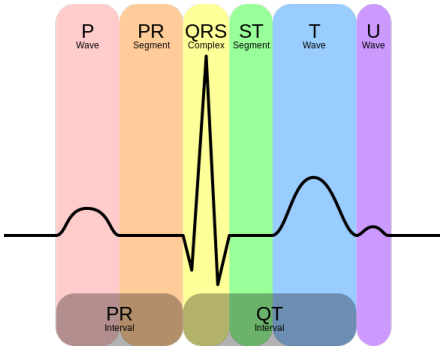
Gør-det-selv EKG måler



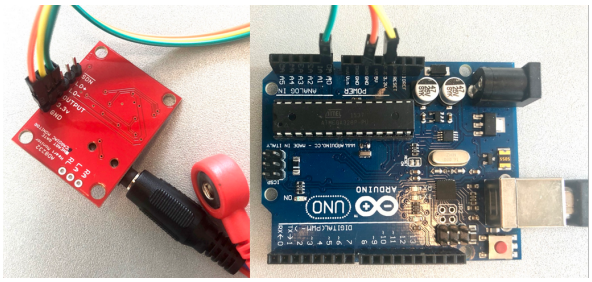
Når hjertet slår, sker det i et velkoordineret mønster. Synkroniseringen heraf foregår via elektriske signaler, som man kan måle via elektroder på kroppen, via elektrokardiogrammer eller EKG-målinger, som vist th.

I denne øvelse, skal I programmere et program, som skal kunne indlæse og afbilde EKG signaler via en lille mikroprocesser; en såkaldt Arduino Uno, samt kunne beregne og visualisere pulsen ved at analysere signalet.

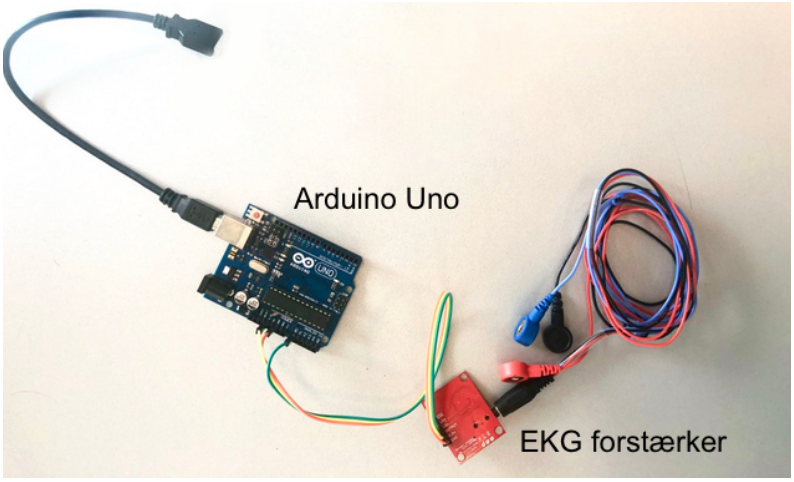
Materialer

Til denne opgave skal I bruge:

- En Arduino Uno og tilhørende USB kabel
- En EKG forstærker (AD8232)
  - EKG forstærkeren skal forbindes til Arduinoen:
    - AD8232 **3.3v** skal forbindes til **+3.3V** på Arduinoen
    - AD8232 **GND** skal forbindes til **GND** på Arduinoen (der er 3 på Arduinoen, det er ligegyldigt hvilken I forbinder til)
    - AD8232 **Output** skal forbindes til **A0** på Arduinoen (analog pin 0)



- Et bundt med 3 Harwin han-hun ledninger
- Et EKG kabel med et mini-jack stik i den ene ende og et rødt/blåt/sort stik i den anden ende



Software

Start med at hente og installere Arduino 1.8.10 IDE programmeringsmiljøet på nedenstående adresse (findes både til Mac og Windows).

https://www.arduino.cc/en/Main/Software

Hardware forbindelser

Forbind dernæst EKG modtageren med Arduinoen, som vist på side 1.

Start af Arduino IDE

Første gang programmeringsmiljøet startes ser det ud som th.

Som det ses, er der to hoved områder; et kaldet **setup**, hvor man skriver kode, som kun skal køres en gang til at starte med (typisk opsætning) samt et såkaldt **loop**, som er en betegnelse for kode, der skal køres igen og igen - faktisk al den tid Arduinoen er tændt! Foran både setup og loop er der et ord ved navn **void**. Dette indikerer, at funktionerne ikke skal returnere nogen værdier. Tuborgklammerne indikerer hvor koden, som hører til funktionerne starter og slutter.

Til at programmere EKG optageren, skal der bruges såkaldt **variabler** til at gemme tal fra gennemkørsel til gennemkørsel. Disse skal defineres øverst, altså ovenover **void setup()** linjen.

Opgave 1: Den første EKG måling

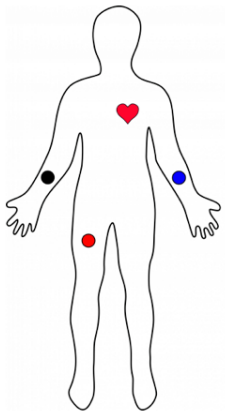
NB: I må IKKE have computeren tilsluttet 230V mens I måler EKG - det er potentielt livsfarligt! Den må KUN køre på batteri!!!

Til at begynde med, programmeres blot et simpelt program til at optage og vise signalet. Dette gøres ved at definere en variabel øverst, som skal indeholde den nuværende værdi af EKG-signalet. At indlæse værdier digitalt kaldes at *sample*. Når man definerer variabler i Arduino IDE, skal man starte med at angive hvilken *type* variablen er. I dette tilfælde skal den kunne indeholde heltal fra 0 til 1024, hvilket kræver variabler af typen *integer*, som forkortes *int*. Ydermere skal variablen *initialiseres*, dvs. sættes til en start værdi. Alt dette gøres ved at skrive `int sample = 0;` på første linje, dvs. over void setup () linjen. HUSK det afsluttende semikolon (;) - det markerer, at I er færdige med den nuværende linje.

I setup funktionen, skal programmet nu kommunikere med Arduinoen. Dette gøres gennem en seriel forbindelse via USB stikket, som etableres med kommandoen **`Serial.begin(9600);`** hvor 9600 betegner hvor mange værdier, der maksimalt kan indlæses i sekundet. Indtast denne linje inde i setup() funktionen.

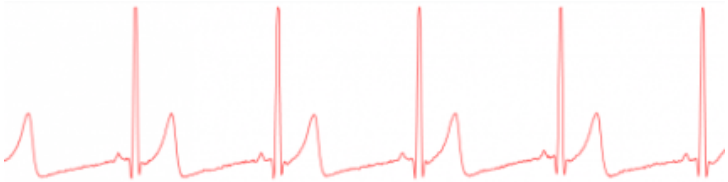
Nu skal værdierne indlæses. Dette gøres i loop() funktionen idet værdierne skal indlæses kontinuert. I indlæser værdier med kommandoen analogRead(port). Værdierne skal gemmes i den variabel I oprettede tidligere ved navn *sample*. Som bekendt, forbandt I EKG forstærkerens *Output* port med Arduionen via porten *A0*, hvilket indikeres via kommandoen **`sample = analogRead(A0);`**

Indsæt denne linje i loop() funktionen. Dernæst skal værdien udskrives, så vi kan aflæse den. Dette gøres via kommandoen **`Serial.println(sample);`** Indsæt denne linje under linjen I skrev lige før. Til slut skal programmet pause en lille smule mellem hver gang I kører det for at holde styr på hvor mange gange I *sampler* signalet pr. sekund, den såkaldte *samplingsfrekvens*. Denne angives med kommandoen *delay(millisekunder)*, der sætter programmet til at pause i det angivne antal millisekunder. Prøv i første omgang at sætte ventetiden til 20 millisekunder via kommandoen **`delay(20);`**



Prøv nu at påsætte EKG elektroder som vist til højre (sæt dem på håndled og ankler) og forbind dem til EKG forstærkereren med EKG kablet. på.

Prøv nu at køre programmet ved at trykke på Kør knappen øverst. Som udgangspunkt kan I ikke se noget, men hvis I vælger menuen *Værktøjer/ Tools* og derpå *Serial Plotter*, skulle I meget gerne få et EKG signal frem på skærmen, som ligner dette:



De høje takker kaldes *R-takkerne* og indikerer de tidspunkter, hvor hovedkamrene i hjertet trækker sig sammen og sender blod ud i kroppen - det er det I kan mærke, hvis I tager pulsen ved at trykke på håndleddet eller halsen.

Tidligere satte I *delay* værdien til 20 millisekunder. Det betyder med andre ord, at jeres program indsamler værdier med 20 millisekunders mellemrum, også kaldet *samplingsfrekvensen*. Normalt angiver man dette som hvor mange værdier man indsamler pr. sekund, som måles i Hertz (Hz) - i dette tilfælde altså 1000/20 millisekunder = 50 Hz. Prøv nu at justere denne til at måle med værdier 1 Hz, 10 Hz, 20 Hz, 50Hz og 100Hz, ved hver gang at regne ud hvor stort et delay i millisekunder I skal indsætte og køre programmet.

Hvor lav en samplingsfrekvens kan I bruge og stadig se R-takkerne tydeligt?

Næste skridt bliver at programmere kode til at finde disse R-takker for at beregne pulsen. Hertil skal bruges en grænseværdi (threshold), som adskiller R-takkerne fra resten af EKG signalet (se figuren herunder).

Aflæs en værdi, som kun R-takkerne overstiger (*threshold*), som I skal bruge senere.

Hvor stort skal jeres threshold være?

Opgave 2: Puls aflæsning

Nu skal I programmere kode til at kunne finde R-takkerne, beregne tiden mellem hver R-tak og bruge denne til at beregne pulsen. Hertil skal bruges en række variabler, der skal defineres under *sample* variablen, som I indsatte allerøverst i programmet tidligere. Variablen threshold skal indeholde den talværdi I fandt tidligere som grænseværdi, der tydeligt adskilte R-takker fra resten af EKG-signalet (erstat ***XXX*** med grænseværdien):

```
int sample = 0;
int BPM = 0;
int beat_old = 0;
float beats[10];
int beatIndex=0;
float threshold = XXX;
boolean belowThreshold = true;
```

Int variabler kender I fra tidligere. *Float* er betegnelsen for en variabeltype, som kan indeholde decimaltal mens *boolean* er en variabel type, som kan være sand eller falsk. Variablen ved navn *beats* ser lidt speciel ud - firkantklammerne angiver, at denne variabel skal kunne indeholde 10 værdier - den bruges til at udregne gennemsnittet af tiden mellem hvert hjerteslag over de sidste 10 hjerteslag. Dette kaldes også et *array* med plads til 10 værdier. Hvad variablerne ellers skal bruges til gennemgås herunder.

Til at udregne pulsen, skal I nu programmere en funktion, som I kan kalde, når I rammer en R-tak. Funktionen skal fuldstændig se ud som *setup* og *loop* funktionerne, der allerede er tilstede i jeres program, dvs. således:

```
void calculateBPM () {
}
```

Funktionen skal bygges op, så den konstant udregner tiden mellem det nuværende og sidste hjerteslag, der bruges til at beregne pulsen.

Først skal tiden for det nuværende hjerteslag gemmes. Dette gøres i en midlertidig variabel inde i calculateBPM funktionen. Til forskel fra de globale variabler, der står øverst i programmet, kan man også oprette midlertidige variabler, som kun er tilstede mens man

kører underfunktionen. Indsæt en linje i calculateBPM funktionen mellem tuborg-klammerne, der gemmer tiden for det nuværende hjerteslag i millisekunder i *beat_new*:

```
int beat_new = millis();
```

Millis er en funktion, der aflæser det nuværende tidspunkt i millisekunder. Beregn dernæst forskellen mellem det forrige hjerteslag og det nuværende og gem tidspunktet for det nuværende hjerteslag i beat_old, så denne tid kan bruges næste gang hjertet slår:

```
int diff = beat_new - beat_old;
beat_old = beat_new;
```

Forskellen *(diff)* på det gamle (*beat_old*) og nye hjerteslag (*beat_new*) kan nu bruges til at udregne pulsen. Denne er defineret som antal slag pr. minut. Et minut er som bekendt 60 sekunder - i millisekunder bliver dette 60000. Skal pulsen dermed beregnes kan man gøres det ved at dividere 60000 med differencen mellem *beat_new* og *beat_old*, som blev gemt i variablen *diff* herover:

```
float currentBPM = 60000 / diff;
```

Forskellen mellem hjerteslag vil hele tiden variere lidt idet kroppen konstant justerer hjerterytmen baseret på det aktuelle behov for ilt. Rejser I jer op til pulsen stige lidt, løber I vil den også stige, sover I vil den falde osv. For at kunne beregne og angive en mere stabil puls skal I derfor gemme de sidste 10 puls værdier i arrayet *beats*. For at holde styr på hvor det skal gemmes bruges en tæller (*beatIndex*), som angiver hvilken plads den nuværende puls skal gemmes på. Hvis den blot konstant blev talt op, ville I på et tidspunkt komme over de 10 pladser der er til rådighed - for at undgå det bruges en såkaldt *modulus* operation (skrives med % tegn) - den angiver blot, at I skal dividere tælleren med 10 og gemme det, som ikke går op i 10. Et eksempel: 23 % 10 vil således give 3 mens 107 % 10 vil give 7 osv.

```
beats[beatIndex] = currentBPM;
beatIndex = (beatIndex + 1) % 10;
```

Nu skal middelværdien af alle pulsværdierne udregnes. Som I givetvis ved, udregnes middelværdier ved at lægge dem alle sammen (*sum*) og dividere med antallet, som er 10 i dette tilfælde. Start derfor med at definere en *sum* variabel af typen *float* (den skal kunne indeholde decimaltal), dvs. I skal indsætte en linje bygget op som denne (erstat selv variabeltype, variabelnavn og standardværdi med de rigtige værdier):

```
Variabeltype variabelnavn = standardværdi;
```

Den letteste måde at lægge alle tallene sammen på er at bruge en såkaldt *for* løkke. Denne starter ved en angiven værdi, tæller den op med en konstant værdi hver gang for loopet kører (typisk tælles der op med 1; en såkaldt *inkrementering*, der kan skrives med ++ efter et variabel navn) og slutter når en given værdi er nået. Her bruges *i* til at indeholde tællerværdien, som inkrementeres hver gang for loopet køres, og slutter efter 10 gennemkørsler (også kaldet *iterationer*):

```
for (int i = 0; i < 10; i++) {
}
```

I hver *loop iteration* skal en variabel summeres op med værdien på nuværende plads af *arrayet beats*, der indeholder de 10 sidste puls-værdier.

Inde i for loopet skal værdien på pladsen *i* lægges til *sum* variablen:

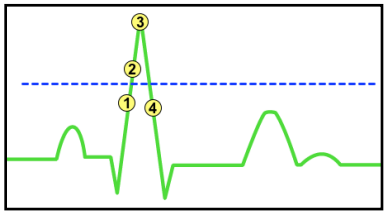
```
sum = sum + beats[i];
```

Efter loopet kan den gennemsnitlige puls udregnes ved at dividere *sum* med 10 (NB: husk at gøre det *efter* for loopet - det skal kun gøres en gang og ikke hver gang for loopet køres):

```
BPM = int(sum / 10);
```

Funktionen er imidlertid blot en hjælpefunktion - den skal også kaldes løbende inde i selve hoved-loop funktionen fra tidligere - men hvornår? Det letteste vil være at kalde den hver gang *sample* værdien bliver større end det *threshold* I definerede tidligere, der angav den unikke værdi, som kun R-takkerne oversteg.

MEN, EKG signalet er ikke kun over denne grænseværdi i et enkelt punkt - den vil forblive over værdien i et stykke tid og altså flere samples. *calculateBPM* skal kun kaldes en gang for hver R-tak - så what to do? Det letteste er at definere hvornår signalet igen kommer under grænseværdien - og det er det som variablen *belowThreshold* bruges til.



Koden I skal skrive lige om lidt består af to sammenligninger i første *if* statement:

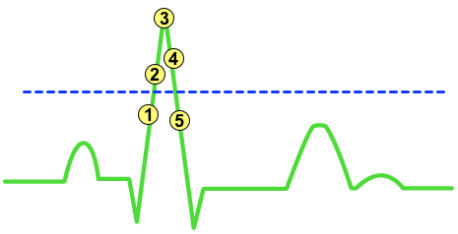
1. Er nuværende punkt (*sample*) over grænseværdien (*threshold*)?
2. Er variablen *belowThreshold* sand?

Hvis begge sammenligninger er sande, kaldes *calculateBPM*.

I et separat *if* statement tjekkes det om *sample* værdien er under *threshold* - hvis det er tilfældet sættes *belowThreshold* til *true* og ellers sættes *belowThreshold* til *false*.

I punkt 1 i figuren herover er værdien ikke over *threshold*, så der kaldes *calculateBPM* ikke. I punkt 2 er værdien over, så vi tjekker også den anden sammenligning. Idet variablen *belowThreshold* initialiseres til *true* og punktet før punkt 1 også var under *threshold*, vil den også være opfyldt ➠ vi kører *calculateBPM*.

Det betyder, at selvom punkt 3 opfylder første sammenligning (større end *threshold*), opfylder det ikke den anden fordi *belowThreshold* nu er *false*. I punkt 4 fejler første sammenligning igen, da værdien nu er under *threshold* - og det bliver ved med at være tilfældet indtil næste R-tak hvor det hele starter forfra.



Vil jeres kode/algoritme også virke i situationen til venstre, hvor I har tre punkter (punkt 2-4) over threshold i QRS komplekset?

Når vi skal sammenligne værdier, bruges indenfor kodeverdenen ofte såkaldt *if*-sætninger, der bruges til at køre noget kode, hvis betingelserne er opfyldt og noget andet kode hvis de ikke er. Opbygningen af *if*-sætninger ligner meget *funktioner* og *for* loops:

```
if (sammenligning) {
     kode som skal køres, hvis sammenligningen er sand;
} else {
     kode som skal køres, hvis sammenligningen er falsk;
}
```

Sammenligningen skal være af typen *boolean,* der netop kan indeholde værdien *sand* eller *falsk*. Når man kombinerer flere variabler, som alle skal være sande gøres det med en såkalt *AND* operator, der skrives med to amperesander: &&.

Indsæt disse to if-sætninger lige under **`sample = analogRead(A0)`** kommandoen fra tidligere:

```
if (sample > threshold && belowThreshold == true) {
  calculateBPM();
}

if (sample < threshold) {
  belowThreshold = true;
} else {
  belowThreshold = false;
}
```

Nu er I næsten klar til at køre programmet igen - imidlertid skal vi kunne se hvad pulsen bliver. Den letteste måde at tjekke det på, er ved at udskrive pulsen til seriel porten, fuldstændig som I tidligere gjorde med **`Serial.println(sample);`** kommandoen. MEN denne kommando skriver data ud mange gange i sekundet, mens jeres puls kun udregnes ca. en gang i sekundet - derfor vil *sample* værdierne fuldstændig overdøve pulsen. Start derfor med at *udkommentere* denne kommando, dvs. fortælle programmet, at den skal ignorere denne linje. Dette gøres ved at indsætte to skråstreger foran kommandoen:

```
//Serial.println(sample);
```

Indsæt derpå en **`Serial.println`** kommando i *calculateBPM* funktionen, på sidste linje, der udskriver *BPM* variablen istedet.

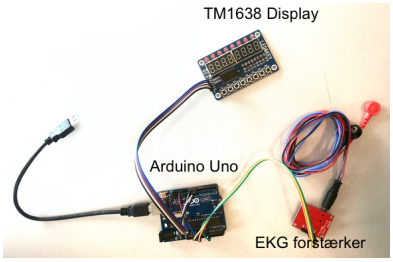
Kør derpå programmet, men luk nu *Serial Plotter* vinduet, og start istedet en *Serial Monitor* under *Værktøjer*/*Tools*. I skulle nu gerne få en konstant opdateret liste af puls værdier; ca. en i sekundet hvis altså jeres hvilepuls er omkring 60 slag i minuttet.

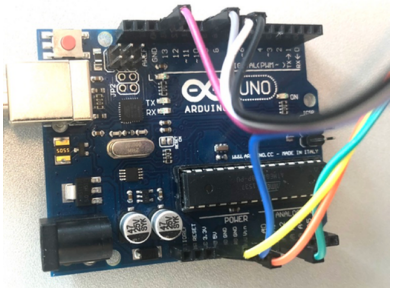
Opgave 3:Udskrivning af puls til display

Den netop programmerede måde at vise pulsen på virker - men den er ikke videre brugbar. For det første kan vi ikke længere se selve EKG signalet. For det andet får vi ikke noget overblik fordi puls værdierne konstant scroller henover skærmen. Derfor skal I nu forbinde TM1638 display modulet, som skal bruges til at vise puls værdierne.

Hardware:

- Et TM1638 display med LED-dioder, trykknapper og 7-segment display
- Et bundt med 5 Harwin han-hun ledninger
- Displayet skal forbindes til Arduinoen:
  - TM1638 **VCC** skal forbindes til **+5V** på Arduinoen
  - TM1638 **GND** skal forbindes til **GND** på Arduinoen (der er 3 grounds på Arduinoen, det er ligegyldigt hvilken I forbinder til)
  - TM1638 **STB** (strobe) skal forbindes til **10** på Arduinoen
  - TM1638 **CLK** (clock) skal forbindes til **9** på Arduinoen
  - TM1638 **DIO** (data in/out) skal forbindes til **8** på Arduinoen





Kode:

Start med at fjerne kommenteringen foran *Serial.prinln(sample)* kommandoen ved at slette de to skråstreger. Udkommenter nu *Serial.println(BPM)* linjen i calculateBPM funktionen.

Forbind displayet som angivet på side 1. For at kunne tilgå modulet, skal I først installere et såkaldt *library*, som er en samling af hjælpefunktioner, der gør kommunikationen med eksterne enheder langt lettere.

Gå ind i menuen **Sketch -> Include Library -> Manage Libraries…** og skriv *TM1638* i søgefeltet øverst. Dette skulle gerne give jer *TM1638lite*. Klik på den og vælg *Install*. Tryk derpå på *Luk*.

For at kunne bruge dette *library*, skal I fortælle programmet at I gerne vil inkludere det. Dette gøres lettest via menuen **Sketch -> Include Library -> TM1638lite**. Dette vil automatisk indsætte denne linje allerøverst i jeres program, som netop fortæller Arduino, at I gerne vil kunne tilgå displayet:

```
#include <TM1638lite.h>
```

For at kunne bruge displayet, skal det først defineres under globalt variabler. Dette gøres med kommandoen `TM1638lite tmdisp(10,9,8);`

TM1638lite er variabel typen, *tmdisp* er variabel navnet vi kan bruge til at tilgå displayet, mens *10,9,8* er de porte I forbandt displayet til Arduinoen med tidligere.

Først skal displayet nulstilles i *setup* funktionen. Indsæt kommandoen **`tmdisp.reset();`** under *Serial.begin* kommandoen.

Dernæst skal pulsen udskrives - ideelt set skulle vi gerne få udskrevet både den nuværende puls, som er gemt i *currentBPM* variablen, og middelværdien af de sidste 10 puls slag, som er gemt i *BPM* variablen. Dette kan gøres ved at definere en *String* variabel, som simpelthen er en variabel, der kan indeholde tekst, og sætte den til at indeholde de to variabler adskilt af 2-3 mellemrum:

```
String BPMtxt = String(BPM) + "    " + int(currentBPM);
```

Som det kan ses, skal BPM *type castes*, dvs. konverteres fra heltal (*int*) til tekst (*String*), hvilket blot gøres ved at skrive *String(BPM)*. Tilsvarende udskrives *currentBPM* som et heltal hvilket gøres ved at *typecaste* fra *float* til *int*. Indsæt denne linje som den sidste i *calculateBPM* funktionen. Derpå skal tekststrengen *BPMtxt* udskrives:

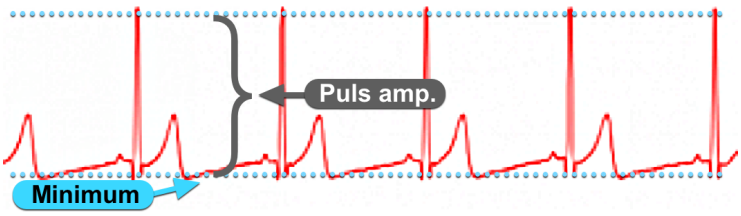
```
tmdisp.displayText(BPMtxt);
```

Kør programmet og tjek om puls og middelværdi vises på TM1638 displayet.

Bonus opgave 4: Visning af puls med LED-dioder

Til slut skal vi programmere en funktion, som viser pulsen som via de LED-pærer, der er øverst på TM1638 display modulet. Hvis I tæller dem, vil I kunne se, at der er 8 i alt. For at kunne vise EKG signalet, skal vi med andre ord have omdannet værdierne, der vises i Serial Plotter vinduet, til at gå fra 0->8.

Start EKG målingen igen, og vis igen *Serial Plotter* vinduet.



Aflæs den mindste værdi (minimum) og aflæs hvor langt der er fra den mindste til den største værdi (puls amplituden).

Indsæt denne funktion nederst i programmet:

```
void ecgLEDs(float val) {
  for (int pos = 0; pos < 8; pos++) {
    if (val > pos+1) {
      tmdisp.setLED(pos,1);
    } else {
      tmdisp.setLED(pos,0);
    }
  }
}
```

Som det kan ses, skal funktionen kunne modtage en decimalværdi ved navn *val*. Denne værdi skal være et tal mellem 0 og 8 - hvis det er 8 skal alle 8 LED pærer lyse, er tallet 2.5 skal 2 pærer lyse og så fremdeles. For at tænde/slukke pærerne køres kommanden *tmdisp.setLED(pos,værdi)* - værdien er enten 1 eller 0 for at hhv. tænde/slukke LED pæren.

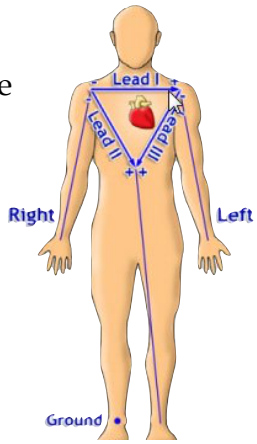
For at kunne tilgå hver enkelt LED bruges et *for loop* ligesom tidligere, der denne gang looper henover de 8 LED pærer. For hver pære tjekkes det hvorvidt *val* er over den nuværende position (*pos*) + 1 og hvis det er sandt tændes pæren - hvis ikke slukkes den.

Nu skal funktionen kaldes fra hoved loop funktionen. Indsæt denne linje under *Serial.println(sample);* linjen:

```
ecgLEDs(((sample-minimum)/puls amplitude)*8);
```

Dette omsætter sample værdien til et tal mellem 0 og 8. Kør nu programmet igen - hvis I har valgt minimum og puls amplituderne korrekt burde LED pærerne lyse fra 0 -> 8 i takt med EKG signalet. Hvis I ikke helt får slukket alle pærerne er jeres minimum værdi for lille - skru op for den. Hvis ikke I helt får tændt alle 8 pærer er jeres puls amplitude for lille - skru op for for den.

Bonus opgave 5: Sammenligning af 1., 2. og 3. afledninger

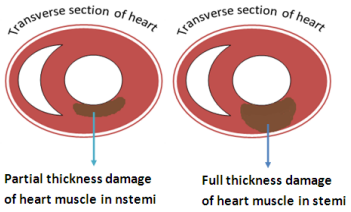


Som bekendt har vi i disse øvelser målt på 1. afledning, dvs. mellem højre og venstre arm, mens højre ben er brugt til at fjerne støj.

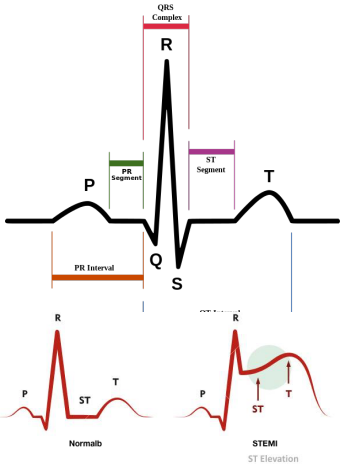
Prøv nu at gentage målingerne fra tidligere, men mål imellem højre arm og venstre ben (2. afledning) samt venstre arm og venstre ben (3. afledning).

Prøv at sammenholde udseendet ved hver afledning og diskuter hvorfor højden/udseendet af EKG signalet varierer afhængigt af målepunkterne.

Bonus opgave 6: ST elevations myokardieinfarkt



Når man får en blodprop i hjertet, er det af afgørende betydning af finde ud af om blodproppen har ramt hele hjertevæggen, eller kun en mindre del - det afgør hvorvidt man skal have en ballonudvidelse, eller blot blodfortyndende medicin, dvs. skal patienten køres til et centralsygehus, eller det nærmeste regionale sygehus?



For at afklare dette, måles et EKG i ambulancen, hvor der ses efter ST elevation - dvs. højden af segmentet mellem S og T takken sammenlignes med baseline, altså grundlinjen for EKG.

Prøv at skrive en *algoritme*, altså kode, der kan aflæse højden i ST-segmentet (typisk omkring 0.08-0.1 sekund efter R-takken) og sammenligne med TP-segmentet, altså linjen mellem T og P-takkerne. Hvis den er større end 0.1 mV er der tale om ST-elevation, altså et ST Elevations Myokardie Infarkt (STEMI), der skal behandles med ballonudvidelse, fx. på Aarhus Universitetshospital Skejby.