8.0. Arduino Programmeren voor Beginners – Deel 8: Arrays

We zijn nu bij het achtste les van Arduino Programmeren voor Beginners, en in dit deel gaan we een beetje dieper in op Arrays. Array hebben we al gezien bij het werken met strings, maar nu wat meer aandacht aan wat we nog meer met arrays kunnen doen.

Inhoudsopgave

.

8.1. Wat zijn Arrays?

We hebben al kennis gemaakt met een bijzonder array: de string (met een kleine “s”) wat een array van karakters is of in het C/Engels: een Array of Char.

We weten nu ook al dat een “array” (Array Data Type, Array Data Structuur) gezien kan worden als een verzameling van elementen van hetzelfde data type, welke we ieder met een index nummer kunnen benaderen. Een array, in z’n eenvoudigste vorm, kan gezien worden als een lijst, en dat is wat we zagen bij het string voorbeeld: een lijst van Char (karakter).

Uiteraard is het gebruik van een array niet beperkt tot tekst. Een array (lijst) kan gebruikt worden met elk data type die we kennen in de C-programmeertaal zoals we deze gebruiken voor het Arduino Programmeren. We kunnen bijvoorbeeld een lijst met nummers maken, of boolean (b.v. lampen aan of uit), en zelfs arrays van objecten is mogelijk, zoals van bijvoorbeeld het “String”-object.

Een Array kan gezien worden als een lijst van elementen van hetzelfde data type,
waarbij we elementen individueel kunnen benaderen met een index nummer.

We hebben het voorbeeld van de string al gezien: char Naam[5] = "Hans";

Wat resulteerde in een array (lijst) van 5 elementen, waarbij elk element een “char” is:

string Array	
positie (index)	Geheugenlocatie inhoud
0	H
1	a
2	n
3	s
4	NULL

De variabele “Naam” wijst naar de geheugenlocatie van het eerste element, dus de geheugenlocatie die de letter “H” van de string bevat. De index van deze eerste locatie is “0” (nul). Immers: arrays beginnen met element tellen vanaf nul.

Laten we eens een andere array maken, bijvoorbeeld met booleans …

Stel we hebben 5 lampen, en deze kunnen AAN of UIT zijn, wat dus prima is voor het gebruik van een boolean waarbij we AAN als “true” zien, en UIT als “false” zien. We kunnen hiervoor een variabele

maken, b.v. “LampenAan” en maken deze een array van booleans (Engels: Array of Boolean):

bool LampenAan[5];

Zoals met gewone variabelen kunnen we de variabele meteen waarde(n) toewijzen, echter voor arrays gaat dat net een beetje anders dan voor gewone variabelen. In tegenstelling tot gewone variabelen, waar we slechts een enkele waarde toewijzen, willen we bij een array meteen meerdere waarden toewijzen. De programmeertaal C heeft daarvoor een aparte notatie die gebruik maakt van accolades. Doet je vast een beetje denken aan de code blokken die we eerder gezien hebben. Misschien is het nog niet zo gek om het als code blok te zien: we wijzen namelijk een heel blok in 1x toe.

Stel we definiëren onze variabele “LampenAan”, voor 5 lampen, en zetten alle waarden initieel op UIT (=false):

bool LampenAan[5] = { false, false, false, false, false };

We geven dus een lijst met waarden door, ingesloten door accolades, en de waarden gescheiden door een komma.

Net zoals we dat met een string konden, kunnen we voor onze nieuwe variabele ook elk element met een index nummer benaderen (lezen en schrijven) zoals we laten zien in dit voorbeeld:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    bool LampenAan[5] = { false, false, false, false, false };
6
7    // schakel lampen
8    LampenAan[2] = true;   // Zet de 3de lamp AAN
9    LampenAan[4] = false;  // Zet de 5de lamp UIT
10
11   // kijk of de 2de lamp aan staat
12   if(LampenAan[1]==true) {
13     Serial.println("Tweede lamp staat AAN!");
14   }
15
16   // Kijk of de eerste lamp aan staat
17   if(LampenAan[0]) {
18     Serial.println("Eerste lamp staat AAN!");
19   }
```

20	
21	// Als de 4de lamp niet aanstaat, zet dan ook de 3de lamp aan
22	if(!LampenAan[3]) {
23	Serial.println("4de lamp was niet aan, 3de lamp werd dus aangezet!");
24	LampenAan[2] = true;
25	}
26	}
27	
28	void loop() {
29	// leave empty for now
30	}

Dit voorbeeld is natuurlijk helemaal niet spannend, en het doet ook eigenlijk niks leuks, maar het geeft een voorbeeld hoe we met een array kunnen werken.

Merk wel op, en we hebben dit al eerder gezien, dat we met een array van booleans werken. Dat wil dus zeggen dat we daar handig gebruik van kunnen maken in het “if” statement, welke daardoor korter wordt.

De langere methode, voor het “if” statement (zie regels 12 en 17), is als we LampenAan[1]==true hadden gezet in de conditie van het “if” statement. Maar omdat de waarde al een boolean is, hebben we al een boolean waarde voor onze conditie. Maar omdat LampenAan[1] vergelijken met een boolean weer een boolean oplevert (en eigenlijk niets bijdraagt) kunnen we daar net zo goed alleen maar de boolean waarde wegzetten als conditie.

Je ziet dat we in regel 22 ook de logische operator “not” hebben gebruikt – en de “not” operator draait een boolean gewoon om, dus true wordt false en false wordt true. In het voorbeeld willen we zien op “LampenAan[3]” NIET aan staat.

Omdat we de variabele naam en het gebruik van de waarden (true=AAN en false=UIT) een beetje logisch hebben uitgekozen, kun je dit ook gewoon zo lezen.
Dus “als NIET LampenAan[3] doe dan …“.

Als de lamp dus aanstaat, dan hebben we een “true” (waar), en de “not” operator maakt er “false” (onwaar) van. Dus de “if” conditie faalt.
Maar als de lamp uit staat dan geeft LampenAan[3] een “false”, en door de “not” operator wordt dit “true”, dus de “if” conditie slaagt en de code wordt uitgevoerd.

8.2. Waarom beginnen Arrays met tellen bij nul?

Waarom beginnen Arrays met tellen bij nul?

Ehm, je zei de 4de lamp, maar in de code type je een 3, is dat niet een foutje?

Kun je nog herinneren dat we bij een string (array van karakters) zeiden dat we beginnen te tellen met nul?
Nou, dat geldt dus voor alle array … bij een array beginnen we altijd met nul voor de index nummers.

Arrays zijn ZERO INDEXED (nul geïndexeerd), wat wil zeggen dat we
altijd beginnen te tellen met nul en dus het eerste index nummer is ook nul!

Laten we even kijken hoe dat in z'n werkt gaat.

Dus de variabele wijst naar de geheugenlocatie van het eerste element van de array. Die geheugenlocatie bevat dus de waarde van het eerste element. Dit noemen we een zogenaamde "pointer" en deze wijst dus naar waar de "lijst" (array) begint.

Een pointer (Engels voor: Aanwijzer) wijst dus naar een geheugenlocatie. Als we de waarde van het eerste element willen lezen, dan kijken we naar de geheugenlocatie waar de pointer naar toe wijst plus nul.

Voor het volgende element, kijken we naar het geheugen adres + 1. De daaropvolgende staat in geheugen adres +2, etc.

Dus het index nummer wordt bij het geheugen adres opgeteld om het geheugen adres te krijgen van het gewenste element in de array.

Dat was de versimpelde uitleg, want in werkelijkheid is het een beetje complexer. Misschien kun je nog herinneren dat niet ieder data type even veel geheugen in beslag neemt. Zo gebruiken een byte en een char maar 1 byte, terwijl een int(eger) 2 bytes gebruikt.

Dat wil dus zeggen dat simpel weg "1" gebruiken voor iedere volgende geheugenlocatie niet altijd even lekker zal werken. Het is handig om te weten dat een geheugenlocatie altijd een enkele byte bevat. Dus als we een datatype gebruiken die meer dan 1 byte nodig heeft, dan zullen we dit probleem anders moeten oplossen.

Laten we als voorbeeld een array maken van int (integer, of te wel gehele nummers): int Nummers[5];

We hebben dus een array van 5 int elementen, en elke int heeft 2 bytes aan geheugen nodig.

Het eerste element is een eitje, want die wijst al meteen naar de juiste locatie.
Maar als we bedenken dat de geheugenlocatie met de volgende reken truc wordt bepaalt, dan wordt het e.e.a. misschien duidelijker:

geheugenlocatie + (index nummer * 2 bytes)

Het index nummer voor het eerste element is nul, dus de berekening wordt:

geheugenlocatie + (0 * 2 bytes)

Vermenigvuldigen met nul levert altijd nul, dus de uitkomst wordt:

geheugenlocatie + 0

Dus we krijgen voor het eerste element netjes de correcte geheugenlocatie.

Als we naar het tweede element (index = 1) van onze int array gaan kijken, dan wordt de berekening:

geheugenlocatie + (index nummer * 2 bytes)

geheugenlocatie + (1 * 2 bytes)

geheugenlocatie + 2

Dus de geheugenlocatie voor het twee element, met index = 1) levert met deze truc dus ook weer de juiste locatie op.

Je hoeft dit echt niet te onthouden hoor, maar het geeft je een idee hoe dit mechanisme werkt en waarom we dus met nul beginnen als we array elementen gaan tellen. Het laat je dus ook zien waarom een array (of variabele) correct gedefinieerd moet worden, waarbij het datatype toch wel belangrijk is zodat de Arduino weet in wat voor stappen het door het geheugen moet gaan om de juiste waarde te vinden.

Vergeet dus niet: Tellen begint bij nul!

8.4. Multi-Dimensionale Arrays

We hebben het al gezegd, de array in z'n eenvoudigste vorm, is een simpele lijst. Het is "plat".
We kunnen in een array ook array's zetten – een beetje complex misschien, maar daardoor kunnen we zogenaamde multi-dimensionale arrays maken.

Wat zeg je nou?

Dimensies

OK, ik kan me voorstellen dat je nog niet met dimensies hebt gewerkt, althans niet bewust, dus ik zal een simpele uitleg proberen te geven.

Laten we voor de eenvoud een dimensie zien als een richting waarin je jezelf kunt bewegen.
Als eerste dimensie: je kunt links en rechts bewegen.
Dus we kunnen horizontaal bewegen en in de wiskunde noemen we dat de X-as. We zouden dat in een tabel bijvoorbeeld een kolom kunnen zien: een simpel lijstje. Lastig is wel dat in de wiskunde we horizontaal bewegen (regel) terwijl we in de programmeertaal C en in ons boodschappenlijstje verticaal bewegen (kolom).

De eerste dimensie is horizontaal (breedte),
zeg maar een KOLOM in een tabel (een lijstje),
of de X-as in de wiskunde.

Bedenk dat we ook omhoog en omlaag kunnen bewegen – en dit is weer een andere dimensie!
Dit is dus verticaal bewegen en in de wiskunde noemen we dat de Y-as, of in een tabel zouden dit kolommen kunnen zijn.
Niet vergeten: de wiskundige verticale beweging is dus net andersom in de taal "C".

De tweede dimensie is verticaal (hoogte),
zeg maar een to be a REGEL in een tabel (meerder lijstjes),
of de Y-as in de wiskunde.

We kunnen nog meer dimensies bedenken, maar in de array voorbeelden zal ik daar niet op in gaan.

Voor wie nieuwsgierig is: we hebben allemaal weleens van 3D gehoord, of het nou voor een 3D film was, of een 3D TV, het maakt niet uit. Allen bieden ze “diepte”.
We kunnen dus vooruit en achteruit bewegen en in de wiskunde noemen we dat de Z-as.

De derde dimensie is diepte,
In de wiskunde de Z-as.

We weten dus nu dat er 3 dimensies zijn, maar we kunnen zo eindeloos doorgaan. Een nadeel echter is dat we meer dan 3 dimensies vaak niet meer kunnen bevatten in ons hoofd. Maar voor wie het wil weten: de 4de dimensie is tijd – dus bewegen in tijd.

Voor onze arrays blijven we lekker bij 1 or 2 dimensies omdat het tekenen van een 3D tabel al lastig wordt, laat staan een 4D tabel.

Enkelvoudige Dimensionale Array

Een enkelvoudige (single) dimensie array kunnen we het eenvoudigste zien als en simpele lijst, of als een gewone KOLOM, zoiets als dit:

1D Array voorbeeld	
Index	Value
0	A
1	B
2	C
3	D
4	E

We bewegen dus horizontaal, links-rechts, of te wl over X-as.

In dit voorbeeld heeft de array 5 karakters (niet te verwarren met een string, want die zou met een NULL-karakter eindigen!!) met de letters “ABCDE”. We weten ook al dat we elementen eenvoudig met een index nummer kunnen benaderen, en om de “C”-waarde te benaderen doen we dus zoiets als: variable[2].

variabele[index]

Het toewijzen van initiele waarden hebben we al gezien: bool LampenAan[5] = { false, false, false, false, false };

Twee Dimensionale Array

Als we nu gaan kijken naar 2 dimensies, dan kunnen we ons dat voorstellen als REGELS en KOLOMMEN van een tabel, dus de X- en Y-as in de wiskunde.
Gelukkig is dat ook niet moeilijk te bevatten. Zie het als een tabel, b.v. een school rooster of een tabel in een programma zoals Excel.

2D Array voorbeeld					
Index 2 Index 1	0	1	2	3	4
0	A	F	K	P	U
1	B	G	L	Q	V
2	C	H	M	R	W
3	D	I	N	S	X
4	E	J	O	T	Y

Onze 2D array bevat dus 25 waarden en wel: “ABCDEFGHIJKLMNOPQRSTUVWXY”.

Omdat dit een 2D array is, heeft het ook 2 indexen die we gebruiken moeten om een element te benaderen:

variabele[index1, index2]

Als we dus de letter “L” willen benaderen, dan doen we dat dus als volgt: variabele[1][2].

Merk op: je kunt dit ook schrijven als: variabele[1,2]

Het initieel toewijzen van een 1D array hebben we al gezien: bool LampenAan[5] = { false, false, false, false, false };
Het toewijzen van waarden voor een 2D array is wat lastiger, het is immers niet meer een simpel lijstje!
Weet je nog dat we zeiden: arrays in een array? Nou dat geldt dus ook voor de waarden die we gaan toewijzen.

We zagen voorheen dat we een “set” met waarden door konden geven door de “set” te omringen met accolades en de waarden te scheiden met komma’s: { 1,2,3 }

Omdat we arrays in een array plaatsen moeten we dus voor elke “kolom” zo’n setje maken en meegeven als waarde. Maar niet vergeten dat we waarden (en dus ook de setjes) scheiden met komma’s en omringen ze met accolades.
We krijgen daarom zoiets als: datatype naam = { { setje1 }, { setje2 }, {setje3} };

Hierbij is zo’n setje dus zoiets als: { 1,2,3 }

In een stukje code, om voorgaand voorbeeld tabel te maken, zou dat er dus zo uit zien.
Niet vergeten: het zijn karakters, dus een enkel aanhalingsteken gebruiken!

```
1boolean variable[5,5] = { { 'A','B','C','D','E' },
2                          { 'F','G','H','I','J' },
3                          { 'K','L','M','N','O' },
4                          { 'P','Q','R','S','T' },
5                          { 'U','V','W','X','Y' } };
```

Zie je dat de setjes gescheiden zijn door komma's en het geheel nog eens extra omringd is met accolades? Ieder setje wordt als een "waarde" gezien.

We zouden nu verder kunnen gaan met een 3-dimensional earray, maar omdat het diepte toe voegt (Z-as), zou dit betekenen dat onze tabel een kubus zou worden.

Een toepassing voorbeeld van een 2D array

Nu vraag je jezelf misschien: wat moet ik nou met een array die meerdere dimensies heeft?
Stel je voor, we gebruiken een Arduino om de lichten aan te sturen in meerdere kamers.
Elke kamer heeft 4 lampen en we hebben 3 kamers.
Een tabel, een 2D array dus, laten we het weer LampenAan noemen, zou dat kunnen omvatten en er zo uit kunnen zien.

2-D Array Voorbeeld			
Kamer: Lamp:	0	1	2
0	true	false	true
1	true	false	false
2	true	false	true
3	true	false	false

Zie dit dus als een lijstje van lampen, voor elke kamer.

Dus als we de status van de lampen in de eerste kamer willen zien (index = 0!!!), dan kijken we naar de volgende waarden: LampenAan[0,0], LampenAan[0,1], LampenAan[0,2] en LampenAan[0,3]. Op deze manier kunnen we onze waarden gemakkelijk opslaan en eenvoudig benaderen, zeker als we een "for"-loop hierbij gaan gebruiken.

Stel we willen alle lampen in alle kamers door lopen om te kijken of ze aan staan of niet. Je zou natuurlijk 12 aparte variabelen kunnen maken (LampAan1, LampAan2, etc), of elke element van de array individueel intypen. Maar met een “for”-loo[in een “for”-loop gaat dat vlotter:

```
1  #define kamers 3
2  #define lampenPerKamer 4
3
4  void setup() {
5    Serial.begin(9600);
6    boolean LampenAan[kamers][lampenPerKamer] = { { true, true, true, true },
7                                                  { false, false, false, false },
8                                                  { true, false, true, false } };
9
10   for(int kamer=0; kamer<=kamers-1; kamer++) {     // Tel de KAMERS
11     for(int lamp=0; lamp<=lampenPerKamer-1; lamp++) {   // Tel de LAMPEN voor elke KAMER
12       Serial.print("Kamer ");
13       Serial.print(kamer);
14       Serial.print(" Lamp ");
15       Serial.print(lamp);
16
17       if(LampenAan[kamer][lamp]) {
18         Serial.println(" is AAN");
19       }
20       else
21       {
22         Serial.println(" is UIT");
23       }
24     } // einde lamp loop
25
26     Serial.println(); // lege regel tussen kamers
27   } // einde kamer loop
28 }
29
30 void loop() {
31   // leave empty for now
32 }
```

Dus even kort: we hebben 3 kamers, met elk 4 lampen, dus: 3 lijsten, met elk 4 items op de lijst!

Zie je hoe ik de “#define” stiekem heb gebruikt om een constante te definiëren?
Ik heb dat expres gedaan omdat we deze twee waarden op meerdere plaatsen gebruiken in ons programma:

Bij de array definitie

In de “for”-loops

Zie je ook dat ik in de “for”-loop de betreffende constante gebruik maar vermindert met 1?
Dat heb ik gedaan omdat wij mensen 3 kamers tellen: Kamer 1, 2 en 3.
Maar … je raad het al: bij array’s starten we met tellen bij nul.
Dus: Kamer 0, 1, en 2.
Vandaar dus de “-1” om op dezelfde telling uit te komen zoals we die voor arrays moeten gebruiken als we elementen gaan aanspreken.

Zie je ook dat ik zinvolle namen heb gekozen voor de variabelen?
Het programma wordt veel duidelijker door zinvolle namen te geven aan variabelen en het gebruik van dit soort constanten te gebruiken in plaats van zinloze namen of gewoon nummers?

Wist je overigens als we een constructie, zoals de “for”-loop, in eenzelfde constructie (dus ook een “for”-loop) plaatsen, dat men dit “nesting” noemt? Leuk om te weten misschien maar niet superbelangrijk te onthouden hoor.

Zie je ook dat als we 2 van dit soort “for”-loops in elkaar zetten, dat de code soms wat lastiger wordt om te lezen? Dat is nou precies de reden waarom ik bij de “accolade sluiten” (}) even een opmerking plaats zodat ik weet waar de accolade bij hoort.

De uitleg voor deze code is misschien wat voor de hand liggend, maar voor de zekerheid even:
We starten een “for”-loop om de kamers te gaan tellen (kamer) en voor iedere “kamer” gebruiken we een “for”-loop om de lampen te tellen en doorlopen.
Net na de “for”-loop voor de lampen, maar nog steeds in de “for”-loop van de kamers zie je dat ik ook een lege “Serial.println()” heb geplaatst – dit resulteert in een lege regel tussen de kamer lijsten.

Vergeet dus niet dat de “for”-loop voor de lampen voor elke kamer herhaald wordt!

De output nog even voor de volledigheid:

```
Kamer 0 Lamp 0 is AAN
Kamer 0 Lamp 1 is UIT
Kamer 0 Lamp 2 is AAN
Kamer 0 Lamp 3 is UIT

Kamer 1 Lamp 0 is UIT
Kamer 1 Lamp 1 is UIT
Kamer 1 Lamp 2 is UIT
Kamer 1 Lamp 3 is AAN

Kamer 2 Lamp 0 is AAN
Kamer 2 Lamp 1 is UIT
Kamer 2 Lamp 2 is AAN
Kamer 2 Lamp 3 is UIT
```

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 9: Tekst Invoer