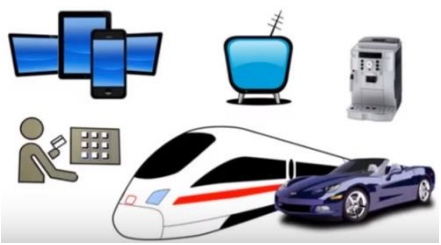
Om programmering



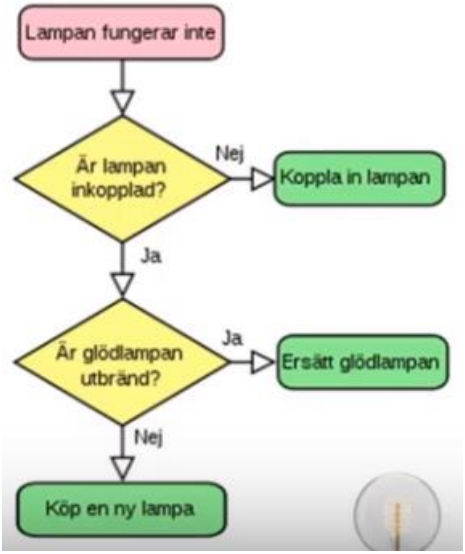
Världen vi lever i är full med prylar som är programmerade. De kallas för "intelligenta". Man pratar om *artificiell intelligens*. Men prylarna kan inte tänka själva. Någon har programmerat dem, närmare bestämt de elektroniska komponenterna i dem – små datorer. Det är de som styr all funktionalitet.

Programmering är ett av de mest spännande kapitlen i teknologihistorien. Inte bara därför att den har lagt grunden till den moderna IT-industrin och på gott och ont revolutionerat världen. Den har också bidragit till att förverkliga den urgamla mänskliga drömmen att förenkla mödosamma arbeten. Istället för att plåga sig instruerar man en maskin med idéer. Programmering realiserar önskemålet att låta datorn göra jobbet för att ha mer tid över för annat i livet.

När man tröttnat på att använda program som andra skrivit – maila, surfa eller lyssna på musik – är det dags att börja programmera själv. Det är roligare att köra en bil än att bara åka med. Det är kreativiteten och det fria skapandet som lockar. Med programmering kan du testa helt nya egna idéer.

*"Everyone in this country should learn how to program a computer. Because it teaches you how to **think**."*

Steve Jobs



Men hur programmerar man?

Egentligen gör vi det varje dag utan att vara medvetna om det. Är t.ex. en lampa trasig följer vi ungefär det som kan beskrivas med bilden till höger, ett s.k. *flödesschema*. I praktiken löser vi problemet att ersätta en trasig lampa genom att tänka och göra så utan att någonsin rita ett flödesschema. Flödesschemat illustrerar och dokumenterar dock *algoritmen*, dvs tillvägagångssättet för problemets lösning. När den en gång är ritad skulle den kunna användas av vem som helst som vill byta en

trasig lampa. Den blir en slags allmängiltig manual för just detta problem. Men ännu viktigare är att metodiken kan tas över till svårare problem.

Ett annat vardagligt exempel är matlagning. Vare sig vi använder ett recept ur en kokbok eller lagar efter känsla, följer vi en algoritm som dessutom – till skillnad från lampalgoritmen – även har en *input*, råvaror och en *output*, maträtten. Hårdvaran som hjälper oss är köket med alla sina instrument. Matreceptet är mjukvaran dvs programmet. Det är precis samma struktur när vi kör ett program på datorn, matar in indata och får ut utdata som resultat. Programmet vi använder är avgörande för resultatet, precis som matreceptet samt dess förverkligande är avgörande för om vi lyckas med maträtten.

Algoritmiskt tänkande

Båda exemplen visar: Det är algoritmer som medvetet eller omedvetet styr *hur* vi gör – ett sätt att tänka vars gemensamma drag kan generaliseras så här:

1. Att formulera problemet och definiera målet. Hur når vi målet – problemets lösning?

2. Genom att bryta ner problemet i mindre, överskådliga och enklare delar, s.k. *moduler*. Varje modul ska i princip kunna utföras av vem som helst. Detta kallas för *modularisering* som är en allmän princip inte bara i programmering utan i all problemlösning.

3. Genom att ge *instruktioner* som leder till problemets lösning. De måste formuleras på ett entydigt sätt så att de inte kan tolkas på olika sätt. För datorer gör exakt som vi säger. Det har visat sig att det vanliga språket inte lämpar sig för detta ändamål, för det är tolkbart. Skönlitteraturen är ett praktexempel för olika tolkningar av språket. Det vore synd om det inte vore så. Därför har man i programmering hittat på andra, speciella programmeringsspråk vars vokabulär och syntax följer strikta regler som är entydiga. Datorn kan tolka dessa regler endast mekaniskt.

4. I denna process uppstår situationer där vi måste träffa ett *val* – samma sak som att besvara en *fråga*. Den första frågan i algoritmen "Att byta lampa" är "Är lampan inkopplad?" (ovan). Valet mellan "Ja" och "Nej" avgör hur algoritmen fortsätter. Ytterligare val följer.

Det är avgörande att skilja mellan *instruktion* och *val*. En instruktion är ett *kommando* som måste *utföras* medan ett val är en *fråga* som måste *besvaras*. I flödes-

planen till lampalgoritmen är *instruktion* (grön) och *val* (gul) markerade med olika färger. Deras distinktion blir avgörande när man går över från flödesplan till kod.

Algoritmers byggstenar

Man delar in algoritmers viktigaste ingredienser i tre kategorier och kallar dem för *kontrollstrukturer*, eftersom de är generella strukturer som styr och kontrollerar algoritmerna och ger dem den karakteristiska ordningen. Dessa grundläggande kontrollstrukturer är *sekvens*, *selektion* och *repetition* och kommer att tas upp i boken. De anses vara algoritmers byggstenar. Alla algoritmer är uppbyggda av dem.

Avgörande för en algoritms funktionalitet är ingrediensernas *inbördes ordning*. Tar man in i en kokande gryta potatisen först och köttet sedan – istället för tvärtom – blir det mos istället för maträtt. I detta sammanhang hör även algoritmens korrekta avslutning. Utan ett exakt formulerat *avslutningskriterium* som uppnås i ändlig tid uppstår evighetsloopar. När sådant inträffar brukar vi ofta säga att datorn "hängt sig". I själva verket är orsaken en algoritm med ett inkorrekt konstruerat avslutningskriterium. Allt detta kommer att behandlas utförligt i boken.

Ytterligare en ingrediens av algoritmer är *logik*. Datorer kan ingen logik. Människan måste föra över logiken in i datorn. Det är det som kallas för *artificiell intelligens*. Bl.a. formuleringen av korrekta avslutningskriterier i val och loopar, men även modularisering och strukturering kräver logiskt tänkande.

Att upptäcka *mönster* är också en förmåga som ofta behövs i konstruktion av algoritmer, vilket vi kommer att se i våra programexempel som följer i boken.

I valet av instruktioner som ska tas med i en algoritm är det en självklarhet att man sorterar bort allt som är mindre relevant och tar in endast det som är relevant. Dvs även att avgöra *relevansen* av saker och ting för att uppnå det definierade målet (punkt 1) hör till programmerarens uppgifter.

Val av programmeringsspråk

I denna kurs har vi bestämt oss för programmeringsspråket C/C++ för att introducera till programmering. Men språket är bara ett medel av underordnad betydelse. Målet är att lära sig *tankesättet* och *tekniken* att programmera, oberoende av språk. Har man en gång förstått de grundläggande koncept som är gemensamma för alla språk, blir det närmast en teknikalitet att på egen hand lära sig ett nytt språk.