C++ Fokus på sproget #06

in C++ Fokus with 0 Comments

Funktioner med parametre

I afsnittet med variabler kom vi kort ind på begrebet 'scope'. Scope handler om det fænomen, at variabler har et anvendelsesområde, hvor de er kendte, og kun her kan de anvendes. Uden for scope eksisterer de ikke. Langt de fleste variabler er på denne måde lokale og det er god skik at begrænse anvendelsen af globale variabler, der jo ellers gør ens variabler kendte overalt i koden. En af grundene skal søges i, at det er lettere at læse koden, hvis hændelserne foregår lokalt. Nogle programmører undgår derfor helt at anvende globale variabler.

Men hvad gør man så, for ens funktioner skal jo typisk have noget input at arbejde med. Hvorledes udveksler funktioner oplysninger om variabler? Hvordan videregiver de værdier, som den næste funktion skal bruge for at kunne udføre sine rutiner og foretage sine beregninger? Den opgave har man løst ved at indrette funktioner med et lille vindue, hvor man 'udefra' kan vise dem en variabel og hvor de så at sige kan kigge ud mod verden og se, hvad der bliver vist i vinduet. Kigger vi på funktionen 'setup', som alle Arduino sketches skal indeholde, skal den erklæres således:

- void setup()

Erklæringen slutter med to parenteser, og det er netop disse to parenteser, der udgør funktionernes vindue mod omverdenen. void setup() gør ikke brug af vinduet, men det er der alligevel. Forestil dig, at du skal sælge din bil og du har en funktion, som kan beregne prisen. Den kunne f. eks. hedde beregnPrisenPaaMinBil og skal i sagens natur returnere et tal. Din bil er gammel og kan højst indbringe 65 535 kr. Funktionen kan derfor erklæres således:

- unsigned int calculateCarValue()

Men hov, den bliver jo nødt til at vide lidt om din bil før funktionen kan beregne. Den skal vide, hvor langt bilen har kørt. Den har brug for et input. Lad os fastsætte, at denne information oplyses som et tal, der angiver km-standen i tusinde km og at din bil højst kan have kørt 255 000 km og lad os kalde informationen for OdometerReading. Nu kommer erklæringen til at se således ud:

- unsigned int calculateCarValue(unsigned long OdometerReading)

Inde i parentesen kan man med andre ord skrive navnet på en variabel, hvori man kan lægge en værdi, som funktionen kan arbejde med. I vinduet viser vi vores variabel OdometerReading og funktionen kan indefra se, hvilken værdi, der ligger inden i OdometerReading. Vi siger, at funktionen har en parameter. Bemærk, at vores variabel OdometerReading IKKE kan anvendes af andre funktioner. Den er udelukkende til brug INDE i funktionen calculateCarValue. Andre funktioner har adgang til at lægge værdier ind i OdometerReading, med de kan IKKE selv bruge den. Med andre ord kan den kun bruges lokalt, – dens scope er den funktion, som den er parameter til. Den kan kun bruges *inde* i funktionen.

Men andre funktioner kan altså overføre en værdi til den.

Hvis du skal have et billede af, hvad der er på spil her, kan du tænke på de huse, der ved siden af hoveddøren har en lille boks muret ind. Boksen har en låge på ydersiden og en anden låge på indersiden. Så kan posten, mælkemanden eller avisbudet lægge noget i boksen og ejeren af huset kan åbne den indefra og tage sine varer. På samme måde kan man tænke om en parameter. Det er en boks, som man kan lægge noget i. Men sammenligningen med mælkeboksen holder selvfølgelig ikke 100%. Avisbudet kan jo komme tilbage og tage avisen med sig igen. Når man har kaldt en funktion med en parameter, er der ingen mulighed for at fortryde. Funktionen registrerer, hvilken værdi funktionen har og gør derefter sin ting.

Når jeg vil sætte min funktion i arbejde, foretager jeg et kald til den. Det kan jeg gøre på to måder. Jeg kan direkte lægge et km-tal i boksen, sådan her:

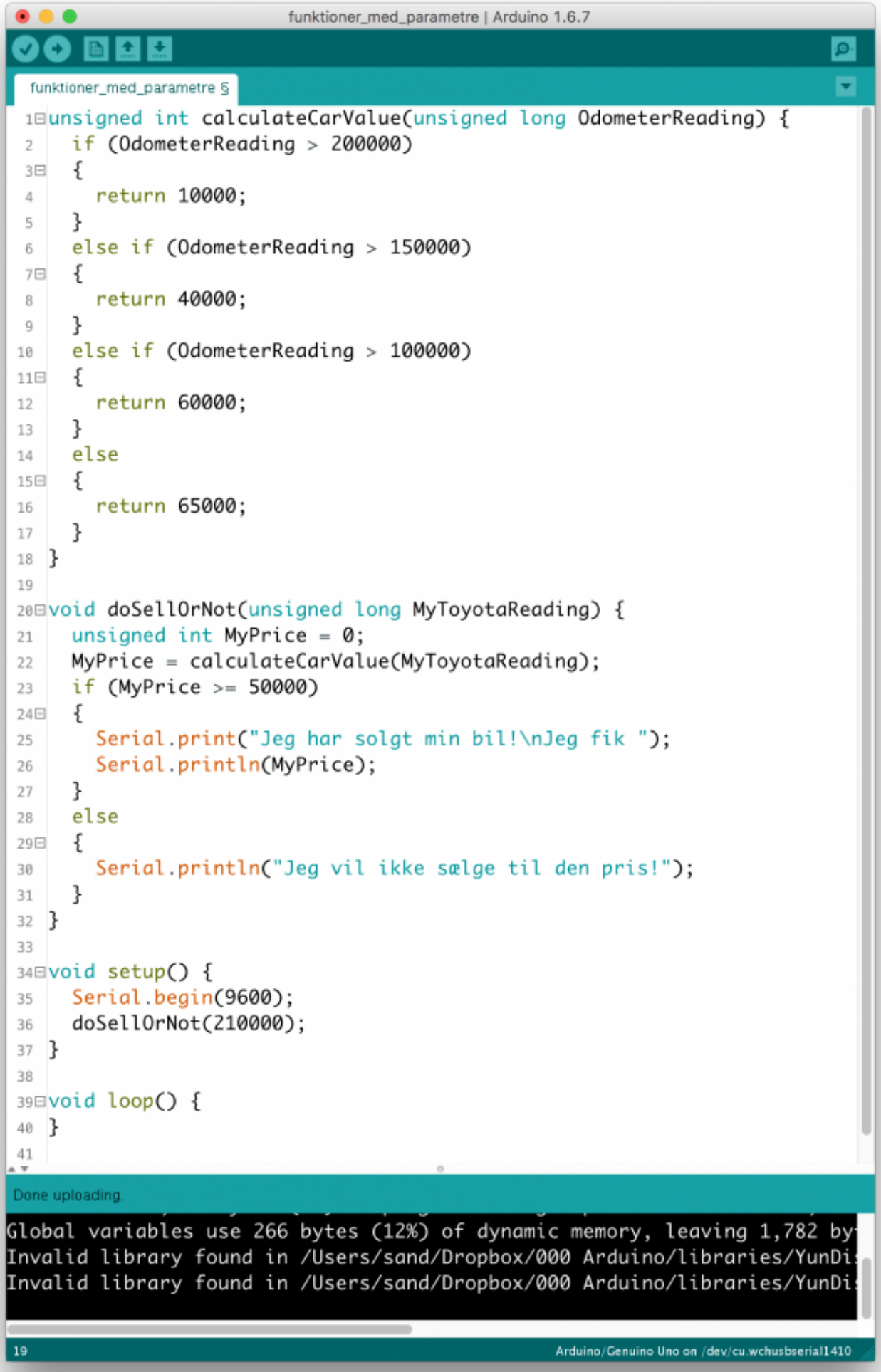
- calculateCarValue(125000);

Eller jeg kan lægge km-tallet ind i en variabel, som jeg så efterfølgende lægger ind i boksen. Jeg kunne f. eks. have en variabel som hedder ToyotaKm og så ville der et sted i min kode skulle stå:

- MyToyotaReading=125000;
- calculateCarValue(MyToyotaReading);

Det første tilfælde kaldes ‘hardcoding the value’. I det andet tilfælde overfører man en værdi fra en variabel til en anden og det er sådan man hyppigst arbejder med parametre. Man lader den modtagende funktion få et kig ind i en variabel, der hører til den funktion, hvor funktionskaldet kommer fra. På den måde kan man overføre værdier mellem variable, som hver for sig er lokale inde i hver sin funktion.

Koden kunne f. eks. komme til at således ud:


funktioner_med_parametre | Arduino 1.6.7
funktioner_med_parametre §
unsigned int calculateCarValue(unsigned long OdometerReading) {
if (OdometerReading > 200000)
{
return 10000;
}
else if (OdometerReading > 150000)
{
return 40000;
}
else if (OdometerReading > 100000)
{
return 60000;
}
else
{
return 65000;
}
}
void doSellOrNot(unsigned long MyToyotaReading) {
unsigned int MyPrice = 0;
MyPrice = calculateCarValue(MyToyotaReading);
if (MyPrice >= 50000)
{
Serial.print("Jeg har solgt min bil!\nJeg fik ");
Serial.println(MyPrice);
}
else
{
Serial.println("Jeg vil ikke sælge til den pris!");
}
}
void setup() {
Serial.begin(9600);
doSellOrNot(210000);
}
void loop() {
}
Done uploading.
Global variables use 266 bytes (12%) of dynamic memory, leaving 1,782 by
Invalid library found in /Users/sand/Dropbox/000 Arduino/libraries/YunDi
Invalid library found in /Users/sand/Dropbox/000 Arduino/libraries/YunDi
19
Arduino/Genuino Uno on /dev/cu.wchusbserial1410