Appendix 1 - Debugging

"Om debugging är processen att ta bort bugar, då måste programmering vara processen att lägga till dom."

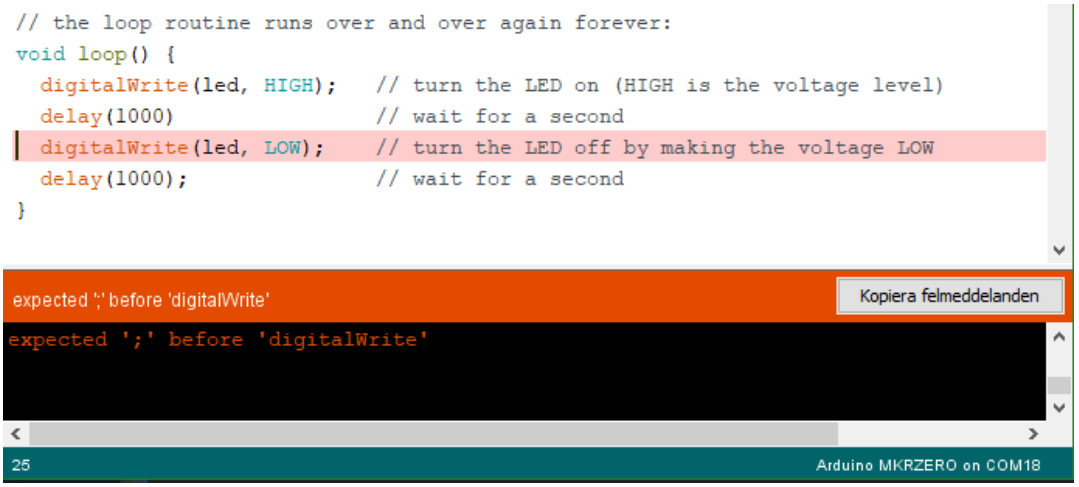
- E. W. Dijkstra

För att tycka om upplevelsen att programmera och för att bli en bra programmerare behöver man kunna hantera fel och felmeddelanden. Detta appendix är menat att ge instruktioner om det. Detta dokument är inte labb eftersom det inte passar någonstans i ordningen, utan är menat att användas vid behov under tiden de ordinarie labbarna görs.

Hur hanterar man ett fel eller felmeddelande? Generellt består processen av tre steg,

1. Identifiera problemet
2. Bestäm lösningen på problemet
3. Implementera lösningen

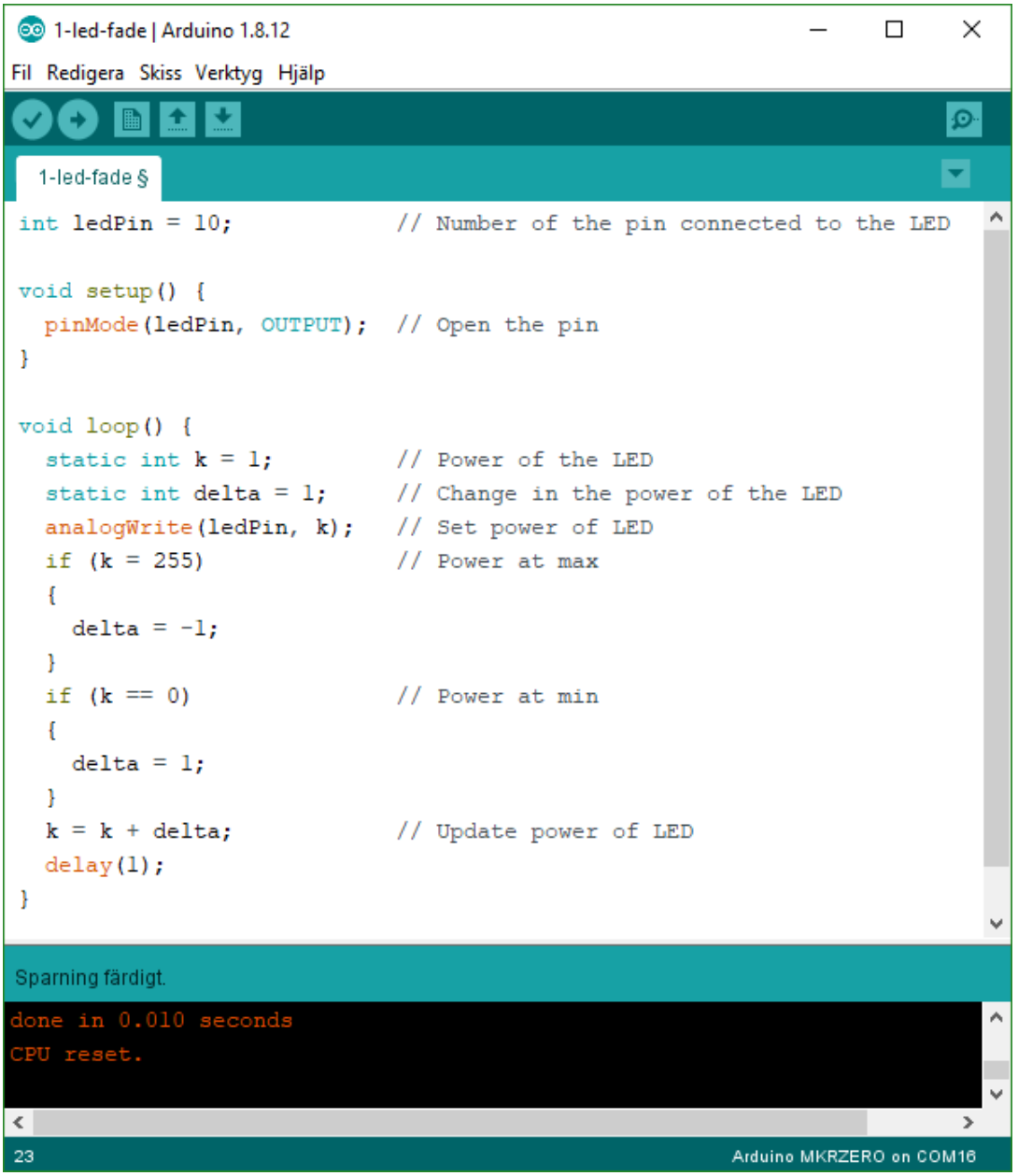
Men det är lättare sagt än gjort. Hur identifierar man ett problem? Ta detta som ett exempel.



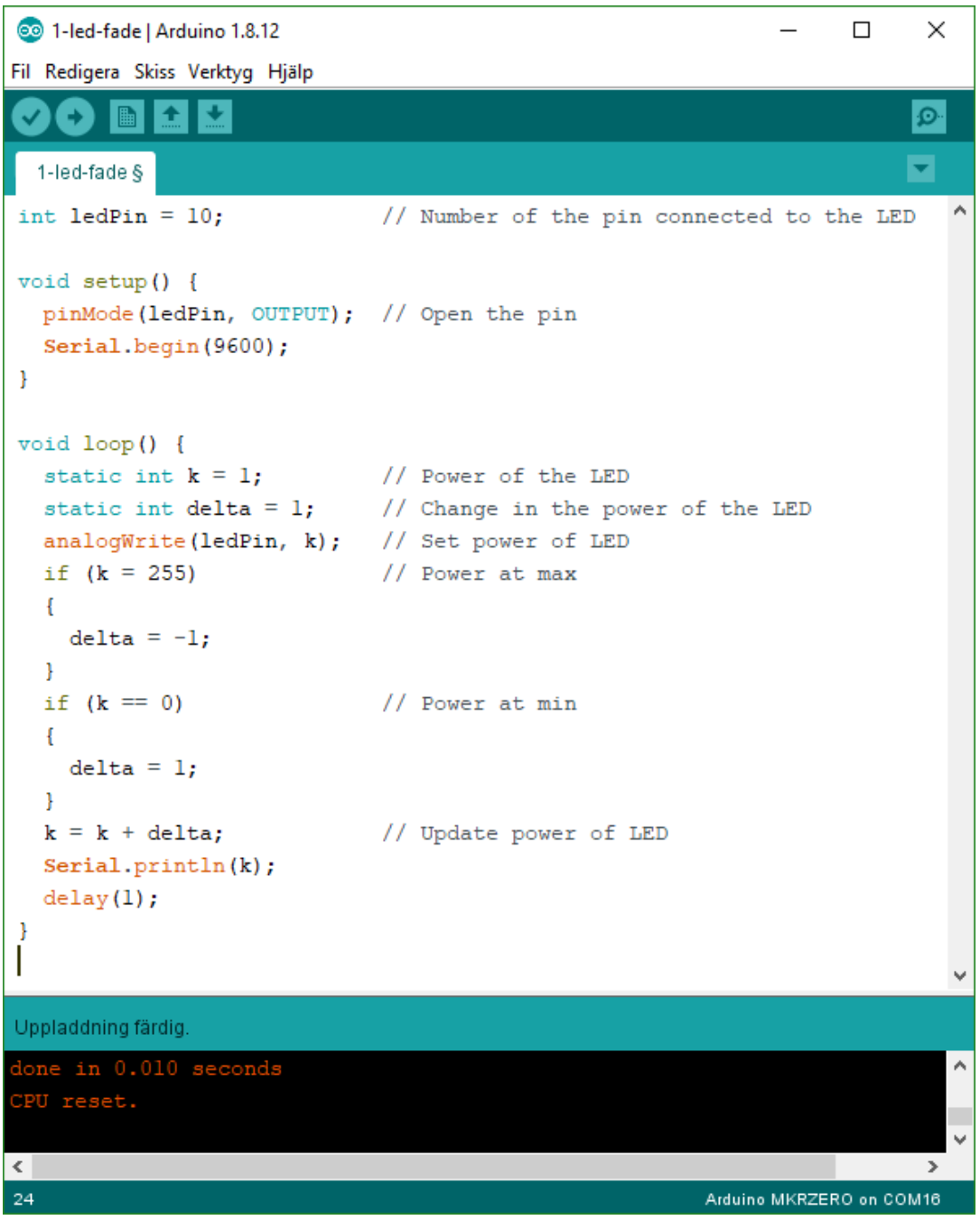
Felmeddelandet säger `expected ';' before 'digitalWrite'` och raden med `digitalWrite` är rödmarkerad. Detta exempel är ganska rättframt, programmet påstår att den förväntade sig en markering för att en rad är slut (;) innan funktionen `digitalWrite`, men fann ingen sådan. Då kollar man på raden ovanför den markerade och ser efter om där finns ett

E-post: cansat@au.se
Telefon: 070-000 90 56

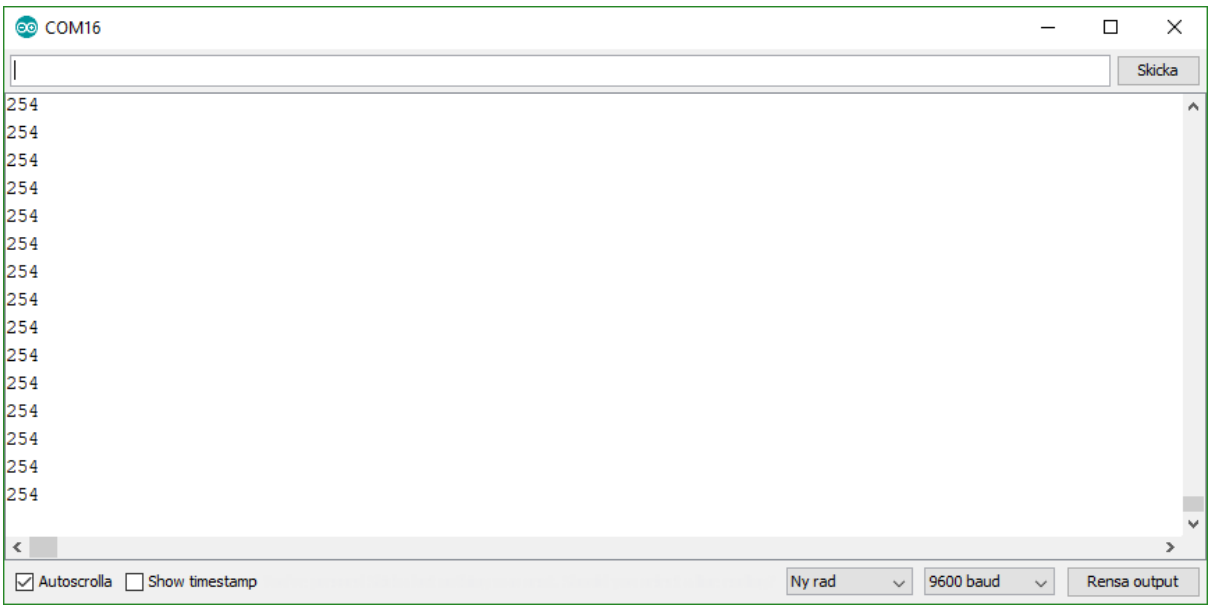
semikolon, och då ser man att det inte gör det. Problemet är att det fattas ett semikolon, lösningen är att skriva dit ett.



Denna är betydligt svårare. Koden ger inga felmeddelanden men lysdioden håller en konstant, hög, ljusstyrka istället för pulserna den ska ge. En bra strategi när programmet inte säger att något är fel är att få det att skriva ut mer data. Det görs med Serial monitorn. Genom att modifiera programmet enligt bilden nedan kan man se vad som händer.

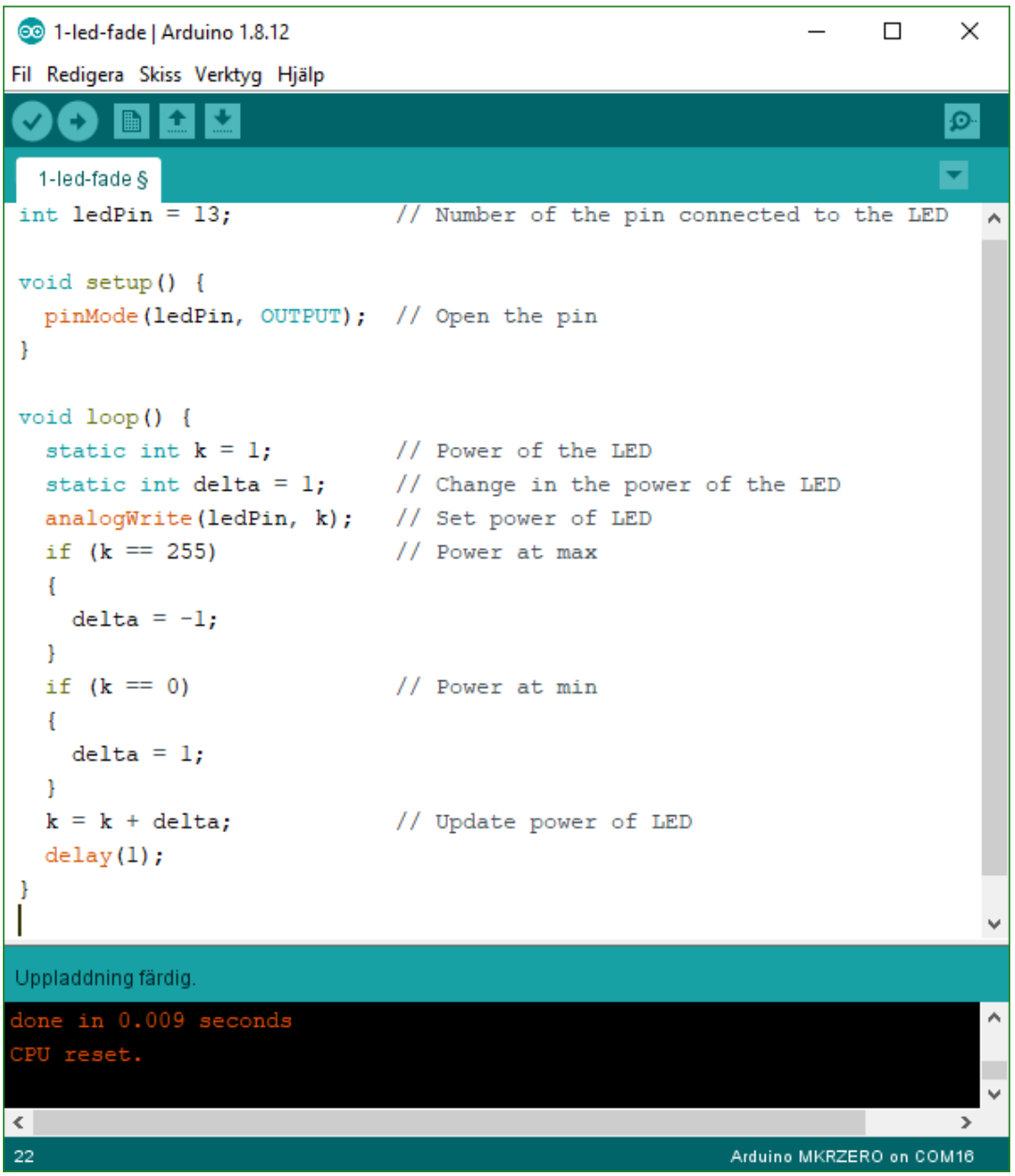


Nu kommer programmet tala om vilket värde `k` har varje gång det går igenom loopen. Genom att kolla i Serial monitor ser vi följande.

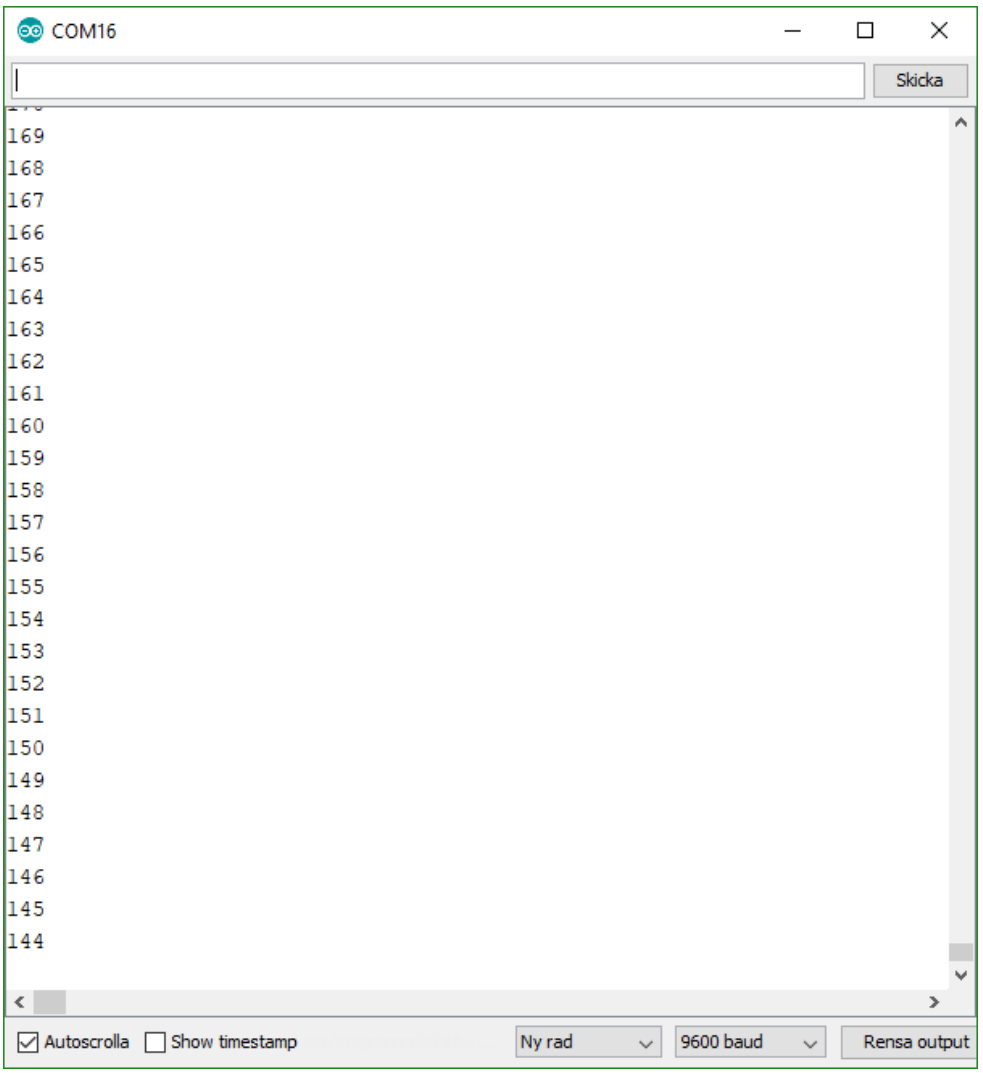


`k` har värdet 254 varje gång och det verkar inte ändras. Det sista som händer innan `k` skrivs ut är att beräkningen `k = k + delta;` görs, så `k` är 255 och `delta` är -1. Om vi fortsätter att kolla tidigare i programmet ser vi att om `delta` är -1 måste den ha fått det värdet i den första if-satsen. När vi tittar närmare på den if-satsen ser vi att kriteriet är fel. Det är ett enkelt likamedtecken istället för ett dubbelt.

Ett enkelt likamedtecken tillskriver ett värde, vilket (om det lyckas) är sant. Ett dubbelt likamedtecken är en boolesk operator som jämför två värden och blir sant om båda är lika. Lösningen är att skriva till det saknade likamedtecknet.



Nästa exempel har en anna typ av problem. Felet denna gång är att lysdioden inte pulserar utan blinkar. Den går direkt från maximal till minimal ljusstyrka utan något mellanting. Vi skriver in koden för Serial monitor igen, för att se vilka värden programmet har.



Resultatet visar att k ändrar värde som den ska. Därför tror vi att problemet inte ligger i koden utan i hårdvaran. En koll i dokumentationen för funktionen analogWrite[1] (vilket är den som ska skriva värdena till lysdioden) avslöjar att den måste använda en specifik pin, och vilka pins som fungerar beror på vilken Arduino man använder. Vi använder en MKR ZERO och kan därför välja på pin nr 0-8, 10, A3 eller A4. Koden men problemet använder pin nr 13 vilken inte är på listan. Problemet var att Arduinon inte kunde skriva ut värden med så små skillnader som vi bad den att göra på den pin-en. Lösningen var att byta pin till en som var med på listan.

[1] https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/

Från dessa tre exempel kan vi se en process för problemlösning. Den ser ut ungefär såhär

1. Läs och tolka eventuella felmeddelanden
2. Få programmet att skriva ut mer information
3. Läs dokumentationen
4. Googla problemet

Den fjärde punkten är inte något vi har pratat om än. Anledningen till att Arduino används i dessa labbar är att det är många andra som har använt Arduino tidigare. Dessa människor har antagligen redan haft alla problem vi kommer stöta på. Genom att klistra in ett felmeddelande i Google kommer ett svar nästan säkert dyka upp.