Les 8 23 april 2019   arrays

Nogmaals arrays, maar nu met wat meer diepgang.
In zijn eenvoudigste vorm een array is een lijst van karakters, getallen, objecten, etc.
De datatypen in deze lijst is voor elk item gelijk! (Zie vorige verslag/les -7-)
Je kan een lijst van booleans maken, bv 5 lampen aan of uit true of false: **bool lampenaan[5];**

Net als bij alle variabelen, kan je ook hier aan elke variabele een waarde toekennen.

```
bool lampenaan[5] = { false, false, false, false, false };
```

De lijst staat tussen **{**accolades**}** de onderlinge data is gescheiden door komma's **,**

```
void setup() {
  Serial.begin(115200);              // set the speed for the serial monitor:
  bool lampenaan[5] = { false, false, false, false, false }; // lampen uit
  LampenAan[2] = true;               // Zet de 3de lamp AAN
  LampenAan[4] = false;              // Zet de 5de lamp UIT
  if(LampenAan[1]==true) {           // controleer of de 2de lamp aan staat
    Serial.println("Tweede lamp staat AAN!");
  }
 if(LampenAan[0]) {                  // Kijk of de eerste lamp aan staat
    Serial.println("Eerste lamp staat AAN!");
  }
 if(!LampenAan[3]) {                 // Als de 4de lamp uit is, zet dan de 3de lamp aan
    Serial.println("4de lamp was niet aan, 3de lamp werd dus aangezet!");
    LampenAan[2] = true;
  }
}

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

Iets dergelijks hadden we ook al in les -7- met Hans en zijn neefjes Bram en Max.
Vergelijkingen zijn booleaans. Omdat de lampen ook booleaans worden aan/uit gezet, kan de vergelijking een stuk eenvoudiger: **if(!LampenAan[3]) {**
Arrays beginnen ALTIJD met -0- te tellen, dus de 4e lamp wordt aangegeven met lamp[3].

De waarde van een array is niet de data zelf, maar de geheugen locatie. (op pos -0- + index)
Deze index is de pointer (aanwijzer) van de eerste locatie van het array die dus op 0 begint.

Andere variabelen gebruiken meerdere geheugenposities. Omdat het array alleen één byte per keer aanwijst, en telkens één byte opschuift, ontstaan er problemen bij getallen.
Een karakter en byte neemt één byte in beslag, daar werkt het dus prima.
Gebruikt een data item meerdere bytes, moet er een kunstgreep worden toegepast.
Een **int** heeft bijvoorbeeld 2 bytes nodig, daar begint het gedonder al.

Dit is als volgt opgelost: geheugen locatie + (**index nummer** * 2 bytes)
Het **index nummer** voor het eerste element is **nul**, dus de berekening wordt:
geheugen locatie + (0 * 2 bytes) Vermenigvuldigen met nul levert altijd nul op.
Dus de uitkomst wordt: geheugen locatie + 0
Hierdoor komt voor het eerste element netjes de correcte geheugen locatie: positie -0-.

Voor het tweede element (index = 1) van de int array geldt:
geheugen locatie + (index nummer * 2 bytes) geheugen locatie + (1 * 2 bytes)
Dus: geheugen locatie + 2   Dit gebeurd allemaal op de achtergrond, dus kan je vergeten.

Het geeft wel aan, waarom een array met -0- begint te tellen!

Arrays met meerdere dimensies

Je kan in de X-as bewegen, maar ook haaks daarop, in de Y-as en zelfs dwars daarop, de Z-as.
Horizontaal noemen we 'regel' (=row) verticaal heet 'kolom' (=colom)
Hierbij is + naar rechts of naar boven en - de tegenovergestelde richting, omlaag en naar links.
Wij kunnen ons in 3 dimensies bewegen, maar rekenkundig zijn er eindeloos veel dimensies.
De benodigde hoeveelheid geheugen neemt kwadratisch toe, voor je het weet kom je te kort!
Twee dimensionaal kan je voorstellen als een vel 'ruitjes papier' 3 dimensionaal een blokkendoos gevuld met kleine blokjes waarvan elk blokje een 'doosje' met data is.
De hoeveelheid geheugen is dan ene derde macht
Één rij van 10 vakjes = 10 geheugenplaatsen
Een vel van 10 bij 10 'ruitjes' kost al 100 geheugenplaatsen
Een blokkendoos van 10x10x10 blokjes neemt 1000 geheugenplaatsen in beslag.
Dus bij 4 dimensies komt deze structuur op 10.000 geheugenplaatsen uit.
Fysiek kan je een vierde dimensie niet voorstellen, maar dat is in de rekenkunde niet nodig.

Een 2D array geeft je aan met twee indexen: variabele[index1, index2]
Een (één dimensionaal) array is een 'setje' data.
Een twee dimensionaal array is een setje van datasetjes, dus een setje in een setje
We krijgen dan zo iets als: datatype naam = { { setje1 }, { setje2 }, {setje3} };
Hierbij is zo'n setje dus zoiets als: { 1,2,3 }
Dat ziet er als volgt uit:

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

Dit mag je ook allemaal achter elkaar schrijven, maar dan wordt het heel onoverzichtelijk!
Hieronder een voorbeeld met 12 lampen. Er zijn 3 kamers met elk 4 lampen

```
#define kamers 3                    // er zijn drie kamers  0-1-2
#define lampenPerKamer 4            // er zijn per kamer 4 lampen 0-1-2-3
                                    // #define is een andere manier van declareren géén ;
void setup() {
  Serial.begin(115200);             // de communicatie snelheid
  boolean LampenAan[kamers][lampenPerKamer]={ // data type variabele naam, [row, colom
    {true,true,true,true},          // kamer 0 alle lampen aan
    {false,false,false,false},      // kamer 1 alle lampen uit
    {true,false,true,false}};       // kamer 2 twee aan, twee uit
// De booleaanse variabele: LampenAan  heeft twee dimensies  [kamers] en [lampenperkamer]
// Daarna worden de schakel situaties (aan of uit) per kamer aangegeven
  for(int kamer=0; kamer<=kamers-1; kamer++) {          // Tel de KAMERS
    for(int lamp=0; lamp<=lampenPerKamer-1; lamp++){    //Tel de LAMPEN perKAMER
      Serial.print("Kamer ");         // tekst 'kamer'
      Serial.print(kamer);            // nummer van de kamer
      Serial.print(" Lamp ");         // tekst 'lamp'
      Serial.print(lamp);             // status van de lamp
      if(LampenAan[kamer][lamp]) {    // als de lamp 'aan' staat
        Serial.println(" is AAN");    // print de tekst 'is aan'
      }
      else
      {
        Serial.println(" is UIT");
      }
    }                                 // einde lamp loop

    Serial.println();                 // lege regel tussen kamers
```

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

De uitvoer is:

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

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

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

In 't kort: we hebben 3 kamers, met elk 4 lampen, dus: 3 lijsten, met elk 4 items op de lijst!
Zie hoe de "#define" is gebruikt om een constante te definiëren! ➔Bij de array definitie
➔In de "for"-loops
In de "for"-loop is de betreffende constante gebruikt maar verminderd met 1.
Dat is gedaan omdat mensen bij -1- beginnen te tellen 1,2,3
Arrays beginnen bij -0- te tellen 0,1,2
Vandaar dus de "-1" om op de telling uit te komen die we die voor arrays moeten gebruiken.
Kies ook zinvolle namen voor de variabelen en constanten dat is duidelijker.

-=-=-=-=-

De huiswerkopdracht:

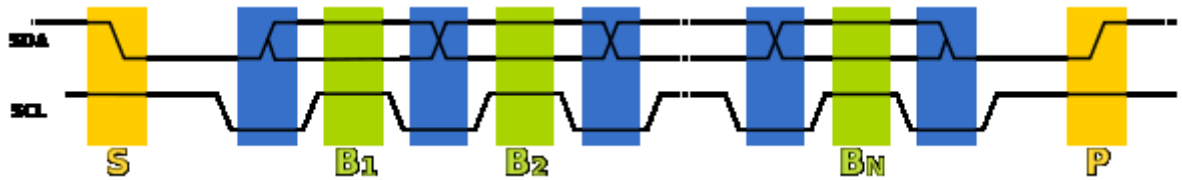
Maak iets waarbij strings op het scherm en/of LCD kan worden geplaatst, al dan niet met I^2C.
De communicatie tussen Arduino en PC is eigenlijk al gedaan, maar wat is I^2C?

I^2C Geschiedenis

De I^2C-bus werd in 1979 door Philips ontwikkeld en in 1982 gepatenteerd.
Soms spreekt men van *Two-Wire Interface* (TWI) wanneer men het I^2C-protocol bedoelt.
De communicatie gaat op lage snelheid, over korte verbindingen meestal binnen één apparaat.

Werking

I^2C werkt op basis van twee buslijnen, namelijk SDA (serial data) en SCL (serial clock).
Over de SDA-lijn wordt de data- en over de SCL-lijn wordt het kloksignaal verzonden.
In het onderstaande timingdiagram wordt verduidelijkt hoe de SDA en SCL samenwerken:



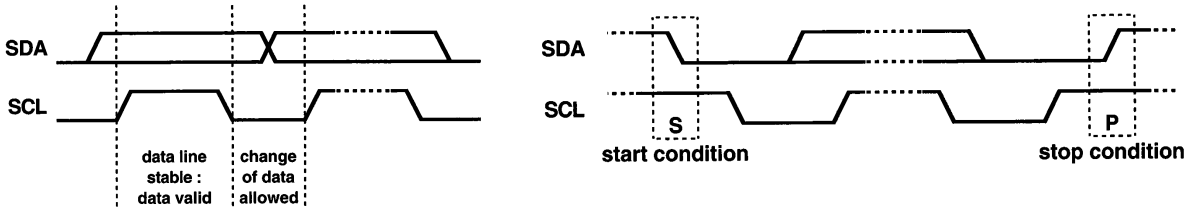
De werking van I^2C dataoverdracht:

1. Data verzenden wordt geïnitieerd met een STARTbit (S) die de SDA het signaal geeft om omlaag getrokken te worden, terwijl de SCL hoog blijft.
2. SDA zet de eerste databit gelijk, terwijl SCL laag gehouden wordt (gedurende de blauwe tijdsbalk.). De data wordt ontvangen als SCL naar omhoog gaat (groen).
3. Als de overdracht compleet is wordt een STOPbit (P) verzonden door de datalijn vrij te geven en deze zo in staat te stellen om omhoog getrokken te worden, terwijl SCL continu hoog gehouden wordt.
4. Ten einde valse detecties te voorkomen wordt het niveau van de SDA veranderd op de dalende flank (overgang van hoog naar laag) van SCL. Het uitlezen gebeurt op de stijgende flank (de overgang van laag naar hoog) van SCL.

Er kan maar één master actief zijn, het protocol bepaald wie de actieve master is.
Elk apparaat heeft een uniek eigen adres, dat hardware matig is vastgelegd.
De master geeft een oproep aan de bus, dat kan alleen als de bus vrij is.
Het apparaat dat door de master wordt aangesproken, geadresseerd, is de slave.

Alle uitgangen vormen een wired AND functie. Dus open collector/drain!
Daarom is er één pull-up weerstand nodig per buslijn. (SDA en SCL) voor alle apparaten samen. Bij een vrije bus, zijn alle datalijnen -1-

Alle ingangen van de apparaten vormen een wired OR functie.
Als beide lijnen minimaal 4,7 µs hoog zijn is de bus vrij.

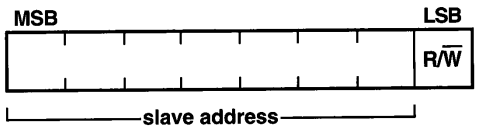


Als de clock hoog is, mag de data **NIET** veranderen.
Alleen bij **starten en stoppen** veranderd de data tijdens een hoge clock.

Adressering was oorspronkelijk 7-bits, maar kan ook 10-bits. (wij gebruiken 7-bits adressen)
Om beide tegelijkertijd mogelijk te maken is de 10-bits modus wat geforceerd.
Dat forceren is een moeilijke bezigheid, wordt daarom weinig toegepast.

7-bits adressering

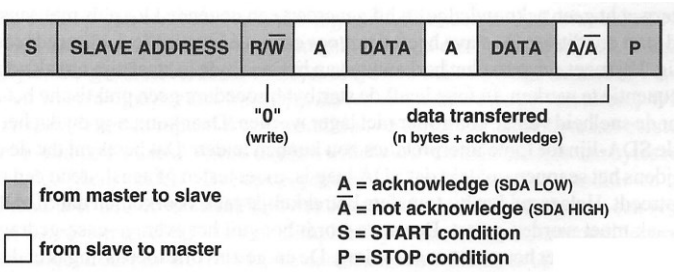
De bus wordt geclaimd door een start-conditie op de lijn te zetten. (Neergaande dataflank bij hoge clock) Daarna volgt het byte dat het adres van de gewenste slave bevat. De eerste 7 bits vormen het adres, het 8e bit is **-1-** om te **lezen** of **-0-** bij **schrijven**



Alle slaves lezen het adres, de gekozen slave gaat verder, de anderen haken af.
De geadresseerde slave reageert met een acknowledege bit.
Gebeurd dit niet, dan is het een ongeldig adres, of de slave is kapot.
Tijdens dit adresseren kunnen langzame slaves clockstreching uitvoeren. (= tijdrekken)
Het slave-adres is (hardware)matig vastgelegd.

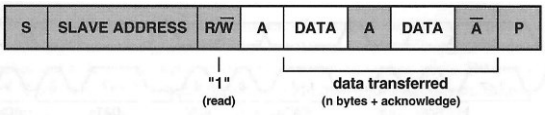
Een bijzonder adres is: 0000 000 0 7 bits 0 en één bit 0 = schrijven
Dit is broadcast, waarbij ALLE slaves worden geadresseerd!

Van master naar slave

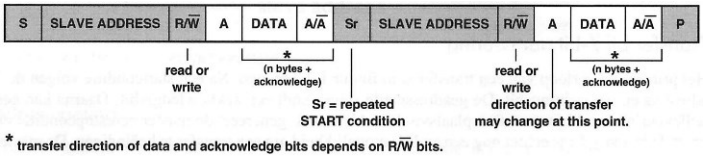


De master kan stoppen zoals hierboven omschreven, maar ook stoppen door een nieuw startbit met startconditie geven aan dezelfde slave, maar dan een andere transmissie richting, maar ook een andere slave activeren.

Van slave naar master



Heen en weer



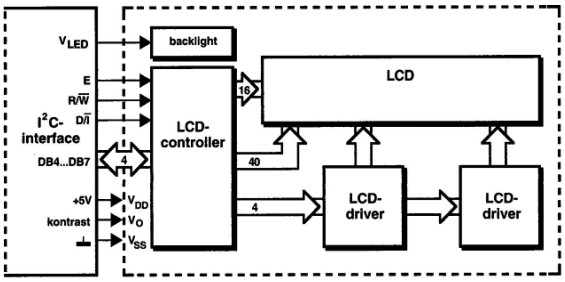
Bij repeated start wordt de startbyte procedure NIET toegepast.

Stap voor stap

-1- Zodra de bus vrij is (SCL en SDA hoog) kan de master de bus beleggen door een startconditie (SDA neergaande flank tijdens hoge CLK) te genereren.
-2- Het eerste byte dat verzonden wordt bevat het 7-bits slave adres met het R/W bit. (0 = schrijven 1 = lezen)
-3- Is de slave aanwezig, dan genereerd hij ACK puls. (= een 0)
-4- Was de R/W puls -0- dan stuurt de master data naar de slave tot hij geen ACK pulsen meer ontvangt, of totdat alle data verzonden is.
-5- Was de R/W puls -1- dan genereerd de master klokpulsen waarin de slave zijn data kan verzenden na ieder ontvangen byte genereerd de master een ACK puls. Dit gaat door totdat de master geen ACK pulsen meer geeft.
-6- Tot slot geeft de master de bus weer vrij door een stopconditie te genereren of een nieuwe startconditie etc. (= neergaande flank)

De **PCF8574p** met LCD (universele chip)
Er worden 4 data bits gebruikt: 4-5-6-7.
4 stuurlijnen: R/W, RS, Enable, V_{LED}.

Met RS (register select) wordt aangegeven of het een instructie voor de LCD controller is, of het weer te geven karakter.
Met V_{LED} word de backlight aan/uit gezet.

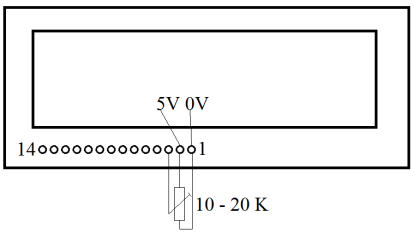


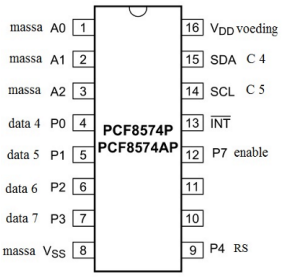
Het interrupt ontstaat als er iets op de **ingang** van de chip veranderd.
Als uitgang kunnen ze enkele micro ampères leveren, maar elke uitgang kan max 25 mA naar massa opnemen. De 8574 is er in twee varianten: 8574 met basisadres 0100 xxx r/w (dec 64 hex 40) waarbij xxx de in te stellen laatste 3 adresbitjes zijn.
(64 is schrijven, 65 is lezen. Een volgende chip kan 66=schrijven en 67=lezen)
Zo zijn er 8 adressen = 4 keer lezen, en 4 keer schrijven = 4 chips van dit type.

Wil je meer van deze chips op dezelfde adresbus, dan is er de 8574A met basisadres 0111 xxx r/w (dec 112 hex 70) met wéér 4 mogelijkheden.

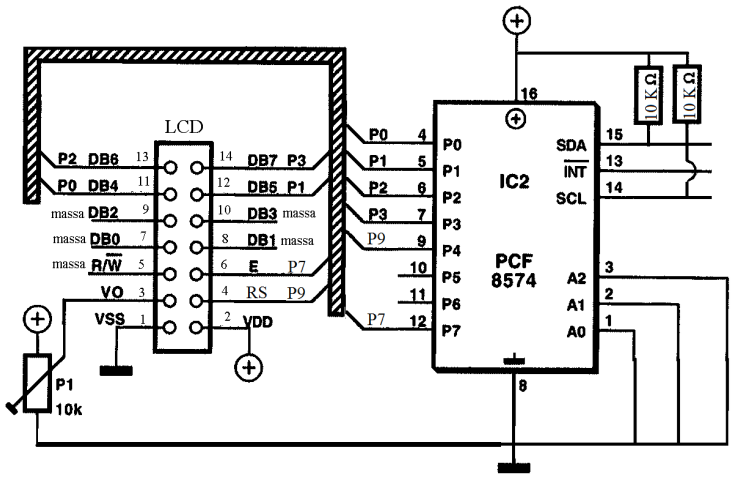
Dit LCD (DMC162007N-B*) heeft 4 datalijnen en 2 stuurlijnen (RS en E)
R/W wordt aan massa gelegd, V_{led} is er niet, geen backlight.

-1- massa 0V Vss
-2- + 5V (max 7V) Vcc logica
-3- + 5V (max 13V) Vee LCD zelf
-4- RS register select
-5- R/W H = read L = write
-6- Enable (geen interne pull-up)
-7- data 0 (8 bitsmode)
-8- data 1 (8 bitsmode)
-9- data 2 (8 bitsmode)
-10- data 3 (8 bitsmode)
-11- data 4 (4/8 bitsmode) 0V op pinnen: 1- 5 - 7 - 8 - 9 - 10
-12- data 5 (4/8 bitsmode) 5V op pin 2 Potm op pin 3
-13- data 6 (4/8 bitsmode) pin 4 & 6 register & enable
-14- data 7 (4/8 bitsmode) datapinnen 11-12-13-14





Hieronder deze I²C chip met aansluitingen zoals ik die op mijn LCD heb aangesloten.
SDA en SCL zijn voorzien van een pull-up weerstand van 10 K Ω.



Meer info over I²C :
http://www.i2c-bus.org/i2c-bus/
Zonder de gereserveerde adressen blijven er 112 (7 bit's) adressen beschikbaar.

De PCF8574p is een universele I²C chip link naar de datasheet:
https://www.nxp.com/docs/en/data.../PCF8574_PCF8574A.pdf

Er is er ook een 16-bits variant, de MCP23017, datasheet vind je hier:
https://researchdesignlab.com/projects/RDL_IO%20expander.pdf

dan is er nog het door Louis gebruikte LCDisplay: zonder I²C
https://www.banggood.com/1Pc-1602-Character-LCD-Display-Module-Blue-Backlight-p-978160.html?rmmds=search&cur_warehouse=UK

met I²C
https://www.banggood.com/IIC-I2C-2004-204-20-x-4-Character-LCD-Display-Module-Blue-p-908616.html?rmmds=search&cur_warehouse=UK

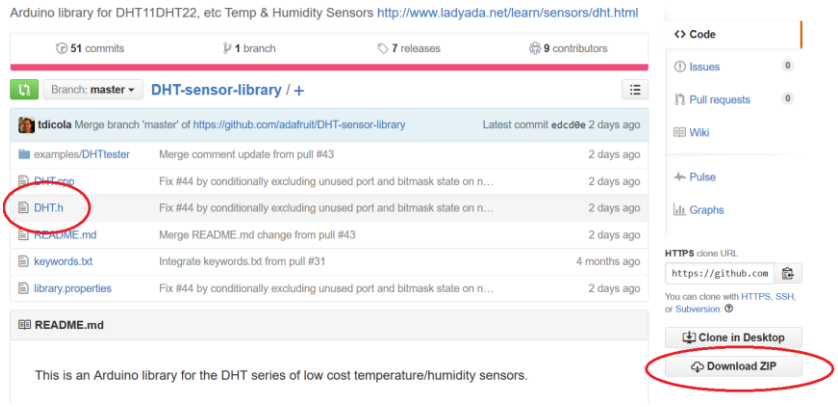
Een los I²C printje:
https://www.banggood.com/LCD1602-Adapter-I2CIICTWI-Serial-Interface-Module-Board-For-Arduino-p-1056942.html?rmmds=search&cur_warehouse=CN

Hoe je hiermee om moet gaan, zie je hier: https://www.youtube.com/watch?v=q9YC_GVHy5A

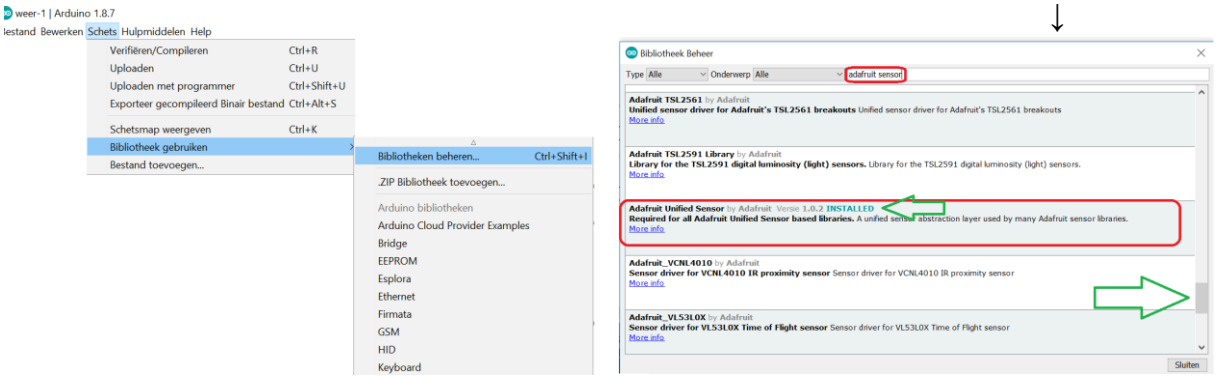
Library invoegen:

Die hebben we al eerder gedaan met het installeren van het weerstation januari 2019
Klik op: https://www.arduino.cc/en/guide/libraries voor meer info.

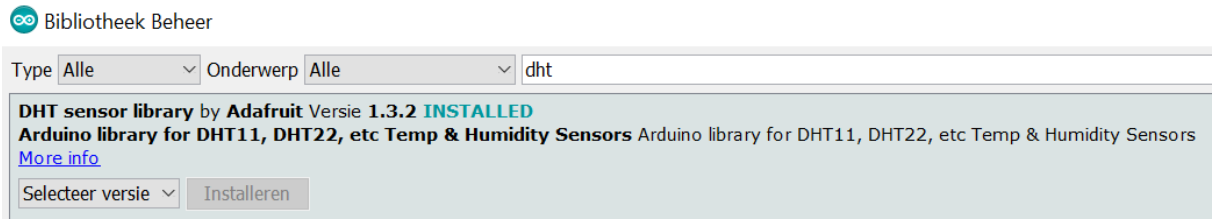
Even een herhaling: Op https://github.com/adafruit/dht-sensor-library vind je de bibliotheken.
Kies voor **DHT.h** en download het ZIP bestand



Vervolgens open je het Arduino programma en klik achtereenvolgens op:
#include "DHT.h" // de bibliotheek voor DHT sensoren
Om die te kunnen gebruiken, open je in schets bibliotheek beheren. Je krijgt dan dit beeld:



Hierin vul je eerst dat je zoekt naar een algemene **Adafruit sensor**. (bovenste rode kadertje)
Je krijgt een overzicht van Adafruit sensoren, scroll helemaal naar beneden (zie scrollpijl) en installeer deze universele sensor: **Adafruit Unified Sensor**. (ik heb hem al geïnstalleerd)
Rechts onderaan staat "installeren" klik daarop! De library wordt geinstalleerd.
Daarna zoek je (weer op de plek van het kleine rode kadertje) de DHT sensor, vul in: **DHT**

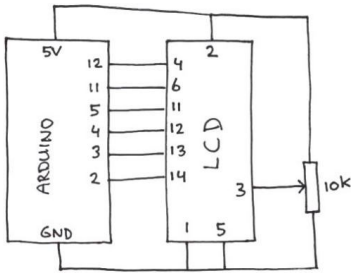


Ook deze heb ik al geïnstalleerd. Klik vervolgens rechts onderaan op **sluiten**.

Om de LCD te kunnen gebruiken moet je de LiquidCrystal library gebruiken:
#include <LiquidCrystal.h>.
Zoek via de Arduino IDE Help→Referentie→Libraries de LiquidCrystal library op.
Na installeren de Arduino IDE opnieuw opstarten

6.8 Liquid Crystal Display

Wil je tekst en gegevens zien zonder de Arduino via een USB kabel met de computer te verbinden, dan kan je een LCD aansluiten. Arduino's LiquidCrystal library ondersteunt displays die werken volgens het Hitachi HD44780 protocol. Er zijn veel displays die werken met dit protocol. Je herkent ze aan de 16 aansluitingen. De meest voorkomende displays zijn die met 2 regels van 16 karakters en die met 4 regels van 20 karakters. Het display heeft 6 digitale signalen van de Arduino nodig. Je moet dus 6 digitale uitgangen van de Arduino gebruiken om de LCD aan te sluiten.
Sluit een LCD met twee regels van 16 karakters aan volgens het volgende schema:



Met de potmeter kun je het contrast regelen. De meeste displays hebben ook LED's ingebouwd voor de achtergrondverlichting. Deze vind je meestal op pinnen 15 en 16.

Elk karakter van het display bestaat uit 7 rijen van 5 pixels.
Je kunt ook je eigen karakters maken, zoals een smiley of een rechthoekje.
Je kunt maximaal 8 eigen karakters definiëren.

Een eigen karakter maken:

```
#include <LiquidCrystal.h>       // gebruik de LCD library
LiquidCrystal lcd(12,11,5,4,3,2); // LCD RS, E, D4, D5, D6, D7 op pootjes 12,11,5,4,3,2
byte hokje[] = { B11111, B10001, B10001, B10001, B10001, B10001, B11111};
                // hierboven wordt elke LCD regel voor het hokje gedefinieerd  5x7 pixels
void setup(){
  lcd.createChar(1, hokje);          // hier wordt karakter -1- gecreëerd (max 8 stuks)
  lcd.begin(16,2);                   // hier wordt het gebruikte display aangegeven
  lcd.write(1);                      // hier wordt het zojuist gemaakte karakter weergegeven
}
void loop(){}                        // hier wordt even niet gebruikt wel verplicht aanwezig.
```

Een analoge waarde van een sensor kun je op het display weergeven met een rij blokjes van 0 tot 16 blokjes.
Je kunt de resolutie ook verhogen naar 48 streepjes.
Hiervoor moet je eerst drie karaktertjes maken: met één streepje, met twee streepjes en met drie streepjes.

Een LCD kan je rechtstreeks op de Arduino aansluiten, je hebt dan 6 pootjes nodig
Wanneer je hetzelfde LCD dmv I^2C aansluit, dan heb je aan twee pootjes voldoende!
Bovendien, als je eenmaal I^2C aan hebt staan, kunnen er veel meer zaken over die twee pootjes worden geregeld.
Hieronder twee manieren van LCD aansturen, rechtstreeks en dmv I^2C

```
// LCD rechtstreeks aangesloten
#include <LiquidCrystal.h>                    // include the library code:
const int rs = 2, en = 3, d4 = 4, d5 = 5, d6 = 6, d7 = 7;     // pootjes benoemen
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);                    // pootjes naar subroutine.

void setup() {
  lcd.begin(16, 2);                           // een LCD 2 regels van 16 karakters
  lcd.print("hallo wereld!");                 // print deze tekst naar het LCD.
}
void loop() {
  lcd.setCursor(0, 1);                        // zet de cursor op pos 0 regel 1
  lcd.print(millis() / 1000);                 // het aantal seconden sinds de laatste reset
}
```

-=-=-=-=-=-=-=-=-

```
//LCD via I2C aangesloten
#include <LiquidCrystal.h>                    // include the library code:
const int rs = 2, en = 3, d4 = 4, d5 = 5, d6 = 6, d7 = 7;     // pootjes benoemen
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);                    // pootjes naar subroutine.

void setup() {
  lcd.begin(16, 2);                           // een LCD 2 regels van 16 karakters
  lcd.print("hallo wereld!");                 // print deze tekst naar het LCD.
}
void loop() {
  lcd.setCursor(0, 1);                        // zet de cursor op pos 0 regel 1
  lcd.print(millis() / 1000);                 // het aantal seconden sinds de laatste reset
}
```

-=-=-=-=-=-=-=-=-

```
// I2C scanner                Er zijn vele sensoren en actuatoren die dmv I2C verbonden kunnen worden.
#include <Wire.h>          // Soms heb je een 'ding' waarvan je het adres niet weet.
void setup(){              // Dat zoek je dan op met onderstaand programma:
  Wire.begin();
  Serial.begin(115200);
  Serial.println("\nI2C Scanner");
}
void loop() {
  byte error, address;
  int nDevices;
  Serial.println("Scanning...");
  nDevices = 0;
  for (address = 1; address < 127; address++ )  { // The I2C_scanner uses the return value of the
    Wire.beginTransmission(address);              // Write endTransmission to see if a device did
   error = Wire.endTransmission();                // acknowledge to the address.
   if (error == 0) {                              // er is geen fout gevonden
     Serial.print("I2C device found at address 0x");
     if (address < 16) Serial.print("0");
     Serial.print(address, HEX);
     Serial.println("  !");
     nDevices++;
   } else if (error == 4) {                       // foutcode
     Serial.print("Unknow error at address 0x");
     if (address < 16) Serial.print("0");
     Serial.println(address, HEX);
   }
  }
  if (nDevices == 0)Serial.println("No I2C devices found\n");
  else Serial.println("done\n");
  delay(5000);          // wait 5 seconds for next scan
}
```

Vreemd, deze laatste if - else gebruikt geen accolades en toch werkt het programma.

Groeten, Dré

de huiswerkoplossing:

Louis heeft onderstaand twee regelig LCD gebruikt met I^2C communicatie

```
#include <Wire.h>                                   // nodig voor communicatie met I2C
#include <LiquidCrystal_I2C.h>    // bevat de functies om het LCD te bedienen
LiquidCrystal_I2C lcd(0x27, 16, 2);
   /* Definieer het object met de naam lcd in van het type LiquidCrystal_I2C
      0x27 is het hexadecimale I2C adres van het LCD (afhankelijk van merk en type).
      16,2 wil zeggen een display met 2 regels van elk 16 karakters. */
char karakter = 0;                                  // via Serial gelezen karakter
String tekst = "";                                  // tekst string voor de lichtkrant
String voorNaTekst = "   *  *  *  *  *   ";        // voorloop en naloop tekst
int lengte;                                         // lengte van 'tekst'
boolean tekstIngevoerd = false;                     // vlag voor of er al tekst is ingevoerd
int plaats;                                         // plaats in de string 'tekst'
unsigned long vorigTijdstip;                        // nodig voor wachtlus
byte pijl[] = {B00000, B00010, B00100, B01000, B11111, B01000, B00100, B00010};
                                                    // definitie van een speciaal karakter (pijl naar links)
void setup() {
  Serial.begin(115200);
  Serial.println("Typ de tekst voor de lichtkrant en sluit af met -Enter-.");
  lcd.init();                                       // initialiseer het LCD scherm met de naam lcd.
  lcd.backlight();                                  // zet de achtergrondverlichting aan.
  lcd.createChar( 0, pijl);                         // het zelfgemaakte karakter 'pijl' krijgt volgnummer 0.
  lcd.setCursor(3, 1);                              // zet de cursor op plaats 3 van regel 1
  for (int i = 1; i < 5; i++) {
    lcd.write(0);                                   // plaats het zelfgemaakte karakter met volgnummer 0
(pijl)
    lcd.print("  ");                                // print twee spaties
  }
}
void loop() {
  if (Serial.available() > 0) {                     // is ereen karakter beschikbaar?
    karakter = Serial.read();                       // lees het karakter
    if (karakter != '\n') {                         // indien geen line feed
      if (tekstIngevoerd == true) {
        tekstIngevoerd = false;
        tekst = "";
      }
      tekst += karakter;
    }
    else {                                          // einde van de ingevoerde tekst
      tekstIngevoerd = true;
      tekst = voorNaTekst + tekst;                  // clear the string for reuse
      tekst += voorNaTekst;
      lengte = tekst.length();
      Serial.println(tekst);
      Serial.println(lengte);
    }
  }
  if (tekstIngevoerd == true) {
    for (plaats = 0; plaats < lengte - 16; plaats++) {
      while (millis() - vorigTijdstip < 300) {}
      vorigTijdstip = millis();
      lcd.setCursor(0, 0);
      lcd.print(tekst.substring(plaats, plaats + 16));
      Serial.println(tekst.substring(plaats, plaats + 16));
    }
  }
}
```