Logik och det Booleska

Introduktion till Booleska Uttryck och Kontrollstrukturer för Blivande Programmerare

Gustav Hartvigsson

Strömstad Gymnasium

2 maj 2016

Sammanfattning

I detta dokument skall eleven lära sig den formella notationen för de booleska operatorer. Dokumentet går igenom operatorerna från en enkel nivå genom ekvationer och sanningstabeller.

Eleven skall få en överblick över de operatorer som är mest relevanta för programmering: `och`, `eller`, `exklusiv eller`, `negering`, `implikation` och `omm`.

Eleven presenteras därefter för övningar som *bör* utföras som testar elevens förståelse för booleska operatorer, booleska uttryck och sanningstabeller.

I andra halvan av detta dokument skall eleven lära sig hur kontrollstrukturer ser ut, fungerar och någon form av pseudokod. Eleven skall lära sig hur det går att skriva och använder pseudokod och varför det är av fördel att använda den.

Målet med detta dokument är inte att vara ett substitut till de traditionella läromedlen, utan att vara ett komplement. Det är inte tänkt att eleven skall lära sig detta utantill – det är inte en del av kursmålen – utan att exponera eleven för hur det går att tänka i relation till booleska operatorer och logiska uttryck som förekommer inom programmering. Samt lära sig att läsa, skriva och resonera kring kod genom användningen av pseudokod.

Detta dokument finns att få tagpå på `https://launchpad.net/bok-logik` i form av `.tex` och `.pdf`. Förändringar och förbättringar är alltid välkomna.

Tack till:

Teresia Lind (korrekturläsning),
Vincent Longo (förtydliganden),
Tobias Genberg (moraliskt stöd och idébollplank)

Innehåll

1 Booleska Operatorer, Uttryck och Logik

Ordet boolesk (*boolean* på engelska) kommer av den brittiske matematikern George Boole som introducerade konceptet i sina böcker *The Mathematical Analysis of Logic (1847)* och *An Investigation of the Laws of Thought (1854)*. Ordet lär ha myntats av Henry M. Sheffer enligt Edward Vermilye Huntington.

Logik hjälper oss att förstå och resonera kring påståenden som ”Det finns ett heltal som inte är summan av kvadratroten ur två heltal”.

I logiken så används oftast $p, q, r, ...$ för att representera något påstående med ett visst sanningsvärde, ett sådant uttryck kan vara ”Pelle har en dator”, ”klockan är sexton fyrtiofem” eller ”mitt hus är mindre än det där huset”.

I detta dokument användes – generellt – notationen S för att representera ett `sant` sanningsvärde för ett uttryck och F för ett `falskt` sanningsvärde för ett uttryck.

Den notationen som Boole använde i sina böcker var 1 för ett `sant` sanningsvärde och 0 för ett `falskt` sanningsvärde.

Denna notation används inte här dels för att det skall vara lättare att läsa och för att skilja detta från digitala signaler, trots att de inte är så stor skillnad.

1.1 ”och” Operatorn

`”och”` operatorn har den formella matematiska notationen $\wedge$. Denna operatorn kallas formellt för konjunktion.

Om vi har sanningsuttrycken p och q och så kan vi skriva ett uttryck $p \wedge q$ och sanningstabellen för detta uttryck kan ses i tabell 1.

Tabell 1: Sanningstabell för $p \wedge q$

p	q	$p \wedge q$
F	F	F
S	F	F
F	S	F
S	S	S

Ett sätt att se på hur $\wedge$ fungerar är att ta meningen ”Pelle **och** Nora håller om varandra”, detta uttryck håller endast om både Pelle och Nora håller om varandra. Om Pelle håller om Nora, men Nora inte håller om Pelle, och tvärt om, så håller inte det uttrycket som vi utgick ifrån.

Ett annat exempel är ”klockan är fyra **och** solen går upp”. Om klockan inte är fyra eller om solen inte går upp så håller inte uttrycket, och är därmed `falkt`. Påståendena här är ”klockan är fyra” och ”solen går upp”.

1.2 ”eller” Operatorn

`”eller”` operatorn har den formella notationen $\vee$ och har de formella namnen *disjunktion* eller *inklusive disjunktion*.

Om vi har uttrycket $p \vee q$ där p och q är något logiskt uttryck som kan vara `sant` eller `falskt` så får vi sanningstabellen som kan ses i tabell 2.

Tabell 2: Sanningstabell för $p \vee q$

p	q	$p \vee q$
F	F	F
S	F	S
F	S	S
S	S	S

Ett sätt att se på hur $\vee$ fungerar är att ta uttrycket ”Pelle har en dator **eller** Noras klocka visar fjorton femton”, detta uttryck består av två delar, båda delarna – eller påståenden – kan ha sanningsvärdena `sant` eller `falskt`, och de utesluter inte varandra när det gäller hela uttryckets sanningsvärde. Alltså om Pelle har en dator och Noras klocka visar fjorton femton så håller uttrycket, samma gäller om bara ett av dessa påståenden är `sant`.

1.3 ”exklusivt eller” Operatorn

Den matematiska notationen för `”exklusivt eller”` är $\oplus$ och har det formella namnet *exklusiv disjunktion*.

En sanningstabell över uttrycket $p \oplus q$ kan ses i tabell 3.

Tabell 3: Sanningstabell för $p \oplus q$

p	q	$p \oplus q$
F	F	F
S	F	S
F	S	S
S	S	F

Ett sätt att se på detta kan vara att använda uttrycket ”Pelle **eller** Nora använder datorterminalen”, detta utesluter varandra ty både Nora och Pelle kan inte använda samma datorterminal samtidigt.

1.4 ”negering” Operatorn

Negering är viktigt att förstå, det använd ofta tillsammans med deluttryck. Den matematiska notationen för negering är $\neg$. En sanningstabell över uttrycket $\neg p$ kan ses i tabell 4.

Tabell 4: Sanningstabell för $\neg p$

p	$\neg p$
F	S
S	F

Ett exempel på hur det går att använda $\neg$ är uttrycket $p \wedge \neg q$, alltså p och inte q. Ett exempel på detta kan vara meningen ”Nora hoppar bungyjump **och** Pelle kollar **inte** på sin mobiltelefon”, uttrycket håller inte om Nora inte hoppar bungyjump eller om Pelle kollar på sin mobiltelefon.

1.5 ”implikation” Operatorn

”`implikaiton`” har operatorn $\rightarrow$. Om vi har ett uttryck p och ett uttryck q så kan det skrivas $p \rightarrow q$ som betyder ”om p så q”. En sanningstabell över $p \rightarrow q$ kan ses i tabell 5.

I $p \rightarrow q$ så kallas p för *hypotesen* och q för *slutsatsen* eller *konsekvensen*.

Tabell 5: Sanningstabell för $p \rightarrow q$

p	q	$p \rightarrow q$
F	F	S
S	F	F
F	S	S
S	S	S

Uttrycket $p \rightarrow q$ är falsk om p är sant och q är falsk, sant i övriga fall. Andra sätt att skriva $p \rightarrow q$ är som ”p är tillräcklighet för q”, ”q när p” och ”q är nödvändigt för p”.

Notera att $p \rightarrow q$ är logiskt ekvivalent till $\neg(p \wedge \neg q)$, eller $p \rightarrow q \equiv \neg(p \wedge \neg q)$.

Ett sätt att se på hur $\rightarrow$ kan användas är ”**om** Nora vinner två miljoner **så** skall hon köpa en hålkortsläsare till Pelle”, detta uttryck håller om Nora inte vinner två miljoner och inte köper en hålkortsläsare till pelle, om Nora köper en hålkortsläsare till Pelle men inte har vunnit två miljoner, samt om Nora vinner två miljoner och köper en hålkortsläsare till Pelle. Uttrycket håller inte om och endast inte om Nora vinner två miljoner men inte köper en hålkortsläsare till Pelle.

1.6 ”omm” Operatorn

”`omm`” betyder *om och endast om* och har den formella notationen $\leftrightarrow$. $\leftrightarrow$ kallas på engelska *iff* (*if and only if*) eller *bidirectional implication* vilket kan ge en indikation över vad operatorn gör.

Notera att $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p) \equiv \neg(p \oplus q)$. En sanningstabell över $p \leftrightarrow q$ kan ses i tabell 6.

Tabell 6: Sanningstabell för $\leftrightarrow$

p	q	$p \leftrightarrow q$
F	F	S
S	F	F
F	S	F
S	S	S

Ett sätt att se på hur $\leftrightarrow$ fungerar är uttrycket ”solen går upp **om och endast om** jorden roterar kring sin axel”, uttrycket är inte `sant` om något av påståendena är `falskt` men håller i om båda är antingen `sant` eller `falskt`. Alltså om jorden inte roterar och solen inte går upp samt om jorden roterar och solen går upp så håller uttrycket, uttrycket håller inte om solen går upp trots att jorden inte roterar eller om jorden roterar men solen inte går upp.

1.7 Booleska Uttryck och Sanningstabeller

Boolesk algebra är viktig att förstå och hur den relaterar till sanningsuttryck. Ovan sågs exempel på boolesk algebra där vi relaterade den till uttrycks sanningsvärde.

I formel 1 kan ses ett exempel på ett booleskt uttryck.

$$(p \vee q) \oplus \neg(p \wedge q) \tag{1}$$

Tabellen 7 är en sanningstabellen för uttrycket 1. Uttrycket har bara två påståenden p och q, dock kan uttryck vara mycket mer komplexa med många fler påståenden.

Tabell 7: Sanningstabell för $(p \vee q) \oplus \neg(p \wedge q)$

p	q	$p \vee q$	$p \wedge q$	$\neg(p \wedge q)$	$(p \vee q) \oplus \neg(p \wedge q)$
F	F	F	F	S	S
S	F	S	F	S	F
F	S	S	F	S	F
S	S	S	S	F	S

Vi ser att $(p \vee q) \oplus \neg(p \wedge q) \equiv \neg(p \oplus q)$ (ekvivalent).

När vi ritar sanningstabeller för komplexa uttryck kan det vara till hjälp att kunna några grundläggande knep.

- Det finns 2^n *rader* i sanningstabellen där n är antalet påståenden $(p, q, r, ...)$. Alltså finns det två påståenden så skall det vara fyra rader, finns det fem påståenden skall det finnas 32 rader.
- Operator ordningen $\neg \wedge \vee \rightarrow \leftrightarrow$.
- Dela upp de minsta deluttrycken i var sin kolumn kombinera dem i i följande kolumner tills hela uttrycket finns att avläsa i sista kolumnen.

- Räkna binärt när påståendena fylls i. Alltså, har vi tre påståenden p, q, r så fylls första raden i med F, F, F, andra raden med S, F, F tredje raden med F, S, F fjärde raden S, S, F hela vägen till S, S, S.

Ett exempel är $(p \vee \neg q) \oplus (r \wedge q)$. Detta uttryck har tre påståenden (p,q och r), detta medför att sanningstabellen skall ha sex rader.

Det minsta deluttrycket är $\neg q$ och kommer i den första kolumnen efter påståendena, det näst minsta deluttrycken är $(p \vee \neg q)$ och $(r \wedge q)$ och följer på detta, och avslutas med en kolumn hela uttrycket.

Tabell 8: Exempel tabell

Påståenden			Deluttryck			Hela uttrycket.
p	q	r	$\neg q$	$(p \vee \neg q)$	$(r \wedge q)$	$(p \vee \neg q) \oplus (r \wedge q)$
...	...	...	...	...	...	...
...	...	...	...	...	...	...

1.8 Kontradiktion och Tautologi

Programmerare kommer i kontakt med koncepten *tautologi* och *kontradiktion*, detta för att det kan förekomma tautologiska och kontradiktoriska uttryck i kod som kan orsaka felaktigt beteende i mjukvara. Det är därför viktigt att ha en förståelse för vad en tautologi och en kontradiktion är.

En tautologi är ett uttryck som är `sant` oavsett vad påståendena har för sanningsvärde. Ett exemplet på en tautologi är $A \vee \neg A$ (A eller inte A), detta uttryck är alltid `sant` oberoende av vad A har för sanningsvärde.

En kontradiktion är ett uttryck som är `falskt` oberoende på vad påståendena har för sanningsvärde. Ett exempel är $A \wedge \neg A$ (A och inte A), detta uttryck är `falskt` oberoende av vad A har för sanningsvärde.

1.9 Kvantifikatorer

Ibland så behöver vi kunna resonera kring existensen av något i en viss domän med en viss egenskap eller om ett påstående gäller för alla värden i en viss domän. I logiken så kan vi göra detta med hjälp av *kvantorer* eller *kvantifikatorer*. Det finns två primära sådana som vi använder inom logiken, *allkvantifikatorn* $\forall$ och *existenskvantifikatorn* $\exists$. Ni kommer här att introduceras till konceptet av kvantifikatorer, dock så kommer vi inte all lägga något krut på detta, det finns mycket som vi skulle kunna gå genom när det gäller regler för detta, dock är det utanför ramarna för detta dokument.

Säg att vi har påståendet ”Det finns en fågel med blå fjädrar”, då kan man resonera kring detta med hjälp av $\exists x P(x)$ där $P(x)$ är ”x har blå fjädrar” och x är en fågel i domänen av fåglar. Detta påståendet är sant om det finns minst en fågel med blå fjädrar.

Om vi har påståendet ”alla böcker i biblioteket har en blank sida” så kan vi skriva $\forall x P(x)$ där $P(x)$ är påståendet ”x har en tom sida” och x är en bok i domänen böcker i biblioteket. Detta påstående är sant om och endast om det gäller för **all** böcker i biblioteket.

Kvantifikatorer kan bara användas på något i en viss domän eller i en mängd. Läs mer om detta om ni är intresserade i Matematik 5 eller Diskret Matematik.

1.10 Ekvivalenser

Tabell 9: De logiska ekvivalenserna

$p \wedge S \equiv p$ $p \vee F \equiv p$	Identitetslagrna
$p \vee S \equiv S$ $p \wedge F \equiv F$	Domänlagarna
$p \vee p \equiv p$ $p \wedge p \equiv p$	Idempotentlagarna
$\neg(\neg p) \equiv p$	Dubbelnegationslagen
$p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$	Kommutativitetslagarna
$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Associationslagarna
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Distributivalagarna
$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$	De Morgans lagar
$p \vee (p \wedge q) \equiv p$ $p \wedge (p \vee q) \equiv p$	Absorptionslagarna
$p \vee \neg p \equiv S$ $p \wedge \neg p \equiv F$	Negationslagarna

Tabell 10: Ekvivalenserna för implikation

$p \to q \equiv \neg p \vee q$
$p \to q \equiv \neg q \to \neg p$
$p \vee q \equiv \neg p \to q$
$p \wedge q \equiv \neg(p \to \neg q)$
$\neg(p \to q) \equiv q \vee q$
$(p \to q) \wedge (p \to r) \equiv p \to (p \wedge r)$
$(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$
$(p \to p) \vee (p \to r) \equiv p \to (q \vee r)$
$(p \to r) \vee (q \to r) \equiv (p \wedge q) \to r$

Tabell 11: Ekvivalenserna för omm

$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$
$p \leftrightarrow q \equiv \neg q \leftrightarrow \neg p$
$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg q \wedge \neg p)$
$\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$

1.11 Övningar

Dessa övningar skall göras individuellt och **för hand** med papper och penna. Det är **inte** viktigt att eleven skriver eller ritar snyggt, dock skall det vara läsbart och tydligt.

När eleven ritar tabellerna kan de använda en linjal för att det skall bli snyggt och tydligt.

Eleven **skall inte** rita, skriva eller fylla i något på detta dokument, de skall använda lösa papper.

Övning 1 Vad skall stå istället för $\square$ i uttrycket $(p \vee q)\square\neg r$ så att r negerar deluttrycket $(p \vee q)$.

Övning 2 Rita en sanningstabell för uttrycket $\neg(p \vee q) \wedge (p \oplus r)$.

Övning 3 Bevisa att $(p \to q) \wedge (p \to r) \equiv p \to (p \wedge r)$.

2 Pseudokod, Kontrollstrukturer och Koppling till Logik

När vi skriver kod så kan det vara nödvändigt att den kod vi skriver kan kan göra något val under några omständigheter. En sådan omständighet är ”om variabel i är mindre än 20 så utför x annars utför y”, detta mönster är mycket vanligt. Detta kallas för *kontrollstrukturer* och styr *programflödet*, alltså vad programmet gör, om det så är att läsa av styrspakarna på din Game Box Station handkontroll och få din karaktär på skärmen utföra något, eller att få en lampa att blinka i ett visst mönster.

I denna sektion kommer vi att gå genom en vanlig notationen för pseudokod, denna notation är viktig att förstå för den hjälper oss att resonera kring programflöden och algoritmer på ett språkoberoende sätt.

Här kommer vi att använda en variant av pseudokod som används av LaTeX paketet `algorithmicx`, ett exempel på detta kan ses i algoritm 1. Koden kommer att använda engelsk notering, men det går även att använda svenska ord i de som ni skriver.

Algoritm 1 Ett exempel på pseudokod.

```
1: if i ≥ maxval then
2:     i ← 0
3: else
4:     if i + k ≤ maxval then
5:         i ← i + k
6:     end if
7: end if
```

Algoritm 2 är ett exempel på hur algoritm 1 kan se ut på svenska.

Algoritm 2 Hur algoritm 1 kan se ut på svenska.

```
1: om i ≥ maxval så
2:     i ← 0
3: annars
4:     om i + k ≤ maxval så
5:         i ← i + k
6:     slut om
7: slut om
```

Pseudokod är har inte någon formell notation, dock så förekommer det vissa vanliga element:

- Börja ett kodblock med ett nyckelord och eventuella krav.
- Avsluta kodblock med ett `slut` eller `end` följt av namnet på nyckelordet som användes för att starta kod stycket.
- All pseudokod *skall* vara indenterad så att det som är innanför kodblocket är lätt att urskilja från det utanför. Undantag finns.
- Skriv vad som skall hända, inte hur det skall utföras.

2.1 `om` p `så` x `annars` y

Inom programmering så är `om` satsen den mest basala kontrollstrukturen. Den – tillsammans med `hopp` – kan användas istället för alla andra kontrollstrukturer.

`hopp` operationen används för att hoppa mellan olika ställen i koden. Denna operation anses generellt vara omodern, och finns inte i många språk.

I algoritm 1 och 2 (som är ekvivalenta) så såg vi några exempel på hur `om` kontrollstrukturen kan användas. Vi kommer nu att gå genom hur det går att ta sig från ett logiskt uttryck till kontrollstrukturer.

Säg att vi har uttrycket ”Om och endast om Nora köper en häst så skall Pelle få rida den”. Detta uttrycks logiskt som: $n \Leftrightarrow p$ där n är ”Nora köper en häst” och p är ”Pelle får rida på Noras häst’. Detta skulle uttryckas i kod som det som visas i algoritm 3.

Algoritm 3 Får Pelle rida?

```
1: om Nora köper en häst så
2:     Pelle får rida på Noras häst
3: slut om
```

Ett annat exempel kan vara ”Om Pelle har mer eller lika med tjugo euro så skall han bjuda Nora på en pizza annars får Nora en hamburgare”, detta uttrycks logiskt i ekvation 2 och uttrycks i pseudokod i algoritm 4.

$$\text{Nora får} \begin{cases} \text{Pizza} & p \leq 20 \\ \text{hamburgare} & \text{annars} \end{cases} \tag{2}$$

Där p är hur många euro Pelle har

Algoritm 4 Får Nora hamburgare eller pizza?

```
1: om p ≥ 20 så                          ▷ Pelle har mer än 20 euro
2:     Nora får Pizza
3: annars                                ▷ Pelle har inte mer än 20 euro
4:     Nora får Hamburger
5: slut om
```

Låt oss nu ta ett mer abstrakt exempel som är mer lik det som vi kan komma att se i riktig kod. I ekvation 3 och uttrycks i kod i algoritm 5.

$$x = \begin{cases} 1 & i \leq 1 \\ 2 & 1 < i \leq 20 \\ 3 & 20 < i \leq 50 \\ 4 & \text{annars} \end{cases} \tag{3}$$

Där $i \in \mathbb{Z}$.

Algoritm 5 Sett x till något värde beroende på vad i har för värde

Require: $i \in \mathbb{Z}$
1: **if** $i \leq 1$ **then**
2: $x \leftarrow 1$
3: **else if** $1 < i \leq 20$ **then**
4: $x \leftarrow 2$
5: **else if** $20 < i \leq 50$ **then**
6: $x \leftarrow 3$
7: **else**
8: $x \leftarrow 4$
9: **end if**

Notera att det går att skriva kontrollstrukturen på ett logiskt ekvivalet sätt som syns i algoritm 6. Att skriva på detta sätt är lite mer arbete och är inte väldigt snyggt. Dock kan det förekomma i äldre kod och i vissa språk som inte har `annars om`.

Algoritm 6 Ett alternativt sätt att skirva samma kod som i algoritm 5

Require: $i \in \mathbb{Z}$
1: **if** $i \leq 1$ **then**
2: $x \leftarrow 1$
3: **else**
4: **if** $1 < i \leq 20$ **then**
5: $x \leftarrow 2$
6: **else**
7: **if** $20 < i \leq 50$ **then**
8: $x \leftarrow 3$
9: **else**
10: $x \leftarrow 4$
11: **end if**
12: **end if**
13: **end if**

2.2 `medans` p `utför` x

`medans` kontrollstationen utför det som finns innan för den medans dess krav är uppfyllda. Ett enkelt exempel på detta kan återfinnas i algoritm 7.

Algoritm 7 Medans p är uppfyllt utför $i \leftarrow i + 1$

1: let $i \leftarrow 0$
2: **medans** $i \leq 20$ **utför**
3: $i \leftarrow i + 1$
4: **slut medans**

Som ni kanske kommer ihåg från undersektion 2.1 så nändes att alla kontroll strukturer kan återskapas med bara `om` kontrollstrukturen och `hopp` operatorn, Detta demonstreras i algoritm 8 och en alternativ skrivning kan ses i 9.

Algoritm 8 Samma som i algoritm 7, fast med hopp

```
1: låt i ← 0
2: Loop:                    ▷ Detta är ett märke eller en markör. Vi kan hoppa till den.
3: om i ≤ 20 så
4:     i ← i + 1
5: annars
6:     hopp Loop                          ▷ Hoppa till markören ovan på rad 2
7: slut om
```

Den alternative skrivningen av algoritm 8 som ses i algoritm 9 är så som det kan utföras i datorer. Att använda `hopp` – om det inte är maskinkod eller om du vet exakt vad du gör – anses vara en osed och bör därmed undviks, använd istället de inbyggda kontrollstationerna som tillhandahålls av det programmeringsspråk som ni använder.

Algoritm 9 Alternativ skrivning av 8.

```
1: let i ← 0
2: Loop:
3: if i > 20 then
4:     goto JumpOut
5: end if
6: i ← i + 1
7: goto Loop
8: JumpOut:
```

2.3 upprepa x tills p – utför x medans p

`upprepa` x `tills` p och `utför` x `medans` p är kan se väldigt lika ut men har olika betydelse. Om vi skulle ta exemplet från undersektion 2.2 så kan det se ut som exemplena i algoritm 10 och 11.

Algoritm 10 Alternativ skrivning av 7.

```
1: let i ← 0
2: utför
3:     i ← i + 1
4: medans i ≤ 20
```

Algoritm 11 Alternativ skrivning av 7.

```
1: let i ← 0
2: upprepa
3:     i ← i + 1
4: tills i > 20
```

I algoritm 10 så utförs allt inuti kodblocket om och endast om uttrycket i `medans` är uppfyllt, när uttrycket inte längre uppfylls så hoppar programmet ur kodblocket. Denna struktur är vanlig inom programmeringen.

I algoritm 11 så upprepas allt inuti kodblocket tills uttrycket i `tills` är uppfyllt. Detta uttryck är inte vanlig i programmeringsspråk, utan används nästan bara inom pseudokod och algoritmbeskrivningar.

2.4 för i tills j utför x

Säg att vi vill summera alla positiva heltal från och med 0 till och med 200. Detta skulle matematiskt uttryckas som i ekvation 4.

$$r = \sum_{i=0}^{200} = 1 + 2 + 3 + ... + 198 + 199 + 200 \tag{4}$$

Detta kan uttryckat som i algoritm 12 eller 13

Algoritm 12 Addera ihop alla heltal mellan och inklusive 0 och 200

1: låt $r \leftarrow 0$
2: **för** $i \in \mathbb{Z}^+, i \leq 200; i \leftarrow i + 1$ **utför**
3: $\quad r \leftarrow r + i$
4: **slut för**
5: Skriv ut r

Algoritm 13 Alternativ skrivning av algoritm 12

1: let $r \leftarrow 0$
2: **for** i between and including 0 and 200 **do**
3: $\quad r \leftarrow r + i$
4: **end for**
5: print r

Här är det viktigt att förstå att det inte finns någon formel notation för pseudokod, dock så brukar den följa samma mönster som vi har använt här. I algoritm 12 så uttrycks `för` på ett sätt som är mer likt det som vi återfinner i vissa programmeringsspråk. Algoritm 13 är mer som när vi resonerar kring kod. Båda dessa är ekvivalenta, trots att de ser olika ut.

Om vi ser på algoritm 12 så ser vi att är det inte stor skillnad på en `för` och en `medans`.

2.5 för alla a i S utför x

När vi arbetar med `fält` (engelska `arrays`) och `mängder` (engelska `sets`) kan det vara viktigt att kunna utföra operationer på varje element i `fältet` eller `mängden`. Säg att vi har en `mängd` med namn som vi vill skriva ut, hur skulle detta gå till? Hur det går till i riktig kod är inte viktigt när vi skriver pseudokod, vad som är viktigt är vad pseudokoden representerar. Ett exempel på det går att skriva pseudokod för att skriva ut alla namnen kan ses i algoritm 14.

Algoritm 14 Skriv ut alla namn i mängden $Namn$

Krav: $Namn$ är en mängd med namn

1: **för alla** n **i** $Namn$ **utför**
2: Skriv ut n
3: **slut för**

Ett annat sätt att skriva detta är med hjälp av $\in$ (element, är ett element av) operatorn, som i algoritm 15.

Algoritm 15 Alternativ skrivning av algoritm 14

Require: $Names$ is a set of names

1: **for all** $n \in Names$ **do**
2: Print n
3: **end for**

2.6 Procedurer och Funktioner

`procedurer` och `funktioner` är så gott som ekvivalenta uttryck, dock så kan det sägas att en `procedur` – i allmänhet – inte returnerar något. Dessa strukturer är – för att förklara det på ett enkelt sätt – kodststycken som kan åkallas från andra ställen i koden. I datorer händer detta genom hopp och undanstoppade av värden på den så kallade `stacken` – dock är detta utanför ramen för det här dokumentet.

Vi kan börja med ett enkelt exempel. I algoritm 16 så räknar vi ut fakulteten av ett 20.

$$n! = fact(n) = \begin{cases} 1 & n \leq 1 \\ fact(n-1) \times n & x > 1 \end{cases} \tag{5}$$

Den (simplifierade) formella definitionen av fakultet

Algoritm 16 Räkna ut fakulteten av 20

1: **funktion** $fact(n)$ ▷ Definiera funktionen
2: **om** $n \leq 1$ **så**
3: **returnera** 1
4: **annars om** $n > 1$ **så**
5: **returnera** $fact(n-1) \times n$ ▷ Detta är en rekursiv funktion
6: **slut om**
7: **slut funktion**
8: Skriv ut $fact(20)$ ▷ Programmet skriver ut $!20$

Låt oss nu ta ett lite mer komplett exempel. Låt säga att vi vill ta ett `associativ fält` (`map` på engelska) som mappar personnummer till namn ($SsnName$) och ett `associativ fält` som mappar personnummer till telefon nummer ($SsnTel$), och så vill vi skriva ut namn och telefonnummer på var och en av personerna. Detta kan ses i algoritm 17.

Algoritm 17 Skriv ut en lista med namn och telefon nummer

Require: $SsnName : Map(ssn, name)$ is a map that maps an SSN to a name
Require: $SsnTel : Map(ssn, tel)$ is a map that maps an SSN to a telephone number

```
1: function createNameToTel(A : Map(name, ssn), B : Map(ssn, tel))
2:     create outSet : Set(Pair(name, tel))
3:     for all pairA : Pair(ssn, name) ∈ A do
4:         create pairB : Pair(name, tel)
5:         pairB.name ← pairA.name and pairB.tel ← B[pairA.ssn]
6:         add pairB to outSet
7:     end for
8:     return outMap
9: end function
10:                                              ▷ Only an empty line...
11: function printPairOfSet(M : Set(Pair(name, tel)))
12:     for all pair : Pair(name, tel) ∈ M do
13:         Print "pair.name has the telephone number pair.tel"
14:     end for
15: end function
16:
17: let NameTel ← createNameToTel(NameSsn, SsnTel)
18: printPairOfSet(NameTel)
```

Koden i algoritm 17 är är skriven på ett väldigt mångordig (verbose) sätt, den skulle nog kunna göras lite mer fåordig.

2.7 Kodramar - Kodomfattning

I kod talar vi om ramar (engelska `scope`) som håller variabler och/eller objekt. Det finns två primära ramar som vi använder: `lokal` och `global`.

Den `globala` ramen innehåller variabler och objekt som är åtkomliga från hela programmet.

De `lokala` kodramarna kan ses som lådor. En låda som kan går att se ut genom, men inte se in i. Ett exempel på en sådan låda är en funktion: De variabler som deklareras i denna kodram kan inte ses från andra funktioner. Ett annat exempel på en lokal kodram är en `om`-satts, om variabel i deklareras i en sådan kodram så är den inte åtkomlig från ramen utanför.

Om vi skulle beskriva vad den lokala kodramen är mer exakt är som följande: Den lokala kodramen, likt den globala kodramen, innehåller variabler och objekt vars tillgång är begränsad till ett lokalt exekverings sammanhang (execution context), så som inuti en funktion eller en loop. Den lokala kodramen har tillgång till allt i den globala kodramen, men en funktion har inte tillgång till variabler i en annan funktion för dess sammanhang är utanför ramen.

Det finns fler sätt att dela upp kod än detta, så som klasser, object, design mönster (design patterns), men det är utanför ramen för detta dokument.

Ett exempel på hur detta fungerar kan ses i algoritm 18.

Något att ha i åtanke är att författaren använder ordet ”kodram”, detta ord är helt påhittat av författaren för han inte kunde hitta ett bättre ord på svenska.

Algoritm 18 Exempel på hur kodramar fungerar.

```
1: låt my_global_var ← 12                                   ▷ En global variabel
2: funktion funk1()
3:     låt funk_1_var ← ”Någon sträng”
4:     /* Det är tillåtet att använda globala
5:        variabler i en funktion */
6:     Skriv ut my_global_var
7:
8:     /* Det är tillåtet att använda vaiabler som
9:        deklareras i denna kodram här */
10:    Skriv ut funk_1_var
11:    om foo så
12:        låt lokal_var ← 1337                          ▷ Deklarerad innuti om-satsen.
13:        my_global_var ← lokal_var                                          ▷ OK!
14:    slut om
15:    lokal_var ← 2          ▷ Går inte att använda lokal_var utanför kodramen.
16: slut funktion
17:
18: funktion funk2()
19:    Skriv ut my_global_var                                              ▷ Helt OK
20:
21:    /* Det går inte att använda denna variablen här,
22:       den är deklarerad i en annan kodram */
23:    Skriv ut funk_1_var
24: slut funktion
```

2.8 Övningar

Dessa övningar skall göras individuellt **eller** i grupp och **för hand** med papper och penna, **eller** på vita tavlan. Det är **inte** viktigt att eleven skriver eller ritar snyggt, dock skall det vara läsbart och tydligt.

Eleven **skall inte** rita, skriva eller fylla i något på detta dokument, de skall använda lösa papper.

Övning 1 Skriv en algoritm som adderar alla tal i en mängden N och skriv ut summan i slutet.

Övning 2 Skriv en algoritm som adderar alla jämna tal i mängden M och skriv ut summan i slutet.

Övning 3 Skriv en algoritm som skriver ut

```
-
--
---
----
-----
------
```

Algoritmen skall använda en loop an något slag.

3 Svar

3.1 Logik

Övning 1 $(p \vee q) \wedge \neg r$.

Övning 2 –

p	q	r	$(p \vee q)$	$(p \oplus r)$	$\neg(p \vee r)$	$\neg(p \vee q) \wedge (p \oplus r)$
F	F	F	F	F	S	F
S	F	F	S	F	F	F
F	S	F	S	S	F	F
S	S	F	S	F	S	F
F	F	S	F	S	S	S
S	F	S	S	S	F	F
F	S	S	S	F	F	F
S	S	S	S	F	F	F

3.2 Pseudokod

Övning 1 -

Algoritm 19 Addera alla tal i en mängd.

Krav: N är en mängd med tal.

```
1: låt tmp ← 0
2: för alla i i mängden N utför                    ▷ för alla i ∈ N
3:     temp ← temp + i
4: slut för
```

Övning 2 -

Algoritm 20 Addera alla jämna tal i en mängd.

Krav: N är en mängd med tal.

```
1: låt tmp ← 0
2: för alla i i mängden N utför                    ▷ för alla i ∈ N
3:     om i är jämt så                             ▷ i mod 2 = 0 alt. i|2
4:         temp ← temp + i
5:     slut om
6: slut för
```

Övning 3 Visa lärare eller handledare.
Ett exempel kan vara.

Algoritm 21 Skriv ut ett streck.

1: Skapa en tom textsträng S
2: $i \leftarrow 0$
3: **utför**
4: lägg till ett ”-” till strängen S
5: skriv ut S
6: $i \leftarrow i + 1$
7: **medans** $i \leq 5$

4 ”Ordlista”

Här följer en liten ”ordlista” över vanliga begrepp. Se http://www.datatermgruppen.se/ordlista.html för fler ord som har med datorer att göra.

Tabell 12: Engelska ⇒ Svenska

iff	omm
if and only if	om och endast om
subset	delmängd
superset	övermängd
map, associative array	associativt fält
array	fält
set	mängd
while	medans
do	utför, gör
for	för
scope	ram, kodram, ramar, kodramar
execution context	exekverings sammanhang

Tabell 13: Symboler

$\subset$	Strikt delmängd.
$\subseteq$	Delmängd eller lika.
$\supset$	Strikt övermängd.
$\supseteq$	Övermängd eller lika.
$\forall$	För alla - Allkvantifikatorn.
$\exists$	Existerar - Existenskvantifikatorn.
$\in$	I, inuti.
$\ni$	Som $\in$ fast åt andra hållet.

Det finns motsatser till vissa av dessa, så som $\not\subset, \not\supset, \not\ni, \notin$ finns även.

Tabell 14: Vanliga mängder

$\mathbb{R}$	Alla reella tal.
$\mathbb{Z}$	Alla heltal, $\mathbb{Z} = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}$.
$\mathbb{N}$	Alla naturliga tal. $\mathbb{N} = \{i \in \mathbb{Z}, i > 0\}$ eller $\mathbb{N} = \{i \in \mathbb{Z}, i \geq 0\}$
$\mathbb{Q}$	Alla bråktal eller rationella tal, $\mathbb{Q} = \{{}^{p}/_{q} \mid p \in \mathbb{Z}, q \in \mathbb{Z}, q \neq 0\}$.
$\mathbb{C}$	Komplexa tal.