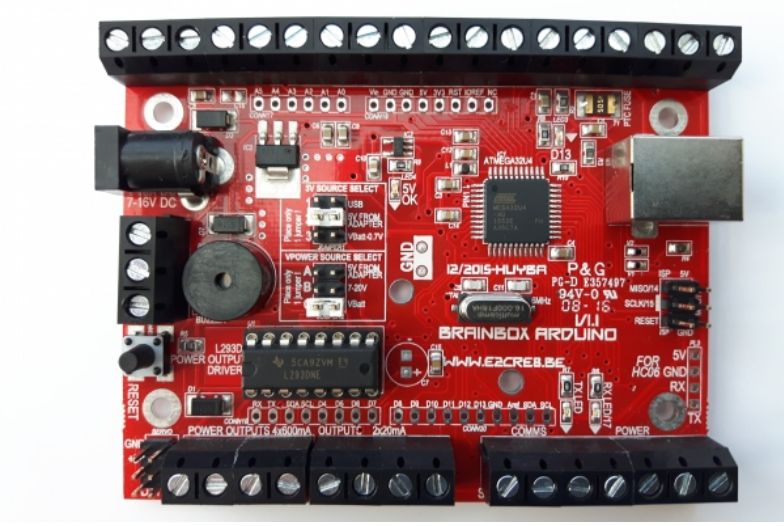
ARDUINO IDE SCHOOLHANDLEIDING

MET THEORIE, OEFENINGEN en EVALUATIETOOL

GEBASEERD OP BRAINBOX AVR HARDWARE



Voldoet aan leerplannen:

- Industriële ICT – derde graad VVKSO
- Industriële Wetenschappen – derde graad VVKSO
- Elektriciteit – Elektronica – derde graad (!enkel als basis)

B. Huyskens – A.Naets

Sept 2018

Arduino Programmeer manual bronnen

De informatie is onder andere verkregen door:
http://www.arduino.cc
http://www.arduino.nu
http://www.wiring.org.co
http://www.e2cre8.be
http://www.stemzone.be

voorwoord

Deze handleiding leert u de standaard programmeerstructuren aan a.d.h.v. het programmeren van een Arduino microcontroller.

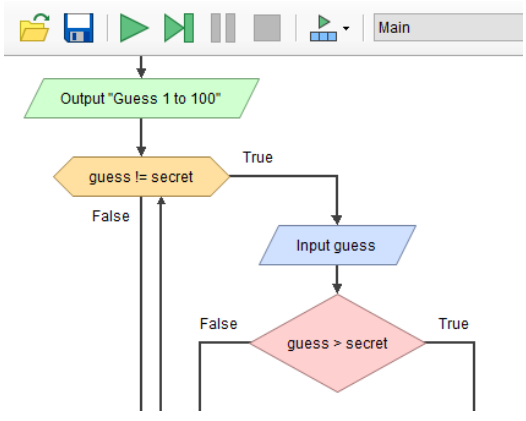
De basis van de Arduino IDE programmeertaal is C wat wereldwijd nog steeds de moeder aller talen is. Om het geheel zo eenvoudig mogelijk te maken hebben de ontwikkelaars van deze Arduino compiler echter soms gekozen voor programmeerstructuren die iets van de C-norm afwijken. In grote lijnen mag echter gesteld worden dat deze taal een goede basis kan zijn om het programmeren aan te leren.

Een Arduino processor is een microcontroller. Om alle functies van een bepaalde microcontrollers te kunnen programmeren in C is een doorgedreven studie van deze microcontroller en al zijn registers noodzakelijk. De ontwikkelaars van Arduino IDE hebben de programmeertaal echter voorzien van een uitgebreide set van voorgeschreven functies (ook wel bibliotheken genoemd) die het mogelijk maken om, zonder een diepgaande studie van de controller, toch de meeste functies van deze controller te kunnengebruiken.

Voor de volledigheid dient het gezegd te worden dat van professionele embedded programmers verwacht wordt dat ze in echte ANSI C programmeren en dat ze een controller wel kunnen programmeren vanuit de complexe datasheet.

Flowgorithm

http://www.flowgorithm.org/index.htm



Flowgorithm is een gratis tool waarmee je snel en duidelijke flowcharts kan maken.

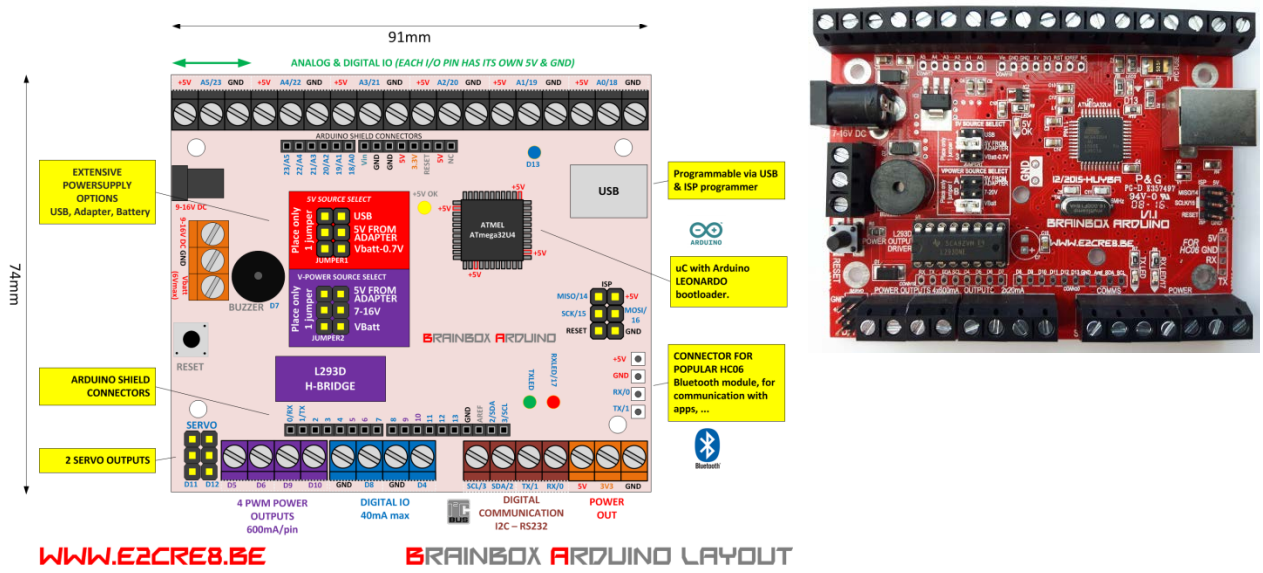
Het maken van een flowchart voordat je begint te coderen is een essentiële stap die vaak wordt overgeslagen bij Studenten.

Daarom vragen we dat er een flowchart gemaakt wordt voor elke opgave die gegeven wordt.

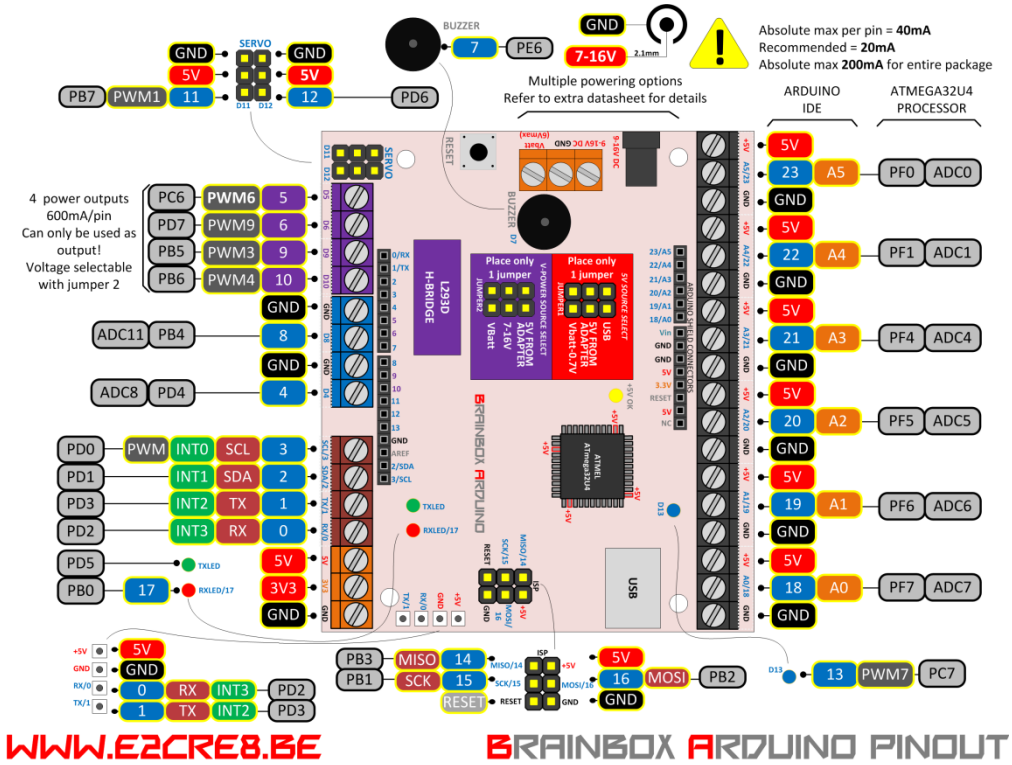
Inhoud

Gebruikte hardware Brainbox AVR

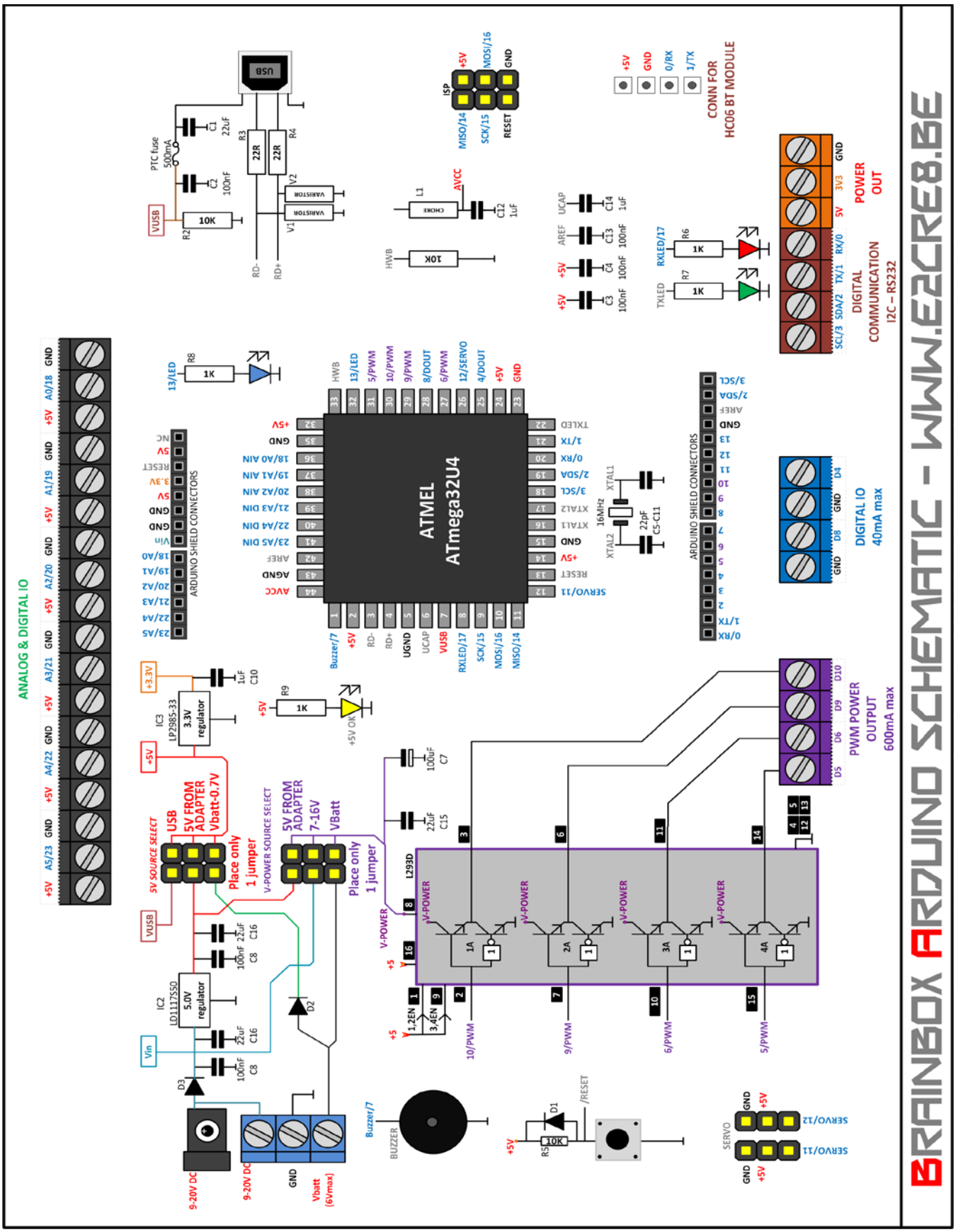
Als hardware gebruiken we de Brainbox AVR van www.e2cre8.be. Dit is een Arduino Leonardo compatibel ontwikkelsysteem dat in Vlaanderen ontworpen en geproduceerd wordt en dat speciaal voor onderwijstoepassingen is voorzien van o.a. stevige schroefconnectoren en poweruitgangen.



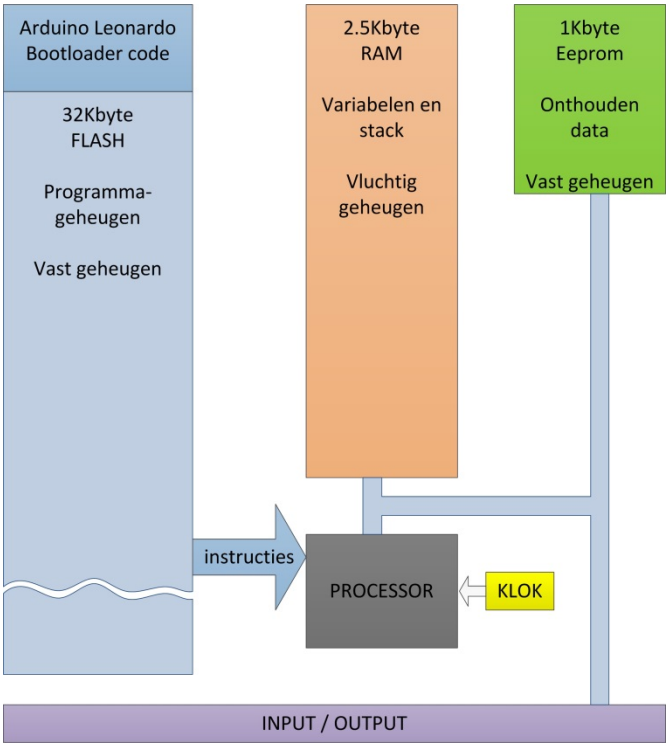
Pinout Brainbox AVR



Schema Brainbox AVR



Vereenvoudigd blokschema Leonardo processor



Dit sterk vereenvoudigd blokschema geeft de samenhang van de verschillende componenten in de ATMEGA32U4 processor weer.

Een microcontroller is heel gelijkaardig opgebouwd als een computer.

Het hart van de µC is de processor. De processor voert alle instructies uit tegen een snelheid die bepaald is door de klok. In dit geval zullen er 16 miljoen instructies per seconde worden uitgevoerd – uitgedrukt als 16MIPS.

De instructies worden regel per regel uit het programmageheugen gehaald. Dit programmageheugen kan 32000 regels code bevatten en is van het FLASH type wat het zelfde geheugen is zoals bv USB sticks of SD cards. Het programma wordt ook na het uitschakelen van de µC onthouden.

De bootloader code maakt van een lege ATMEGA32U4 controller een “Arduino Leonardo” controller. Deze bootloader code moet er met een AVR programmer worden ingeladen en zorgt er vanaf dan onder andere voor dat we het programma in het programmageheugen kunnen overschrijven via een rechtstreekse USB connectie. We hebben vanaf nu dus geen externe programmer meer nodig.

Data die tijdens de loop van een programma kan wijzigen – zoals bijvoorbeeld de waarde van een sensor – wordt altijd opgeslagen in het RAM geheugen. Het RAM geheugen in onze processor kan 2500 bytes vluchtige data (of variabelen) onthouden. RAM geheugen is snel, maar is vluchtig wat betekent dat alle data weg is als de processor wordt uitgezet.

Het RAM geheugen wordt ook gebruikt om te onthouden van welk adres er gesprongen wordt naar andere delen in het programma om later terug naar het eerste adres te kunnen springen. Deze adressen worden onderaan in het RAM geheugen opgeslagen en dat noemt de stack (of stapel)

Als we in de loop van het programma data hebben die we willen blijven onthouden, ook nadat de processor uitgeschakeld is, dan moeten we die in het Eeprom geheugen opslaan. Het eeprom geheugen kan 1000 bytes data bevatten, en is in vergelijking met het RAM geheugen een traag geheugen.

Het doel van microcontrollers is om in real-time (zonder vertraging) Input en Output (IO) pinnen hoog en laag te maken om actuatoren aan te sturen, om sensoren in te lezen of om te communiceren met allerlei rand-componenten.

Onze µC is een 8 bit µC wat betekent dat er steeds slechts met 8 bit getallen kan gerekend worden. Een 8 bit getal kan data bevatten tussen 0 en 255. Het rekenen met grotere getallen is zeker ook mogelijk, maar dan zal onze µC deze getallen opsplitsen in getallen van 8 bits en zo stap voor stap een uitkomst berekenen. U begrijpt dat dit dan wat meer tijd in beslag zal nemen.

Structuur Arduino programma

De basisstructuur van de Arduino programmeertaal is erg simpel. Het bestaat uit minstens twee blokken. Deze twee blokken of functies vormen een aantal statements (programma commando's). Voorbeeld:

```
void setup()
{
    statements;
}

void loop()
{
    statements;
}
```

Waarbij setup() de voorbereiding is en loop() de uitvoering. Beide functies zijn nodig om een programma te kunnen laten werken.

In de setup functie worden variabelen gedeclareerd en bepaald welke pinnen ingang of uitgang worden. De setup functie wordt slechts eenmaal doorlopen.

De loop functie volgt na de setup functie en wordt vaak oneindig herhaald. De loop functie leest vaak de inputs en laat afhankelijk daarvan bepalen wat de outputs moeten doen. Eigenlijk komt het er op neer dat de loop functie de motor van het programma is, dus daar waar al het werk moet gebeuren.

setup()

De setup() functie wordt één keer aangeroepen wanneer het programma start.
Het wordt gebruikt om de mode van de pinnen te initialiseren of begint met seriële communicatie. De syntax ziet er als volgt uit:

```
void setup()
{
    pinMode(pin, OUTPUT);   // maak de 'pin' als uitgang
}
```

loop()

Nadat de setup() functie aangeroepen is, volgt de loop() functie. Deze doet precies wat de naam zegt en loopt oneindig alle instructies binnen de loop af. Wanneer de laatste instructie werd uitgevoerd start de loop terug bij de eerste instructie. De syntax:

```
void loop()
{
    digitalWrite(pin, HIGH);      // zet 'pin' aan
    delay(1000);                  // 1 seconde pauze
    digitalWrite(pin, LOW);       // zet 'pin' uit
    delay(1000);                  // 1 seconde pauze
}
```

{ } krullende haakjes (accolade)

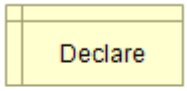
Krullende haakjes geven het begin of het einde aan van een functieblok zoals je ook tegenkomt bij de void loop(). Kijk maar:

```
void loop()
{
      statements;
}
```

De eerste opende accolade (gekrulde haakje) **{** moet altijd afgesloten worden door een (gekruld gesloten) accolade **}**. Het aantal accolades is dus altijd een even
getal. Let daar goed op, want één accolade te weinig en een heel programma kan stoppen met werken.
Vind dan de ontbrekende accolade, maar eens terug.

; puntkomma

Een puntkomma moet gebruikt worden na elke ingevoerde opdracht. Net zoals de gekrulde haakjes, zal bij het ontbreken van een puntkomma een error verschijnen. Voorbeeld:




```
int x = 13;     // declareer variable 'x' as the integer 13
```

Opmerking: Vaak is het zo dat het ontbreken van een puntkomma er voor zorgt dat de Arduino software niet wil compileren en een error aangeeft op een andere plek dan waar de puntkomma vergeten is. Dat wordt dus lastig zoeken.

/*… */ blok commentaar

Blok commentaar zijn gebieden met tekst die door het programma genegeerd worden. Tussen de /* en */ staat meestal uitleg over het programma of over de code die daar staat. Het mooie van “blok commentaar” is dat het over meerdere regels geplaatst kan worden.

```
/* dit is een blok met commentaar Vergeet niet het einde
van het Commentaar aan te geven
*/
```

Commentaar wordt nooit mee geprogrammeerd in de microcontroller. Het neemt dus geen geheugenruimte in beslag. Natuurlijk wordt het wel bewaard in het Arduino programma (Windows/Linux/Mac).

// regel commentaar

Een regel die begint met // en eindigt met tekst of code zal op die regel genegeerd worden. Ook dit neemt geen geheugen in de microcontroller in beslag. Code vooraan in de regel gevolgd door // zal wel uitgevoerd worden alles wat na de // komt op die regel echter niet.

```
// Dit is commentaar op een regel en onderstaande wordt
// niet uitgevoerd omdat het vooraf gaat met //.
// A = B + 3
```

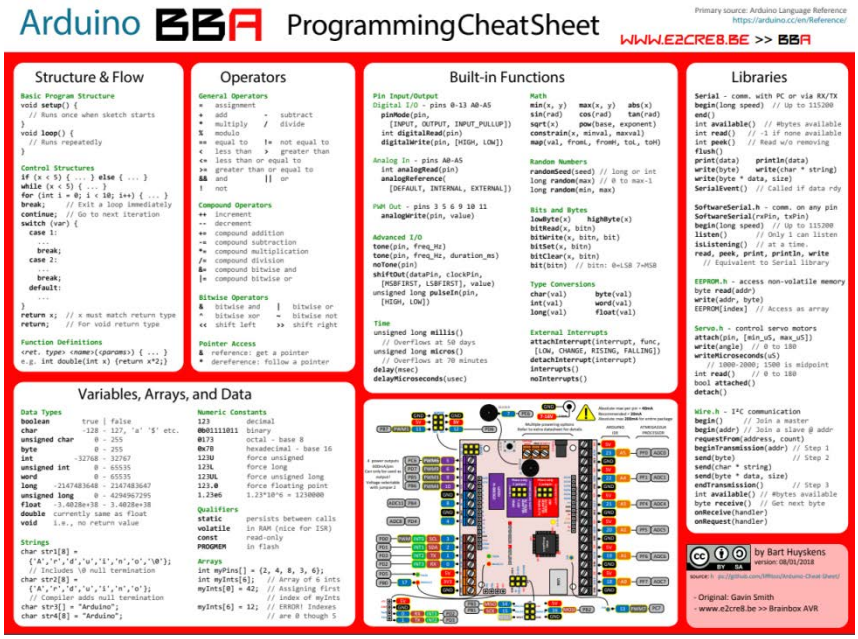
Regel commentaar wordt vaak gebruikt om de genoemde opdrachten uit te leggen.

Regel 1: wij voorzien ALTIJD en ELKE regel van zinvolle commentaar, zodat we zelf of anderen later het programma snel kunnen begrijpen.

Arduino Cheatsheet

Deze 'spiekbrief' verzamelt de meeste Arduino IDE instructies en datatypes op 1 blad, samen met de pinout van de Brainbox AVR. Deze tool mag steeds gebruikt worden, ook tijdens testen en examens.

Deze CheatSheet is te downloaden van www.e2cre8.be >> BBA



Variabelen

Een variabele is een manier om een getal-waarde te bewaren voor later gebruik in het programma. Zoals de naam variabele al aangeeft kan de waarde van een variabele ook regelmatig veranderen in de loop van het programma.

Er bestaan ook zogenaamde constanten. Constanten zijn geen variabelen.

Variabelen worden weggeschreven in het RAM geheugen van de processor. Vermits microcontrollers niet veel RAM geheugen hebben, moeten we zuinig omspringen met variabelen. Daarom is het van heel groot belang dat we de grootte van een variabele duidelijk meegeven aan het programma (dat noemen we 'declareren' van de variabele). Grote variabelen kunnen grotere getallen bevatten, maar nemen ook meer RAM geheugen in beslag en berekeningen met grote variabelen duren ook langer.

Een variabele moet op een juiste manier gedeclareerd worden. In de code hieronder wordt een variabele gedeclareerd genaamd Var1 die vervolgens de waarde krijgt die gemeten wordt op de analoge input pin 2:

```
int Var1 = 0;       // declareer een variabele met de naam Var1
                    // en geef die de waarde 0
                    // met int geeft je aan dat Var1 een 'Integer' //formaat
                    // heeft (16 bit) dat getallen tussen 32767 en -//32768 kan bevatten
Var1 = 1589;        // verander de waarde achter Var1 nu naar 1589
```

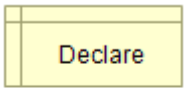
Een voorbeeld: De volgende code test of de Var1 kleiner is dan 100. Als dat zo is, dan krijgt Var1 de waarde 0. Is de waarde hoger dan 100, dan houdt Var1 de waarde die hij al had. In de derde regel zie je dat een pauze gemaakt wordt waarvan de duur bepaald wordt door de waarde die op dat moment in de variabele Var1 zit, gemeten in msec dan.

```
if (Var1 < 100)    // test of de variabele kleiner is dan 100
{
      Var1 = 0;   // zo ja, dan krijgt hij de waarde 0
}

delay(Var1);    // De waarde van de variabele bepaalt de pauze
```

REGEL 2 : Geef variabelen een logische naam bijvoorbeeld TiltSensor of PushButton. Bekijken anderen jouw programma, dan wordt het al een stuk gemakkelijker om het te lezen. Je kunt elk woord voor variabelen gebruiken tenminste als het geen naam is die al gebruikt wordt in de Arduino omgeving.

Variabelen declareren

Alle variabelen moeten eenmalig gedeclareerd worden voordat je ze kunt gebruiken. Er zijn verschillende types zoals int, long, float die elk een ander formaat hebben of andere data kunnen bevatten.



Variabelen bereik

Een variabele kan op verschillende plaatsen in het programma gedeclareerd worden, maar die plaats heeft wel gevolgen voor de bruikbaarheid van deze variabele.

Een **globale variabele** is een variabele die in een heel programma kunt oproepen. Deze variabele declareer je boven void setup(). Globale variabelen zullen gedurende de volledige looptijd van het programma op dezelfde plaats in het RAM geheugen worden onthouden.

Een **lokale variabele** is een variabele die alleen gebruikt kan worden in een klein stukje van een programma. Het is een tijdelijke variabele. Zo'n stukje kan bijvoorbeeld in de void loop() zitten. Lokale variabelen nemen enkel tijdens de uitvoering van de betreffende regels code een bepaalde plaats in het RAM geheugen in. Nadien kan deze plaats door andere lokale variabelen gebruikt worden. Dit is een efficiëntere manier om met RAM geheugen om te springen, maar je moet wel opletten dat deze variabele enkel bruikbaar is binnen de huidige functieblok.

Opmerking: Hoe meer variabelen je gebruikt in een microcontroller des te sneller zal het geheugen vol zitten. Dat kan natuurlijk niet de bedoeling zijn.

REGEL 3: Kies met kennis van zake of u een variabele als lokale of als globale variabele wil declareren. Globale variabelen starten steeds met een hoofdletter per woord. Lokale variabelen noteer je steeds in kleine letters.

Het volgende voorbeeld zal duidelijk maken hoe de verschillende variabelen werken:

```
int Value;        // globale var 'Value' is zichtbaar in het hele programma

void setup()
{}              // geen setup nodig voor dit programma

void loop()
{
      for (int i=0; i<20;) // lokale var 'i' is alleen zichtbaar in de for loop
      {
          i++;                    // i = i + 1
      }
      float f;              // lokale var 'f' is alleen zichtbaar in void loop()
}
```

Op de volgende bladzijde worden de verschillende type variabelen beschreven.

Datatypes of Formaten van variabelen

In de **cheatsheet** staan de verschillende variabelen en opties mooi samengevat. We bespreken hier enkele van de meest gebruikte types.

Variables, Arrays, and Data

Data Types

boolean	true | false
char	-128 - 127, 'a' '$' etc.
unsigned char	0 - 255
byte	0 - 255
int	-32768 - 32767
unsigned int	0 - 65535
word	0 - 65535
long	-2147483648 - 2147483647
unsigned long	0 - 4294967295
float	-3.4028e+38 - 3.4028e+38
double	currently same as float
void	i.e., no return value

Strings

```
char str1[8] =
  {'A','r','d','u','i','n','o','\0'};
  // Includes \0 null termination
char str2[8] =
  {'A','r','d','u','i','n','o'};
  // Compiler adds null termination
char str3[] = "Arduino";
char str4[8] = "Arduino";
```

Numeric Constants

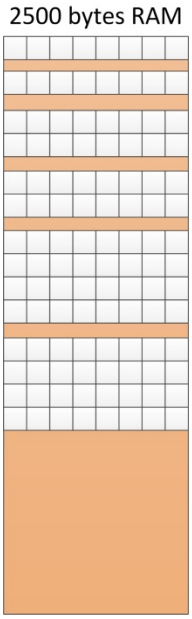
123	decimal
0b01111011	binary
0173	octal - base 8
0x7B	hexadecimal - base 16
123U	force unsigned
123L	force long
123UL	force unsigned long
123.0	force floating point
1.23e6	1.23*10^6 = 1230000

Qualifiers

static	persists between calls
volatile	in RAM (nice for ISR)
const	read-only
PROGMEM	in flash

Arrays

```
int myPins[] = {2, 4, 8, 3, 6};
int myInts[6];   // Array of 6 ints
myInts[0] = 42;  // Assigning first
                 // index of myInts
myInts[6] = 12;  // ERROR! Indexes
                 // are 0 though 5
```



byte x = 27;	8 bit 'byte'	0-255
char y = 'A';	8 bit 'char'	-128 -> +127 of tekens 'A', 'x'
int z = 426;	16 bit 'int'	-32768 -> +32767
unsigned int w = 290;	16 bit 'int'	0 -> 65535
long m = 9999999;	32 bit 'long'	-2,147,483,648 -> 2,147,483,647.
float o = 1.117;	32 bit 'float'	-3.4028235E+38 -> +3.4028235E+38

Variabelen worden opgeslagen in het RAM geheugen van onze µC. Ons RAM geheugen kan 2500 bytes (8 bit getallen) onthouden. Afhankelijk van het type van variabele is het formaat een veelvoud van bytes. Hoe groter de variabele, hoe groter de getallen die er in kunnen worden opgeslagen, maar ook hoe langer het duurt om er mee te werken. Verstandig omgaan met variabelen is dus een belangrijke eigenschap van een goede programmeur.

byte

Byte bewaart een 8-bit numerieke waarde zonder een decimale punt met een bereik van 0-255.

```
byte Button1 = 180;     // declareert 'Button1' als een byte type
```

int

Integers zijn primaire datatypes om getallen te bewaren zonder een decimale punt een 16-bit waarde met een bereik van 32767 tot -32768.

```
int Count4 = 1500; // declareert Count4' als een integer type
```

Opmerking: Een integer variabele kan niet groter zijn dan 32767. Verhoog je 32767 met 1 dan wordt het een negatief getal: -32768.

```
unsigned int Count4 = 1500; // declareert Count4' als een unigned integer
```

Het woord “unsigned” voor de int zorgt ervoor dat deze int var enkel positieve getallen kan bevatten. De range wijzigt dan van 0 tot 65535

long

Datatype voor erg grote getallen, zonder een decimale punt (een soort uitgebreide integer) een 32-bit waarde met een bereik van 2,147,483,647 tot
-2,147,483,648.

```
long SomeVariabele = 90000;                    // declareert 'SomeVariabele’ als een long type
```

float

Een datatype voor getallen met een decimale punt. Dus met getallen achter de komma. Floating datatypes nemen meer geheugen in gebruik dan een integer en worden opgeslagen als een 32-bit waarde met een bereik van 3.4028235E+38 tot
-3.4028235E+38.

```
float SomeVariabele = 3.14;           // declareert 'SomeVariabele' als een float-point type
```

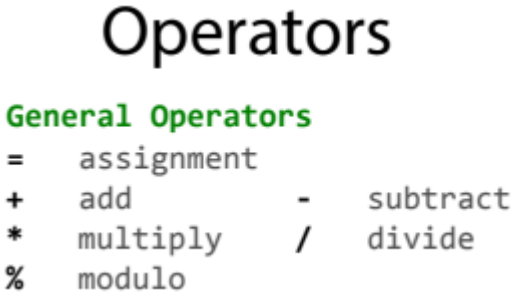
Opmerking: Het is van groot belang dat je het formaat van de variabelen correct declareert. D.w.z niet te klein, maar zeker ook niet te groot.

- Grote variabelen nemen meer plaats in in het beperkte RAM geheugen van de processor
- Het rekenen met grote variabelen kost de processor veel meer tijd als rekenen met kleine variabelen.

REGEL 3: Kies het formaat van uw variabele verstandig. Niet te groot, maar ook niet te klein.

Rekenen

Rekenkundige bewerkingen zijn bijvoorbeeld optellen, aftrekken, vermenigvuldigen en delen. Het is altijd een resultaat van twee getallen (operands). Voorbeelden:




```
y = y + 3;
x = x - 7;
i = j * 6;
r = r/ 5;
z = 11%5; //(uitkomst z = 1, % = modulo - rest van een deling)
```

De berekening wordt uitgevoerd afhankelijk van het gekozen datatype. Als er gekozen is voor een integer dan zal het resultaat 9 / 4 = 2 zijn in plaats van 2.25. Indien de datatypes van het type float waren geweest, dan had de uitkomst wel als een kommagetal zijn weergegeven.

Pas ook op bij rekenkundige bewerkingen voor een "**overflow error"** bij te grote getallen. Een byte kan tot maximaal 255 bevatten, zodat een variabele die gedeclareerd is als een byte de optelling van 250 + 30 niet juist zal weergeven.
250+5 = 255 +1 = (256, maar dat past niet in een byte en wordt dus weergegeven als 0) 0 + 24 = 24. De uitkomst zal dus worden weergegeven als 24.

Zijn de getallen die je gaat bewerken van twee verschillende types, dan wordt het grootste type gebruikt voor de berekening. Bijvoorbeeld als één van de getallen een integer is en het andere getal een float dan zal de uitkomst een getal zijn van het type float.

Datatypes kunnen ook geforceerd worden omgezet naar een ander type tijdens berekeningen:

```
123U        force unsigned
123L        force long
123UL       force unsigned long
123.0       force floating point
1.23e6      1.23*10^6 = 1230000
```

Kies dus altijd een type variabele dat groot genoeg is voor de gewenste berekening. Zorg dat je weet wat er gebeurt als de gebruikte variabele door een optelling ineens van positief veranderd in negatief. Weet je het niet zeker lees dan een paar pagina's terug hoe je getallen moet declareren. Let echter wel op dat float variabelen veel geheugen in beslag nemen en ook de microcontroller zwaarder belasten (lees: langzamer werken).

Samengestelde opdrachten

Samengestelde opdrachten voor simpele wiskundige berekeningen kent de Arduino ook. Ze worden veel gebruikt in loops en worden later nog beschreven in deze manual. De meest voorkomende samengestelde opdrachten zijn:

```
x ++   // is hetzelfde als x = x + 1, of verhoog x met +1
x --   // is hetzelfde als as x = x - 1, of verlaag x met -1
```

Onderstaande samengestelde opdrachten komen ook voor in sommige programma's, maar wij gebruiken die niet omdat deze de leesbaarheid van een programma niet ten goede komen.

```
x += y  // is hetzelfde als as x = x + y, of verhoog x met +y
x -= y  // is hetzelfde als x = x - y, or of verlaag x met -y
x *= y  // is hetzelfde als x = x * y, of vermenigvuldig x met y
x /= y  // is hetzelfde als x = x / y, of deel x met y
```

Vergelijken van getallen

Variabelen worden vaak met elkaar vergeleken. Op basis daarvan worden er dan beslissingen genomen.

```
if (x == y)    // Als x gelijk is aan y
if (x != y)    // Als x is niet gelijk aan y
if (x < y )    // Als x is kleiner dan y
if (x > y )    // Als x is groter dan y
if (x <= y)    // Als x is kleiner of gelijk aan y
if (x >= y)    // Als x is groter of gelijk aan y
```

Logische berekeningen

Logische berekeningen zijn vergelijkingen die als uitkomst 'waar' of 'niet waar' (TRUE of FALSE) hebben. Er zijn drie logische operaties: AND, OR en NOT die vaak gebruikt worden in zogenaamde if opdrachten:

Logische AND "&&":

```
if ((x > 0) && (x < 5))        // enkel waar als beide vergelijkingen waar zijn
```

Logische OR "||":

```
if ((x > 0) || (y > 0))        // waar als één van de twee vergelijkingen waar is
```

Logische NOT "!":

```
if (!(x > 0))                  // alleen waar als vergelijking niet waar is
```

Constanten

De Arduino taal heeft een aantal voorgedefinieerde waarden, die ook wel constanten worden genoemd. Ze worden gebruikt om een programma gemakkelijker te kunnen lezen of schrijven. Constanten zitten ook in verschillende groepen. Constanten worden steeds met hoofdletters geschreven.

TRUE/FALSE

Dit zijn Boolean constanten die een logisch niveau vaststellen. False wordt gedefinieerd als een 0, terwijl true wordt gedefinieerd als een 1 of iedere andere waarde anders dan een 0. In Boolean is -1, 2 en -900 gedefinieerd als true. Voorbeeld:

```
if (b == TRUE);   //Als variabele b waar is of als b niet gelijk is aan 0
{
        Doe iets;       // willekeurige code
}
```

HIGH/LOW

Deze constanten definiëren een pin niveau van HIGH of LOW en worden gebuikt om digitale pennen te lezen of schrijven. HIGH is een logische 1 en LOW is een logische 0. Een logische 1 is meestal 5 V. Voorbeeld:

```
digitalWrite(13, HIGH);  // maak digitale pin 13 hoog
```

INPUT/OUTPUT

Deze constanten worden gebruikt met de opdracht pinMode() om te definiëren of een digitale pin INPUT of OUTPUT moet worden. Voorbeeld:

```
pinMode(13, OUTPUT);   //Pin 13 is een output
```

LED_BUILTIN

Meestal hangt de vaste led op een Arduino bordje aan pin13, maar als je het niet zeker weet, kan je het woord LED_BUILTIN gebruiken ipv 13. Arduino IDE weet zelf van elk bordje aan welke pin de led hangt en zal dit voor u vertalen.

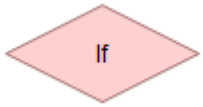
```
pinMode(LED_BUILTIN, OUTPUT);   //Pin 13 is een output
digitalWrite(LED_BUILTIN, HIGH);  // maak digitale pin 13 hoog
```

Programmeerstructuren of FLOW-CONTROL

Onderstaande programmeer-structuren zijn aanwezig in nagenoeg elke bestaande programmeertaal. Het zijn allemaal voorwaardelijke sprong- of herhalingsinstructies die het verloop (of de flow) van het programma kunnen bepalen afhankelijk van bepaalde voorwaarden. Elke flow-control instructie heeft zijn eigen toepassingen en dikwijls kunnen er verschillende instructies gebruikt worden om toch tot een zelfde werkend eindresultaat te komen.

```
Control Structures
if (x < 5) { ... } else { ... }
while (x < 5) { ... }
for (int i = 0; i < 10; i++) { ... }
break;      // Exit a loop immediately
continue;   // Go to next iteration
switch (var) {
  case 1:
    ...
    break;
  case 2:
    ...
    break;
  default:
    ...
}
```

if



De if opdracht test of bepaalde condities bereikt zijn. Denk bijvoorbeeld aan een analoog signaal dat een bepaalde waarde bereikt waarbij ingegrepen moet worden. In dat geval moet er iets gebeuren. Die actie moet dan plaats vinden binnen de haakjes (zie het voorbeeld hier onder). Wordt er niet aan de voorwaarde voldaan, dan wordt de actie tussen de haakjes overgeslagen.

Voorbeeld:

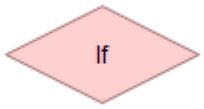
```
if (waardeVariabele == waarde)          //Als waardeVariabele groter of gelijk is aan waarde
{
        Doe iets;                       // voer deze code uit als de voorwaarde waar is, spring dan naar x?
}
                                        // deze code wordt uitgevoerd als de voorwaarde niet waar is.
```

In het bovenstaande voorbeeld wordt de waardeVariabele vergeleken met een andere waarde.

Opmerking: Een veelvoorkomende schrijffout: **if(x=10).** Deze foute code (=-teken vergeten) zal geen foutmelding geven omdat dit programmatorisch geen fout is. Het geeft x de waarde 10 en heeft als resultaat dus altijd TRUE, omdat x nu niet gelijk is aan 0. Gebruik dus altijd met zorg '==' zodat bij de opdracht if(x==10) getest wordt of x gelijk is aan de waarde 10 of niet. Bedenk: bij '=' aan de term "maak gelijk aan" en bij '==' aan de vraag "is gelijk aan".

if… else

De if… else opdracht maakt het mogelijk hoe dan ook een beslissing te laten nemen. Bijvoorbeeld je meet dat een digitale input pin hoog is, in dat geval wil je dat actie_A start. Is de pin echter laag, dan moet actie_B starten. Dat zou er als volgt uit kunnen zien:.




```
if (inputPin == HIGH)   // controle of inputPin hoog is
{
        Voer aktie_A uit;      // gevolg als inputPin hoog is
}
else
{
        Voer aktie_B uit;      // gevolg als inputPin laag is
}
```

Else kan ook een andere procedure zijn zodat je meerdere testen in dezelfde lus kunt verwerken. Bekijk het volgende voorbeeld eens:

```
if (inputPin < 500)
{
        Voer aktie_A uit;   // gevolg als inputPin kleiner is dan 500
}
else if (inputPin >= 1000)
{
        Voer aktie_ B uit;// gevolg als inputPin groter of gelijk is aan 1000
}
else
{
        Voer aktie_C uit; // gevolg als inputPin tss 500 en 999
}
```

Opmerking: Kijk goed naar de haakjes en de puntkomma's dat wil op deze wijze best wel eens ingewikkeld worden.

For (loop)

Het for commando wordt gebruikt om een aantal commando's een bekend aantal keren te laten herhalen. Via een teller wordt bijgehouden hoe vaak de lus zich moet herhalen. Het commando ziet er als volgt uit:


```
for (variabele; conditie; expressie)
{
        doeiets;
}
```

Dat lijkt lastig, maar het is een eenvoudige en vaak gebruikte opdracht. Een voorbeeld:

```
for (byte i=0; i<20; i++)          // declareer i, en test of i kleiner is dan 20, i wordt elke loop met 1 verhoogd
{
  digitalWrite(13,  HIGH);         //  zet pin 13  aan
  delay(250);                      //  1/4 second  pauze
  digitalWrite(13,  LOW);          //  zet pin 13  uit
  delay(250);                      //  1/4 second  pauze
}
```

In de eerste regel wordt i gedeclareerd en wordt meteen getest of i kleiner is dan 20. Daarna wordt i met 1 verhoogd (en krijgt de waarde 1). Aangezien i nul was, wordt alle code die er onder staat tussen de haakjes uitgevoerd. Daarna wordt regel 1 opnieuw uitgevoerd. Variabele i heeft nu de waarde 1 en er wordt weer getest of i kleiner is dan 20, hetgeen nog steeds het geval is. Nu wordt i met 1 verhoogd, zodat i de waarde 2 krijgt. Dat blijft zich herhalen tot i de waarde 20 bereikt. Daarna wordt de code tussen de haakjes niet meer uitgevoerd.

Opmerking: In C is de for loop veel flexibeler in te vullen dan in sommige andere computertalen zoals bijvoorbeeld BASIC. De variabele, conditie en expressie kan je naar wens aanpassen. Let op: ze worden wel gescheiden door een puntkomma

While (loop)

De while loop heeft wel wat weg van de for loop. Hij is gemakkelijk uit te leggen met: “Zolang je aan die voorwaarde voldoet, moet je deze instructies uitvoeren”. Die voorwaarde zou bijvoorbeeld het testen van een sensor kunnen zijn. De loop stopt pas als hij niet meer aan de voorwaarde voldoet. Een voorbeeld:

```
while (someVariable == value)
{
        doe iets;
}
```

Het volgende voorbeeld test of de someVariabele kleiner is dan 200. Als dat waar is, blijft de lus zich herhalen totdat someVariabele niet langer kleiner is dan 200. Onderstaande loop wordt dus 200 maal doorlopen.

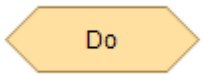
```
byte someVariabele = 0;      // declareer deze var en zet deze op 0

while (someVariabele < 200)  // test of someVariabele kleiner
                                        // is dan 200
{
        Doe iets;            // voer programmacode uit
        someVariabele++;         // verhoog variabele met 1
}
```

Opmerking: indien de laatste regel vergeten zou worden, zal het programma in een oneindige lus terecht komen!

do… while

De do while loop werkt grotendeels hetzelfde zoals de while loop. Het verschil zit hem in het feit dat de conditie onderaan staat in plaats van bovenaan, zoals bij de while loop. Ongeacht de voorwaarde worden de instructies binnen een do while loop altijd minimaal 1 keer doorlopen.

```
do
{
        doeiets;    // instructies binnen de do-while loop
                    // worden altijd minimaal 1 keer doorlopen
}
while (someVariable == value); // zolang deze voorwaarde TRUE is.
```



Switch…case

De switch case instructie gelijkt heel erg op de if instructie, maar geeft de programmeur de kans om op een verkorte manier hele specifieke gevolgen te geven aan specifieke situaties.

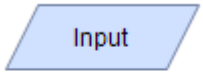
Met de switch instructie geef je aan welke variabele moet worden bekeken. Wanneer een case-statement wordt gevonden waarvan de waarde overeenkomt met die van de variabele, wordt de code in die case-instructie uitgevoerd.

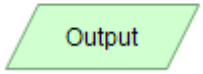
Het break-sleutelwoord verlaat de switch-instructie en wordt meestal gebruikt aan het einde van elke case. Zonder een break-statement blijft de switch-instructie de volgende uitdrukkingen uitvoeren tot een pauze, of het einde van de instructie switch wordt bereikt.
De default wordt uitgevoerd wanneer geen van de cases een match heeft opgeleverd.

```
switch (var) {
 case 1:      // wanneer var gelijk is aan 1
   statements
   break;     // spring uit de case structuur
 case 2:      // wanneer van gelijk is aan 2
   statements
   break;     // spring uit de case structuur
 default:     // in alle andere gevallen
   statements }
```

Opmerking: switch case instructies kunnen enkel een gelijkheid testen en niet groter of kleiner dan. Daarvoor moet je kiezen voor een if instructie.

Input output instructies





pinMode(pin, mode)

Wordt gebruikt in de void setup() loop om een specifieke pin te configureren als een INPUT of als een OUTPUT.

```
pinMode(pin, OUTPUT); // sets pin to output
```

Arduino's digitale pinnen zijn standaard geconfigureerd als inputs. Je hoeft ze dus niet per sé te declareren als inputs met pinMode(). Pinnen die geconfigureerd zijn als INPUT bevinden zich in een hoogohmige toestand.

Deze hoogohmige input mode is voor ons een veilige toestand. Zolang we hier geen spanningen op aansluiten die boven de 5V stijgen kunnen we deze pin niet stuk krijgen. De hoogohmige weerstand die in deze situatie intern aan elke pin hangt, zal deze ingang beveiligen tegen te grote stromen.

Van het moment je echter een pin definieert als uitgangspin, moet je wel heel erg goed opletten wat je er aan hangt. De hoogohmige interne beveiligingsweerstand staat er in dit geval niet meer zodat deze pin nu wel stuk kan gemaakt worden door ondoordachte schakelingen. De schakelingen op de werkbladen die je kan terugvinden op we website www.e2cre8.be >> BBA zijn hierop berekend.

Pinnen die geconfigureerd zijn als OUTPUT staan in een laagohmige toestand en kunnen maximaal 425 mA leveren. Dit is ruim genoeg voor een LED, maar veel te weinig voor een motor of bijvoorbeeld een relais.

Kortsluiting tussen verschillende poorten kunnen de poort of de gehele Atmega chip onherstelbaar beschadigen. In dat geval moet de chip vervangen worden (inclusief een nieuwe bootloader).

digitalRead(pin)

Leest de waarde uit van een specifieke pin met als resultaat HIGH of LOW. De pin is gespecificeerd al een variabele of een constante (0-13).

```
Value = digitalRead(Pin);          // maak 'value' gelijk aan de input pin
```

digitalWrite(pin, value)

Schrijf de waarde naar een specifieke pin met als niveau HIGH of LOW. De pin is gespecificeerd als een variabele of een constante (0-13).

```
digitalWrite(pin, HIGH);   // maak 'pin' hoog
```

Het volgende voorbeeld leest een drukknop uit die is aangesloten op pin 7 en laat een LED, aangesloten op pin 13 aan gaan als de drukknop ingedrukt is:

```
int led = 13;   // LED op pin 13
int pin = 7;    // drukknop op pin 7 int value = 0;     // variabele value

void setup()
{
        pinMode(led, OUTPUT);   // maak van pin 13 een output
        pinMode(pin, INPUT);    // maak van pin 7 een input
}

void loop()
{
        value = digitalRead(pin);   // maak van 'value' de input pin
        digitalWrite(led, value);   // zet value over naar de 'led'
}
```

analogRead(pin)

Leest de waarde van een specifieke analoge pin in een 10 bit resolutie. Deze functie werkt alleen op pin 0 t/m 6 – aangegeven met een 

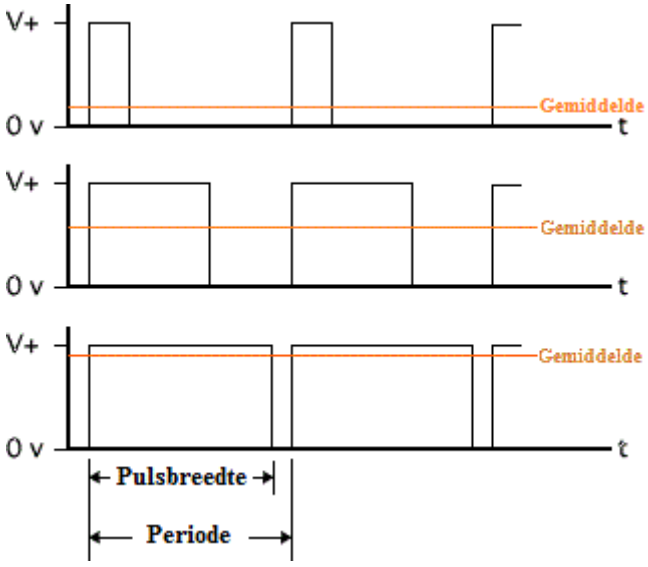
. De uitkomst is een integer waarde tussen 0 to 1023.

```
waarde = analogRead(pin);                    // maak van waarde' wat gelezen
                                                    // wordt op de 'pin'
```

Opmerking: Analoge pinnen hoeven niet te worden gedeclareerd als INPUT of OUTPUT. Het zijn automatisch al digitale inputs.

analogWrite(pin, value)

Niet tegenstaande de naam is dit eigenlijk geen analoge uitgang. Onze uC heeft namelijk geen analoge uitgangen, maar we kunnen wel dat effect creëren door op een uitgang een “PWM” puls-breedte (Pulse Width Modulation) modulatie signaal te genereren.



Een PWM signaal is een digitaal signaal met een constante frequentie, waarvan de aan-tijd varieert t.o.v. de uit-tijd.
De gemiddelde spanning stijgt dan naarmate de aan-tijd langer wordt tov de Periode (ook wel duty cycle genoemd)

$$duty\ cycle = \frac{Aan - tijd}{Periode}$$

Doordat we zo snel aan en uit schakelen lijkt het voor leds of motoren alsof we de analoge spanning laten stijgen of dalen.

Zo'n PWM signaal genereren kan enkel op pins die gemarkeerd zijn met 

.
Voor onze Brainbox AVR is dat pin 3, 5,6,9,10,11,13.

De value of waarde kan worden weergegeven met een getal tussen 0 en 255.

```
analogWrite(pin, waarde);   // schrijf 'waarde' naar analoge 'pin'
```

Een waarde van 0 geeft 0 V als output op de gespecificeerde pin. Een waarde van 255 geeft 5 V als output op de gespecificeerde pin. Voor waarden tussen 0 en 255 vindt er een evenredige pulsbreedte modulatie plaats, waarbij opgemerkt moet worden dat de breedte van de pulsen evenredig groter wordt naarmate de waarde stijgt.

Omdat dit een hardware functie is, zal de uitgegeven pulsbreedte continue het zelfde zijn totdat er een nieuwe analogWrite wordt gedaan.

Het volgende voorbeeld leest een analoge waarde van een analoge INPUT pin. Vervolgens wordt deze waarde aangeboden aan een PWM OUTPUT pin. Let op de gelezen waarde is van 0 – 1023, maar de maximale aangeboden PWM waarde mag niet meer zijn dan 255. Daarom wordt de gelezen waarde herschaald met de map instructie.

```
int led = 10;   //  LED met 220 weerstand op pin 10
int pin = 0;    //  potentiometer op analoge pin 0
int value;  //  value om te lezen

void setup(){}  //  geen setup nodig

void loop()
{
        value = analogRead(pin);    //  lees 'value' op 'pin'
        value = map(value,0,1023,0,255) //  converteer 0-1023 to 0-255
        analogWrite(led, value);    //  outputs PWM signaal naar led
}
```

Tijd

delay(ms)

Wachtlus (pauze) weergegeven in milliseconden, waarbij de waarde 1000 gelijk staat aan 1000msec of 1 seconde.

```
delay(1000); // wacht een seconde
```

delayMicroseconds(us)

Wachtlus (pauze) weergegeven in microseconden, waarbij de waarde 1000 gelijk staat aan 1000usec of 1 milliseconde.

```
delayMicroseconds(1000); // wacht een milliseconde
```

millis()

Laat zien hoeveel milliseconde het Arduino board in werking is na de start van het lopende programma. De weergave is een long variabele.

```
unsigned long Looptijd;    // looptijd kan een heel grote waarde aannemen
                           // en moet dus in een grote var gezet worden

Looptijd = millis(); // looptijd wordt gevuld met millis()
```

Note: Het weer te geven getal zal na verloop van tijd een overflow veroorzaken. (Een reset naar nul), na ongeveer 50 dagen.

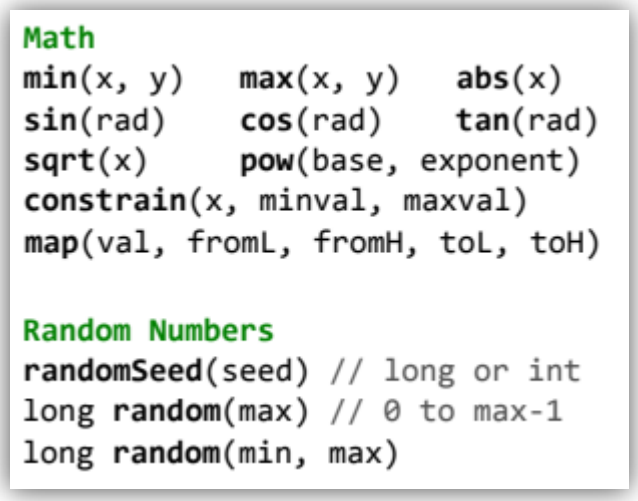
micros()

Laat zien hoeveel milliseconde het Arduino board in werking is na de start van het lopende programma. De weergave is een long variabele.

```
unsigned long Looptijd;    // looptijd kan een heel grote waarde aannemen
                           // en moet dus in een grote variabele gezet worden

Looptijd = micros(); // looptijd wordt gevuld met micros()
```

Note: Het weer te geven getal zal na verloop van tijd een overflow veroorzaken. (Een reset naar nul), na ongeveer 70 minuten

Wiskundige functies



1 UIT CHEATSHEET

We bespreken hier enkele van de mogelijke wiskundige functies.

min(x, y)

Berekent het minimum van twee getallen en geeft het kleinste getal weer.

```
waarde = min(getal, 100);
// 'waarde' is gelijk aan 100
// of kleiner dan 100 als getal
// kleiner dan 100is
```

max(x, y)

Bereken het maximum van twee getallen en geef het grootste getal weer.

```
waarde = max(getal, 100);        // 'waarde' is gelijk aan 100
                                 // of hoger dan 100 als getal
                                 // hoger dan 100 is
```

map(value, fromLow, fromHigh, toLow, toHigh)

De map instructie wordt gebruikt om waarden te herschalen.

map(value, fromLow, fromHigh, toLow, toHigh)

Stel dat x een variabele is die getallen kan bevatten tussen 0 en 1023, maar dat we die waarde willen omzetten naar een evenredige waarde tussen 0 en 255 die we opslaan in variabele y, dan kunnen we dat zo schrijven:

```
y = map(x, 0, 1023, 0, 255);
```

Evaluatiesysteem

Op de volgende pagina's staan – per onderwerp – telkens een aantal opgaven. Met de leerkracht wordt afgesproken welke opgaven op welke data geëvalueerd zullen worden.

Dit zijn de evaluatiecriteria die wij hanteren:

			1 = waar	
Opgave werd niet uitgevoerd		0		0
Werking programma is ondermaats		1		0
Opgave werd bijna perfect uitgevoerd		2		0
Opgave perfect uitgevoerd		4	1	4

Alle regels werden van zinvolle commentaar voorzien	JA	1	1	1
	NEEN	0		0

Variabelen kregen zinvolle namen	JA	1	1	1
	NEEN	0		0

Het formaat van de variabelen is verstandig gekozen	JA	1	1	1
	NEEN	0		0

Het programma is vlot leesbaar door derden	JA	1	1	1
	NEEN	0		0

De leerling loopt achter op de opgestelde planning		0		0
De leerling loopt gelijk met de opgestelde planning		1		0
De leerling loopt voor op de opgestelde planning		2	1	2

Geen of weinig eigen inbreng in het programma		0		0
Er is een bepaalde eigen inbreng die het programma een meerwaarde geeft		1		0
Het programma overstijgt de opgave door de eigen inbreng		2	1	2

Eindresultaat Evaluatie Arduino IDE programma	op 12	12

Opgaven

A Digitale output

- Gebruik de handleiding 0-2LED op www.e2cre8.be >> BBA - we gebruiken de twee vaste leds op de BBA
- A1: schrijf een programma dat beide leds afzonderlijk laat flikkeren met een frequentie 10Hz

- Gebruik de handleiding 0-20LED op www.e2cre8.be >> BBA - je sluit nu zelf minimaal 1 led correct aan op de uitgangen van de BBA.
- A2: schrijf een programma waarmee je de traagheid van uw ogen test. Vanaf een bepaalde snelheid lijkt het alsof de led constant aan is, terwijl deze toch aan en uit flikkert. Zoek die grens proefondervindelijk op.

B Geluid

- Gebruik de handleiding **0-BUZZER** op www.e2cre8.be >> BBA - we gebruiken de vaste buzzer op de BBA
- B1: Maak een programma dat een sirene laat horen op de buzzer van de BBA, enkel gebruik makend van delay() en delayMicroseconds() instructies
- B2: Maak een eenvoudige melodie, gebruik makend van de tone() functie. Onderzoek eerst de werking van de tone() instructie op de Arduino website. Installeer indien nodig een extra bibliotheek.

C Dig input

- Gebruik de handleiding **I-DIG SWITCH** op www.e2cre8.be >> BBA - sluit op een correcte manier (met weerstanden!!) een schakelaar aan op een digitale ingang van de BBA.
- C1: Maak een programma dat de toestand van een schakelaar weergeeft op een led. Als de schakelaar ingedrukt is moet de led branden, als de schakelaar wordt losgelaten mag de led niet branden.
- C2: Verander het vorige programma zodat de led uit gaat als de schakelaar ingedrukt wordt en aan wanneer die wordt losgelaten.
- C3: sluit twee schakelaars aan – op twee verschillende digitale inputs. De led mag pas aan gaan als beide schakelaars tezamen ingedrukt zijn (EN-POORT)
- C4: sluit twee schakelaars aan – op twee verschillende digitale inputs. De led mag pas aan gaan als minimaal 1 van de schakelaars ingedrukt is (OF-POORT)
- C5: sluit twee schakelaars aan – op twee verschillende digitale inputs. De led mag pas aan gaan als slechts 1 van de schakelaars ingedrukt is (EXOR-POORT)
- C6: Schrijf een programma dat de led aan laat gaan als SW1 wordt ingedrukt en uit laat gaan als SW2 wordt ingedrukt.
- C7(U): Schrijf een START-STOP programma met 1 schakelaar. Wanneer de schakelaar de eerste maal wordt ingedrukt moet de led aan gaan, wanneer de zelfde schakelaar een tweede maal wordt ingedrukt moet de led weer uitgaan. Dit is niet zo'n eenvoudig programma. Je zal rekening moeten houden met eventuele dender (zoek dit op) van de schakelaar, je zal iets moeten schrijven dat enkel op de positieve flank van de schakelaar reageert en je zal een bepaalde geheugenfunctie moeten programmeren die onthoudt wat de stand was van de led.

D Serial monitor / Serial plotter

De seriële monitor is een zeer nuttige extra feature die in Arduino IDE is ingebouwd. Zolang uw Arduino bordje via USB verbonden is met uw PC, kan er data van uw Arduino Bordje naar de PC gestuurd worden. Deze data kan dan door de Serial Monitor als tekst gevisualiseerd worden. Deze methode is ideaal om te volgen wat er juist in de µC gebeurt.

Onderstaand programma demonstreert de functies van de serialmonitor

```
void setup() {
  Serial.begin(9600);     // open the serial port at 9600 bps:
}

void loop() {
  Serial.print("NOFORM");      // print tekst
  Serial.print("\t");         // prints a tab (vaste spatie)

  Serial.print("DEC");
  Serial.print("\t");

  Serial.print("HEX");
  Serial.print("\t");

  Serial.print("BIN");
  Serial.print("\t");
  Serial.println("");            // begin een volgende regel - print niets


  for(byte x=0; x< 64; x++){    // Enkel de eerste 64 waarden

    // print deze waarden in 4 verschillende talstelsels:
    Serial.print(x);              // print as an ASCII-encoded decimal
    Serial.print("\t");                    // prints a tab

    Serial.print(x, DEC);       // print as an ASCII-encoded decimal
    Serial.print("\t");         // prints a tab

    Serial.print(x, HEX);       // print as an ASCII-encoded hexadecimal
    Serial.print("\t");         // prints a tab

    Serial.println(x, BIN);     // print as an ASCII-encoded binary
                                      // then adds the carriage return with "println"
    delay(200);                 // delay 200 ms, om het geheel te vertragen
  }
  Serial.println("");           // begin een nieuwe lijn
}
```

- D1: test bovenstaand programma uit. Om in combinatie met onze Brainbox AVR de Serial monitor te activeren nadat een nieuw programma werd ingeladen, is het soms nodig om enkele seconden te wachten totdat het programmeren volledig afgelopen is en de poort terug vrijgegeven is voor de Serial Monitor. Soms moet ook de juist com poort even terug geselecteerd worden.
- D2: Schrijf een programma dat de toestand van 2 correct aangesloten schakelaars **I-SWITCH** schakelaars weergeeft op de Seriële Monitor, met de nodige tekst erbij.

E Timing

- Gebruik de handleiding **I-DIG SWITCH** op www.e2cre8.be >> BBA - sluit op een correcte manier (met weerstanden!!) een schakelaar aan op een digitale ingang van de BBA.
- E1: Gebruik de millis() en micros() functies – in combinatie met de Seriële monitor om het aantal milliseconden weer te geven. De timer wordt gestart met SW1 en gestopt met SW2.
- E2: Schrijf een programma dat een spel maakt waarbij je met de twee schakelaars zo dicht mogelijk een tijd van 3 seconden moet benaderen. De timer wordt gestart met SW1 en gestopt met SW2. De tijd moet mooi worden weergegeven, maar ook één van de volgende commentaren : Veel te kort, Een beetje te kort, Proficiat – helemaal juist, Een beetje te lang, Veel te lang.
- E3(U): Gebruik de handleiding **I-DIG LICHTSLUIS** op www.e2cre8.be >> BBA - sluit op een correcte manier 2 lichtsluizen aan en geeft de tijd weer tussen de twee lichtsluizen. Zet eventueel de twee lichtsluizen op 1 meter afstand en geef ook de snelheid weer in meter/sec.

F Analog input

- Gebruik de handleiding I-AN Potmeter op www.e2cre8.be >> BBA - sluit de potmeter correct aan.
- F1: Schrijf een programma dat de waarde van de potmeter weergeeft op de seriële monitor als een waarde tussen 0 en 2013 en als een waarde tussen 0 en 255. Voorzie deze waarden ook van de nodige tekst en tabs.
- F2: Arduino IDE heeft ook een Seriële plotter ingebouwd die meetwaarden grafisch kan weergeven. Onderzoek hoe deze werkt en schrijf een programma dat de meetwaarde van een potmeter grafisch kan weergeven.
- F3: Gebruik de waarde tussen 0 en 255 uit vorige oefening om:
    - Led 13 en 17 uit te laten gaan als de waarde onder 75 ligt
    - Enkel led 13 aan te laten gaan als de waarde ligt tussen 75 en 120
    - Enkel led 17 aan te laat gaan als de waarde ligt tussen 120 en 190
    - Beide leds aan te laten gaan als de waarde boven 190 ligt
    - Houdt ook rekening met uw voorwaarden. Voor elke mogelijke situatie moet een gevolg geprogrammeerd zijn.
    - De waarde van de potmeter moet ook op de Serial monitor te volgen zijn.
- F4: kies nu zelf van de BBA I-AN pagina een analoge sensor (geen potmeter) die je correct aansluit op de BBA en waarvan je het meetresultaat correct weergeeft.

G PWM output

- We gebruiken de led die is aangesloten op pin 13 van de BBA.
- G1: Gebruik de analogWrite() functie om de led rustig harder te laten branden en dan rustig te laten doven. De huidige PWM waarde moet zichtbaar gemaakt worden op de Serial Monitor
- G2: Sluit een potmeter aan (I-AN POTMETER) en bepaal hiermee hoe hard de led op D13 gedimd wordt.
- G3: Sluit een DC motortje aan zoals in O-PWM DC MOTOR en bepaal met een potmeter de snelheid van de DC motor.
- G4(U): Sluit een RGB led aan (0-20 RGB LED) op 3 PWM uitgangen. Sluit 3 potmeters (I-AN POTMETER) correct aan op 3 analoge ingangen. Maak met de 3 potmeters alle mogelijke mengkleuren met een RGB led.

H LCD

- We gebruiken een I2C LCD en sluiten die aan zoals in de manual I2C LCD
- H1: Visualiseer de waarde van de potmeter op de LCD, zowel als waarde tussen 0 en 1023 en als waarde tussen 0 en 255.
- H2: Laat een woord van 5 letters over de breedte van het scherm over en weer schuiven.
- H3: Maak met behulp van twee drukknoppen en de LCD een keuzemenu waarin de gebruiker kan kiezen voor een programma waarin led 13 gaat knipperen, of een programma waarin een sirene hoorbaar

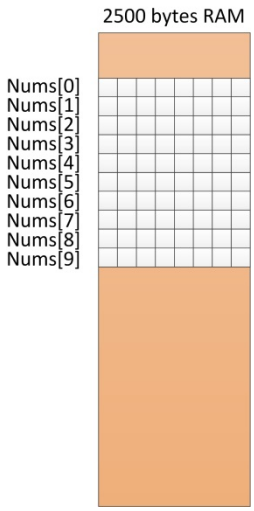
gemaakt wordt. Let wel op dat je de lopende programma's ook moet kunnen stoppen en dat de menustructuur nooit mag vastlopen. Test dit uitgebreid uit.

I Arrays

Het gebruik van Arrays om variabelen te declareren, kan u in een aantal gevallen veel winst opleveren qua programmeerwerk. In een array declareren we een groep variabelen, allemaal met hetzelfde formaat en enkel van elkaar verschillend door een indexnummer. Er zijn verschillende manieren om in Arduino IDE een Array te declareren:

```
int myInts[6];
// declareert een array met 6 variabelen van het type int (16bit)
// de naam van elke variabele in "myInts[index]"

byte myPins[] = {2, 4, 8, 3, 6};
// declareert een array met een niet gedefinieerd aantal elementen
// het formaat is byte (8bit) en de naam is myPins[index]
// We vullen deze array met 5 waarden, Arduino IDE beslist zelf dat
// deze array een array van 5 elementen zal zijn.

int mySensVals[6] = {2, 4, -8, 3, 2};
        // declareert een array van 6 elementen van het int type
        // De array wordt gevuld met 5 waarden.
        //De 6e variabele: mySensVals[5] blijft dus leeg

char message[6] = "hello";
        // declareert een array van het type char.
        //Dit type kan ook tekst bevatten (als ascii)
        //Dit type array noemen we ook een STRING
        // Het formaat moet telkens 1 plaats groter zijn dan het
        //aantal tekens om het '0' karakter (einde woord) ook mee
        // te kunnen opslaan.
```

Voorbeeldprogramma Arrays:



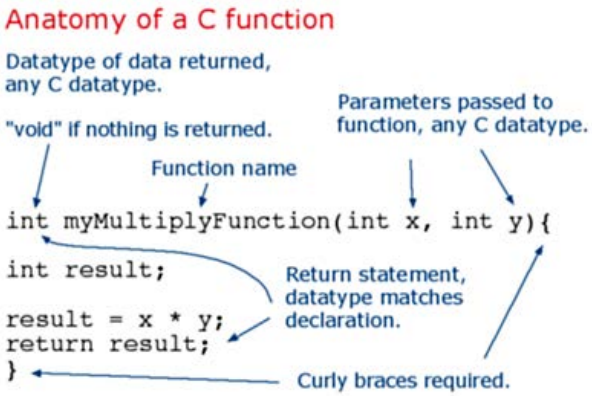

```
const int AantalWaarden = 10; // aantal waarden in array
byte Nums[AantalWaarden] = {2, 4, 8, 3, 6, 6, 7, 2, 5, 7};
// declareer een 10 byte array en vul die met waarden
int Som = 0; // declareer een variabale SOM van het type int
int Gemiddelde = 0;

void setup() {
 Serial.begin(9600);   // open the serial port at 9600 bps:
}

void loop()
{
 for(byte x=0; x< AantalWaarden; x++)
  {
   Som = Som + Nums[x]; // alle Values optellen
  }
  Gemiddelde = Som / AantalWaarden; // Gemiddelde berekenen
  Som = 0; // resetten SOM voor volgende loop
```

```
 Serial.print("Gemiddelde: ");    // prints a label
 Serial.print("\t");           // prints a tab

 Serial.print(Gemiddelde);    // print de uitkomst
 Serial.println("");        // prints another carriage return
 delay(500);              // wacht een halve seconde
}
```

- I1: Pas het vorige programma aan zodat deze het gemiddelde neemt van 20 verschillende getallen
- I2: Pas het vorige programma aan zodat de array ook automatisch en in een for loop gevuld wordt met 20 Random waarden (binnen de grenzen van een byte). Bestudeer dus eerst de Random functie in Arduino IDE.
- I3: Pas het vorige programma aan zodat nu ipv de berekening van het gemiddelde, de 20 getallen gesorteerd worden. Bestudeer hiervoor eerst verschillende sorteerroutines – je mag geen voorgeschreven Arduino sorteerroutine gebruiken.

J Functies

Het gebruik van functies maakt uw programma korter en overzichtelijker. C heeft een hele knappe functiestructuur. De typsiche Arduino bibliotheken en instructies zijn niet meer dan een verzameling van voorgeschreven functies die in Arduino IDE zijn ingewerkt.



Er bestaan 3 types van functies die alle 3 in onderstaand (nutteloos) programma worden gedemonstreerd.

```
void Wacht(void)        // functie die een delay van 1,5msec maakt
{
  delay(1);
  delayMicroseconds(500);
}
```

Deze functie krijgt geen data mee van de aanvrager (void = leeg) en geeft ook geen data terug aan de aanvrager. In dit geval wordt er een delay van 1.5msec gegenereerd. Zeker als je die meermaals in je programma nodig hebt is het interessant om deze regels telkens aan te roepen met een functie.

```
void WachtAantal(int x)   // functie die een delay van x aantal keer 1,5msec maakt
{
 for (int y = 0; y<x; y++)
 {
  delay(1);
  delayMicroseconds(500);
 }
}
```

Dit is een functie die wel data meekrijgt van de aanvrager (int x), maar geen data teruggeeft aan de aanvrager(void). Bij de aanvraag staat de data die wordt meegegeven tussen de haakjes: WachtAantal(10);

Die 10 wordt in de functie ingelezen in de variabele x die binnen de functie gebruikt wordt om een aantal keer de delay van 1.5msec te herhalen.

```
int Vermenigvuldig(char a, char b) // functie die a en b vermenigvuldigt en de uitkomst terugstuurt als een integer
{
    int resultaat;
    resultaat = a*b;
    return resultaat; // geef de uitkomst terug mee naar de functieaanvrager
}
```

Deze derde functie krijgt data van de aanvrager – in dit geval twee variabalen die in de functie in de lokale variabelen a en b worden gekopieerd. Binnen deze functie wordt hiervan het product gemaakt en de uitkomst wordt met de return instructie als een integer terug naar de aanvrager gestuurd. De functie wordt als volgt aangeroepen:

```
k = Vermenigvuldig(i,j);  //k heeft nadien de waarde i*j
```

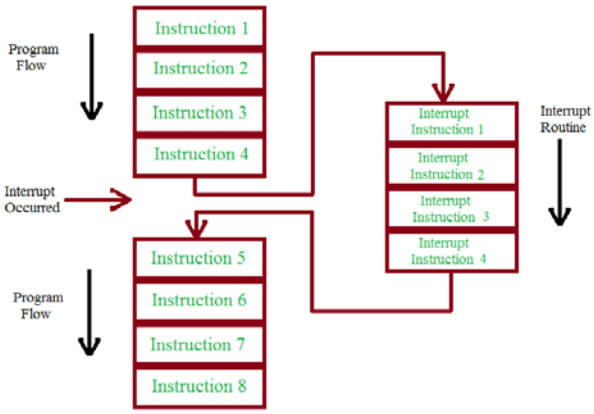
ZINLOOS VOORBEELDPROGRAMMA FUNCTIES

```
byte i = 2;
byte j = 3;
byte k = 0;

void setup() {
 Serial.begin(9600);     // open the serial port at 9600 bps:
}

void loop()
{
   Wacht();              // roep functie Wacht aan
   WachtAantal(10);        // roep functie wacht aan en geef waarde 10 mee
   k = Vermenigvuldig(i,j);  // roep functie vermenigvuldig aan en geef de waarde van var i en j mee
                     // de uitkomst moet in var k komen staan
   Serial.print("k: ");    // prints a label
   Serial.print("\t");     // prints a tab

   Serial.print(k);        // print de uitkomst k
   Serial.println("");      // prints another carriage return
   delay(500);             // wacht een halve seconde
}

void Wacht(void)          // functie die een delay van 1,5msec maakt
{
  delay(1);
  delayMicroseconds(500);
}

void WachtAantal(int x)   // functie die een delay van x aantal keer 1,5msec maakt
{
   for (int y = 0; y<x; y++)
   {
      delay(1);
      delayMicroseconds(500);
   }
}
```

```
int Vermenigvuldig(char a, char b) // functie die a en b vermenigvuldigt en de uitkomst terugstuurt als een integer
{
 int resultaat;
 resultaat = a*b;
 return resultaat; // geef de uitkomst terug mee naar de functieaanvrager
}
```

K Interrupts

Interrupts zijn een zeer krachtige feature van microcontrollers en maken het mogelijk om een lopend programma éénder waar te onderbreken om meteen te kunnen reageren op een bepaalde belangrijke event door een stukje code uit te voeren (interrupt routine) dat bij dit event hoort en dan terug verder te gaan met het uitvoeren van het hoofdprogramma.



Onze ATMEGA32U4 heeft vele mogelijke interrupt sources. Zo kan er een interrupt gegenereerd worden wanneer een timer overloopt, of wanneer er nieuwe data binnenkomt via de UART of wanneer er een flank gedetecteerd wordt op bepaalde pins.

We moeten echter in het achterhoofd houden dat we geen ATMEGA32U4 aan het programmeren zijn, maar wel een Arduino Leonardo controller. De bootloader code die van onze ATMEGA een Arduino maakt, heeft al een aantal van deze interrupts in gebruik en Arduino is niet altijd even open over welke interrupts al in gebruik zijn. Daarom wordt het gebruik van interrupts in een Arduino programma niet echt gepromoot. Tijdens een interrupt routine onder Arduino IDE telt bv millis()niet op en seriële data wordt op dit moment niet ontvangen.

De interrupts op enkele externe pins van de Arduino Leonardo zijn echter wel te gebruiken, dus die zullen we hier gebruiken om het begrip interrupts te duiden. De pins waarop deze externe interrupt mogelijk is voor onze Arduino Leonardo zijn: **0,1,2,3,7**

!! Een interrupt routine moet zo snel mogelijk worden uitgevoerd. Delay() mag niet gebruikt worden.

Syntax Arduino interrupt

attachInterrupt(digitalPinToInterrupt(pin), ISR, mode); (recommended)

Parameters Arduino Interrupt

pin: the pin number **0,1,2,3,7**
ISR: the Interrupt Service Routine to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.
mode: defines when the interrupt should be triggered. Four constants are predefined as valid values:

LOW to trigger the interrupt whenever the pin is low,

CHANGE to trigger the interrupt whenever the pin changes value

RISING to trigger when the pin goes from low to high,

FALLING for when the pin goes from high to low.

Voorbeeldprogramma Interrupt

```
#define ledPin 13 // ledpin zal bij compilatie vervangen worden door 13
                  // in het hele programma - gebruikt geen verder
                                    //geheugen van de uC
const byte interruptPin = 2;
                        // interruptPin zal vervangen worden door 2 in de loop
          // van het programma - gebruikt dus wel geheugen van de uC

volatile byte state = LOW;
             // variabelen die op de meest onlogische momenten kunnen
             // veranderen declareren we als volatile om de compiler aan te geven
             // dat deze variabele zeker in het ram geheugen moet gezet worden.

void setup() {

  pinMode(ledPin, OUTPUT);  // maak pin 13 output (LED)
  pinMode(interruptPin, INPUT);  //maak pin 2 input (schakelaar)
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
             // maak de interrupt actief op pin 2, elke keer als de
             //toestand van pin 2 wijzigt (CHANGE)
             // en roep dan telkens interrupt routine "blink" aan.
}

void loop() {
  digitalWrite(ledPin, state);
// als state = 1, is de led aan, anders is de led uit
}

void blink() {
  state = !state;   // wijzig de toestand van state
}
```

- K1: Wijzig bovenstaand programma zodat de led enkel van toestand verandert bij een stijgende flank.
- K2: Maak gebruik van interrupts om een periodemeter te maken die de periode van een blokgolf (10Hz – 500Hz (0V/5V) van functiegenerator of andere µC) meet in usec.
- K3: Vorm bovenstaand programma om tot een frequentiemeter.

L Sleepmode

- Microcontrollers gebruiken relatief gezien weinig energie, maar veel microcontrollers draaien op een batterij en dan is elke milliWatt belangrijk. Programma's die gedurende een bepaalde tijd niets uitvoeren en daarvoor een delay gebruiken, kunnen de controller dan beter in sleepmode laten gaan voor een bepaalde tijd. Gedurende die sleepmode verbruikt de controller minder energie. Er zijn verschillende sleepmodes – elk met hun specifieke energiebesparing en wake-up time (hoe dieper in slaap – hoe langer het duurt om de controller volledig terug wakker te maken)
- L1: Zoek op de Arduino website naar instructies die de microcontroller in sleepmode kunnen brengen en experimenteer hiermee. Het gebruik van een mA of een uA meter is hierbij verplicht. Als resultaat geef je een minimaal 3 programma's af – zonder sleepmode, zelfde functionaliteit met sleepmode 1 en nogmaals met sleepmode 2 en een tabel waarin het stroomverbruik van de 3 programma's vergeleken wordt.

N EEprom

Het Eeprom geheugen van onze Arduino Leonardo kan gebruikt worden om data of instellingen op te slaan die we ook na een power-down van de µC nog willen bewaren.

- N1: Zoek op de Arduino website de instructies op die te maken hebben met het Eeprom geheugen en schrijf een programma waarin je dit geheugen zinvol gebruikt (met zowel lees als schrijfinstructies)

O Communicatieprotocol

- O1: Voor deze opgave moet je uw bordje verbinden met dat van uw gebuur met verschillende draadjes (bepaal zelf hoeveel je er nodig hebt). Aan het bordje van de zender hangt een potmeter. Met deze potmeter moet de led op het bordje van de ontvanger gedimd kunnen worden. Bedenk zelf een serieel communicatieprotocol om dit klaar te krijgen.
- O2: Als uitbreiding op vorige opgave moet dit in twee richtingen kunnen, dus vanaf nu moet ook de ontvanger met een potmeter aan zijn/haar bordje de led op het andere bordje kunnen dimmen.
- O3: Gebruik handleiding I-DIG LICHTSLUIS om deze verbinding draadloos en nu dus via IR licht te maken.