deze tweede methode een nieuwe “String” object maken met de tekst “Bram”, terwijl we in de eerste methode alleen maar de tekst “Bram” doorgeven. De tweede methode wordt vaak gebruikt om b.v. een “string” om te zetten naar een “String”.

Beiden methoden zijn goed, en we zien dat het wijzigen van een String eenvoudiger is.

Laten we nu eens door het voorbeeld gaan waarbij we de string langer gaan maken, zoals we eerder hebben gedaan. Merk hierbij op dat de lengte van de String onbelangrijk is voor onze code – het object regelt dat allemaal voor ons automatisch.

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    String Naam = "Hans";
6
7    Serial.print("Naam = ");
8    Serial.println(Naam);
9
```

```
10   Naam = "Hans heeft 2 neefjes: Bram en Max!";
11
12   Serial.print("Naam = ");
13   Serial.println(Naam);
14  }
15
16  void loop() {
17    // leave empty for now
18  }
```

Voorheen, met de “string” moesten we dus goed opletten of de tekst in de lengte van de array zou passen. Met het String object is dat echter niet het geval. We gaan dus ook geen maffe output zien als de String langer wordt dan initieel de lengte was..

Als dat nieuwe “String” object dan zo handig is,… waarom zouden we dan nog kijken naar die oude “string” (array of char)?
Daar zijn twee mogelijke redenen voor: Als eerste neemt het object meer geheugen in beslag – je kunt het als een aanvulling zien. Het object bevats namelijk ook de “string” intern. Dat laatste is dus een tweede reden waarom we wat meer willen weten van de “string” met de kleine “s”.

String object Methoden (functies)

Nu moet je weten dat het “String” object veel meer methoden (functies) heeft, welke het een en ander eenvoudiger maakt.

Hieronder een voorbeeld van enkele methoden, kijk ook eens bij de Arduino String Object Reference. Dit is helaas in het Engels, maar het laat je nog meer methoden zien.

Length(), Concat() en Nummers converteren

In het volgende voorbeeld gaan we Strings (hoofdletter “S”) aan elkaar plakken, lengte van de String bepalen en nummers aan onze string toevoegen, zonder enige moeite.

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    String Naam = "Hans";
6
7    Serial.print("Naam = ");
8    Serial.println(Naam);
9
10   // determine length
11   Serial.print("De lengte van Naam met strlen: ");
12   Serial.println(Naam.length());
13
14   Name.concat(" heeft 2 neefjes: Bram en Max!");
15   Serial.print("Naam = ");
16   Serial.println(Naam);
17
18   // determine length
19   Serial.print("De lengte van Naam met strlen: ");
20   Serial.println(Naam.length());
21
22   // Alternatieve manier om tekst toe te voegen
23   Naam = "Hans"; // even terug naar de oorspronkelijke tekst
24   Serial.print("Naam= ");
25   Serial.println(Naam);
26
27   Naam += " heeft 2 neefjes";
28   Naam = Naam + ", ze heten Bram" + " en Max";
29   Serial.print("Naam = ");
30   Serial.println(Naam);
31
32   // het volgende werkt niet met arrays
33   Naam = "Een groot nummer is: ";
34   Naam = Naam + 32768;
35   Serial.print("Naam = ");
36   Serial.println(Naam);
37 }
38
39 void loop() {
40   // leave empty for now
41 }
```

De output:

Naam = Hans
De lengte van Naam met strlen: 4
Naam = Hans heeft 2 neefjes: Bram en Max!
De lengte van Naam met strlen: 42
Naam = Hans
Naam = Hans heeft 2 neefjes, ze heten Bram en Max
Naam = Een groot nummer is: 32768

Laten we eens stap-voor-stap door dit programma gaan:

Uiteraard definiëren (declareren) de variabele “Naam” als een String object “Naam” en geven het de initiële waarde “Hans” (regels 7 en 8), wat we vervolgens kunnen weergeven via “Serial” zoals we dat al eerder gedaan hebben.

In regel 12, bepalen we de lengte van de String tekst, en dat is in dit geval zonder het NULL-karakter mee te tellen. Hiervoor gebruiken we de “length()” methode van het “String” object: Naam.length() . Deze methode geeft een nummer terug als antwoord, welke we weergeven via “Serial.print”.

In regel 14, gebruiken we de methode “concat()” (Engels voor “Samenvoegen”) om de tekst ” heeft 2 neefjes: Bram en Max!” aan de String “Hans” te plakken. Zoals je ziet werkt dat prima. Maar … er is zelfs een eenvoudigere methode om “String”’s aan elkaar te plakken. Gewoon door het gebruik van het plus (+) teken. We kunnen dit zelfs doen met de compound operators die we eerder hebben gezien. Zie regels 27 en 28, waar we “+=” en het gewone “+” gebruiken.

Het “String” object laat nog meer van z’n goede kant zien als we een nummer willen omzetten en in een String willen zetten. Met een array is dat een heel gedoe, maar het String object is het net zo eenvoudig als tekst toevoegen- zie regel 34 – dit werkt met een string met een kleine letter “s” dus helemaal niet!

Nu we weten dat String("some text") een “String” object terug geeft, en dat we eenvoudig tekst aan elkaar kunnen plakken met het plus (+) teken, kan ik je ook laten zien hoe we de “Serial.println()” stukken eenvoudig kunnen gebruiken en vele zaken die we voorheen in meerdere regels moesten doen, ineens in een enkele regel kan. Als voorbeeld:

1	Serial.println(String("Het nummer zeven = ")+String(7));
2	
3	int test = 10;
4	Serial.println(String("Hoeveel vingers heb ik? ")+String(test));
5	
6	Serial.println(String("Tien = ")+
7	String(test));

De eerste regel kunnen we lezen als
– Maak een String object met de tekst “Het nummer zeven = “,
– Maak een String object met de tekst voor het nummer 7,

– Voeg beiden Strings samen (+),
– Geef het resultaat daarvan als parameter voor de “Serial.println()” methode (functie).

Als eindresultaat zullen we dus zien: “Het nummer zeven = 7”.

In regel 3 definieren we een integer (int) variabele met de naam “test”.
In regel 4 zien we net zoiets als in regel 1, waarbij we in plaats van het nummer (7) een variabele (test) gebruiken, wat ook prima werkt.

In de regels 6 en 7 zien we iets nieuws, en dat heeft niets te maken met strings of objects.
Herinner je jezelf nog dat we een statement afsluiten met een punt-komma?
In regel 6 sluiten we de regel echter niet af, dus de compiler gaat er vanuit dat het statement op de volgende regel verder gaat.
Het voordeel van het opbreken van een statement over meerdere regels kan zijn dat het geheel beter leesbaar wordt. Het is overigens niet verplicht natuurlijk, maar in bepaalde omstandigheden maakt het werken met de code een stuk prettiger.

Probeer nu dit voorbeeld eens en zet het voorbeeld in de “setup()” function. Speel er mee en kijk eens of je er zelf een beetje uit komt.

7.5. Strings Vergelijken

Het is allemaal wel leuk een aardig dat we strings kunnen maken, bewerken en weergeven, maar kunnen we strings ook vergelijken? Natuurlijk kunnen we dat!

string – array of char

Met de array of character “string”, zien we iets raar, want als we twee gelijke strings gaan vergelijken, dan merken we dat dit gewoon niet werkt …
In onderstaand voorbeeld zou je verwachten dat we “Hallo Hans” zouden zien omdat “Hans” == “Hans”, niet waar?

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    char Naam[] = "Hans";
6
7    Serial.print("Naam = ");
8    Serial.println(Naam);
9
10   if (Naam=="Hans") {
11     Serial.println("Hallo Hans");
12   }
13   else
14   {
15     Serial.println("Jij bent niet Hans");
```

```
16    }
17  }
18
19  void loop() {
20    // leave empty for now
21  }
```

De reden waarom dit fout gaat heeft te maken met iets wat we al een paar keer vermeld hebben. De variabele wijst namelijk naar een geheugenlocatie – het is een zogenaamde “pointer” (Engels voor: aanwijzer). Uiteraard zijn beiden niet gelijk daardoor, dus zullen we een speciale functie moeten gaan gebruiken en wel: “strcmp()” (lees dat als “string compare”, wat Engels is voor “string vergelijk”)

```
1   void setup() {
2     // set the speed for the serial monitor:
3     Serial.begin(9600);
4
5     char Naam[] = "Hans";
6
7     Serial.print("Naam = ");
8     Serial.println(Naam);
9
10    if (strcmp(Naam,"Hans")==0) {
11      Serial.println("Hallo Hans");
12    }
13    else
14    {
15      Serial.println("Jij bent niet Hans");
16    }
17  }
18
19  void loop() {
20    // leave empty for now
21  }
```

De functie “strcmp()” geeft als antwoord nul (0) als beide strings identiek zijn wat tekst betreft.

Deze functie geeft een antwoord kleiner dan nul als de eerste string kleiner is dan de tweede string die we als parameter meegeven, of een nummer groter dan nul als de eerste string groter is dan de tweede string.

Maar er zit een haakje aan deze functie …

Wanneer deze functie gaat vergelijken, dan kijkt het naar de ASCII-waarden van de individuele karakters. Als het een karakter tegen komt in de eerste string, welke een hogere waarde heeft, in vergelijk met het karakter in dezelfde positie van de andere string, dan besluit de functie dat de eerste string groter is. En dat geeft nu een beetje onverwachte antwoorden … als mens zou je zeggen dat “Hans” groter is dan “Hi”, maar bij deze functie is dat dus niet het geval, want deze functie zal zeggen dat “Hoi” groter is dan “Hans”. Dat komt natuurlijk ook omdat we mensen een langer woord als “groter” zien.

Als we hier stap voor stap doorheen gaan, dan zien we dat het eerste karakter van “Hans” en “Hi”, gelijk zijn.
Echter voor het tweede karakter, “a” (ASCII: 97) en “i”(ASCII: 105), ziet de functie dat “i” groter is dan “a” en is daarmee klaar.
“strcmp(“Hans”, “Hi”)” zal dus een negatief nummer geven … want “Hans” is kleiner dan “Hi”.

strcmp() kun je het beste gebruiken om te bepalen of twee strings identiek zijn (en dus nul als antwoord geeft als dit het geval is)

Laten we een voorbeeld bekijken wat dit onverwachte gedrag demonstreer:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    // compare
6    Serial.print("strcmp(\"Hans\",\"Hans\") = ");
7    Serial.println(strcmp("Hans","Hans"));
8
9    Serial.print("strcmp(\"Hans\",\"Hansie\") = ");
10   Serial.println(strcmp("Hans","Hansie"));
11
12   Serial.print("strcmp(\"Hans\",\"Hi\") = ");
13   Serial.println(strcmp("Hans","Hi"));
14
15   Serial.print("strcmp(\"Hans\",\"Haa\") = ");
16   Serial.println(strcmp("Hans","Haa"));
17 }
18
19 void loop() {
20   // leave empty for now
21 }
```

De output:

strcmp("Hans","Hans") = 0
strcmp("Hans","Hansie") = -1
strcmp("Hans","Hi") = -1
strcmp("Hans","Haa") = 1

We zien dus dat “Hans” kleiner is dan “Hansie”, zoals wij mensen zouden verwachten.
Onverwacht zien we echter dat “Hans” kleiner is dan “Hi”, ook al zouden wij mensen denken dat een langere string per definitie groter is.

Mocht je dus een langere string zoeken, bepaal dan de lengte van de strings en vergelijk deze.

In mijn optiek maakt dit dus dat de “strcmp()” functie het beste gebruikt kan worden om te bepalen of twee strings identiek zijn.

Je ziet overigens hier een mooie toepassin voor de eerder besproken “Escape character” in onze strings,…

String – object

Het vergelijken van een “String” objecten kan natuurlijk ook, maar ook deze vertoont dezelfde eigenaardigheid als de eerder genoemde “strcmp()” functie. Het verschil is echter de eenvoudigere notatie omdat we gewoon een comparison operator (vergelijking operator) kunnen gebruiken.

Ook hier dus weer, goed om te kijken of twee Strings identiek zijn wat tekst betreft …

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    String Naam = "Hans";
6
7    Serial.print("Naam = ");
8    Serial.println(Naam);
9
10   // compare
11   Serial.print("Gelijk aan: \"Hans\",\"Hans\" = ");
12   Serial.println(Naam == String("Hans"));
13
14   Serial.print("Groter dan: \"Hans\",\"Hansie\" = ");
15   Serial.println(Naam > String("Hansie"));
16
17   Serial.print("Kleiner dan: \"Hans\",\"Hi\" = ");
18   Serial.println(Naam < String("Hi"));
19
20   Serial.print("Niet gelijk: \"Hans\",\"Haa\" = ");
21   Serial.println(Naam != String("Haa"));
```

```
}

void loop() {
  // leave empty for now
```

Het voorbeeld van de string met de kleine “s” waarin we keken of ik Hans was, werkt overigens wel gewoon voor het “String” object (in tegenstelling tot de array of char string):

```
void setup() {
  // set the speed for the serial monitor:
  Serial.begin(9600);

  String Naam = "Hans";

  Serial.print("Naam = ");
  Serial.println(Naam);

  if (Naam == "Hans") {
    Serial.println("Hallo Hans");
  }
  else
  {
    Serial.println("Jij bent niet Hans");
  }
}

void loop() {
  // leave empty for now
}
```

Als je vragen hebt: stel ze dan hieronder, en bedenk dat er geen domme vragen zijn, behalve dan natuurlijk de vraag die niet gesteld is. We zijn allemaal een keer bij nul begonnen!

Volgende hoofdstuk: Arduino Programmeren voor Beginners – Deel 8: Arrays