Arduino Programming

Theorie Sessie 3

ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

- Contact-dender: nabespreking labo2
- Libraries Introductie
- Arduino memory
- Arrays & char arrays
- Strings in C++
- “Serial read” & input buffer

Contact-dender

Labo ervaring

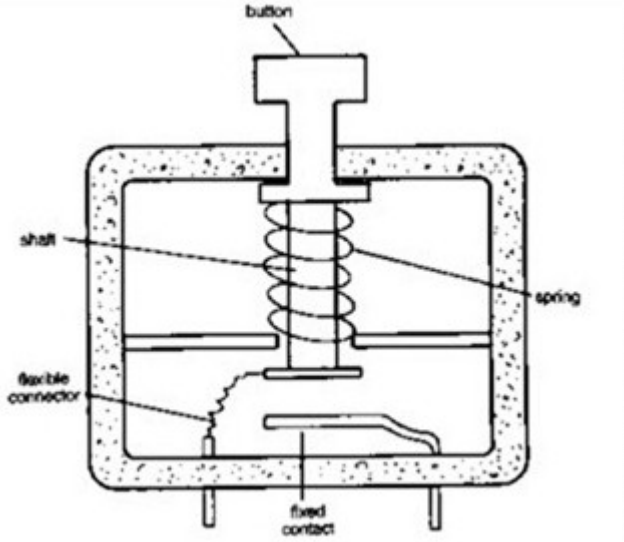
In het labo heb je kunnen kennismaken
met het “contactdender” probleem

Wat kon je vaststellen ?

Contact-dender

Probleem beschrijving

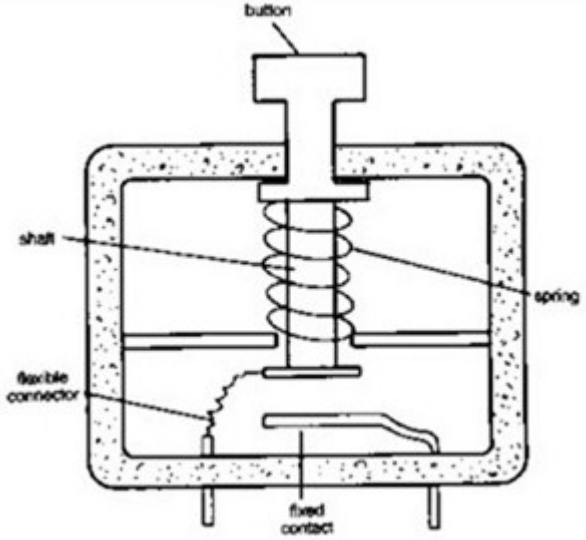
Een drukknop intern is een metalen plaatje
met hieronder een veertje

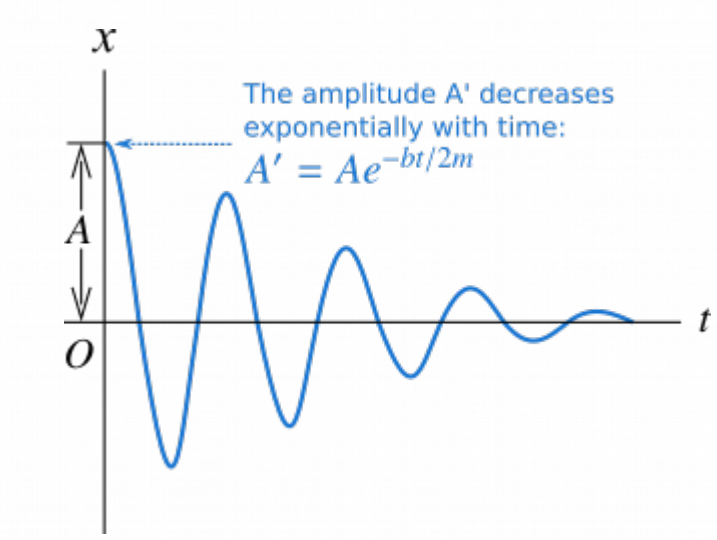


Contact-dender

Probleem beschrijving

Bij het loslaten van de drukknop
zal het veertje even oscilleren voor het stilstaat

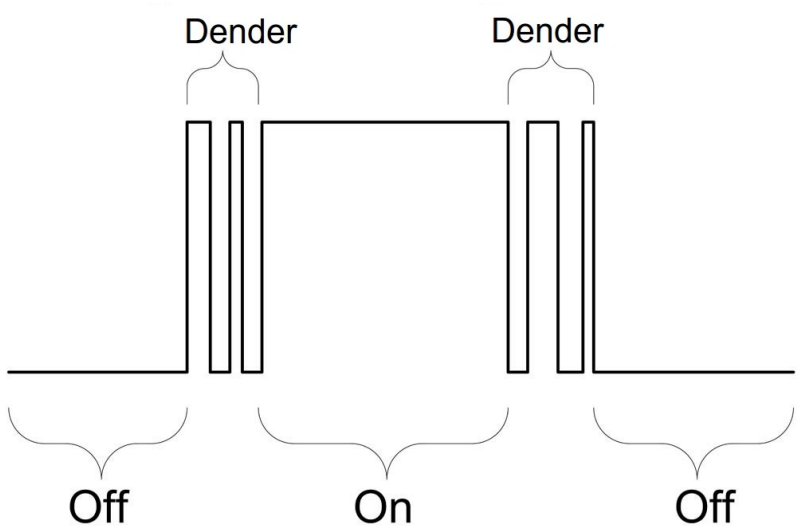




Contact-dender

Probleem beschrijving

Dat wil zeggen dat we niet een mooie 0 – 1 overgang krijgen maar contactdender



Contact-dender

de-bouncing

Met ‘de-bouncing’ duiden we
de oplossingen voor contact-dender aan.

(dender in het engels = bounce)

We kunnen deze opsplitsen in
software en hardware oplossing.

Contact-dender

Hardware debouncing

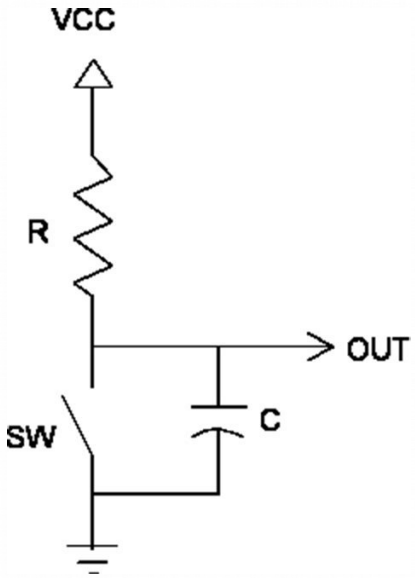
Hardware debouncing
kan op verschillende manieren

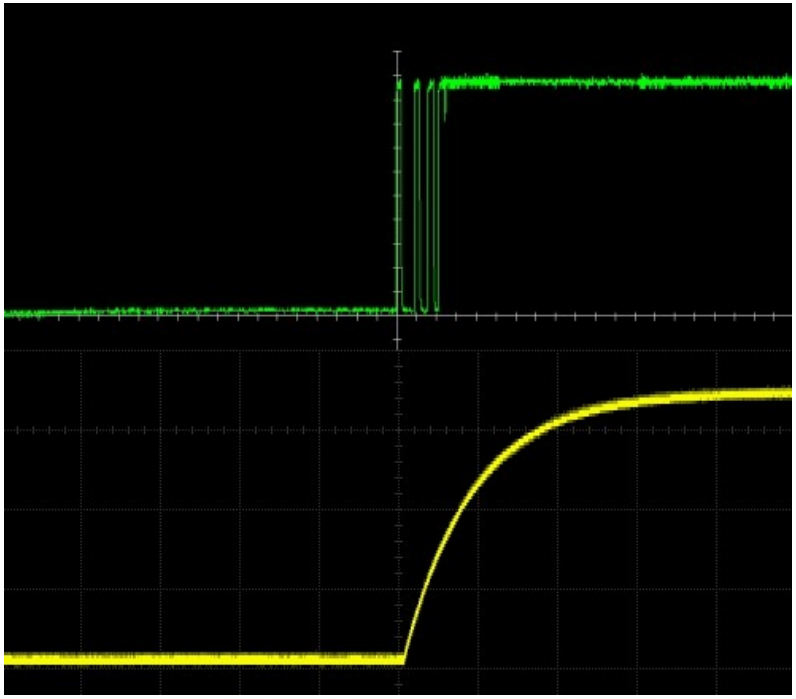
Een eerste manier is
door middel van een RC kring

Deze RC kring werkt
als een laag doorlaat filter:

Contact-dender

Hardware debouncing

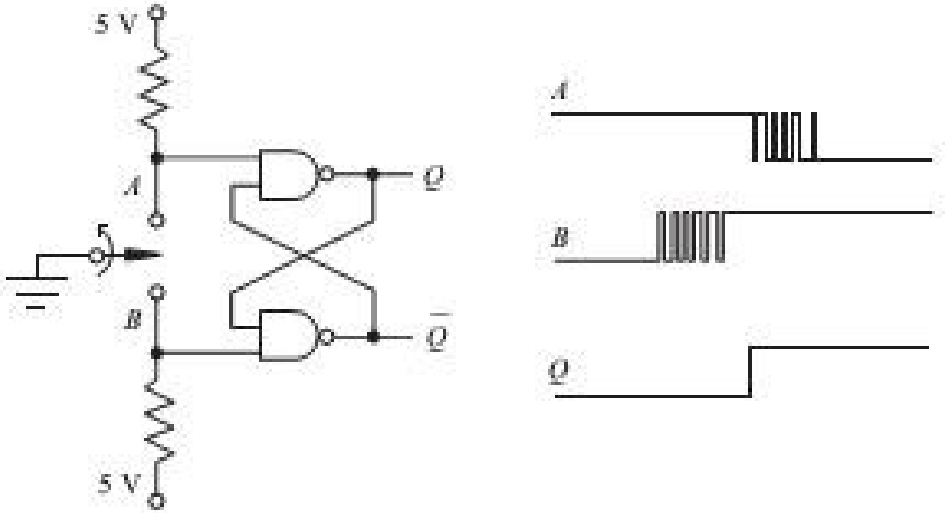




Contact-dender

Hardware debouncing

Een tweede manier is
m.b.v. een flip-flop schakeling



Contact-dender

Hardware debouncing

Voordeel van deze schakeling is
dat deze geen tijdsvertraging heeft

Vooral bij meermalig snel drukken op de knop
geeft dit een vlotte gebruikerservaring

Nadeel is dat de schakeling
een wisselschakelaar nodig heeft

... en werkt enkel indien de dender niet zo groot is
dat ze ook tussen de contacten gaat trillen

Contact-dender

Software debouncing

Software debouncing kan eveneens
op verschillende manieren

De eenvoudigste is het
direct aanroepen van een delay() bij
het vaststellen van 1/0 overgang

if (digitalread(...)) { delay(300); … }

Contact-dender

Software debouncing

Wil men voorkomen
dat de uitvoering vertraging oploopt
kan men gebruik maken van een vlag en millis()

```
if (digitalread(pin)) { ts = millis(); vlag=true; }

if (vlag) ….

if (( millis() - ts ) > 300 ) { vlag=false; }
```

- Libraries Introductie

Libraries Introductie

Omschrijving

De C en C++ programeertalen bevatten
slechts een beperkt aantal instructies,
dito zo bij Arduino code.

Zo bevat Arduino code niet default alle mogelijke commando's om de hele reeks hardware die men op zo'n Arduino kan aansluiten aan te sturen.

Er bestaan immers honderduizenden componenten en wekelijks komen er daar nog tientallen bij

Libraries Introductie

Omschrijving

Er is dus een mechanisme nodig om naar believen functionaliteit aan de taal te kunnen toevoegen.

Dit kan door gebruik te maken van code bibliotheken

Indien extra functionaliteit nodig is, bvb het aansturen van een bepaald hardware onderdeel, dan kan die door simpelweg zo'n bibliotheek toe te voegen aan een Arduino sketch.

Libraries Introductie

Praktisch

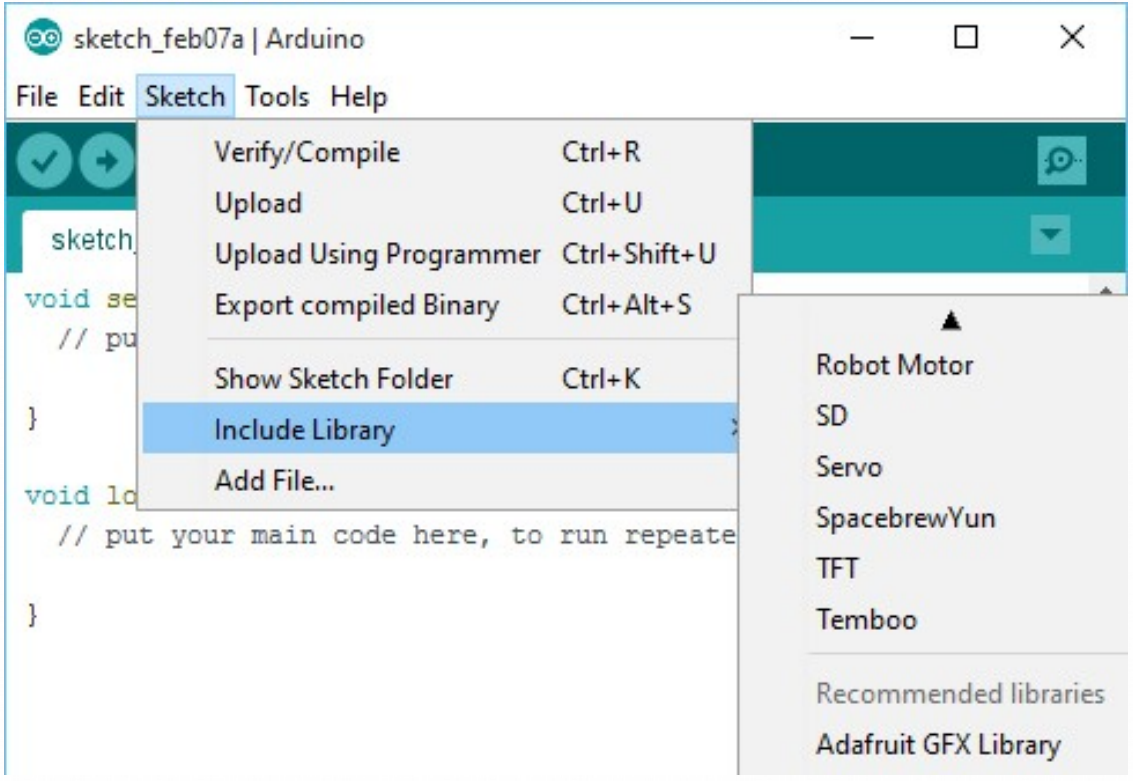
Zo’n bibliotheek toevoegen gaat
door gebruik van het commando “include”

Omdat bepaalde hardware onderdelen
vaak gebruikt worden is de bibliotheek ervan
reeds opgenomen in de Arduino IDE.

We kunnen deze bibliotheken toevoegen door ze
simpelweg te selectioneren in de IDE:

Libraries Introductie

Praktisch



Libraries Introductie

Praktisch

Zoals men kan zien voegt dit een lijntje code toe aan de sketch die start met “ #include … ”

Uiteraard kan men ook direct deze lijn code zelf intikken om ze toe te voegen aan de sketch.

Voorwaarde is wél dat de bibliotheek (engels “library”) aanwezig is in de juiste map op de harddisk van onze PC

Libraries Introductie

File locatie

“De juiste map” kan echter *twee* dingen betekenen

De libraries die je kan selecteren via
de eerder vermelde methode ofwel
de “ingebouwde” libraries bevinden zich
in de “libraries” map die zich ergens tussen
de installatie bestanden van de IDE schuil houdt

Om deze libraries toe te voegen gebruiken we:

#include <letopdehaakjeshier>

Libraries Introductie

File locatie

Een tweede mogelijkheid is echter
om de libraries nodig voor een sketch
samen met die sketch te bewaren.

Een library die zich in zo'n sketch folder
bevindt voegen we dan toe als volgt:

#include "aanhalingstekenshier"

Pas dus op bij het schijven van het include
statement dat je de juiste schrijfwijze gebruikt !

Libraries Introductie

Toevoegen libraries

Uiteraard bevat ook de IDE niet reeds
alle libraries voor alle hardware onderdelen.

Alleen enkele vaak gebruikte,
dus is er ook een manier om nog andere
of nieuwe libraries toe te voegen aan de IDE.

Dat gaat door het toevoegen van “zip” files.

Libraries Introductie

Toevoegen libraries

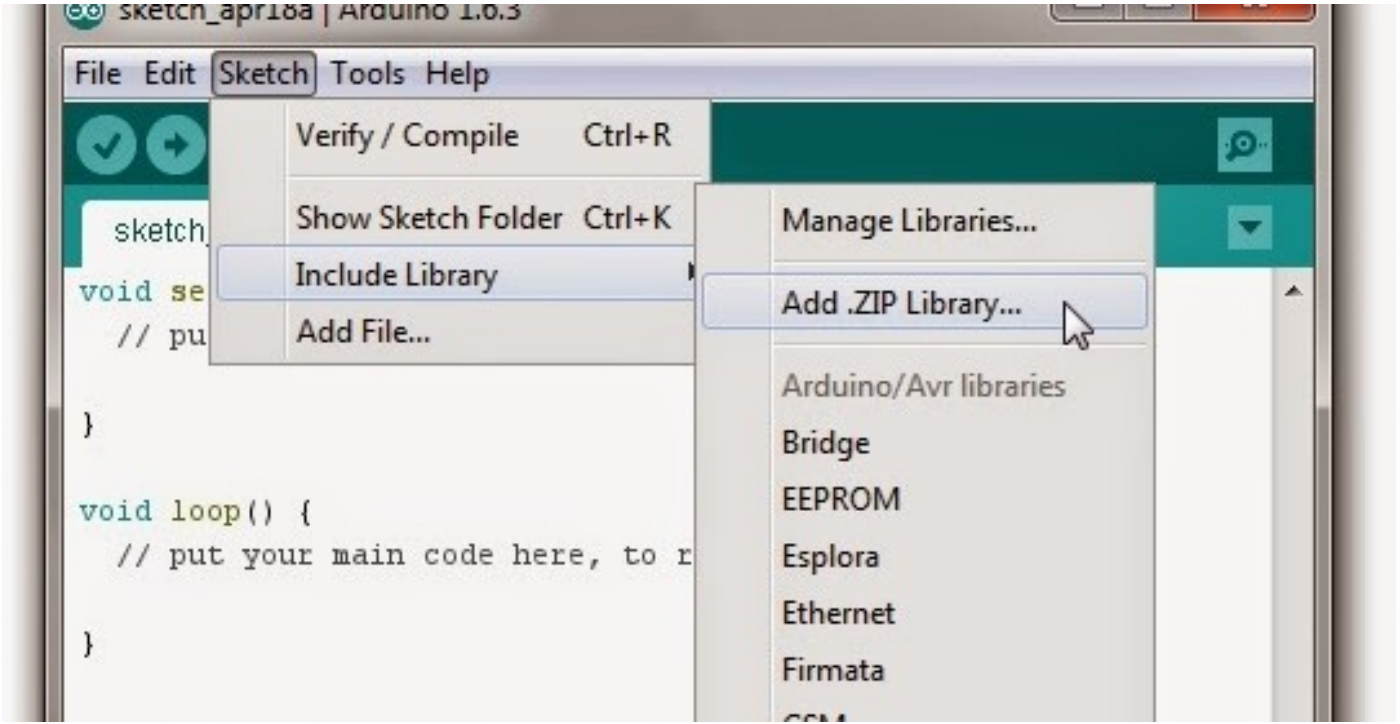
Eerst ga je op zoek naar zo'n zip library
van het hardware onderdeel of de extra
functionaliteit die je wenst toe te voegen.

Een hele reeks van deze libraries zijn te vinden op

https://www.arduino.cc/en/Reference/Libraries

Eens de zip file gedownload gaat het toevoegen zo:

Libraries Introductie



Libraries Introductie

Toevoegen libraries

Hierna krijg je een pop-up venster waarin
je de zip file kan selecteren.

Eens toegevoegd komt deze library
in de lijst te staan tussen alle andere
en kan je ze op dezelfde manier
toevoegen aan een sketch
als de “ingebakken” bibliotheken

Libraries Introductie

Toevoegen libraries

Het is ook mogelijk om zelf bibliotheken te schrijven.

Gebruikelijkerwijze zullen deze dan in de sketch folder komen en dus toe te voegen met #include “...”

Hoe zelf zo’n bibliotheek te maken komt in een van de volgende theorie sessies aan bod..

- Arduino Memory

Arduino Memory

ATMega328p

Het geheugen van de Arduino komt integraal van de microcontroller op de PCB.

Er is verder geen extern geheugen aanwezig

Dit geheugen is samengesteld uit 3 verschillende types geheugens:

Arduino Memory

Memory layout

- 32 KiloBytes Flash geheugen:
  - Non-volatile geheugen die +- 10.000 keer kan herschreven worden voor het faalt.
  - Het geheugen waar programma in terecht komt
- 2 KiloBytes SRAM geheugen:
  - volatile geheugen, geen schrijfbeperking
  - Hier komen o.a. variabelen terecht tijdens runtime.
- 1 KiloByte EEPROM geheugen:
  - Non-volatile geheugen “Electrically Erasable”
  - Kan +- 100.000 keer herschreven worden.

Arduino Memory

Memory layout

Dit EEPROM geheugen kan niet direct aangesproken worden in Arduino code.

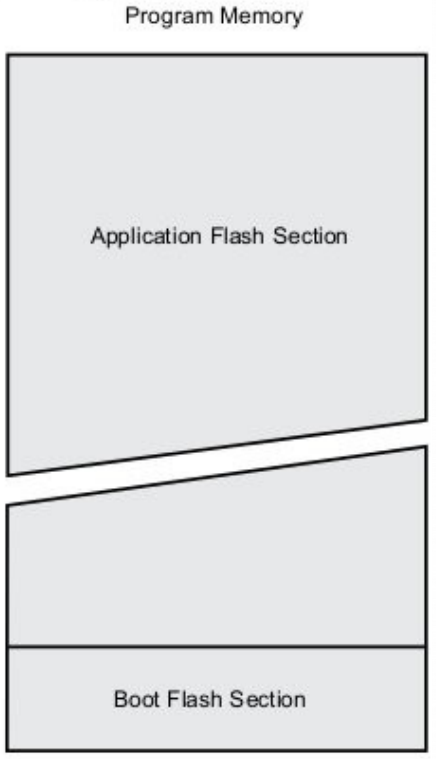
Er is uiteraard wel een library die we kunnen toevoegen aan onze sketch die dit mogelijk maakt...

Dit EEPROM geheugen is interessant indien we waardes van variabelen ook na het wegvallen van de voedingsspanning willen hergebruiken

Arduino Memory

Layout Flash

Program Memory Map ATmega328/P



0x0000 → begin adres

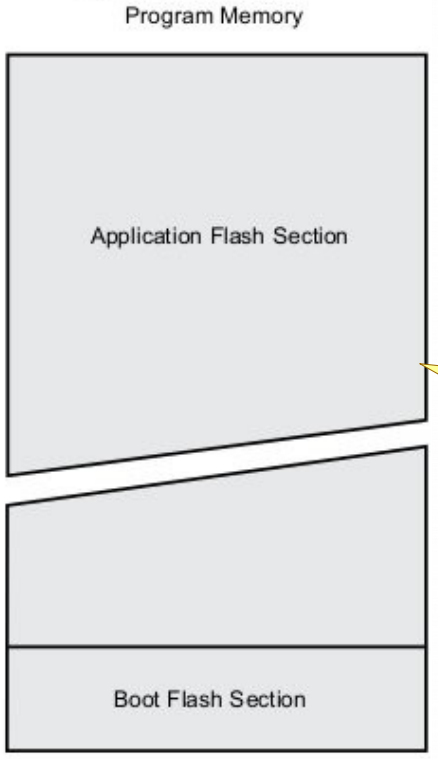
Een deel van het flash geheugen wordt reeds gebruikt door de bootloader

0x3FFF → eind adres

Arduino Memory

Layout Flash

Program Memory Map ATmega328/P



0x0000 → begin adres

We hebben dus
niet de volle 32 KiloByte ter beschikking
voor onze software...

0x3FFF → eind adres

- Arrays & char arrays

Arrays & char arrays

Arrays

Net zoals in andere programeertalen is het vaak nodig verzamelingen aan variabelen te hebben.

In Arduino sketch alsook in C/C++ kan dit door het toevoegen van een index bij het declareren

```
int rij[10];
```

Dit statement creëert een array van 10 variabelen van het type “int” aanspreekbaar via de index.

Arrays & char arrays

Arrays

De declaratie bepaalt het aantal elementen

```
int  nogeenarray[n];
```

… **doch de index voor het aanspreken is steeds van 0 tot n-1**

int rij[10] heeft dus de elementen rij[0] tot rij[9] ...

Arrays & char arrays

Arrays

In Arduino code, net als C/C++,
zijn arrays default nooit dynamisch qua formaat.

Een declaratie van het exacte aantal
elementen is m.a.w. verplicht.

Men kan wél indirect het aantal elementen geven bij
declaratie door onmiddelijke assignatie:

```
int rij[ ] = { 1, 2, 3 };
```

Arrays & char arrays

Arrays

Bij deze declaratie wijze

```
int rij[ ] = { 1, 2, 3 };
```

kan de compiler het exacte aantal elementen afleiden uit de assignatie lijst.

merk op : C# kent hier “ArrayList” voor arrays met dynamische grootte
“*ArrayList” bestaat dus niet in C/C++*

Arrays & char arrays

Char arrays

Arrays kunnen aangemaakt worden
in alle basis types. (int, bool, float, etc)

Indien een array uit char’s bestaat
spreekt men van een “char-array”

Deze kent extra eigenschappen t.o.v de andere types,
met name de declaratie kan op een extra manier:

```
char zin[ ] = “dit is een char array”;
```

Arrays & char arrays

Char arrays

Ook het toekennen van een waarde
tijdens runtime kan op een andere manier:

```
char letters[10];
…

letters =  “abcdefgxyz”;
```

Ook hier is het opnieuw niet mogelijk
om de grootte van de array onbepaald te laten.

Arrays & char arrays

Char arrays

Al ziet deze declaratie er uit
als een string in andere programmeertalen,

een char-array is niet hetzelfde
als een string in C/C++ !

- Strings in C++

Strings in C en C++

String vs char array

In de eerste versies had de C programmeertaal
geen strings, deze werden pas later toegevoegd.

Strings in C zijn niet hetzelfde als in C++ !

in C zijn dit speciale char-arrays, niets meer

Strings in C++ zijn objecten
Dit is eveneens zo in Arduino code.
We bespreken hier verder enkel de C++ versie

Strings in C++

String vs char array

In tegenstelling tot een char array
moet een String afgesloten worden
met het “\0” character op het einde

Tijdens assignatie met aanhalingstekens
zal de compiler dit zelf toevoegen

String zin = “dit is een string”;

Strings in C++

String vs char array

Merk op:

strings behoeven géén rechte haakjes
bij het declareren ervan.

Een String mét rechte haakjes
tijdens declaratie kán en mág wel…
… maar is dan een array van Strings

String zinnen[ ] = { “dit is zin een”, “dit is zin twee” };

Strings in C++

String vs char array

Merk op:

De characters van een String kunnen net als een char array aangesproken worden met een index

```
String zin = “dit is een string”;
char a = zin[3];
```

Deze index echter loopt één element verder dan de identieke char array (hier staat de “\0” !)

- “Serial read”

- “Serial read” input buffer

“Serial read”: input buffer

Omschrijving

Tot nu toe hebben we alle elementen besproken
om tot de kern te komen van deze sessie.

Met “Serial.write” hadden we reeds gezien hoe we
vanuit een Arduino sketch tekst op de terminal van
de Arduino IDE kunnen plaatsen.

Om iets te doen als “Serial.read” echter
is het noodzakelijk we eerst begrijpen
wat een “input buffer” is en hoe er mee te werken

“Serial read”: input buffer

Post vergelijking

Wanneer we zelf tekst naar de terminal sturen
vanuit onze arduino code kunnen we
bij wijze van voorbeeld dit beschouwen
als het zelf naar het postkantoor gaan
voor het posten van een brief.

We nemen dit op in onze takenlijst,
kunnen iets doen voordien en
we kunnen reeds iets plannen voor erna

“Serial read”: input buffer

Post vergelijking

Eens de brief in de bus zit
van het postkantoor is onze taak afgerond

Bij het ontvangen van een brief
zit het echter iets complexer.

Veronderstel hierbij voorlopig even
dat we géén brievenbus hebben

“Serial read”: input buffer

Post vergelijking

We zouden een taak kunnen uitvoeren
en als we klaar zijn even gaan kijken
als de postbode voorbijkomt

Vervolgens doen we nog een taak,
kijken opnieuw als de postbode voorbijkomt etc.

De kans is groot dat de postbode ofwel nog niet
geweest is ofwel reeds voorbij gereden is...

“Serial read”: input buffer

Post vergelijking

De kans dat hij voorbij komt nét op het moment dat we gaan kijken is eerder klein.

De enige oplossing is géén taken uitvoeren en blijven wachten aan de deur tot de brief er is.

Gebruiken we nu wél een brievenbus dan kan het eerste scenario wel werken.

“Serial read”: input buffer

Post vergelijking

We gaan regelmatig kijken naar de brievenbus

Vaak zal de brievenbus leeg zijn,
soms eens één brief,
soms eens een hele hoop.

Maar vooral kunnen we voor en na
elk bezoek aan de brievenbus
telkens een taak uitvoeren.

“Serial read”: input buffer

Post vergelijking

Feit blijft wel dat we regelmatig
naar de brievenbus moeten gaan kijken.

Doen we dit niet zal de brievenbus overvol geraken
en is de kans reëel dat we post gaan missen.

“Serial read”: input buffer

Serial lezen.

Willen we nu tekst lezen van de terminal is zijn de problemen equivalent aan ons voorbeeld.

Onze code bij uitvoering

- weet niet exact wanneer er tekst op komst is,
- weet niet hoeveel tekst dit zal zijn
- en de code zou onuitvoerbaar worden indien de microcontroller permanent moest blijven wachten op seriële data.

“Serial read”: input buffer

Serial input buffer

De oplossing hiervoor is voorzien
in de vorm van een input buffer.

Dit kan men best interpreteren als een char array:

waar indien een teken via de
seriële ingang ontvangen wordt dit teken
het éérste element van de char array wordt.

En alle reeds ontvangen tekens
ééntje opgeschoven worden in de array

“Serial read”: input buffer

Serial input buffer

Werking is volgens het “FiFo” principe

“ **First in = First out** ”

inkomende data wordt dus in de buffer geduwd

bij het lezen wordt éérst
de data genomen
die het langst in de buffer zit

“Serial read”: input buffer

Lezen van de buffer

Om de buffer te lezen maken we
gebruik van 2 commando’s:

```
Serial.available()
```

Hiermee gaan we kijken
of er iets in de buffer zit

Serial.available geeft aan hoeveel
ongelezen bytes er actueel in de buffer zitten

“Serial read”: input buffer

Lezen van de buffer

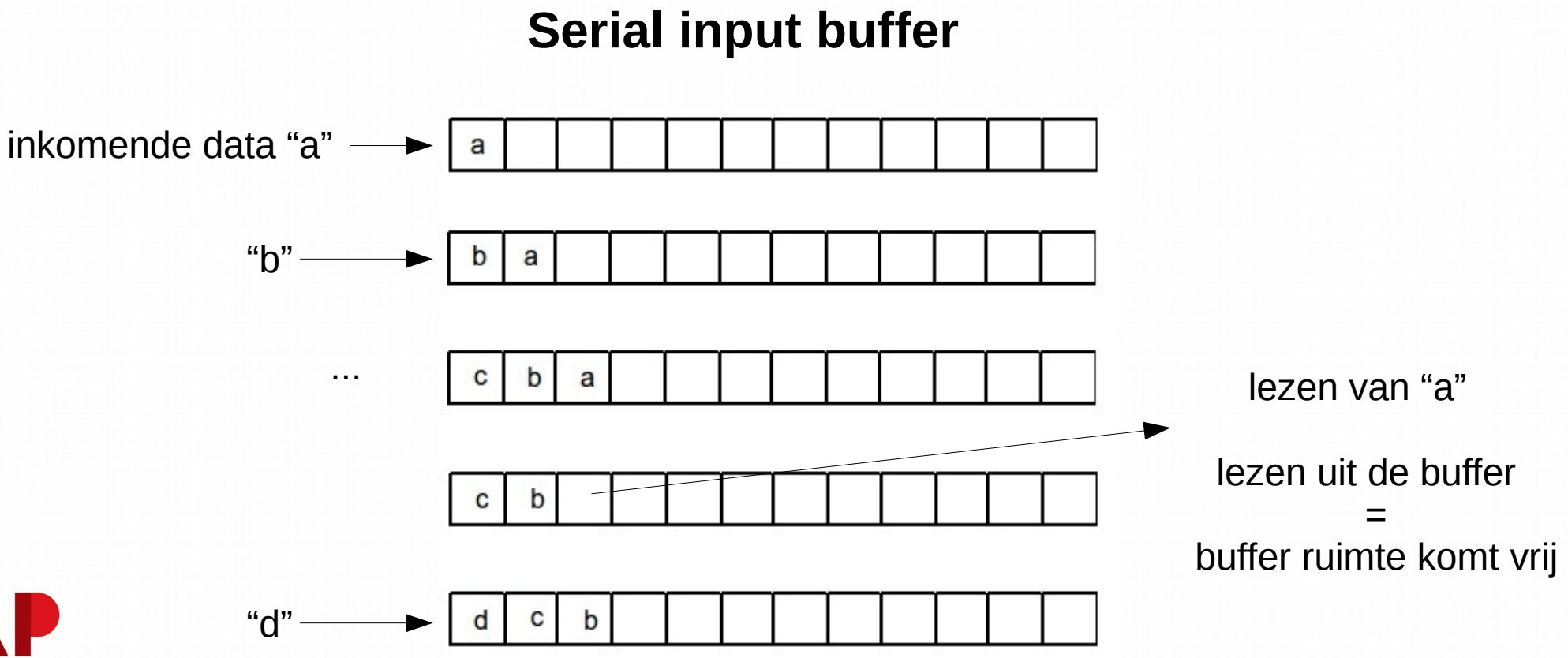
Eens we weten dat er iets in de buffer zit
kunnen we deze uitlezen met het commando

```
Serial.read()
```

Serial.read zal één byte uit de buffer lezen.
Eens gelezen wordt deze byte
direct uit de buffer gewist.

Volgende tekening en stukje code
maakt dit alles iets duidelijker:

“Serial read”: input buffer



“Serial read”: input buffer

Lezen van de buffer

```
int ontvangenByte = 0;

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

void loop() {

   if (Serial.available() > 0) {

         ontvangenByte = Serial.read();

         Serial.println(ontvangenByte);
    }
}
```

“Serial read”: input buffer

Lezen van de buffer

in dit code voorbeeld zal Serial.read()
dus énkel het laatste teken lezen,
daarna gaat de loop direct verder

Het voordeel van deze lus is dat tussen
het bekijken van de “brievenbus” of buffer in
andere taken kunnen uitgevoerd worden

“Serial read”: input buffer

Lezen van de volledige buffer

```
int ontvangenByte = 0;

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

void loop() {

   while (Serial.available() > 0) {
          ontvangenByte = Serial.read();
    }

}
```

“Serial read”: input buffer

Lezen van de volledige buffer

Door het gebruik van een “while”
zullen *alle* tekens nog in de buffer
een na een uitgelezen worden
voor de code uit de while gaat.

Gezien Serial.available het exacte aantal
tekens in de buffer weergeeft is kan men ook
gebruik maken van het volgende commando:

```
Serial.readBytes( opslagvar, aantal );
```

“Serial read”: input buffer

ReadBytes

```
Serial.readBytes( opslagvar, aantal );
```

“opslagvar” is hier een character array waarin de bytes uit de buffer in zullen gekopiëerd worden

“aantal” is het aantal bytes
die uit de buffer gelezen moeten worden

Merk op: alle gelezen bytes verdwijnen uit de buffer

“Serial read”: input buffer

Buffer size

Wat nu als we heel veel code hebben in onze loop
en té lang wachten om de buffer te lezen ?

De buffer ruimte is niet oneindig groot…

Bij de ATMega328p:

Seriële buffer = 64 bytes groot

“Serial read”: input buffer

Buffer size

Dat wil zeggen dat als data
niet tijdig uit de buffer gelezen wordt
deze data verloren gaat !

In ons postbode voorbeeld is het equivalent
de brief die terug wordt gezonden naar
de afzender als “onbestelbaar”

“Serial read”: Buffer inhoud

ASCII

Tekst via de seriële monitor worden
verzonden als “ASCII” tekens

Volgende lijn is m.a.w géén fout,
ook niet als we enkel letters intikken...

```
int ontvangenByte = Serial.read();
```

ASCII codes zijn immers
nummerieke waarden
tussen 0 en 127

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

“Serial read”: Buffer inhoud

ASCII

Men kan het resultaat van Serial.read()
dus zowel toekennen aan een char
als aan een int of byte.

<u>Men mag niet vergeten echter dat deze int waarde
niet overeenkomt met wat men intikt !!</u>

Als we “5” intikken in de terminal zal onze int
immers de waarde 35 krijgen en niet 5…

“Serial read”: Omzetting ASCII

Decimale cijfers

Een héél eenvoudige en snelle manier
om ASCII tekens “0” tot “9” om te zetten naar
decimale cijfers van 0 tot 9 is dan ook de volgende:

ASCII “0” → waarde 30
ASCII “1” → waarde 31
...
ASCII “9” → waarde 39

dus simpel weg de ASCII waarde verminderen
met 30 geeft ons het decimale cijfer terug.

“Serial read”: Omzetting ASCII

Integer parsing

Vaak is de bedoeling om via de terminal
echter een integer door te geven,
niet gewoon één enkel cijfer.

Gezien ingetikte tekst als ASCII verzonden wordt
is deze omzetting dan ook ietwat omslachtig...

Omdat dit zo vaak voorkomt hebben de Arduino
onwikkelaars hiervoor een commando voorzien:

```
Serial.parseInt()
```

“Serial read”: Omzetting ASCII

Integer parsing

Belangrijk hier is even stil te staan
bij het feit dat een int 2-bytes groot is !

Serial.parseInt() zal dus 2 bytes lezen uit de buffer.

Men kan m.a.w. best volgende code schrijven:

```
if (Serial.available() > 1) {

            mijnint = Serial.parceInt();
}
```

“Serial read”: Omzetting ASCII

String input

Ook voor het lezen van een string
bestaat een specifieke methode:

```
Serial.readStringUntil( eindchar )
```

Hierbij is “eindchar” het teken waarmee de
te lezen string afgesloten zal worden.

Indien het eindecharacter niet zou komen
zou de code hier blijven hangen.
Om dit te voorkomen is een timeout voorzien

“Serial read”: Omzetting ASCII

String input

Standaard staat deze timeout op 1 seconde.

Indien het m.a.w langer dan een seconde duurt
voor het eindcharacter komt gaat
de code uitvoering alsnog terug verder.

Deze timeout kan men aanpassen en
instellen op een zelf gekozen waarde met:

```
Serial.setTimeout( tijdsduur )
```