Datalogi 0 GA

Kursusbog 2003

Bind 1

Generelle oplysninger

Nils Andersen

juni 2003

Datalogisk Institut
Københavns Universitet
2003 *HCØ-Tryk*

Redaktion: © Nils Andersen 2003
Tryk: HCØ-Tryk
Oplag: 350 eksemplarer
ISSN 0108-3708

Forord

Kursusbogen for Datalogi 0 GA udgives i fem bind:

1. Generelle oplysninger
2. Vejledning i brug af DIKUs EDB-system
3. Laboratorieøvelser
4. Noter
5. Uddrag af L. C. Paulson: ML for the Working Programmer, 2^{nd} edition

Bindene kan købes på det datalogiske studiekontor på DIKU, efterhånden som de bliver færdige.

Dette første bind indeholder en række afsnit af vidt forskellig karakter; benyt først givne lejlighed til at kigge det hurtigt igennem. Læg mærke til, at nogle af de givne oplysninger og anvisninger først vil blive aktuelle på et senere tidspunkt. Det er ikke nødvendigt at læse det hele nu, men det er vigtigt at lægge mærke til, afsnittene findes.

En række yderligere oplysninger kan først gives i løbet af kursets udvikling og vil blive annonceret på hjemmesiden `http://www.diku.dk/undervisning/2003e/dat0ga/`

Kommentarer, forslag, oplysninger om trykfejl og lignende modtages gerne. De kan sendes til `nils@diku.dk`

- Vær aktiv
  - Følg de seneste kursusmeddelelser på hjemmesiden
  - Læs regelmæssigt elektronisk post (mindst to gange ugentligt)
  - Planlæg studietiden. Jo mere, man har planlagt, jo mere når man også
  - Afpas læsehastigheden efter teksten. I faglitteratur er der en mening med hvert ord, der står, og hver linje skal forstås. At læse en enkelt teoretisk tekstside kan godt tage en time
- Vær kritisk
  - Tvivl på autoriteterne
  - Stil spørgsmål ved forelæsningerne
  - Søg alternative kilder
  - Når du præsenteres for en oplysning, så overvej, hvordan det ellers kunne forholde sig
- Vær åben
  - Prøv at forstå studiekammeraternes spørgsmål og problemer
  - Find nogle at studere sammen med
  - Få også tid til andet end studierne
- Følg med
  - Kom ikke bagud med stoffet
  - Regn de stillede opgaver
  - Deltag i øvelserne
  - Indberet trykfejl og kommentarer til læreren (læg besked i hans postrum på DIKU, eller send elektronisk post til `nils@diku.dk`)
- Overhold tidsfristerne
  - Aflever obligatoriske opgaver rettidigt
  - Husk eksamenstilmelding
  - Konsekvenserne af efterladenhed på disse områder kan være vidtgående (såsom udelukkelse fra fortsatte universitetsstudier eller bortfald af uddannelsesstøtte)

Indhold

Kapitel 1

Om Datalogi 0 GA

1.1 Studiemæssig indplacering

Ud af "Matematik 4", som var indslaget om numerisk analyse på "den rene matematiklinje" efter studieordningen af 1960, udvikledes det introducerende datalogikursus "Datalogi 0", som var et helårskursus på 20 ECTS-points (dvs. med en belastning på $\frac{1}{3}$ af fuld tid). I studieåret 2000-01 blev Datalogi 0 blev udbudt for sidste gang; derefter blev det delt op i to kurser på 10 points hver: Datalogi 0 GA, som ligger om efteråret, og Datalogi 0 GB, som kun afholdes om foråret.

Disse introduktionskurser er såvel beregnet for studerende, der vil gøre datalogi til hovedfag, som for dem, der vil bruge datamatik og datateknik (eng.: *computer science, information technology*) som hjælpemiddel i andre fag.

I efteråret 2003 udbydes Datalogi 0 GA for sidste gang, idet det naturvidenskabelige fakultet fra efteråret 2004 overgår til en ny studiestruktur.

Indhold Edb-systemer til bestemte formål (korrespondance, tekstbehandling, billedbehandling, bogføring, værktøjsmaskiner og så videre) kan som regel benyttes efter kort tids træning og uden dybtgående kendskab til de bagvedliggende funktioner. De to Datalogi 0-kurser udgør et grundkursus i *programmering* ud fra det synspunkt, at skal man som akademiker beskæftige sig med EDB-systemer, er det nødvendigt at få personlig erfaring med styring af computere.

Selv om hurtighed og brugervenlige grænseflader er afgørende for EDB-systemers praktiske værdi, ofres disse egenskaber ikke synderlig opmærksomhed på Datalogi 0 GA, hvor hovedvægten lægges på fremgangsmåder og hjælpemidler til forståelse, konstruktion og beskrivelse af *korrekte* systemer.

Blandt andet af denne grund er det besluttet at knytte undervisningen på Datalogi 0 GA til et *værdiorienteret funktionsprogrammeringssprog*. (På Datalogi 0 GB benyttes et objektorienteret tilstandsprogrammeringssprog.)

Forudsætninger I matematik kræves der kundskaber svarende til obligatorisk niveau på gymnasiets matematiske linje (B-niveau). (Studienævnet har vedtaget, at kravet skal ændres til matematik på A-niveau. Denne beslutning vil formentlig få virkning for nyoptagne fra og med september 2005.)

Belastning Datalogi 0 GA indgår i bachelorstudiet med 10 ECTS-points, svarende til en tredjedel af et studiesemester. Er de nævnte forudsætninger til stede, bør kurset derfor kunne gennemføres med en arbejdsindsats på omkring 15 timer ugentligt.

Erfaringsmæssigt har mange studerende været fristede til at lægge en meget ujævn indsats i kurset: Umiddelbart forud for afleveringsfristen for en rapport er alle andre fag og gøremål lagt til side, mens aktiviteten i de øvrige perioder har været lav. Prøv at undgå denne fristelse! Sørg hele tiden for at være ajour med stoffet — så bliver rapporterne mindre tyngende. Ved at aflyse forelæsninger og øvelser i perioder med intensivt rapportarbejde forsøger vi at udjævne belastningen.

1.2 Undervisningens form

Undervisningen på Datalogi 0 GA veksler mellem *forelæsninger*, *øvelser* og praktisk *opgaveløsning* med brug af datamat. I perioderne med rapportopgaver tilbydes i visse tidsrum *instruktorvagt* i maskinstuen. De studerende opfordres derudover til at danne læsegrupper, hvor diskussion og tilegnelse af pensum kan foregå, uden at den enkelte lades i stikken.

Det forudsættes, at deltagerne på Datalogi 0 GA behersker eller tilegner sig den nødvendige studieteknik til at læse bøger og tilegne sig deres indhold. Som vejledning i studieteknik kan anbefales Thomas Harboe & Jakob Ravn: Kunsten at studere, Teknisk Forlag 2000, ISBN 87 571 2315 2 eller C.A. Mace: The Psychology of Study, Penguin Books 1968 (specielt side 44–67).

Selv om der ikke er mødepligt til forelæsninger og øvelser, er kurset beregnet på aktiv deltagelse, og man kan *ikke* regne med, at alle oplysninger bliver tilgængelige på internet.

1.2.1 Forelæsninger

Forelæsningerne vil blive brugt til gennemgang af pensum, besvarelse af faglige spørgsmål, orientering i tilgrænsende emner, fremlæggelse af rapportopgaver samt i øvrigt meddelelser, der mest hensigtsmæssigt gives samlet til alle kursusdeltagere.

1.2.2 Øvelser

De studerende fordeles på mindre øvelsehold, hvor hvert hold ledes af en ældre studerende, der er ansat som instruktor. Øvelserne bruges til vejledning i brug af det datamatiske system, diskussion eller gennemregning af stillede opgaver samt til drøftelser af rapportbesvarelser, efter at de er blevet bedømt af instruktorerne.

Det forudsættes, at de studerende er velforberedte (herunder, at de har fulgt forelæsningerne og forsøgt at regne alle de stillede opgaver), og de forventes at deltage aktivt i diskussionerne af opgaver og det faglige indhold i øvrigt.

Instruktorens opgave er

- At hjælpe de studerende med at få afviklet deres opgaveløsninger på instituttets EDB-system
- At tydeliggøre det gennemgåede stof og eventuelt drøfte det med deltagerne

- At give tips ved problemer i de stillede opgaver og diskutere deltageres ufærdige løsninger
- At rette opgavebesvarelser og rapporter
- At støtte deltagerne i at opnå en hensigtsmæssig studieteknik

En instruktors opgave er *ikke*

- At forelæse
- At servere kundskaber for holdet
- At løse en stillet opgave fra grunden
- At give en færdig 13-skala-karakter for en rapport
- At dominere øvelserne

Øvelser afholdes i DIKU's undervisningslokaler i nordfløjens stueetage ("øvelsesgangen") med lokalebetegnelserne N 037, N 034, N 030, N 026, N 022, N 018, N 014, N 010 og N 004.

Ønsker man at skifte øvelsehold, er det nødvendigt at følge den af bachelorstudieadministrationen foreskrevne procedure. Ajourførte holdlister ophænges løbende på øvelsesgangens opslagstavler.

1.3 Målbeskrivelse

Problemanalyse: Deltagerne skal lære at afklare og analysere et problem, hvis manuelle løsning er velkendt eller let at finde. De skal kunne påpege væsentlige uklarheder, som måske ikke hindrer manuael behandling (hvor man kan anvende sin sunde fornuft), men som må afklares, før automatisk behandling kan finde sted.

Kodning: Deltagerne skal kunne foreslå forskellige mulige repræsentationer af informationen under databehandlingen. De skal kunne forslå forskellige udfomninger af inddata, forskellige interne repræsentationer og forskellige præsentationer af resultaterne. Deltagerne skal kunne påpege de forskellige muligheders fordele og ulemper til brug for valg imellem dem.

Programmering: Ved at gennemgå kurset skal deltagerne lære velstrukturerede højere programmeringssprog at kende i en sådan grad, at de selvstændigt kan udarbejde programmer eller programdele på en størrelse af op til 500 linjers programtekst.

Indkøring: Der skal opnås færdighed i at indtaste et program og afprøve det systematisk, så det bliver korrekt og robust i den forstand, at det fungerer for indgangsværdier inden for det specificerede område og reagerer rimeligt for alle mulige værdier.

Rapportering: Deltagerne skal gøre rede for en løsning på en klar måde. De skal kunne forklare valgene af data- og programstrukturer og systematikken bag afprøvningen. Deltagerne skal desuden lære at udforme vejledninger i anvendelse af det udviklede programsystem for brugere uden forhåndskendskab til det.

Beskrivelse: Forelagt en programdel eller et kortere program skal en studerende kunne give en kort, klar og overskuelig beskrivelse af dets virkning.

Der lægges stor vægt på, at deltagerne udtrykker sig klart, præcist og sprogligt korrekt.

1.4 Undervisningsmateriale

Ud over kursusbogen benyttes følgende obligatoriske lærebog:

[**H&R**] Michael Reichhardt Hansen & Hans Rischel: *Introduction to Programming Using SML*, Addison-Wesley 1999, ISBN 0-201-39820-6

1.4.1 Supplerende litteratur

Et andet bind af kursusbogen indeholder uddrag af nedenstående bog, som det imidlertid kan anbefales, at man anskaffer sig i sin helhed:

- Larry C. Paulson: *ML for the Working Programmer*, 2nd edition, Cambridge University Press 1996, ISBN 0 521 57050 6 (indbundet), ISBN 0 521 56543 X (hæftet)

En nænsom indføring i SML fås gennem

- Greg Michaelson: *Elementary Standard ML*, UCL Press Limited 1995, ISBN 1-85728-398-8 PB

som dog desværre indeholder en del trykfejl (trykfejlsliste kan findes via kursets hjemmeside). Kursusdeltagere, der kan programmere på forhånd (i et andet programmeringssprog), vil nok foretrække beskrivelsen af SML i

- Jeffrey D. Ullman: *Elements of ML Programming, ML97 Edition*, Prentice-Hall 1998, ISBN 0-13-080391-X

Holder man af at få trukket principielle sider matematisk op, kan man måske have glæde af

- Richard Bosworth: *A practical course in functional programming using Standard ML*, McGraw-Hill 1995, ISBN 0-07-707625-7

Andre gode lærebøger (hvor Wikströms dog beskriver en tidlig version af SML) er:

- Åke Wikström: *Functional Programming Using Standard ML*, Prentice Hall 1987, ISBN 0-13-331968-7 (indbundet), ISBN 0-13-331661-0 (hæftet)

- Colin Myers, Chris Clack, Ellen Poon: *Programming with Standard ML*, Prentice Hall 1993, ISBN 0-13-722075-8

En underholdende præsentation i utraditionel stil (udformet som en fremadskridende dialog) foreligger som

- Matthias Felleisen, Daniel P. Friedman: *The Little MLer*, The MIT Press 1998, ISBN 0-262-56114-X

Den ultimative definition af programmeringssproget STANDARD ML er ikke let tilgængelig, men ledsages af en forklarende kommentar. Disse to bøger er:

- Robin Milner, Mads Tofte, Robert Harper, David MacQueen: *The Definition of Standard ML (Revised)*, The MIT Press 1997, ISBN 0-262-63181-4

- Robin Milner, Mads Tofte: *Commentary on Standard ML*, The MIT Press 1991, ISBN 0-262-13271-0 (indbundet), ISBN 0-262-63137-7 (hæftet)

Hertil kommer en beskrivelse af standardbiblioteket, som er under udarbejdelse.

Til sprogsystemet MOSCOW ML, der benyttes i den indledende del af dette kursus, hører tre velskrevne manualer

- Sergei Romanenko, Claudio Russo, Peter Sestoft: *Moscow ML Owner's Manual*
- Sergei Romanenko, Claudio Russo, Peter Sestoft: *Moscow ML Language Overview*
- Sergei Romanenko, Claudio Russo, Peter Sestoft: *Moscow ML Library Documentation*

der alle kan findes fra MOSCOW ML's hjemmeside
`http://www.dina.kvl.dk/~sestoft/mosml.html` hvor man også kan finde oplysninger om, hvordan systemet (vederlagsfrit) kan hentes hjem. Det kan afvikles under de fleste operativsystemer (Unix, MS Windows 95/98/NT/2000/XP, Macintosh (68k og PPC) med MacOS (ikke-X), Macintosh PPC med MacOS X samt (kun version 1.42) MS/DOS, Windows 3.1 og OS/2).

Den seneste version (2.00) af MOSCOW ML implementerer hele 1997-definitionen af STANDARD ML.

En anden mulighed er at skaffe sig det (noget mere omfattende) system SML OF NEW JERSEY, der også har en hjemmeside:
`http://cm.bell-labs.com/cm/cs/what/smlnj/index.html`.

1.5 Praktisk brug af datamat

Studerende, som har adgang til en datamat hjemme, kan gratis skaffe sig systemet MOSCOW ML. (Se nærmere via den ovennævnte hjemmeside.) Kursusdeltagelse forudsætter dog ikke, at man selv anskaffer sig en maskine: Til brug for de praktiske øvelser stiller DIKU et antal EDB-terminaler til rådighed for de studerende på Datalogi 0 GA (om adgangsforhold: se afsnit 2.3), og **selv om man skulle have et system hjemme, er det også nødvendigt at sætte sig grundigt ind i brugen af DIKU's terminaler og de tilhørende systemer:**

Via DIKU's terminaler kan man læse kursets hjemmeside og sende og modtage elektronisk post, og selv, hvis man har netadgang hjemmefra, vil der være visse opgaver, som kun kan løses med de systemer, der ligger på DIKU's anlæg. **Fortrolighed med brugen af DIKU's terminaler er en forudsætning for at kunne løse rapportopgaverne på den fastsatte tid.** Se i øvrigt kapitel 3 med hensyn til rapportopgaver og deres besvarelse.

1.6 Eksamensfordringer

Kursusets hovedvægt ligger dels på det stof, som gennemgås ved forelæsningerne, dels på arbejde med datalogirapporter. Eksamen i Datalogi 0 GA er på tilsvarende måde delt i to uafhængige dele: rapporterne, med bestemte afleveringsfrister i årets løb, og en afsluttende to-timers skriftlig prøve.

Hver af disse to eksamensdele bedømmes med en karakter på 13-skalaen, og de to dele er i øvrigt uafhængige og skal bestås (dvs. mindst med karakteren 6) hver for sig: det er ikke nødvendigt at bestå de to dele ved samme eksamenstermin eller i en bestemt rækkefølge.

Karakteren for kurset Datalogi 0 GA findes som gennemsnittet af de to karakterer (og karakteren for Datalogi 0 beregnes som gennemsnittet af karakteren for Datalogi 0 GA og for Datalogi 0 GB).

Den ordinære skriftlige prøve afholdes i januar, med en omeksamensmulighed i maj/juni. Rapportprøven vedrører semesterets rapportopgave(r), som bedømmes i januar måned.

For begge eksamensdele gælder det, at der skal finde en *forhåndstilmelding* sted. Bemærk tilmeldingsfristerne, og overhold dem — **efter fristens udløb modtages tilmelding ikke!** Tilmelding sker via internet (studieinformationssystemet) eller ved personlig henvendelse på fakultetskontoret (Observatoriebygningerne, Øster Voldgade 3).

Der findes to typer rapportopgaver: Karakteropgaver (K-opgaver) bedømmes samlet efter 13-skalaen, mens godkendelsesopgaver (G-opgaver) bedømmes bestået/ikke bestået. På Datalogi 0 GA stilles én G-opgave og én K-opgave.

En forudsætning for, at den studerende kan indstille sig til prøven i rapportopgaverne, er, at K-opgaven er afleveret, og at besvarelsen af G-opgaven er godkendt. De to opgaver skal være fra samme semester. (Hvis G-opgaven ikke godkendes i første omgang, gives en uge til forbedring af besvarelsen, regnet fra at den studerende har modtaget den retur.)

1.7 Stopprøver

Der er fra 1994 indført *obligatoriske førsteårsprøver* (også kaldet "stopprøver") på universitetet. Se studieinformationssystemet for nærmere oplysninger.

Kapitel 2

Praktiske forhold

2.1 Kursusadministration

Datalogi 0 GA administreres fra det fælles datalogiske studiekontor i mellemfløjens stueetage (M 004).

Her melder man sig til øvelserne, og herfra sælges kursusbogens bind. Det er også her, tilmelding til og aflevering af rapportopgavebesvarelser foregår. Adresseændring skal meddeles bachelorstudieadministrationen (*skriftligt* — på særlige fortrykte blanketter hertil).

Kontorets adresse er

Datalogisk Institut
Bachelorstudieadministrationen
Universitetsparken 1
2100 København Ø

og telefonnummeret er 3532 1363.

Bemærk, at DIKU's kontorer har lukket mellem kl. 12 og kl. 13, og at det datalogiske studiekontor kun har åbent fra kl. 11 til kl. 12 og igen fra kl. 13 til kl. 15.

2.2 Kommunikation

Ud over ved forelæsninger og øvelser vil meddelelser til de studerende blive givet ved opslag og elektronisk post, men den vigtigste kilde til den seneste information om kurset er hjemmesiden på internet: `http://www.diku.dk/undervisning/2003e/dat0ga/`

Til opslag (skrevet på papir) benyttes opslagstavlen foran kontorerne i mellemfløjens stueetage samt dat0ga-opslagstavlen på øvelsesgangen (stueetagen i DIKUs nordfløj). Ved siden af dat0ga-opslagstavlen findes en tavle, hvor hvert øvelseshold har et felt.

Gør det til en vane at læse dat0ga-siden og den elektroniske post et par gange om ugen! Mens det er læreres og instruktorers ansvar at formulere informationen, er det helt og holdent de studerendes eget ansvar at opsøge den!

2.3 Bygninger

Skitsen viser

- Datalogisk Institut (DIKU), Universitetsparken 1

  (Bygningsplan nedenfor)

- H. C. Ørsted Institutet (HCØ), Universitetsparken 5

  Her ligger auditorium 1–10 og en kantine med betjening

- August Krogh Institutet, Universitetsparken 13

  I stueetagen ligger Universitetsbogladen/Naturfagsbogladen

  På første sal en betjent kantine

- Bosch-bygningen, Jagtvej 155

  Visse undervisningslokaler er placeret Jagtvej 155 B og 155 D

  Studentercenteret for naturvidenskab (hjemsted for "Caféen?") og Videnskabsbutikken for naturvidenskab på KU ligger Jagtvej 155 D

DIKU's adresse er Universitetsparken 1, 2100 København Ø. (Der er også indgangsdøre i Universitetsparken 3 og Nørre Allé 63 samt adgang gennem store auditorium, Nørre Allé 61.)

Telefonnummeret 3532 1818 går til omstillingsbordet på H.C. Ørsted Institutet, mens vores lokale informationskontor har nummeret 3532 1400 og telefax 3532 1401. Den direkte telefonlinje til kantinen (ved køkkenet i ikke ryger-kantinen) er 3532 1435. (Telefoner under Københavns Universitets central har femcifrede lokalnumre 2*dddd*. De kan alle nås direkte udefra med 3532 *dddd*, og fra dem alle kan der stilles om.)

I lokalebetegnelserne står	N	for	nordfløjen (gul farvekode),
	M	for	mellemfløjen (grøn farvekode) og
	S	for	sydfløjen (blå farvekode)

Også lille og store auditorium ligger på DIKU.

De ni øvelseslokaler ligger som nævnt på øvelsesgangen, nordfløjens stueetage, med lokalenumre N 037, N 034, N 030, N 026, N 022, N 018, N 014, N 010 og N 004.

Edb-terminalerne er stillet op på første sal, i sydfløjen (S 111) og mellemfløjen (M 103, M 107, M 108 og M 111). Mellem de to maskinstuer ligger operatørkontoret (S 102).

På anden sal ligger en selvbetjeningskantine i mellemfløjen, og et rygerum er indrettet i nordfløjen.

Adgang Der er adgang til maskinstuer, kantine og undervisningslokaler i instituttets åbningstid, mandag til fredag kl. 7^{30} – 18^{00}.

Uden for dette tidsrum giver KU's identitetskort (i forbindelse med den tilhørende kode) adgang gennem de døre, der er koblet til magnetkortlæsere. Studerende med særlige behov kan (mod depositum på kr. 200) låne nøgle til øvelseslokaler, bibliotek, kantinedepot og lignende.

Hvis man møder vagten, som runderer uden for åbningstiden, skal man kunne legitimere sig. Alle må derfor være parat til at forevise gyldigt identitetskort til vægteren. Det er ikke tilladt at medtage gæster uden for normal arbejdstid, medmindre særlig tilladelse er givet af institutbestyrelse eller -leder. Vægteren har ret til at bortvise antrufne personer uden gyldigt identitetskort eller særlig tilladelse.

Skabe De små "blå skabe", som findes flere steder i bygningen, stilles til de studerendes disposition. Hver studerende kan (højst) råde over ét skab, som den studerende selv forsyner med hængelås. Alle låse skal hvert år være fjernet i perioden mellem den 15. juli og den 15. august.

Regler De aktuelle adgangsregler og andre nyttige praktiske oplysninger kan ses på nettet: `http://www.diku.dk/diku/praktiske.oplysninger/`

2.4 Ansvarlighed, fri adgang og ordensregler

Vi forsøger at bevare DIKU som en rar og velfungerende arbejdsplads, og synspunkterne udtrykt i nedenfor aftrykte rundskrivelse deles fuldt ud af DIKUs bestyrelse og af (næsten) alle lærere og studerende:

> Siden DIKUs start har vi haft frie forhold: En bygning, der stort set er tilgængelig 24 timer i døgnet, og en kantine, der ligeledes er "åben" 24 timer i døgnet for selvbetjening (inklusive selvafregning for det, man tager).
> Som studerende nyder du godt af denne frihed. Det er ganske praktisk, at man både kan låse sig ind (for eksempel i weekenden) og tillige få sig en kop kaffe eller lave mad. Men det er en frihed under ansvar — et ansvar, som alle, der anvender bygningen, kan og skal leve op til.
> Desværre indser ikke alle fuldtud dette, og derfor er det nødvendigt med ordensregler (se de generelle ordensregler for Københavns Universitet).
> For DIKUs vedkommende vil jeg særligt fremhæve følgende problemer
>
> - Eksamenssnyd, specielt i forbindelse med store opgaver; (jo, det bliver faktisk opdaget!)
> - Indtagelse af mad eller drikke i terminalrum. Det skal hermed indskærpes, at end ikke kaffekopper hører hjemme i terminalrum, PC-rum og lignende.
> - Manglende overholdelse af regler for brug af datamater, for eksempel spil på datamaterne (specielt støjende adfærdi i forbindelse hermed), misbrug af konti, omgåelse af regler angående pladelagerforbrug, misbrug af vore netforbindelser til omverdenen.
> - Manglende oprydning. Husk, at oprydderen, det er **dig**.
> - Rygning i terminalrum, undervisningslokaler eller ikke-ryger-kantinen.

- Man er forpligtet til at kunne legitimere sig på opfordring, når man befinder sig på DIKU, det vil sige vise studiekort[a]. I enkelte tilfælde har personale, der har anmodet om legitimation, været udsat for trusler — hvilket ikke alene er i strid med ordensreglerne, men også strafbart.

Slutteligt: Lad det ikke være **dig**, der bliver årsag til en opstramning af vore regler!
Hilsen institutbestyrer *Jørgen Bansler*

[a]Som anført ovenfor er instruksen til vægteren senere ændret, så gyldig adgangsdokumentation er ID-kort eller særlig tilladelse.

2.5 Universitetets administrative organer

Kompetencen vedrørende pensum, undervisning og eksamensform på datalogistudiet ligger i studienævnet for datalogi, hvis medlemmer er fem lærere og fem studerende. Nævnets aktiviteter refereres på en opslagstavle i mellemfløjens stueetage, og møderne er offentlige (se også hjemmesiden `http://www.diku.dk/undervisning/snaevn/`). Hvert efterår afholdes der valg, hvortil også studerende på Datalogi 0 GA er valgbare og har stemmeret, se opslag.

Studienævnets aktuelle sammensætning kan ses på hjemmesiden.

Fagstudienævn bør ikke forveksles med studenter-fagrådet, som er en studenterpolitisk forening (blandt flere sådanne).

Hidtil har de studerende også kunnet vælge repræsentanter til fakultetsrådet og konsistorium og haft stemmeret ved valg til posterne som rektor, prorektor, dekan og prodekan. Disse forhold kan ændre sig i takt med indførelse af den nye styrelseslov for universiteterne, der formelt træder i kraft den 1. juli 2003.

Kapitel 3

Rapportopgaver

Arbejdet med rapportopgaver tillægges en væsentlig betydning på datalogistudiet, og på Datalogi 0 GA vægtes rapportopgavekarakteren og den skriftlige prøve lige tungt. Da Datalogi 0 GA (og GB) er de kursus, hvor denne arbejdsform indlæres, indtager rapportopgaverne her en særlig vigtig stilling.

I de efterfølgende afsnit gives en kort motivering for at lade en rapport ledsage et program, nogle observationer og råd vedrørende grupearbejde, oversigter over struktur for og indhold af datalogirapporter, og til sidst præsenteres et eksempel på en besvaret rapportopgave.

3.1 Motivering

Som det nærmere forklares på dette kursus, har et datamatisk system to bestanddele: det er en EDB-maskine (materiellet) med indlagt programmel. Ud fra en snæver betragtning skal programmet kun fortolkes af maskinen, og hele systemet fungerer korrekt, hvis blot programmet giver maskinen de rigtige ordrer.

Drejer det sig derfor om at løse en bestemt opgave, som ikke senere vil ændre sig hverken med hensyn til funktionalitet eller med hensyn til, hvilket system den skal afvikles på, og hvis man kan overskue opgaven fuldstændigt og er i stand til at løse den fejlfrit alene i én uafbrudt arbejdsgang, og hvis arbejdet ikke skal bruges i forbindelse med ens egen eller andres løsning af tilsvarende opgaver ved senere lejligheder, så kan man blot lægge programmet ind i datamaten uden yderligere dokumentation.

Som konsekvens af, at disse forudsætninger aldrig alle vil være opfyldt, skal programmer ledsages af forklarende tekster. Denne dokumentation skal tjene en lang række formål:

- Den skal hjælpe til at finde og rette eventuelle opdukkende fejl.
- Opstår der nye ønsker eller krav til EDB-systemet, skal man kunne finde ud af, hvordan det kan udbygges.
- Hvis flere arbejder sammen om at konstruere EDB-systemet, har de brug for at fastholde fælles beslutninger og retningslinjer.
- Også hvor der kun er en enkelt programmør, vil processen som regel strække sig over et længere tidsrum, og dokumentationen er nødvendig, for at man kan komme ind i de rigtige tankebaner igen efter en afbrydelse.

- Hvis der er særlig smarte konstruktioner i systemet, bør de beskrives, så de ikke går tabt ved eventuelle senere ændringer, og så de kan genbruges i andre sammenhænge.

- Hvert programmelprojekt er en forberedelse til det næste. Indsigter vundet og formidlet i ét projekt kan forbedre udførelsen af kommende projekter.

Programdokumentation Afhængigt af de forskellige kategorier af personer, som berøres af et programsystem, kan man i erhvervslivet kræve mange forskellige dokumentationstyper. Ofte arbejdes der for eksempel med en *brugermanual* orienteret mod "slutbrugerne", dvs. dem, systemet i sidste ende skal betjene, en *installerings-* eller *operatørmanual*, som gør rede for tilpasning og afvikling af systemet på EDB-anlægget, og *vedligeholdelsesdokumentation* med beskrivelse af systemets indre opbygning, så det bliver muligt at rette fejl i systemet eller ændre det eller tilføje ny funktionalitet.

Læreproces Hvis præstationer skal udvikle sig og blive bedre, er det nødvendigt at gøre dem bevidste. Det blev ovenfor nævnt, at det også kan være hensigtsmæssigt at opfatte den praktiske arbejdsproces som en fortsat oplæring, men i en formaliseret undervisningssituation kommer yderligere hensyn til:

3.1.1 Rapportarbejde i undervisningssammenhæng

Typisk præsenteres en problemstilling, som det er opgaven at analysere, hvorefter et datamatisk system i forbindelse med det beskrevne genstandsområde skal konstrueres og dokumenteres.

Selv om vi forsøger at give opgaverne et realistisk skær, kommer man ikke uden om, at situationen er arrangeret: Projektets formål er ikke at tilgodese behovene hos en faktisk bruger (men derimod at demonstrere den studerendes tilegnelse af stoffet). Tvivlstilfælde, som man i praksis ville afklare i dialog med brugeren, må der derfor enten være taget stilling til i opgaveteksten, eller den studerende må træffe en afgørelse på egen hånd.

Normalt kan rapportopgaver ikke gives så stort et omfang, som praktiske projekter vil have. Opgaven er derfor meget omhyggeligt gennemarbejdet med henblik på at reducere arbejdet, så det passer med undervisningssituationen. Typisk vil opgaveteksten indholde nogle begrænsninger, anvisninger og vink, som eventuelt kan kritiseres, men som ubetinget bør følges.

I praktisk systemdokumentation kan der være brug for beskrivelser af arbejdsgange eller apparater eller andet, som ikke direkte har med det udviklede program at gøre, men i undervisningssammenhæng er det først og fremmest interessant at se det beskrevet, undervisningen har drejet sig om

Hertil kommer, at undervisning ønsker at formidle nogle anvisninger på, hvordan programmer mest hensigtsmæssigt konstrueres. Derfor bør opgavebesvarelser indeholde afsnit, der sætter underviserne i stand til at "kigge den studerende over skulderen" og vurdere, i hvilket omfang disse anvisninger er fulgt.

3.2 Gruppearbejde

(Omarbejdet efter et forlæg af Bo Kjellerup og Svend Frydenlund.) En vigtig side af datalogiuddannelsen er opøvelse af evnen til effektivt samarbejde, og midlet hertil er projektarbejde udført i grupper.

3.2.1 Gruppesammensætning

Kvaliteten af resultatet af et gruppearbejde afhænger i høj grad af gruppens evner til samarbejde. For at samarbejde kan blive effektivt, bør medlemmerne i en gruppe opfylde de samme kriterier med hensyn til datalogiske evner og ambitionsniveau.

Datalogiske evner

Det er meget vigtigt, at en gruppes medlemmer er ligeværdige, da ligeværdige personer diskuterer bedst.

To personer med ligeværdige ikke-fænomenale datalogiske evner vil ofte have to muligheder for at indgå i gruppearbejde med en tredje person, nemlig at

1. lave gruppearbejde med Karl Koder, hvor den væsentligste deltagelse i gruppearbejdet består i at heppe på og opvarte Karl Koder, eller

2. lave gruppearbejde med en ligeværdig person, hvor der arbejdes for at få lavet en perfekt analyse, en ganske god brugervejledning, en god programbeskrivelse samt en virkelig god planlægning af afprøvningen og slutteligt et så godt som det kan blive-program. Selv om programmet ikke kan bringes til at køre, kan det sagtens blive vurderet som værende ret godt. Et program bliver som regel vurderet som værende godt, hvis grundlaget for programmet er i orden.

I stort set alle de projektarbejder, det er muligt at blive udsat for, er mulighed 2 bedst, netop fordi en veldiskuteret løsning er at foretrække. Mulighed 2 er ikke nødvendigvis altid den bedste løsning, men kan oftest anvendes med fordel, hvis opgaveteksten *ikke* eksplicit forudsætter, at det udviklede program skal virke.

Mulighed 1 vil ofte kunne resultere i, at rapporten vurderes til at være bestået, medens mulighed 2 ofte vil kunne resultere i, at rapporten vurderes til at være meget god.

Ambitionsniveau

Lige så vigtigt som at have de samme evner er at have det samme ambitionsniveau. Hvis dette ikke er tilfældet, kan man komme ud for, at et problems væsentlighed bedømmes forskelligt af gruppens medlemmer, så et eller flere gruppemedlemmer sidder og triller tommelfingre, medens resten af gruppen tumler med problemet. Hvis ikke alle har samme ambitionsniveau, vil det oftest ende med, at ambitionsniveauet ender på laveste fællesnævner til stor irritation for dem med det højeste ambitionsniveau.

Et realistisk ambitionsniveau for en opgaveløsning bør fastlægges, inden besvarelsen af opgaven påbegyndes. Et ambitionsniveau, der er for højt, vil give et ufuldstændigt resultat, og et ambitionsniveau, der er for lavt, vil give en fuldstændig løsning, som man ikke synes er god nok. Ved valg af ambitionsniveau kan følgende retningslinjer eventuelt anvendes:

Tidsbelastning vil oftest være en begrænsende faktor. Derfor bør en gruppes medlemmer på forhånd blive enige om, hvor meget tid hver enkelt person skal afsætte, og hvornår fællesmøder skal afholdes.

Kvaliteten af det, der skal præsteres, bør aftales, før arbejdet påbegyndes, således at niveauet er ensartet gennem hele arbejdsprocessen. Hvis en gruppes medlemmer på forhånd er blevet enige om målet for deres arbejdsresultaters kvalitet, bør det ikke senere diskuteres, hvilken kvalitet man vil gå efter, med mindre alle gruppemedlemmer har indset, at det ønskede resultat er opnået for tidligt eller ikke kan opnås.

3.2.2 Planlægning

Et problem skal som regel løses i et bestemt tidsrum. Et problem kan oftest deles op i et mindre antal del-problemer. For at få forsøgt at løse alle del-problemer ordentligt inden tidsfristen udløber, kan det være en fordel at planlægge, hvornår de enkelte del-problemer skal løses. Derfor er det en god ide at lave en tidsplan for opgaveløsningen – og bestræbe sig på at overholde den!

For at kunne skrue en tidsplan fornuftigt sammen er det vigtigt at have opnået en overordnet forståelse for det problem, der skal løses. Sørg derfor for at finde ud af, hvad opgaven går ud på, og hvordan opgaven skal opdeles, før tidsplanen laves. Meget stramme tidsplaner kan sjældent overholdes; sørg derfor for, at der er tid i overskud ved de enkelte punkter i tidsplanen.

Jo mere uddybende en tidsplan er, jo større er sandsynligheden for, at der ikke mangler væsentlige tidskrævende punkter i den. Hav derfor følgende med i de overvejelser, der gøres i forbindelse med planlægningen:

- Egne og opgave-fastsatte tidsfrister.
- Til rådighed værende tid.
- Hvilken indsats det vil kræve at løse og besvare et del-problem, der indgår i opgaven.
- Kan det enkelte del-problem behandles af et enkelt gruppemedlem, eller skal det behandles i fællesskab?
- Hvem har tid hvornår?
- I hvilken rækkefølge de enkelte dele af besvarelsen laves. Der skal som regel laves følgende afsnit og handlinger:
  - Analyse.
  - Brugervejledning, opstilling af afprøvning og program-uafhængig del af software-designet.
  - Programmering og program-afhængig del af software-designet.
  - Gennemførsel af afprøvning.
  - Konklusion.
  - Sidste gennemlæsning, opstilling og udskrivning.

Programmet kan afhænge af analysen, brugervejledningen, software-designet samt strategien for afprøvningen og de enkelte afprøvningstilfælde. Hvis dette arbejde er lavet så godt som muligt, inden programmeringen påbegyndes, vil der stort set ikke skulle bruges tid på at rage kastanierne ud af ilden under programmeringen; ved dårligt planlagt arbejde kan dette ofte tage 90 procent af den samlede tid, der bruges på at lave besvarelsen.

3.2.3 Arbejdsform

Gruppearbejdsformen er for de fleste på førsteårskurserne på universitetet velkendt fra gymnasiet og folkeskolen som noget, der en sjælden gang er blevet anvendt og oftest har udartet sig til et af følgende to tilfælde:

1. En person laver alt arbejdet, mens de resterende spiller kort.

2. Alle i gruppen laver det samme samtidigt (hvilket vil sige, at en person tager notater, mens resten på skift citerer, hvad der skal skrives) og får på den måde lavet det samme, som et enkelt gruppemedlem kunne lave på lidt mere end den halve tid.

Ingen af tilfældene er specielt velegnede som gruppearbejdsformer. Det har ganske givet været sjovt for de fleste, men ikke specielt givende i det lange løb.

Valg af arbejdsform

En god arbejdsform er at veksle mellem at arbejde sammen og at arbejde hver for sig. Som et udgangspunkt for, hvilken arbejdsform der er mest velegnet i en given situation, kan tages, om situationen kræver, at der skal diskuteres, eller situationen kræver, at der skal skrives. En enkeltpersonsdiskussion vil oftest mangle mange vitale dele, og mere end to hænder på et tastatur giver sjældent et læsbart resultat. Jo større en del af et arbejde, der laves hver for sig, jo vigtigere er det at have styr på, hvad gruppen som helhed laver. Sørg for at synkronisere arbejdet ofte.

Retningslinjer for gruppearbejde

Det er vigtigt, at en gruppes medlemmer er enige om visse retningslinjer:

- Fælles forhold til arbejdsdisciplin (om aftaler skal overholdes, hvor præcist mødetider angives, om manglende forberedelse til møder kan tillades, arbejdsform med videre).

- Hvilke sociale aktiviteter der skal inddrages i gruppearbejdet, såsom pauser og andre uformelle aktiviteter (eksempelvis madlavning, cafe-besøg og lignende). Vær opmærksom på, hvilke fællesaktiviteter der kan betragtes som formelt rapport-arbejde, og hvor der kan forekomme løs snak, der ikke tages referat af.

- Hvorvidt man har gensidig respekt for ideer. Om man kan blive enige om at lytte frem for at afbryde. Hvornår en diskussion er afsluttet. Hvordan man bliver enige (eventuelt ved flertalsafgørelse, eller endnu bedre ved at blive færdige med at diskutere problemet).

- Om fælles grundlæggende forståelse for opgaven skal være på plads. Hvorvidt "dumme" spørgsmål til hver en tid skal kunne stilles. (Hvis de andre gruppemedlemmer heller ikke kan svare på spørgsmålene, viser det blot, at de andre gruppemedlemmer også befinder sig midt på Lars Tyndskids mark.) Ofte viser det sig, at visse spørgsmål ikke er så dumme endda. Læg mærke til, at der er forskel på "dumme" spørgsmål og irrelevante spørgsmål.

- Det kan være en fordel at veksle mellem arbejdspladser. Sørg derfor for at beslutte, hvornår I arbejder hjemme, og hvornår I vælger at benytte jer af instituttets faciliteter (såsom skærminaler, kantine, køkken, og at der er medstuderende og instruktorvagt i nærheden at spørge til råds).

Det er ret væsentligt, at alle en gruppes medlemmer følger de fælles retningslinjer for samarbejde, hvad enten de er enige i retningslinjerne eller ej.

3.2.4 MindMap

Når en ny opgave angribes, er der ved de indledende møder ofte stor usikkerhed om, hvordan problemerne skal gribes an. Den klassiske beskrivelse af, hvordan sådanne problemer løses, er gennem en systematisk nedbrydning af et problem i mindre og mere overskuelige delproblemer. Denne form kan være ønskelig, når gruppens løsning skal dokumenteres i rapporten, men er sjældent en god beskrivelse af, hvordan løsninger fremkommer.

Især de indledende diskussioner er ofte præget af at være meget ustrukturerede, idet der tilsyneladende mere eller mindre tilfældigt veksles mellem identifikation af nye problemer, en nuancering af allerede fundne problemer, mere eller mindre konkrete løsningsforslag og -skitser samt identificering af sammenhænge mellem problemer og løsninger.

Når denne kaotiske diskussion har stået på nogen tid gennem et eller flere møder, vil der ofte dannes en overordnet forståelse af, hvordan problemerne skal løses. Ofte forekommer de problemer, der i begyndelsen var uoverskuelige, nu trivielle, og det kan i visse tilfælde være svært igen at få øje på, hvad problemet egentligt var. Når analysen skal dokumenteres i rapporten, er det imidlertid vigtigt, at alle de relevante problemer analyseres. Det er derfor godt at have dokumenteret diskussionerne, så det fastholdes, hvilke problemer det er vigtigt at få med i rapporten, samt de relevante alternativer, der bør diskuteres. En sådan dokumentation letter arbejdet med rapportskrivningen væsentligt.

Til møderne kan et "mindmap" være en god måde at tage notater på.

"Mindmap"-teknikken udnytter vores associative hukommelse, idet sammenhænge og associationer noteres ned ved hjælp af nøgleord, der skal hjælpe hukommelsen til at genkalde det noterede. I stedet for hele sætninger består et "mindmap" af en række nøgleord, der er forbundet med streger, som viser sammenhængen mellem nøgleordene. Starten på et "mindmap" for en indledende analyse af et HT-takstsystem kunne se ud som i figur 3.1.

Ovalen betegner emnet for noterne. For hvert nyt emne, der drages ind i diskussionen, tegnes en streg ud fra cirklen, og nøgleordet placeres på linjen. Når mødet skrider frem, vurderes det, hvor nye oplysninger passer ind. Der er ingen faste regler for, hvordan et "mindmap" skal udformes. Når man har arbejdet med teknikken nogle gange, opfindes ofte specielle symboler eller farvekoder. Grøn kan f.eks. angive noget vigtigt, mens rød betegner noget uvæsentligt. En tankeboble omkring et ord kan angive en god ide eller et alternativ,

Figur 3.1: “Mindmap” for HT-takstsystem

en kasse kan repræsentere noget vedtaget, mens et dødningehoved kan angive noget forkastet eller en dårlig ide.

Fordelene ved “mindmap” er, at det er hurtigt at tage notater, mens man deltager i diskussionen, ligesom den tager hensyn til, at der hyppigt skiftes emne. “Mindmap”’et giver endvidere en struktur til de diskuterede emner, der kan bruges som udgangspunkt til at skrive et struktureret referat eller en rapport efter. Blandt ulemperne er, at det kan være svært at forstå et “mindmap”, der er tegnet af en anden. Endvidere kan det ikke forventes, at al information kan udledes af et “mindmap”, hvis det har ligget i lang tid. Derfor kan et “mindmap” med fordel skrives om til et referat inden for overskuelig tid. “Mindmap” er resumeret i figur 3.2.

3.2.5 Programmering

Selv om de enkelte dele af programmet formentlig er uddelegeret, så gennemlæs også de andres programafsnit! Det er vigtigt, at rapporten bevarer en fælles terminologi. Der kunne jo også være noget at kritisere eller kommentere, eller I kunne have konstrueret samme hjælpefunktion, hvilket ser klodset ud. (Med mindre funktionerne ikke umiddelbart kan genbruges, fordi de er pakket ind i strukturer.)

Bliv enige om regler for navngivning af variable, funktioner og parametre, før I programmerer; den slags kan være besværligt at rette bagefter, og en god systematik gør det faktisk også lettere at finde rundt, mens man programmerer. Bliv også enige om reglerne for indrykning, så programkoden kan fremtræde ensartet.

Figur 3.2: Resumé af "mindmap"

3.2.6 Tekstskrivning

Når en rapport skal skrives, vil der som oftest være et eller andet at gå ud fra. Eksempelvis vil der oftest være nogle gode diskussionsnoter, se afsnit 3.2.4, hvorpå en analyse kan baseres. Når man ikke står på bar bund, vil tekstskrivningen som regel ende med omtrent det samme indhold, ligegyldigt hvem fra gruppen der skriver teksten. Hvis alle en gruppes medlemmer skriver det samme samtidig, vil det ofte ende med, at man får nogle diskussioner, som man muligvis ikke ville være kommet igennem, hvis en enkelt person skrev teksten. Nogle af disse diskussioner kan være relevante for indholdet, medens de fleste diskussioner handler om tekstens formulering, der let kan rettes til senere. Skrives teksten samtidigt af alle gruppemedlemmerne, ender det ofte med en omgang ordkløveri, der kan spolere alle tidsplaner, til gengæld ender man med at blive enige om at have skrevet det samme (selvom indholdet nok ikke afspejler det i det lange løb).

En fordel ved gruppearbejde er, at man har nogle personer omkring sig, der ved, hvad man har skrevet om, og derfor vil være ganske gode til at komme med kommentarer til det, man har skrevet. Lad andre gruppemedlemmer gennemlæse den tekst, der er blevet skrevet. Sørg for at tage rettelserne alvorligt, og fortæl, hvilke rettelser der ikke kan opnås enighed om; det viser sig ofte, at enigheden ikke kan opnås, fordi et eller andet ikke er blevet gennemdiskuteret. Vær kritisk med, hvad der sendes til gennemlæsning; det er ikke meningen, at der skal rettes stavefejl, der stjæler opmærksomheden fra indholdet.

Ved gennemlæsning og retning af en tekst er der visse retningslinjer, det kan være en fordel at rette sig efter. Blandt andet:

- Sørg for at få så mange kommentarer med som muligt. Hvis noget kan kommenteres,

er det sandsynligvis forstået. Jo mere overmalet en tekst er blevet af at blive rettet de første par gange, jo mere forståelig har den været. (I sjældne tilfælde kan man komme ud for, at et første udkast er så godt, at der ikke er ret meget at kommentere.)

- Stil spørgsmål til teksten, og lad forfatteren selv besvare dem, når rettelserne indføjes i teksten.

- Læs teksten med målgruppens øjne. Det vil ofte vise sig, at et eller flere brugte begreber ikke kan forstås af målgruppen, og at visse afsnit ikke helt passer til målgruppens niveau.

- Hvis et afsnit ikke forstås ved første gennemlæsning, vil det nok heller ikke blive forstået af andre, eksempelvis en læser fra målgruppen. Sørg for at kommentere, hvorfor det er uforståeligt.

Ved indskrivning af tekst og rettelser til en tekst er der visse retningslinjer, det kan være en fordel at rette sig efter. Blandt andet:

- Lad være med at skrive i den samme fil samtidigt. Det går galt, med mindre I bruger et versionskontrolsystem såsom cvs!

- Hvis en tekst ikke får ret mange rettelser, tyder det på, at teksten ikke er forstået af de øvrige gruppemedlemmer. Jo flere rettelser, jo mere af teksten er forstået af modtageren.

- Benyt muligheden for at tilføje tekst, der blev glemt i første omgang, eller er blevet oplagt som følge af rettelserne.

- Tag alle rettelser alvorligt. Hvis en rettelse forkastes, fortæl da retteren hvorfor.

(Slut på Bo Kjellerups og Svend Frydenlunds afsnit.)

3.3 Struktur og indhold

En teknisk rapport om et programmelprojekt skal formidle *alle relevante* resultater og indsigter i forbindelse med projektet og *ingen irrelevante* dele. Det er givetvis svært at afgøre, hvad der er "relevant", og hvad der ikke er, men det er vigtigt at huske, at det at skrive *mere* ikke altid forbedrer kvaliteten af rapporten og *ofte* endda forværrer den. Irrelevante detaljer stiller sig i vejen for en *klar* fremlæggelse af de relevante dele og gør deres formidling vanskeligere.

Rapporten skal være bygget logisk op og være så kompakt som muligt. Den skal dog indeholde alle væsentlige beslutninger samt relevante afvejninger og redegørelser.

Rapporten skal få programmet til at fremstå som resultat af en bevidst og systematisk konstruktionsproces. Det domæne, hvori programmet skal virke, må have været analyseret grundigt, og en fuldstændig afprøvning skal have sikret mod indtastningsfejl. Først og fremmest må hver programdels indre virkemåde være sikret korrekt gennem en passende kombination af ræsonnementer og test.

Ideelt set består arbejdet af en analysefase, som afklarer opgaver og krav, efterfulgt af en syntesefase, hvorunder systemet gradvis opbygges i stadig flere detaljer. Rapporten kunne have følgende opbygning:

1. *sammenfatning*
2. *problemorienteret analyse*
3. *programmeringsovervejelser*
4. *programbeskrivelse*
5. *afprøvning*
6. *brugervejledning*

men afhængig af naturen af programmelprojektet kan andre opbygninger være mere hensigtmæssige. Nedenfor gives nogle uddybende beskrivelser af enkelte dele; derefter følger nogle råd om rapporters form.

3.3.1 Sammenfatning

Sammenfatningen er på et enkelt afsnit og formidler kortfattet projektets formål, indhold og resultat.

Denne del af rapporten skrives sidst, og man finder undertiden disse oplysninger i et afsluttende kapitel under overskriften "konklusion". Da dette kapitel imidlertid ofte vil være det første, en læser blader hen til, anbefaler vi her, at det placeres forrest.

3.3.2 Problemorienteret analyse

I analysefasen finder man frem til kravene til det system, der skal udvikles.

Et program kan opfattes som en model af en isoleret del af virkeligheden, og analysen skal beskrive denne model, så senere rettelser og tilføjelser kan bygge videre på den uden at "forbedre" til det værre.

Problemstillingen vil ofte være åben eller uklar, og i så fald skal rapporten indeholde en analyse af det stillede problem med opdeling i underproblemer og en afgrænsning af, hvad der præcis tages op til maskinel behandling. Under analysen gøres rede for de uklarheder, selvmodsigelser eller ufuldstændigheder, der er fundet i opgavens formulering, og de beslutninger, der er truffet for at udbedre disse mangler, med begrundelser for de trufne beslutninger.

Eventuelt kan der være en kort diskussion af alternative muligheder.

Under analyse af det domæne, programmet skal virke i, kan det undertiden være relevant at gøre rede for, hvilke begreber man har udisoleret fra det, og forklare, hvilke områder de fundne størrelser varierer indenfor, og hvilke sammenhænge der er imellem dem. Det gælder om at afdække alle problemdomænets egenskaber og lovmæssigheder.

Selv om analysen fører frem til, hvilke dele der skal være i det konstruerede program, bør den kunne formuleres i generelle termer, der ikke er specifikke for noget bestemt programmeringssprog, ligesom analysen under ingen omstændigheder må henvise til programmet.

Analysen skal føre frem til en fuldstændig afklaring af, *hvad* programmet skal gøre, uden nogen forpligtelse til, *hvordan* det skal gøres.

Man kan forestille sig den problemorienterede analyse som et selvstændigt dokument, der er så entydigt, at lod man en anden bruge det som grundlag for et system, der eventuelt endda kunne skrives i et andet programmeringssprog, ville resultatet alligevel få samme funktionalitet som det aktuelle system.

3.3.3 Programmeringsovervejelser

Dette kapitel er bindeled mellem kravspecifikation og implementering og kan også benævnes systemorienteret analyse, programkonstruktion eller "software design". Her begrundes og redegøres for, hvorledes problemdomænets størrelser repræsenteres og behandles i det datamatiske system (herunder i det benyttede programmeringssprog).

Formålet er at give en mere overordnet beskrivelse af, hvordan systemet virker, end den som ligger i selve programteksten, og at finde et passende niveau er en stor udfordring. Tegninger og diagrammer kan ofte være til hjælp i formidlingen.

Ved større opgaver kan beskrivelsen for eksempel opdeles i

1. Systemarkitektur: Systemets overordnede opbygning

2. Modulbeskrivelser: De enkelte modulers funktionalitet

3. Algoritmer og datastrukturer: Valg og opbygning af datastrukturer og algoritmer ved implementering af modulerne.

(Andre former for strukturering er mulige og kan i konkrete tilfælde være mere hensigtsmæssige.)

Man kan sige, at programmeringsovervejelserne skal kunne bruges som programmeringsgrundlag, sådan at hvis man gav dem til en anden, ville vedkommende kunne konstruere et system, der omend det ikke blev magen til det aktuelle dog både fik samme funktionalitet og samme struktur.

3.3.4 Programbeskrivelse

Programbeskrivelse (eller "implementering", som kapitlet også kan benævnes) fungerer som en læsevejledning til programmet.

Målgruppen er programmører, der senere vil få til opgave at vedligeholde eller ajourføre programmet. Programbeskrivelsen kan altså skrives til en læser med samme forudsætninger som en Datalogi 0 GA-studerende.

En programbeskrivelse skal supplere og ikke dublere kommentarerne i programmet. Er programmet velstruktureret og godt kommenteret, kan programbeskrivelsen i mange tilfælde gøres kort.

Indholdet af programbeskrivelsen vil variere fra program til program, men kan alt efter størrelsen og kompleksiteten indeholde nedenstående dele. Det er vigtigt at understrege, at det i hvert enkelt tilfælde må vurderes, hvilke dele der skal med. Er programbeskrivelsen ikke kort og præcis, kan læseren miste overblikket.

Udviklingsværktøj Går der mange år, førend programmet skal vedligeholdes, kan det være praktisk med informationer om, hvilket programmeringssprog og hvilket oversættersystem (navn og version), der er benyttet.

Læsevejledning Denne bør indeholde både en syntaktisk og semantisk læsevejledning. En syntaktisk læsevejledning er nærmest en indholdsfortegnelse til programudskriften, med beskrivelse af programmets statiske struktur. En semantisk læsevejledning beskriver programmets overordnede virkemåde og sammenhængen mellem de enkelte programdele.

Figurer kan medvirke til et godt overblik over programmet.

Typer og datastrukturer Dette afsnit skal beskrive de benyttede typer samt de datastrukturer, der har væsentlig betydning for programmet. Hvis der benyttes større datastrukturer, bør der være en forklaring til, hvordan de bruges i programmet, gerne med eksempler og figurer, så typerne hurtigt kan overskues.

Vær omhyggelig med navne. Giv typer, variable, funktioner og parametre navne, der siger noget om deres indhold og anvendelse. Brug så vidt muligt enten danske eller engelske termer og ikke en blanding. (Man kan dog ikke komme uden om bibliotheksfunktionernes engelske navne.)

Hvis den systematik, der er brugt til navngivning i programmet, ikke er indlysende, skal den forklares.

Afvigelser fra standarden Hvis der er benyttet faciliteter i programmeringssproget, som ikke er dokumenteret i den anvendte litteratur, skal disse nævnes. Ligeledes skal anvendte egenskaber, der er særlige for den benyttede oversætter, begrundes. Hvis det i stedet havde været muligt at anvende standarden, bør det begrundes, hvorfor den ikke er fulgt.

Enkelte programdele Som hovedregel bør funktioner beskrives gennem kommentarer i programmet. Er der imidlertid dele af programmet, som er vanskelige at forstå, kan det være relevant at fremdrage dem her.

Hvis der benyttes særlige repræsentationer eller vanskeligt gennemskuelige "tricks", skal de forklares.

Har der været problemer med at skrive programmet, bør disse problemer anføres. Der kan for eksempel være tale om funktioner, der ikke virker, eller funktioner, der kræver særlig opmærksomhed, for eksempel i forbindelse med manglende robusthed.

Programmets kildetekst

En udskrift af selve programmet skal med som bilag. Er der tale om et større programsystem, bør der være tydelige overskrifter til de forskellige dele, sammen med korte kommentarer om delenes indhold og formål. Med mindre det er *meget* indlysende, skal hver funktion ledsages af en kort kommentar om, hvad den gør.

Det er vigtigt at stille programmet læsevenligt op. Brug sigende navne; ryk ind på linjerne, når noget nyt begynder, og ud igen, når det slutter; hold beslægtede ting sammen, og skil uafhængige dele.

Omfangsrige kommentarer i programmet bør undgås og erstattes af henvisninger til det relevante afsnit af rapporten (i problemorienteret analyse, programmeringsovervejelser eller programbeskrivelse).

3.3.5 Afprøvning

Overordnet skal dette afsnit argumentere for, at programmet er korrekt (eller — hvis det ikke er: klart vise, hvilke dele der virker, og hvilke der ikke gør).

Først og fremmest skal beskrivelsen derfor forklare, hvilken tankegang der ligger til grund for afprøvningen, og hvilke betingelser der ville skulle opfyldes, for at man kunne hævde, programmet var i orden.

Afprøvningsarbejdet lettes naturligvis, hvis programmet er opbygget af små uafhængige moduler (der da kan sikres korrekte hver for sig), men under alle omstændigheder bør programmet være konstrueret, så det er "afprøv-bart".

Man skelner undertiden mellem *ekstern* og *intern* afprøvning:

Ved *ekstern afprøvning* tages opgavespecifikationen som udgangspunkt, mens programmet betragtes som en uanalyseret helhed, en "sort kasse". Programmet underkastes derfor data, som enten er typiske for problemet eller er valgt omkring yderpunkterne af de områder, hvori inddata varierer.

Ved *intern afprøvning* går man ud fra programteksten, og først og fremmest vælges testdata, så alle forgreninger i hver valgkonstruktion afprøves, og så udførelse har været gennem alle programmets dele. (Kan det ikke lade sig gøre, er der jo overflødige dele, som bør fjernes!) Intern afprøvning godtgør således programmets "indre sammenhæng" uden hensyn til, om dets virkning er den ønskede.

Hvad afprøvningen har vist om programmets fejl og mangler, konkluderes sidst i afsnittet.

Omfanget af testudskrifter bør begrænses mest muligt (uden tab af informationsindhold), og lange udskrifter kan det eventuelt være praktisk at placere bagest som bilag.

En instruktor har udarbejdet følgende vejledning:

Ekstern afprøvning

(Forfatter: Erik Dam.) Afprøvning er ofte det, man laver til sidst i en rapport. Ikke fordi det er en god ide at udskyde det — det er det ikke — men ofte er man under tidspres, og motivationen er heller ikke altid stor. Hvis programmet kører, kan det virke som spild af tid, og hvis det ikke gør, vil man hellere bruge tiden på at få det til at virke. Men afprøvning behøves ikke at tage lang tid. Faktisk er det heller ikke nødvendigt at vente til sidst i rapportfasen med at lave en ekstern afprøvning. Lige så snart, man har lavet sine specifikationer, det vil sige beskrevet, hvad programmet skal kunne, kan man lave afprøvningen. Man kan selvfølgelig ikke køre sine tests, men det er også den mindste del af det, når man har bygget dem ordentligt op. Dette afsnit er et forsøg på at give et overblik over, hvad begrebet ekstern afprøvning dækker over. Derudover er der nogle gode råd, der er direkte relevante for projektarbejde her på DIKU.

Intern kontra ekstern afprøvning

Først må det være på sin plads at få fastslået forskellen mellem de to vigtigste typer afprøvning, vi opererer med på DIKU.

En *intern afprøvning* bruger selve programteksten som udgangspunkt. Uden af gå i for mange detaljer så checkes her, om programmet er "fornuftigt" programmeret.

En *ekstern afprøvning* tager derimod udgangspunkt i kravene til et program og afprøver, om programmet lever op til disse krav. Selve udseendet af programmet er irrelevant, blot det virker korrekt. Derfor kaldes denne afprøvning også *black box*-afprøvning, idet programmet blot opfattes som en sort kasse.

For en ordens skyld skal det understreges, at ekstern afprøvning kan foretages på flere niveauer: for eksempel på system-, modul-, eller funktions-niveau. Systematikken er nøjagtigt den samme — afprøvningerne foregår blot på forskellige abstraktionsniveauer.

Formål med ekstern afprøvning

Der er to overordnede formål med en ekstern afprøvning:

Fejlfinding Enhver, der har lavet programmer på over 10 linjer, har set, hvordan fejl kan opstå de mest overraskende steder. For at checke alle krinkelkroge i større programmer er det derfor bydende nødvendigt med en systematisk metode, der sikrer, at hele programmet afprøves. Ideelt set bør denne metode være uafhængig af programmørens (subjektive) forventninger til, hvor i programmet der kan være fejl.

Dokumentation Det andet formål med afprøvningen er at gøre det muligt at **overbevise en udenforstående** om, at programmet er (rimeligt) fejlfrit. Denne udenforstående kunne eksempelvis (grebet frit ud af luften) være en censor. Dette stiller yderligere krav til, at metoden skal være systematisk og gennemskuelig.

Hvad skal afprøves ?

Det skal afprøves, at programmet virker korrekt. Det vil i praksis sige, at programmet skal overholde alle *krav* fra *opgavetekst*, *problemanalyse* og *brugervejledning*. Det skal dog understreges, at det er overholdelsen af *specifikationen*, det vil sige problem- og systemorienterede krav, der skal afprøves. Programorienterede krav, for eksempel vedrørende algoritmer og datastrukturer, skal *ikke* afprøves i en ekstern afprøvning, med mindre disse er del af systemspecifikationen. (Dette kunne være tilfældet, hvis man skulle udvikle et modul med funktioner, som andre programmører skulle kunne udnytte i deres programmer.)

Strategi

Ud fra kravene formuleres en *afprøvningsstrategi*, der anviser, hvordan det kan afprøves, at alle kravene bliver overholdt. Strategien kan eksempelvis omfatte en fornuftig *gruppering af kravene* til programmet. For hver gruppe kan man diskutere, hvordan man kan konstruere *inddata* til afprøvning af kravene i gruppen, hvordan *uddata* kan *dokumenteres*, og endelig hvordan det kan *kontrolleres*, om kravene er overholdt.

En meget vigtig del af strategien er *begrænsning af antallet af afprøvningstilfælde*. I praksis er det uoverkommeligt at afprøve programmet med alle de mulige inddata. Derfor skal det diskuteres, hvordan de mulige inddata til programmet kan grupperes i klasser af afprøvningstilfælde, således at det må formodes, at programmet behandler alle tilfælde i en klasse ensartet. Klasserne skal altså konstrueres, således at hvis programmet virker korrekt med ét tilfælde fra klassen, så må det formodes, at programmet også virker med alle andre tilfælde fra klassen. Disse klasser betegnes *ækvivalensklasser*.

Erfaringen viser, at fejl i programmet ofte opstår i overgangene mellem de forskellige ækvivalensklasser. Derfor kan man med fordel opstille afprøvningstilfælde, der ligger i dette grænseland. En gennemgang af, hvilke afprøvningstilfælde der ligger på grænsen mellem flere ækvivalensklasser, benævnes en *grænseværdianalyse*.

Ovenstående begreber er meget generelle. Dette skyldes ganske enkelt, at hverken ækvivalensklasseinddelingen eller "grænselandet" mellem klasserne er entydigt bestemt. Derfor er det en meget svær og udfordrende opgave at finde fornuftige ækvivalensklasser og grænseværdier. Her gælder det om at bruge sin fantasi, intuition og sunde fornuft, og det er derfor, at man siger, at det er en kunst af afprøve EDB-programmer.

Opstilling af afprøvningen

Når man har lavet sin strategi, skal man ud fra anvisningerne i denne opstille sin konkrete afprøvning. For hver ækvivalensklasse skal udvælges et eller flere *afprøvningstilfælde*.

Hvis der er fundet grænseværdier for en given ækvivalensklasse, skal disse bruges som afprøvningstilfælde for denne klasse. Hvis der ikke er fundet fornuftige grænser for ækvivalensklassen (for eksempel er en mængde ikke ordnet, og er derfor uden naturlige grænser), afprøver man blot med et vilkårligt medlem fra klassen.

Af hensyn til overskueligheden skal afprøvningen opstilles systematisk i *skemaer*. For hvert krav til programmet laves et skema, der henviser til de ind- og uddata, der dokumenterer afprøvningen af kravet. Da det skal være muligt for læseren at gennemskue systematikken, skal for hvert afprøvningstilfælde kort beskrives de karakteristiske egenskaber ved inddata, samt de forventede egenskaber ved uddata. Desuden skal der laves en kolonne, hvor det senere angives, om de faktiske uddata er lig de forventede.

Afvikling

Dette er langt den mest trivielle og kedelige del af afprøvningen. Den ligger typisk i slutningen af rapportperioden, hvor man efterhånden er godt træt af den tåbelige opgave. Derfor er der en ret overhængende risiko for, at man overser fejl eller endda sletter dele af afprøvningsmaterialet. Klokken fire om natten er det ofte også meget nemt at argumentere for, at den sidste halvdel af afprøvningstilfældene faktisk er ret overflødige.

For at sikre, at man med en *overkommelig arbejdsindsats* får lavet en tilstrækkelig *overbevisende afprøvning*, er det derfor vigtigt at sætte arbejdet i system.

For det første kan man spare megen energi i selve *konstruktionen af inddatasættene* svarende til de enkelte ækvivalensklasser. Man kan ofte dække mange ækvivalensklasser med samme inddatasæt. Man skal blot huske på, at man aldrig må kombinere afprøvningstilfælde, der forventes at resultere i en fejlsituation. Det skal nemlig være muligt at identificere, hvilket afprøvningstilfælde der resulterede i fejlsituationen.

For det andet gør man sig selv en stor tjeneste, hvis man *automatiserer* selve afviklingen af programmet med de forskellige inddatasæt. På en PC kan det ske ved hjælp af en `BAT`-fil, mens man på UNIX-systemet kan lave et `shell script`. Den nemmeste løsning er dog ofte at kode afprøvningen som del af selve systemet. På denne måde kan man have adgang til programinterne datastrukturer og behøver ikke at læse inddata til de programdele, som skal afprøves, fra en ekstern fil.

Hvis et program kun modtager inddata fra filer, kan man faktisk afvikle hele afprøvningen tilbagelænet med fødderne solidt plantet på skrivebordet. Samtidig er automatiseringen en kæmpe fordel, hvis man senere er nødt til at køre afprøvningen om igen. Det kan eksempelvis skyldes, at man har fundet og rettet en afgørende fejl i en central del af programmet. For en ordens skyld skal det nævnes, at hvis et program modtager alt sin inddata fra tastatur eller mus, så er det muligvis umuligt at automatisere afviklingen.

Endelig skal det understreges, at afprøvningen ideelt set bør udføres af en anden person end programmøren. Der er en tendens til, at udenforstående finder langt flere fejl. Dette skyldes, at hvis programmøren har glemt et krav til programmet under programmeringen, så vil han/hun sandsynligvis også glemme det i afprøvningen. Samtidig afsløres også, hvis gruppen er uenig om detaljer i programmets ønskede virkemåde.

Dokumentation

Som tidligere nævnt er et af formålene med afprøvningen at overbevise læseren om programmets korrekthed. Ud over at være grundig skal afprøvningen derfor også virke *overbevisende*. Dette indebærer for det første, at strategien skal være fornuftigt beskrevet. Men lige så vigtigt er det, at skemaerne og de tilhørende ind- og uddata er *troværdige*.

Det er derfor meget vigtigt, at skemaerne er *overskuelige*. Det skal være tydeligt, hvilken egenskab ved et bestemt krav til programmet, som der afprøves i hvilket afprøvningstilfælde. Samtidig skal det være fuldstændigt entydigt, hvor henholdsvis ind- og uddata er dokumenteret. Det er derfor en god ide at forklare læseren, hvordan skemaerne skal læses. I hvilket bilag finder man inddata? Hvordan er *referencerne* til bilagene bygget op? Og så videre.

Selve ind- og uddata skal også præsenteres overbevisende. Det betyder, at de skal være så *rå* som overhovedet muligt. Man skal altså ikke redigere i dem for at spare plads, eller for at sætte små forklarende tekster ind mellem linjerne.

Endelig er det meget, meget vigtigt, at afprøvningen fremstår *redelig*. Med andre ord: Hvis det ser ud, som om I har snydt, så bliver I slagtet!

Konklusion

Afprøvningsafsnittet skal sluttes af med en sammenfatning af afprøvningsresultaterne. Hvis der er fundet fejl, skal det nævnes. Det kan også være en god ide at nævne fejl, som er blevet rettet. Disse fejl er godt nok ikke længere synlige i programmet, men afprøvningen virker ofte mere overbevisende, hvis den har afsløret nogle "gode" fejl.

3.3.6 Brugervejledning

(Delvis omarbejdet efter et forlæg af Svend Frydenlund.) Mens den øvrige del af rapporten henvender sig til en læser med programmeringskendskab, har brugervejledningen en anden

status: Dette afsnit skal udformes med henblik på en person, der kunne ønske at benytte programmet uden at sætte sig ind i flere af dets detaljer end strengt nødvendigt. Der bør ikke henvises fra brugervejledningen til andre dele af rapporten (og slet ikke til programteksten).

Man kunne argumentere for, at brugervejledningen som det første, der skulle læses, burde placeres forrest, men vi foreslår i denne vejledning, at afsnittet placeres efter de andre, fordi man normalt vil udarbejde brugervejledningen til sidst, efter at programmet er konstrueret og afprøvet. (Selv om den endelige udformning af en brugervejledning må ligge til sidst, kan man undertiden have glæde af at skrive et udkast til brugervejledning på et meget tidligt stadium af opgaveløsningen for som et led i problemanalysen at overveje, hvordan det planlagte system skal tage sig ud for en bruger.)

Typisk kan man give brugervejledningen tre dele:

- en kort beskrivelse af programmets funktion — hvilken nytte, det kan gøre
- vejledning i at udforme inddata til programmet, med omtale af form og indhold
- beskrivelse af programmets uddata: hvilken form de har, og hvordan de skal forstås. Uddata vil både sige egentlige resultater (i henhold til programmets specificerede virkning) og fejlmeldinger, der er resultat af inkonsistente eller på anden måde fejlagtige inddata.

Disse forklaringer gives ofte bedst gennem eksempler.

I brugervejledningens indledning kan det være hensigtsmæssigt at beskrive, hvilke forudsætninger brugeren forventes at have, for eksempel i form af den arbejdssituation, programmet skal fungere i, eller den forhåndsviden, der skal være til stede hos en bruger. Ofte antages brugeren at være uden kendskab til EDB, men derimod at være godt inde i det problem, programmet løser. Ofte vil det være overladt til forfattergruppens sunde dømmekraft at afgøre, hvilken type bruger der skrives til.

Derefter er det vigtigt, at man tager udgangspunkt i brugerens situation og overvejer, hvad vedkommende har brug for at vide, og hvordan det skal præsenteres. Særlig vigtigt er det at overveje ordvalget: er brugeren ikke EDB-kyndig, er EDB-tekniske fagtermer bandlyst; brug i stedet velkendte ord og begreber fra brugerens fagområde eller arbejdssituation.

Her på Datalogi 0 GA[1] skal brugervejledningen kun beskrive, hvad det konstruerede program selv er ansvarligt for. Man skal altså for eksempel *ikke* beskrive, hvordan der tændes eller slukkes for systemet, hvordan tastaturet er indrettet, hvordan SML-systemet aktiveres, hvordan man kopierer filer eller lignende.

3.3.7 Rapporters form

På grund af fagets stilling som hjælpevidenskab kommer de allerfleste, som beskæftiger sig med databehandling, ud for at skulle forklare sig over for andre, og evnen til pædagogisk fremstilling hører med til en professionel datalogs færdigheder.

[1] I praktiske erhvervssammenhænge kan der gælde andre regler

Sprog. Sørg for at forklare tingene kort og præcist, og undgå indforstået eller poppet sprog. Brug så vidt muligt danske betegnelser, og sæt uoversatte engelske termer i anførelsestegn. Sprogfejl (forkludrede sætninger, stavefejl, kommafejl) virker distraherende og sinkende. Med til at kunne udtrykke sig hører at skrive fejlfrit dansk.

Vær konsekvente i ordvalget, således at samme ting altid benævnes på samme måde[2]. Det viser sig meget ofte, at gode forklaringer kræver en opgavespecifik terminologi. Ved at bruge lidt ekstra plads på at indføre nogle præcise begreber kan man ofte afkorte senere forklaringer betydeligt. Derfor: vær dristige nok til at definere udvalgte termers betydning i jeres rapport, og brug de indførte termer i forklaringerne. Figurer og eksempler kan ofte hjælpe med til at forklare ellers vanskelige ting, men de skal udarbejdes med omhu for ikke at forvirre mere, end de gavner. Figurer bør ikke erstatte en forklaring.

Gentagelser skal undgås: henvis i stedet til det sted i rapporten, hvor emnet tidligere blev behandlet. Ud over at mindske læsbarheden og øge omfanget kan gentagelser gøre teksten upræcis (eller ligefrem selvmodsigende).

Struktur. For de fleste vil den største vanskelighed ved skriftlig fremstilling nok ligge i at finde en hensigtsmæssig disposition: I de første udkast viser det sig, at man har behandlet samme forhold flere steder i rapporten, og først i takt med, at man får overblik over opgaven, finder man ud af, hvilke dele der hører sammen.

Som allerede anført er de seks ovennævnte hovedafsnit som regel forlangt i Datalogi 0 GA-rapporter, men ellers påhviler opdeling af besvarelsen i afsnit og underafsnit den studerende selv. Det er meget vigtigt, at dette gøres med omhu. Opdelingen skal tjene til, at læseren kan få det overblik, der er nødvendigt for at forstå den plads, hver detalje skal have i helheden. Hvert underafsnit bør udformes således, at det behandler en velafgrænset del af problemet — en del, som bekvemt kan overskues under ét. Underafsnittet bør gives en omhyggelig valgt overskrift, som kort og klart identificerer indholdet, så læseren senere bliver mindet om det blot ved at se overskriften. Inddeling i kapitler, hver med underafsnit, kan være hensigtsmæssig. (Men mere end tre niveauer er sjældent af det gode.)

Underafsnit, der udformes efter disse krav, vil ofte blive korte: mindre end en side og kun sjældent mere end et par sider. Hvis længden vokser ud over denne grænse, bør man forsøge at foretage en yderligere opdeling af stoffet.

Klarhed. I udformningen må der frem for alt lægges vægt på klarhed og tydelighed — det dunkelt sagte er det dunkelt tænkte. Tilstræb korthed og overskuelighed, og undgå lange og indviklede forklaringer. Erstat dem med en klarere opdeling af problemet, eventuelt under indførelse af nye hjælpebegreber. Vær konsekvente: brug altid samme term om samme begreb, selv om fremstillingens sprog måske derved taber i smagfuldhed.

Ved den efterfølgende bedømmelse vil uklarhed i sig selv være at regne som en fejl, der ikke kan opvejes, selv om den uklare fremstilling rummer en værdifuld tanke.

Præsentation. Al tekst i besvarelserne skal være skrevet på maskine. En traditionel skrivemaskine kan benyttes, og i så fald accepteres overstregninger og sammenklipninger (men

[2]Kravene til tekniske rapporter er andre end til skønlitterære tekster, hvor smukt sprog ofte skal være varieret.

ikke håndskrevne tilføjelser), men de fleste vælger at bruge et tekstbehandlingssystem, hvor rettelser, tilføjelser og omplacering af afsnit er væsentlig enklere at foretage. Desværre er det i næsten alle tekstbehandlingssystemer meget tungt (om overhovedet muligt) at medtage tegninger og diagrammer. Sideløbende med sine talrige fordele har den udstrakte brug af tekstbehandling den ulempe, at man næsten er holdt op med at lave figurer, hvilket er meget beklageligt, da mange udredninger vinder umådeligt i klarhed ved at blive ledsaget af illustrationer.

Tegninger, figurer, diagrammer og lignende i rapporten kan godt være håndskrevne og bør inkluderes, hvor det er relevant.

Der skal kun skrives på den ene side af papiret, og besvarelsen må ikke hæftes sammen, men skal indlægges i et klart chartek med to åbne sider.

Forside og paginering. Det er naturligt at indlede sin besvarelse med titel, forfattere, dato og lignende oplysninger. Typisk vil man gerne skrive noget i retning af "Dette er en besvarelse af den og den rapportopgave på Datalogi 0 GA, udarbejdet af de og de studerende ..." Da vi både på Datalogi 0 GA og på andre datalogikurser modtager op imod hundrede af den slags besvarelser hvert semester og er forpligtet til at bedømme dem og derefter opbevare dem i fem år, er det hensigtsmæssigt at give dem et præg, der gør det let at identificere besvarelser af samme opgave og at skelne dem fra andre kursers opgaver.

Derfor udleveres sammen med opgaveteksten fortrykte sider, der skal udfyldes og underskrives og benyttes som side 1 i besvarelsen. **Standardforsiden indeholder alle de nødvendige administrative oplysninger, *som derfor ikke skal gentages.*** Meningen er, at de studerende på side 2 kan gå direkte til sagen uden yderligere indledning.

Vi forlanger, at rapporten pagineres fortløbende. Man kan derfor ikke nøjes med at nummerere de enkelte kapitler hver for sig (à la side 1.1-1.5, 2.1-2.4 og så videre), som det undertiden bruges i tekniske manualer (for at det skal være lettere at udskifte enkeltdele separat). Da standardforsiden er side 1, skal resten af rapporten pagineres fra side 2 og opefter. Hvis ens tekstprogram ikke er i stand til at nummerere på den måde, må man sætte sidenumrene på i hånden. Pagineringen skal omfatte alle dele (og alle dele betyder *alle dele*) af rapporten, inklusive bilag, programmer, udskrifter og så videre.

Opstilling. Det skal være let at orientere sig i rapporten. Som udgangspunkt er det vigtigste, at opstillingen er konsekvent, så den letter læserens overblik. Nedenstående anføres en række gode råd, der kan medvirke til en god opstilling.

Afstande Ved valg af marginer og linjeafstande er det vigtigt, at der er plads til notater og til eventuel hæftning, hulning og lignende. Både i siderne og i top og bund bør der derfor være en afstand på cirka 25 mm mellem teksten og papirets kant.

Skrifttyper Som hovedregel bør der kun anvendes én skrifttype. Det kan dog være en god ide at bruge en afvigende skrifttype til programdele, der er hentet ind i hovedteksten.

Brug fremhævelse af ord og linjer med omtanke. Skal flere linjer fremhæves, er indrykning ofte en bedre ide end kursivering eller fed skrift. Skriftstørrelsen bør være omkring 12 punkter og som absolut minimum 10 punkter.

Programmer og dokumentation af EDB-kørsler skal skrives med en "skrivemaskineskrift" (med fast bredde af skrifttegnene), men kan eventuelt være reduceret til halv størrelse (så to oprindelige sider nu kan stå på den anden led på en side).

Nummerering Alle kapitler, afsnit, bilag og figurer bør nummereres, så brugeren lettere kan finde rundt i rapporten. Nummereringen skal afspejle rapportens struktur, men det er sjældent nogen god ide at anvende mere end 3 afsnitsniveauer. Bilagene bør nummereres, så henvisninger fra hovedteksten er entydige. (Bilag "nummereres" ofte med bogstaver.) Figurer nummereres normalt ud fra kapitlet, således at figur nr. 2 i kapitel 6 betegnes "Fig. 6.2".

Overskrifter bør være beskrivende for indholdet af afsnittet. Der kan med fordel anvendes en større eller fremhævet skrift for at synliggøre dem for læseren. Der bør endvidere være en vis ekstra afstand til det foregående afsnit. (Denne afstand kan varieres afhængigt af overskriftsniveauet.) Kapitler (overskriftsniveau 1) bør begynde øverst på en ny side.

Indholdsfortegnelse vil for rapporter på Datalogi 0 GA med de krævede standardafsnit blot være irriterende spild af plads.

I større rapporter bør der dog altid være en indholdsfortegnelse. Den bør også omfatte en oversigt over bilagene, og derudover kan der efter behov være yderligere oversigter (for eksempel over tabeller, figurer, symboler, definitioner.) Figuroversigten medtages ofte på en separat side.

Referencer og litteraturliste Hvis der refereres til forskellige kilder, er det vigtigt at angive en reference i teksten. Typiske kilder er opgaveformuleringen og lærebøgerne. Alle anvendte kilder bør fremgå af en litteraturliste, der placeres bagest i rapporten (så den er let at finde frem til). Litteraturlisten sorteres alfabetisk efter hver references førstnævnte forfatter. For hver kilde bør der anføres forfatter(e), titel, udgave, forlag, by og udgivelsesår.

Redelighed i besvarelser

Redelighed er et fundamentalt krav til videnskabelige redegørelser og betyder, at man tilstræber at fremlægge fakta uden at forsøge at skjule mangler.

Såfremt dele af en besvarelse er overtaget fra andre (for eksempel fra bøger eller andre publikationer eller fra venner eller medstuderende), skal det tydeligt angives, hvad der er overtaget og hvorfra. I tvivlstilfælde bør man naturligvis hellere angive for meget end for lidt. Uredelig anvendelse af andres arbejde, det vil sige uden angivelse af kilden, er ikke blot en fejl, men et brud på studiedisciplinen og vil blive straffet hårdt. På den anden side er redelig anvendelse af videnskabelig litteratur jo netop studiets og videnskabens inderste væsen. Det kræves ikke direkte i alle sammenhænge på Datalogi 0 GA, men vi opfordrer hertil.

Den grad af færdiggørelse, der er nået i besvarelsen af den stillede opgave, skal beskrives tydeligt. Specielt skal det ved opgaver, der sigter mod at udvikle et program, siges klart, såfremt programmet slet ikke, eller kun delvis, er blevet bragt til at køre. Det er

ikke tilstrækkeligt, at graden af færdiggørelse kan afledes gennem en nøjere granskning af kørselsudskrifter.

Redegørelsen for graden af færdiggørelse hænger sammen med den valgte formuleringsmåde. Såfremt formuleringen følger den løsningsproces, der er gennemløbet under arbejdet med besvarelsen, hvilket ofte vil være naturligt i de afsnit, der behandler problemanalysen og afprøvningen, vil færdiggørelsen umiddelbart fremgå. Hvis derimod formuleringen beskriver det tilstræbte resultat, typisk i brugervejledning og programbeskrivelse, vil det normalt være nødvendigt at tilføje særlige oplysninger om, hvad der faktisk er nået.

Sikkerhedskopi

Som sikkerhed mod bortkomst af skriftligt arbejde ved fejltagelser, brand, tyveri og andet kan opgaveteksten kræve besvarelser afleveret i to enslydende eksemplarer. Det ene eksemplar betegnes *hovedeksemplar*, det andet *sikkerhedskopi*. (Hvis de to eksemplarer ikke er helt ens, vil det være hovedeksemplaret, som er gældende.)

En anden mulighed for sikring er, at opgaveteksten kan forlange, at et erstatningseksemplar kan fremskaffes med 24 timers varsel. (Derved kan fotokopiering undgås i de tilfælde, hvor den studerende opbevarer besvarelsen på elektronisk form.)

Bemærk: hvis alle kommer i sidste øjeblik, vil det ikke være muligt at komme til at kopiere på instituttets kopimaskine og få afleveret inden fristens udløb. Det er helt og holdent den enkelte studerendes ansvar, at hans eller hendes besvarelse afleveres rettidigt.

3.3.8 Ressourceforbrug

Det forudsættes, at såvel valget af løsning som arbejdsform under udarbejdelsen af den sker under hensyn til en rimelig ressourceøkonomi.

Under ressourcer regnes her såvel arbejdsindsats som forbrug af maskintid og materialer, specielt papir. Som i alle større projekter bør man så vidt muligt lægge en plan for ressourceforbruget og forsøge at overholde den! At lære dette er et delformål med rapportopgaverne på Datalogi 0 GA.

Der stilles på Datalogi 0 GA som oftest mindst én rapportopgave med meget vide rammer for individuel fastlæggelse af ambitionsniveau. Det er så de studerendes "behagelige" opgave at lægge et realistisk sådant og demonstrere, at de kan håndtere det! (Akkurat som i store dele af resten af det frie studium, i videnskab og i allehånde samfundsmæssige erhvervssammenhænge.) Her ligger formentlig det vanskeligste i at frigøre sig fra skolens mere "gør præcis sådan"- og "går det skævt, er det nok lærerens skyld"-tradition.

Klager er ikke særligt befordrende for det frie studium, men medleven er i høj grad ...

3.4 Aflevering og bedømmelse

Af opgaveteksten vil afleveringsfrist og -sted fremgå. Det er vigtigt at gøre opmærksom på, at *afleveringsfristen skal overholdes*. Ingen forhold — heller ikke problemer med EDB-anlægget — accepteres som begrundelse for forsinkelse.

Studienævnet har dog indført en enkelt undtagelse: studerende kan opnå udsættelse af fristen for aflevering af en rapportopgave med tre døgn (3 gange 24 timer), når følgende er opfyldt:

a. Der skal foreligge en *særlig grund* til udsættelsen. Som sådan anerkendes blandt andet egen sygdom og sygdom eller dødsfald inden for den nærmeste familie i perioden fra opgaven stilles, til den afleveres.

b. Ønsket om udsættelse skal fremsættes *skriftligt* og afleveres eller sendes til DIKU, så det kommer instituttet *i hænde inden* udløbet af den ordinære afleveringsfrist. Anmodningen skal angive navn, studienummer (= personnummer) og udsættelsesgrund.

c. Ved aflevering af besvarelsen inden for udløbet af den overfor angivne udsættelse skal der tillige afleveres dokumentation for den påberåbte udsættelsesgrund i form af lægeattest, attest fra begravelsesmyndighed eller tilsvarende.

Hvis en studerende opnår udsættelse med aflevering af en rapportopgave, der skal afleveres gruppevis, gælder denne udsættelse også de andre studerende i den gruppe, som samarbejder om opgaven.

De angivne 3 gange 24 timer regnes kun af hverdage (mandag til lørdag) og ikke af helligdage.

Gruppetilmelding Besvarelser, der ikke lever op til opgavetekstens krav om gruppestørrelse, *modtages ikke*. Som et tilbud organiseres derfor mulighed for forhåndstilmelding af grupper: deltagerne i en forhåndstilmeldt gruppe forpligter sig til at acceptere hinandens arbejdsindsats, sådan at ingen kan risikere at blive ladt i stikken, når besvarelsen skal afleveres.

Bedømmelse Efter modtagelsen af besvarelserne vil instruktorerne gennemlæse og bedømme dem. Dette munder normalt ud i en udtalelse om besvarelsens gode og dårlige sider samt en vurdering af helhedsindtrykket.

Instruktorens udtalelser må opfattes som en vejledning og kan ikke gøres til genstand for klage.

Derimod kan der klages over den samlede karakter for rapportopgaveprøven efter fakultetets normale regler. (Kapitel 5 bagest i dette bind forklarer, hvordan man finder den nyeste udgave af disse regler.)

3.5 Generelt

Selve opgaveteksten kan forudsættes bekendt og skal hverken gentages i besvarelsen eller vedlægges som bilag. Som retningslinje for, hvilken læser man skal have i tankerne, når man skriver, kan tages: en medstuderende, der er lige så langt i studierne som en selv og har læst opgaveteksten, men ikke løst opgaven.

Løs den stillede opgave: Læs opgaveteksten grundigt igennem, og vær sikker på at have behandlet alle de ønskede punkter.

På den anden side må der også advares mod at løse en mere ambitiøs opgave end den stillede. For at gøre rapportopgaver tiltrækkende, formuleres de ofte med udgangspunkt i en problemstilling med et virkelighedsnært præg. Dybere set vil realismen imidlertid som regel kun være et skær (og det ville også være illusorisk at forvente, man som studerende skulle kunne yde et fuldt professionelt bidrag). Normalt er opgaverne minutiøst beskåret, så de bliver overkommelige for studerende på det pågældende studietrin og inden for den afmålte tid. Man kan gå ud fra, der er lagt megen omhu i opgavens formulering, og selv om teksten kan indeholde begrænsninger og anvisninger, der virker kunstige, må det derfor anbefales at følge dem.

3.6 Eksempel på en besvaret rapportopgave

Sumbevarende afrunding

Dette problem blev stillet som stor opgave på Datalogi 0 (som dengang hed Matematik 4) i november 1969, og en besvarelse (der brugte programmeringssproget Algol 60) blev udarbejdet af lektor Bent Pedersen og optrykt i kursusbøgerne for 1970, 1971, 1972 og 1973.

Den foreliggende besvarelse i Standard ML er skrevet af Nils Andersen.

Nedenfor gengives i redigeret form den relevante del af opgaveteksten.

3.6.0 Opgave

Når mandater efter et valg skal fordeles i forhold til partiernes andel af de afgivne gyldige ikke-blanke stemmer, skal man for hvert parti nå frem til et heltalligt antal repræsentanter.

Her er et uddrag af valgloven fra maj 1964 (§43, stk. 1):

> Det samlede stemmetal for de mandatberettigede partier divideres med totalmandattallet. Med det herved fremkomne tal divideres partiernes stemmetal, og ved denne beregningsmåde udfindes det, hvor mange af de nævnte mandater hvert parti i forhold til stemmetal er berettiget til. Hvis de ved division fremkomne kvotienter ikke er hele tal og derfor, når brøkerne bortkastes, tilsammen ikke giver det hele antal mandater, forhøjes de største brøker, indtil antallet er nået (den største brøks metode).

Konstruer og dokumenter et program i Standard ML til mandatfordeling. Det er ikke nødvendigt at bruge metoden fra den citerede lovtekst, men det samlede antal uddelte mandater skal netop være totalmandattallet, og for hvert parti skal afvigelsen mellem det tildelte antal mandater og det (brudne) antal, partiet efter sit stemmetal var berettiget til, være mindre end en.

3.6.1 Problemorienteret analyse

Fra et matematisk synspunkt er mandatfordeling et udfordrende problem, idet man kan vise, at en række hver for sig rimelige krav til en fordelingsmetode (flere stemmer eller flere

mandater til fordeling skal ikke kunne give et parti færre mandater; det skal ikke kunne betale sig for et parti at dele sig op i flere mindre, ...) umuligt kan opfyldes samtidigt. Derfor findes der en række forskellige fordelingsmetoder, af hvilke den i opgaveteksten nævnte "største brøkers metode" er en og "forholdstalsmetoden" med den d'Hondtske divisorrække $1-2-3-\ldots$ en anden. Ved opgørelse af folketingsvalg i dag bruges ingen af disse metoder, men derimod en forholdstalsmetode med divisorrække $1.4-3-5-\ldots$

For de største brøkers metode gælder blandt andet "Alabamaparadokset": et større antal mandater til fordeling kan give et parti ringere repræsentation:

stemmetal:	232	128	40	
4 mandater:	2.32	1.28	0.40	stemmeandele (brudne tal)
	2	1	1	fordelte mandater
5 mandater:	2.90	1.60	0.50	stemmeandele (brudne tal)
	3	2	0	fordelte mandater

Vi vælger imidlertid i denne besvarelse at programmere metoden beskrevet i den citerede lovtekst.

Hvis p partier har fået stemmetallene s_1, s_2, ..., s_p, og m mandater skal fordeles, foreskriver teksten, at man skal danne $\frac{s_1+s_2+\ldots+s_p}{m}$ og dividere stemmetallene med det. Det giver forholdene $f_1 = \frac{s_1 \cdot m}{s_1+s_2+\ldots+s_p}$, $f_2 = \frac{s_2 \cdot m}{s_1+s_2+\ldots+s_p}$, ..., $f_p = \frac{s_p \cdot m}{s_1+s_2+\ldots+s_p}$, hvis sum er m.

Hvert forhold f_i er et helt tal h_i plus en ægte brøk. Antallet af foreløbigt fordelte mandater er $h = h_1+h_2+\ldots+h_p$, og summen af brøkdelene bliver $m-h$, det antal mandater (mellem 0 og $p-1$), som mangler at blive fordelt.

Ifølge lovteksten skal de ekstra mandater fordeles til partierne med de største brøker, men det ses, at forskriften ikke er fuldstændig: der er ingen anvisning på, hvad man skal gøre, hvis flere brøker er lige store, og der ikke kan blive mandater til dem alle. At denne situation skulle forekomme i praksis er meget usandsynligt, men for at programmere fordelingen vælger vi, at i det tilfælde skal partierne med færrest mandater have endnu et.

Heller ikke efter denne tilføjelse er metoden fuldstændig fastlagt. Tvivl opstår, når flere partier har fået lige mange stemmer og nogle af dem, men ikke dem alle, skal have tildelt mandater. Det er så ikke muligt at skelne mellem partierne alene gennem deres stemmetal, og vi vælger da arbitrært at give dem mandater efter den rækkefølge, de er opført i.

Inddata

Grundlaget for beregningen er stemmetallene samt det totale antal mandater til fordeling. Opgaveteksten nævner intet om, i hvilken form disse data foreligger, men der kan ikke ligge nogen indskrænkning i at beslutte, de skal være indeholdt i en tekstfil; så vil de kunne dannes og rettes af et redigeringsprogram.

Hvert stemmetal må være knyttet til en identifikation (for eksempel et partinavn eller -bogstav), og som indgangsformat er det naturligt at vælge det, man ofte i aviser ser for analyseinstitutters prognoser: en række linjer under hinanden, hver med først en tekst (identifikationen) og derefter et talord (stemmetallet).

Inddata skal også indeholde det totale antal mandater til fordeling, og i forventning om, at programmet skal bruges på flere forskellige sæt af stemmetal, må det være praktisk at kunne forsyne hvert sæt med en identificerende overskrift.

Ligesom linjerne med partiidentifikationer og stemmetal består disse to ekstra oplysninger altså af en tekst og et talord; vi vælger, at de skal stå som første linje af indgangsfilen.

Det valgte indgangsformat er da en tekstfil med følgende struktur:

```
overskrift          totalmandattal
identifikation-1    stemmetal-1
identifikation-2    stemmetal-2
...                 ...
identifikation-p    stemmetal-p
```

For at vi let skal kunne skelne identifikationerne fra stemmetallene, må tallene skrives som cifre uden adskillelse. Til gengæld kan det tillades, at en identifikation slutter med cifre: Når blot identifikationer er adskilt fra stemmetal af mindst et blankt tegn, er der ingen grund til at forlange nogen bestemt opstilling. Tværtimod ville det være forvirrende, hvis linjer ikke kunne slutte med blanke tegn (som jo ikke kan ses). Man kan også acceptere helt blanke linjer (som blot skal springes over).

Før beregningerne kan begynde, skal brugeren spørges, hvorfra inddata skal hentes, og her bør der både være mulighed for at angive "standard input" og en navngiven fil. Da et filnavn ikke kan være tomt, kan de to oplysninger pakkes sammen i en, idet man kan bede brugeren angive et filnavn og oplyse, at et tomt navn vil blive fortolket som "standard input".

Uddata

Også uddata placeres i en tekstfil, og ligesom for inddatas vedkommende må brugeren angive hvilken, før de egentlige beregninger kan gennemføres, og angivelse af et tomt filnavn skal fortolkes som ønske om at se resultaterne på "standard out".

For at vise beregningsgrundlaget og gøre det let at læse korrektur på tallene vælger vi at beholde stemmetallene i uddata, ligesom den overskrift, inddata var forsynet med, naturligvis skal gentages i uddata. Hovedindholdet i uddata skal således være en række linjer under hinanden, hver med identifikation, stemmetal og manddattal.

Resultatet vil være lettest at læse, hvis stemmetal og mandattal opstilles højrestillet i to talkolonner, der naturligt sluttes af med en "sumlinje" med det samlede antal stemmer og mandater. Udgangsfilen får derved denne struktur:

```
overskrift
identifikation-1    stemmetal-1          mandattal-1
identifikation-2    stemmetal-2          mandattal-2
...                 ...                  ...
identifikation-p    stemmetal-p          mandattal-p
-----------------------------------------------------
                    totalt antal stemmer totalmandattal
```

hvor overskrift, totalmandattal, identifikationer og stemmetal stammer fra inddata.

Fejlmuligheder

Det valgte inddataformat betyder, at linjer, der ikke er helt blanke, skal slutte med et talord (som angiver det totale antal mandater henholdsvis de respektive stemmetal). Hvis en linje ikke kun består af opstillingstegn, men alligevel ikke har cifre til sidst, må det meldes som en fejl.

For få linjer i inddata er også en fejl: Indgangsfilen må ikke være helt tom, og består den kun af en enkelt ikke-blank linje, skal det meldes som fejl, at der mangler afstemningsdata.

Med to (ikke-blanke) linjer i inddata vil den første blive fortolket som overskrift og totalt mandattal, den anden som partiidentifikation og stemmetal. Beregningerne kan sådan set gennemføres, selv om de ikke er interessante (og kunne klares uden datamatstøtte): partiet (der er kun ét) vil få alle mandater.

Idet talordene består af de cifre, som er placeret sidst på linjerne, kommer fortegn ikke på tale, så tallene kan ikke blive negative. Derimod foreligger den mulighed, at et eller flere af dem kunne være nul. Et totalt antal mandater på nul vil sandsynligvis være en fejl fra brugerens side, men stiller sig strengt taget ikke i vejen for beregning. Det betyder blot, at også hvert af partierne vil få nul mandater. Dette resultat vurderes at være så iøjnespringende for en bruger, at en separat fejlmelding er overflødig.

Hvis stemmetallet for et eller flere partier er nul, får de pågældende partier ingen mandater, men mandaterne fordeles blot til de øvrige (som om de pågældende partier slet ikke havde deltaget i valget), og man behøver ikke at regne situationen for en fejl.

Hvis det samlede antal stemmer er nul (hvilket for ikke-negative tal jo vil sige, at de alle må være nul), skal der derimod meldes fejl: Metoden indebærer jo, at man skal dividere med stemmesummen, men man kan ikke dividere med nul.

Et patologisk tilfælde Hvis kun ét parti var stillet op, kunne man argumentere for, at det under alle omstændigheder skulle have samtlige mandater tildelt — også, hvis dets stemmetal var nul. I den foreliggende besvarelse udskilles dette tilfælde imidlertid ikke, og det vil give samme fejlmelding som i andre tilfælde med stemmesum nul. Brugeren kan så vælge at fordele mandaterne trods fejlmeldingen.

3.6.2 Programmeringsovervejelser

Lad s betegne summen af stemmetallene, $s = s_1 + s_2 + \ldots + s_p$. Fremgangsmåden ved mandatfordeling indebærer, at man først skal betragte den hele del h_i af brøkerne $f_i = \frac{s_i \cdot m}{s}$ og derefter de ægte brøkdele $f_i - h_i$. Da problemet efter sin natur er heltalligt (det drejer sig om hele antal stemmer og hele antal mandater), virker det ikke naturligt at arbejde med brøker, ligesom der i princippet derved (fordi brøkerne ikke repræsenteres eksakt) kunne indføres fejl. Alle brøkerne vil imidlertid have s som nævner, og da vi kun skal bruge deres *indbyrdes* størrelse, kan man erstatte brøkerne med deres tællere!

Disse tællere r_i vil være resterne for heltalsdivisionen af $s_i \cdot m$ med s (ligesom h_i vil være kvotienten for denne heltalsdivision).

I steden for at løbe listen med resultater igennem gentagne gange for at finde frem til de partier, som skal have deres mandattal forhøjet, kunne man anbringe disse partier forrest. Under analysen fandt vi frem til, at partiet i skulle have sit foreløbige mandattal h_i forhøjet før partiet j, hvis

større brøk: r_i større end r_j

mindre mandattal: eller $r_i = r_j$, men h_i mindre end h_j

tidligere i listen: eller $r_i = r_j$ og $h_i = h_j$ (det vil sige $s_i = s_j$), men i opført forud for j

Sorteres listen af partier og stemmetal efter dette kriterium, vil de partier, som skal have deres stemmetal forhøjet, være de $m - h$ forreste på den sorterede liste.

Når resultaterne vises for brugeren, vil det være forvirrende, hvis partier og stemmetal ikke er opført i samme rækkefølge som i inddata. Efter at mandattallene er forhøjet, så de har m som sum, skal listen derfor bringes tilbage til sin oprindelige rækkefølge ved en fornyet sortering.

For at opnå den ønskede tabulariske opstilling af resultatet må alle partiidentifikationerne gøres lige lange. Beregningerne skal derfor også finde den længste identifikation og tilføje blanktegn til de andre, så de får samme længde. Når oplysningerne skal fremtræde som en tabel, må linjerne på den anden side ikke blive for lange til at kunne være i skærmvinduet, så der bør sættes en øvre grænse for identifikationernes længde. Tabelhovedet "Stemmetal Mandater" har (inklusive det skillende blanktegn) 18 tegn; med 72 positioner i en linje bliver der 54 positioner tilbage, hvilket skønnes at være tilstrækkeligt til partiidentifikation. Længere betegnelser afkortes til deres første 54 tegn.

Programudførelse må omfatte følgende punkter:

1. Hent overskrift, samlet antal mandater til fordeling og de enkelte partiers identifikationer og stemmetal fra inddata.

2. Find den maksimale længde af en identifikation og det samlede antal afgivne stemmer.

3. Nummerer partierne, og tilføj oplysninger om foreløbigt mandattal og divisionsrest.

4. Beregn det fordelte antal mandater (og det manglende antal mandater).

5. Sorter listen af partioplysninger efter det valgte prioriteringskriterium.

6. Uddel de manglende mandater.

7. Sorter listen af partioplysninger efter nummer.

8. Skriv resultatet ud.

De beregninger, som omtales i punkterne 2. og 4., kunne enten udføres i separate gennemløb eller parallelt med behandlingerne i de foregående punkter 1. henholdsvis 3.; vi har valgt den sidstnævnte mulighed.

Under behandlingen foreligger partioplysningerne i forskellige repræsentationer (se figur 3.3). Umiddelbart efter indlæsning har vi en ren liste af tekster

```
ts : string list
```

som derefter opsplittes i partiidentifikationer og stemmetal

```
afstemnres : (string * int) list.
```

I punkt 3. tilføjes numre, kvotienter og rester

```
bData : (int * string * int * int * int) list,
```

`Socialdemokratiet␣␣␣␣␣␣␣␣␣␣␣324249`

Indgangstekst

`Socialdemokratiet`	324249

Talordet er spaltet fra og teksten renset for afsluttende blanke

1	`Socialdemokratiet`	324249	2	1249762

Løbenummer samt kvotient og rest ved divisionen af $324249 \cdot 16$ (der var 16 mandater til fordeling) med det samlede stemmetal 1969111 er tilføjet

Figur 3.3: Afstemningsoplysningernes forskellige repræsentationer under programudførelse.

idet kvintuplerne skal fortolkes som (løbenummer, partiidentifikation, partiets stemmetal, mandattal, divisionsrest). Denne repræsentation bevares under punkterne 5., 6. og 7.

Da alle stemmetal og mandattal repræsenteres i standardtypen `int` af heltal, er det nødvendigt at overveje muligheden for overløb. Variationsområdet for denne type strækker sig i STANDARD ML fra -1073741824 til 1073741823; de negative tal har man ingen glæde af i den aktuelle sammenhæng, men alle ni-cifrede stemmetal (og enkelte ti-cifrede) kan altså accepteres. Både til danske folketingsvalg og til den hypotetiske situation, hvor man for eksempel ville lave en fælles opgørelse af en afstemning i hele Europa, er denne kapacitet tilstrækkelig.

Situationer, hvor kapaciteten blev sprængt, kunne på den anden side tænkes, for eksempel en afstemning i FN's generalforsamling, hvor medlemsstaternes stillingtagen blev vægtet efter indbyggertal. Tilpasning til en sådan situation vil kræve en ændring af programmet.

Til sorteringerne under punkt 5. og 7. benyttes flettesortering ("mergesort"). Da de to sorteringer benytter forskellige kriterier, gives sorteringskriteriet med som ekstra funktionsparameter.

3.6.3 Programbeskrivelse

Overskrifter inddeler programmets funktioner i seks afsnit.

Generelle hjælpefunktioner. Kommentarer ved disse funktioner forklarer udtømmende deres virkning.

Omformning af inddata. Her løses de opgaver, der er omtalt i punkt 1. og 2. ovenfor. Det er værd at bemærke, at `talFraSpejlcifliste` finder værdien af et *spejlvendt* talord (enerne længst til venstre, derefter tierne, og så videre).

`splitITekstOgTal` og `hentOverskrAfstemnresNbrdSTotalNTotal` er de eneste funktioner i dette program, som rejser undtagelser, henholdsvis
`Fail "Tal mangler efter teksten` *`tekst`*`"` og `Fail "Helt tom indgangstekst"` / `Fail "Ingen afstemningsdata"` / `Fail "Ingen stemmer"`.

Beregning. `tilBData` udfører punkt 3. og 4., `prioritOrd` og `opstilOrd` er de sorteringskriterier, som skal benyttes i henholdsvis punkt 5. og 7., `forhoejMandattal` løser punkt 6., og `beregnMandattal` samler punkterne 4., 5., 6. og 7.

Omformning til uddata. Her defineres som konstanter de ønskede bredder af kolonnerne med partibetegnelser, stemmetal og mandattal samt funktionerne `hoved`, `hale`, `visFacitlinje` og `visFacit` til udførelse af punkt 8.

Ind- og udlæsning. Hjælpefunktioner med bivirkning er samlet her. `laesMedStikord` skriver ud på `stdErr` og læser fra `stdIn`. `hentTekstliste (indstrm)` forudsætter indgangsstrømmen `indstrm` åben og lukker den ikke. `gemTekst (udstrm,t)` forudsætter på samme måde udgangsstrømmen `udstrm` åben og lukker den ikke efter udskrivning af `t`.

Hovedfunktion. Her defineres `main : unit -> unit`.

3.6.4 Afprøvning

Størsteparten af funktionerne blev konstrueret med støtte i en intern afprøvning. I tabellen herunder forklares testdata for hver funktion, idet funktionerne behandles i samme rækkefølge, som de har i programmet (se bilag, afsnit 3.6.6). De prøvekørsler, som forklares, er vedlagt (se afsnit 3.6.7). I alle tilfælde blev uddata (i denne afsluttende udgave af programmet) som forventet.

`splitAt`
: Tom/ikke-tom liste `xs`, tal `n` lig med 0 eller positivt.

`splitOn`
: Tom/ikke-tom liste `xs`, kriterium `q` opfyldt/ikke-opfyldt for forreste listeelement.

`repeat`
: Antal `n` lig med 0 eller positivt.

`blanke`
: Længde `n` lig med 0 eller positiv.

`venstretving`
: Bredde `b` lig med 0, mindre end, lig med eller større end tekstens længde.

`hoejrestil`
: Bredde `b` lig med 0 (og dermed mindre end), lig med eller større end tekstens længde.

`streg`
: Blot prøvet for `n` lig med 3, da den i øvrigt er sammensat af allerede testede funktioner.

`flet`
: Begge lister tomme, kun den ene tom, ingen af listerne tomme (og hvor fletning ikke kun består i at sætte den ene liste i forlængelse af den anden). Det ses, at elementer fælles for de to lister gentages i uddata. Funktionen er dog kun afprøvet med voksende lister som argument (da den kun skal bruges sådan af `fsort`).

`fsort`
: Tom liste, singleton-liste og ordnet/ikke-ordnet liste med flere elementer.

`fjernAfslutningsblanke`
: Tom tekst, tekst uden/med afsluttende opstillingstegn

`talFraSpejlcifliste`
: Tomt talord, almindeligt talord og tekst med bogstav i (som ikke giver fejl, men blot indregnes efter sin valør).

`splitITekstOgTal`
: Tom tekst og tekst uden cifre til sidst (som begge skal give fejlmelding). Kun cifre, andre tegn og cifre, specielt med cifre bagest i teksten før de afsluttende cifre (adskilt fra dem af mellemrum).

`rensTekstliste`
: Tom liste, liste af tekster uden/med blanktegn til sidst (herunder helt blanke tekster iblandt).

`hentAfstemnresNbrdSTotal`
: Tom liste, liste med tomme tekster afsluttet med stemmetal, og lister med forskelige kombinationer af tekster og stemmetal, der viser korrekt beregning af stemmesum og maksimal navnebredde (herunder at afsluttende blanke ikke medregnes i navnes bredde).

`hentOverskrAfstemnresNbrdSTotalMTotal`
: De tre fejlmeldinger afprøves: tom liste, singletonliste (med tekst, der ender på talord), og liste af tekster med alle sluttalord nul. Kun ét eksempel med korrekte inddata prøves, da funktionen i det tilfælde henter sit resultat via allerede testede funktioner.

`tilBData`
: Kun prøvet på et enkelt eksempel.

`forhoejMandattal`
: Prøvet, hvor mandatforskellen (antallet af endnu ikke fordelte mandater) er 0, 1, mindre end, lig med og større end antallet af opstillede partier. (Det sidste tilfælde kan ikke forekomme i de kontekster, hvori funktionen bruges.)

`prioritOrd`
: Ikke internt afprøvet.

`opstilOrd`
: Ikke internt afprøvet.

`beregnMandattal`
: Ikke internt afprøvet.

`hoved`
: Typisk eksempel.

`hale`
: Typisk eksempel.

`visFacitlinje`
: Typisk eksempel, og samtidig ses, kolonneopstillingen passer med hoved og hale.

`visFacit`
: Ikke særskilt afprøvet på grund af sin simple sammensætning af allerede testede funktioner.

`laesMedStikord`
: Typisk eksempel.

`hentTekstliste`
: Prøvet med en eksempelfil. Det lykkedes også at få kald af formen `hentTekstliste stdIn` til at fungere (man skulle huske linjeskift forud for "end of input"). Udskrift af dette er ikke vedlagt.

`gemTekst`
: Ikke internt afprøvet.

`main`
: Ikke internt afprøvet.

Eksempler

De ikke internt afprøvede funktioner aktiveres i den eksterne afprøvning vedlagt som bilag (se afsnit 3.6.8).

Med den allerede konstruerede og afprøvede funktion `hentTekstliste` til rådighed var det enkelt at definere en funktion `visfil` til kopiering af en tekstfil til skærmen (se funktionsdefinitionen forrest i bilaget), og den er brugt for at vise inddata til testeksemplerne. De fire første eksempler afprøver fejlmeldingerne. Derefter følger tre korrekte eksempler, hvor fordelingen i det første kan afgøres af brøkerne alene, mens de to andre inddrager de to ekstra kriterier (henholdvis antallet af sikre mandater og placeringen i rækkefølgen). For at afprøve `gemTekst` sendes resultatet af de to første af disse eksempler til en tekstfil, som derefter vises.

Derefter følger som et lidt større eksempel optællingen af afstemningen i Danmark til EU-parlamentet den 10. juni 1999 (kilde: 4. sektion af dagbladet Politiken for den 15. juni 1999).

Det sidste eksempel viser, at afkortningen af lange identifikationer til deres forreste 53 tegn fungerer. Samtidig er eksemplet et forsøg på at hente inddata direkte fra tastaturet (i steden for fra en fil). Dette kom aldrig helt til at fungere: efter at have vist de korrekte uddata slutter programudførelse af med en mystisk fejlmelding (Unknown Error (-21)), og for at komme videre er det nødvendigt at afslutte `mosml`-systemet.

Bortset fra læsning af inddata fra tastatur fungerer alle eksemplerne korrekt.

3.6.5 Brugervejledning

Det konstruerede program kan beregne mandatfordelingen efter et valg. Figur 3.4 viser et eksempel på inddata til programmet.

```
Eksempel ét 5

A           55

B       32
C         13
```

Figur 3.4: Eksempel på indgangsdata.

Metode Det totale antal mandater, der findes som det tal, der står yderst til højre i første indgangslinje, fordeles til de deltagende partier i forhold til deres opnåede stemmetal efter den metode, som kaldes "de største brøkers metode", og som er let at forklare gennem eksemplet: Med 100 afgivne stemmer og 5 mandater kommer hvert mandat til at koste 20 stemmer, og fordelingen til A, B og C skulle derfor egentlig være 2.75, 1.60 og 0.65 mandater. Det giver 2 sikre mandater til A og 1 sikkert mandat til B, men spørgsmålet er, hvem der skal have de 2 sidste mandater. Som angivet af metodens navn lader man brøkdelenes størrelse bestemme: 0.75 er større end 0.65, der igen er større end 0.60, og det fjerde og femte mandat tildeles derfor A og C. Figur 3.5 viser uddata fra programmet.

Hvis lige store brøker ikke alle kan give mandat, vælger programmet at favorisere de mindste partier, se figur 3.6, hvor kun den ene af de to brøker på 0.5 kan give mandat.

```
Eksempel ét

  Stemmetal Mandater
--------------------
A         55         3
B         32         1
C         13         1
--------------------
         100         5
```

Figur 3.5: Udgangsdata fra eksemplet i figur 3.4.

```
Eksempel to 5
P 1           60
P 2          30
P 3           10

inddata
```

```
Eksempel to

    Stemmetal Mandater
----------------------
P 1         60         3
P 2         30         1
P 3         10         1
----------------------
           100         5
uddata
```

Figur 3.6: Ved lige store brøkdele får de mindste partier mandater først.

Kan dette princip heller ikke afgøre fordelingen, fordi stemmetallene står lige, uddeler programmet mandaterne i den rækkefølge, hvori partierne tilfældigvis står opført — se figur 3.7.

```
Eksempel tre 5
A          40
B          30
C          30

inddata
```

```
Eksempel tre

  Stemmetal Mandater
--------------------
A         40         2
B         30         2
C         30         1
--------------------
          100        5
uddata
```

Figur 3.7: Står det lige, uddeles mandater i den rækkefølge, hvori partierne står opført.

Det er værd at bemærke, at de største brøkers metode *ikke altid* giver samme resultat som forholdstalsmetoden, der er den, man normalt benytter ved valg i Danmark (til kommunalbestyrelser, folketing og EU-parlamentet).

Figur 3.8 viser programmets beregning af mandatfordelingen efter valget til EU-parlamentet i 1999 (hvor Centrum-Demokraterne ikke fik nogen repræsentant, mens Venstre faktisk fik 5 valgt ind.)

```
EU-parlamentsvalget den 10. juni 1999 16

Socialdemokratiet           324249
Det Radikale Venstre           180116
Det Konservative Folkeparti           166518
Centrum-Demokraterne           68583
Socialistisk Folkeparti           139845

JuniBevægelsen - Mod Unionen           317051
Folkebevægelsen mod EU           143706
Dansk Folkeparti           114859
Kristeligt Folkeparti           39128

Venstre           460823
Fremskridtspartiet           14233
```

inddata

```
EU-parlamentsvalget den 10. juni 1999

                              Stemmetal Mandater
-----------------------------------------------
Socialdemokratiet                324249        3
Det Radikale Venstre             180116        1
Det Konservative Folkeparti      166518        1
Centrum-Demokraterne              68583        1
Socialistisk Folkeparti          139845        3
JuniBevægelsen - Mod Unionen     317051        3
Folkebevægelsen mod EU           143706        1
Dansk Folkeparti                 114859        1
Kristeligt Folkeparti             39128        0
Venstre                          460823        4
Fremskridtspartiet                14233        0
-----------------------------------------------
                                1969111       16
```

uddata

Figur 3.8: EU-parlamentsvalget opgjort efter de største brøkers metode.

Indgangsformat Som eksemplerne viser, skal indgangsoplysningerne gives i en række linier, blandt hvilke der også kan være blanke linjer. Sådanne linjer springes over, men alle andre linjer skal slutte af med et talord. Det talord, der afslutter forreste linje, skal angive antallet af mandater til fordeling, mens talordene sidst på de øvrige linjer er afstemningsresultatet.

Hvad der står foran mandattallet i første linje opfattes som navn på datasættet. Foran stemmetallene i de øvrige linjer kan man skrive en identifikation af partiet.

De nævnte tekster (datasætnavn og partiidentifikationer) kan godt ende med et ciffer, når der bare er mellemrum hen til talordet bagest (se figur 3.6 og 3.8). Til gengæld er det nødvendigt at skrive mandattal og stemmetal ud i et (uden opdeling ved hjælp af for eksempel mellemrum, punktum eller komma). Vær også opmærksom på, at der er forskel på cifferet "1" og bogstavet "l" og på cifferet "0" og bogstavet "O" eller "o".

Programmet er beregnet til at hente inddata enten fra en tekstfil eller fra tastaturet, men det anbefales, at man bruger en tekstfil, så fejl let kan rettes.

Udgangsformat Navnet fra første linje i inddata bruges som overskrift. Derefter følger (se eksemplerne) en linje med teksten "Stemmetal Mandater" og så mandatfordelingen med en linje til hvert parti. Partiidentifikationer og stemmetal fra inddata gentages her i uddata (til brug for eventuel korrekturlæsning). For identifikationer længere end 53 positioner vises dog kun deres forreste 53 tegn. I skemaets sidste linje anføres det samlede stemmetal og det samlede antal mandater.

Uddata kan enten sendes til en tekstfil eller til skærmen.

Kørsel Hovedfunktionen er `main : unit -> unit`, og programmet aktiveres simpelt hen (efter at være blevet indlæst) med udtrykket

```
main ();
```

Dette vil resultere i to forespørgsler, brugeren skal besvare, om placeringen af henholdsvis ind- og uddata. Man kan enten anføre navnene på de respektive filer eller ved bare at sende en tom linje angive, at der skal læses fra tastatur henholdsvis skrives til skærm.

Fejlmeldinger I visse situationer kan programudførelse blive afbrudt af en melding på skærmen af følgende form:

```
! Uncaught exception:
! Fail "fejlmelding"
```

De fire forskellige fejlmeldinger, der kan komme fra programmet, har følgende betydning:

`Tal mangler efter teksten` *`tekst`* En linje, som begynder med *tekst*, slutter ikke af med et talord (sådan som alle indgangslinjer skal).

`Helt tom indgangstekst`

`Ingen afstemningsdata` Der er kun fundet én linje i indgangsfilen. Var det den rigtige fil?

`Ingen stemmer` Alle stemmetal er angivet til 0. I den situation kan programmet ikke beregne en mandatfordeling.

Forklaring på andre fejl end de her beskrevne må søges i dokumentationen for `mosml`. Der kan blandt andet være tale om fejl i forbindelse med filhåndtering: forsøg på at læse fra en ikke-eksisterende fil eller en fil, systemet ikke har læserettighed til, eller forsøg på at skrive til en fil, systemet ikke kan oprette. Sprogsystemet vil også protestere, hvis talområdet sprænges (men det vil først ske ved stemme- eller mandattal større end 1 000 000 000).

3.6.6 Bilag: program

```
(************************************************************************
 Program hørende til G-opgaven "Sumbevarende afrunding"
 1999 juli 16 Nils Andersen
************************************************************************
 Generelle hjælpefunktioner
************************************************************************)
(* splitAt (xs,n) = (xs0,xs1), hvor xs = xs0 @ xs1 er en opsplitning
   af argumentlisten i sine n forreste elementer og resten *)
fun splitAt ([],_) = ([],[])
  | splitAt (xs as y :: ys,n)
    = if n > 0 then let val (ws,zs) = splitAt (ys,n - 1)
                    in (y :: ws,zs) end
               else ([],xs);

(* splitOn (xs,q) = (xs0,xs1), hvor xs = xs0 @ xs1, og xs0 er den
   længste indledende del af xs, hvori alle elementer opfylder
   prædikatet q *)
fun splitOn ([],_) = ([],[])
  | splitOn (xs as y :: ys,q)
    = if q y then let val (ws,zs) = splitOn (ys,q)
                  in (y :: ws,zs) end
             else ([],xs);

(* repeat (x,n) danner en liste med n x'er *)
fun repeat (x,n) = if n > 0 then x :: repeat (x,n - 1) else [];

(* en tekst bestående af n blanktegn *)
fun blanke n = implode (repeat (#,n));

(* venstretving (t,b) venstrestiller teksten t i et b tegn bredt felt.
   Længere tekster afkortes til deres forreste b tegn *)
fun venstretving (t,b)
    = let val sizet = size t
      in if sizet > b then implode (#1 (splitAt (explode t,b)))
                      else t ^ blanke (b - sizet)
      end;
```

```
(* hoejrestil (t,b) højrestiller teksten t i et b tegn bredt felt *)
fun hoejrestil (t,b) = blanke (b - size t) ^ t;

(* en linje bestående af n minus-tegn *)
fun streg n = implode (repeat (#-",n)) ^ "\n";

(* flet forudfor (xs,ys) sammenfletter filerne xs, ys : 'a list,
   idet forudfor : 'a * 'a -> bool benyttes ved afgørelse af,
   om et element skal gå forud for et andet *)
fun flet _ ([],ys) = ys
  | flet _ (xs,[]) = xs
  | flet forudfor (xs as x :: xs',ys as y :: ys')
    = if forudfor (x,y) then x :: flet forudfor (xs',ys)
                        else y :: flet forudfor (xs,ys');

(* fsort forudfor xs danner ved flettesortering en omordning af
   xs : 'a list, idet forudfor : 'a * 'a -> bool benyttes ved
   afgørelse af, om et element skal gå forud for et andet *)
fun fsort forudfor xs
    = let val lengthxs = length xs
      in if lengthxs < 2 then xs
         else let val halvdelen = lengthxs div 2;
                  val (ys,zs) = splitAt (xs,halvdelen)
              in flet forudfor (fsort forudfor ys,fsort forudfor zs) end
      end;

load "Int"(* Vi skal bruge Int.toString *);

(*************************************************************************
 Omformning af inddata
*************************************************************************)
(* fjernAfslutningsblanke fjerner bagfra
   alle blanktegn og opstillingstegn (linjeskift, vognretur,
   horisontal og vertikal tabulering, sideskift) *)
fun fjernAfslutningsblanke (s : string) : string
    = implode (rev (#2 (splitOn (rev (explode s),Char.isSpace))));

(* talFraSpejlcifliste omsætter en spejlvendt cifferfølge til heltal.
   Argumentet bør kun være cifre, men det kontrolleres ikke *)
fun talFraSpejlcifliste ([] : char list) : int = 0
  | talFraSpejlcifliste (d :: ds)
    = ord d - ord #"0"+ 10 * talFraSpejlcifliste ds;

(* splitITekstOgTal tolker sammenhængende afslutningscifre som
   separat naturligt tal (manglende talord fejlmeldes).
   Fra teksten foran tallet fjernes afsluttende blanke *)
```

```
fun splitITekstOgTal (s : string) : string * int
    = let val (ds,cs) = splitOn (rev (explode s),Char.isDigit)
          val tekst = fjernAfslutningsblanke (implode (rev cs))
      in if ds = []
         then raise Fail ("Tal mangler efter teksten "^ tekst)
         else (tekst,talFraSpejlcifliste ds)
      end;

(* rensTekstliste fjerner fra en liste af tekster
   afsluttende blanke og opstillingstegn samt helt blanke linjer *)
val rensTekstliste : string list -> string list
    = List.filter (fn t => t <> ) o map fjernAfslutningsblanke;

(* hentAfstemnresNbrdSTotal
   opfatter en liste af tekster som hver bestående af et
   identificerende navn og et stemmetal; beregner tillige
   maksimale navnebredde og sum af stemmetallene *)
fun hentAfstemnresNbrdSTotal ([] : string list)
    : (string * int) list * int * int = ([],0,0)
  | hentAfstemnresNbrdSTotal (t :: ts)
    = let val navnOgS as (n,s) = splitITekstOgTal t
          val nbrd0 = size n
          val (afstemnres,nbrd1,sTotal) = hentAfstemnresNbrdSTotal ts
          val nbrd = if nbrd0 < nbrd1 then nbrd1 else nbrd0
      in (navnOgS :: afstemnres,nbrd,s + sTotal) end;

(* hentOverskrAfstemnresNbrdSTotalMTotal
   opfatter forreste tekst i en liste som overskrift og samlet antal
   mandater til fordeling og resten som en liste med identifikationer
   og stemmetal, der tillige summeres. Der meldes fejl, hvis der
   ingen overskriftslinje er, eller kun overskriftslinje, men ingen
   afstemningsdata, eller hvis stemmetallene summerer til 0 *)
fun hentOverskrAfstemnresNbrdSTotalMTotal (tekstliste : string list)
    : string * (string * int) list * int * int * int
    = if null tekstliste then raise Fail "Helt tom indgangstekst"
      else let val ([t],ts) = splitAt (tekstliste,1)
           in if null ts then raise Fail "Ingen afstemningsdata"
              else let val (overskr,mTotal) = splitITekstOgTal t
                       val (afstemnres,nbrd,sTotal)
                           = hentAfstemnresNbrdSTotal ts
                   in if sTotal = 0 then raise Fail "Ingen stemmer"
                      else (overskr,afstemnres,nbrd,sTotal,mTotal)
                   end
           end;
```

```
(*************************************************************************
 Beregning
*************************************************************************)
(* For afstemnres     = [(navn1,s1),(navn2,s2),...]
       sTotal         = s1 + s2 + ...
       mTotal         = antallet af mandater til fordeling
   vil tilBData (afstemnres,sTotal,mTotal) = (bData,mFordelt)
   med bData          = [(1,navn1,s1,f1,r1), (2,navn2,s2,f2,r2),...]
       mFordelt       = f1 + f2 + ...,
   hvor si * mTotal   = fi * sTotal + ri, i = 1,2,...
   idet si er stemmetallene, fi de foreløbige mandattal
   og ri de tilhørende divisionsrester *)
fun tilBData (afstemnres,sTotal,mTotal)
    = let fun tilBeregning (n,[]) = ([],0)
            | tilBeregning (n,(navn,s) :: afstmnres)
              = let val v = s * mTotal
                    val (f,r) = (v div sTotal,v mod sTotal)
                    val (bData,f') = tilBeregning (n+1,afstmnres)
                in ((n,navn,s,f,r) :: bData,f + f') end
      in tilBeregning (1,afstemnres) end;

(* forhoejMandattal (bData,mFordelt,mTotal) vil fordele de
   manglende mandater ved at øge mandattallet med 1 for hver af de
   mTotal - mFordelt forreste elementer i listen bData
   (idet det antages, at 0 <= mTotal - mFordelt <= length bData) *)
fun forhoejMandattal (bData,mFordelt,mTotal)
= let fun forhoej (n,navn,tal,f,r) = (n,navn,tal,f + 1,r)
      val (bData1,bData2) = splitAt (bData,mTotal - mFordelt)
  in map forhoej bData1 @ bData2 end;

(* prioritOrd ((n1,navn1,tal1,f1,r1),(n2,navn2,tal2,f2,r2))
   angiver den valgte prioritering ved mandatforhøjelse:
   leksikografisk ordning efter 1. største brøk (rest),
   2. mindste mandattal, 3. mindste løbenummer *)
fun prioritOrd ((n1,_,_,f1,r1),(n2,_,_,f2,r2))
    = r1 > r2
      orelse r1 = r2 andalso (f1 < f2
                              orelse f1 = f2 andalso n1 < n2);

(* opstilOrd ((n1,navn1,tal1,f1,r1),(n2,navn2,tal2,f2,r2)) angiver
   den valgte rækkefølge ved opstilling af facit: løbenummerorden *)
fun opstilOrd ((n1,_,_,_,_),(n2,_,_,_,_)) = n1 < n2;

(* beregnMandattal ([(navn1,s1),(navn2,s2),...],sTotal,mTotal)
   = [(1,navn1,s1,m1,r1),(2,navn2,s2,m2,r2),...],
   hvor mandattallene m1, m2, ..., med sum mTotal, er beregnet ud fra
```

```
   stemmetallene s1, s2, ..., med sum sTotal *)
fun beregnMandattal (afstemnres,sTotal,mTotal)
    = let val (bData1,mFordelt) = tilBData (afstemnres,sTotal,mTotal)
          val bData2 = fsort prioritOrd bData1
          val bData3 = forhoejMandattal (bData2,mFordelt,mTotal)
      in fsort opstilOrd bData3 end;

(**********************************************************************
 Omformning til uddata
**********************************************************************)
(* opstillingskonstanter *)
val sbrd     = 10 (* antallet af positioner til stemmetal *);
val mbrd     =  9 (* antallet af positioner til mandattal *);
val maxnbrd  = 53 (* maksimalt antal positioner til partibetegnelse *)
fun hoved (nbrd,sbrd,mbrd,ovskr)
    = ovskr ^ "\n\n"
      ^ venstretving (,nbrd) ^ hoejrestil ("Stemmetal",sbrd)
      ^ hoejrestil ("Mandater",mbrd) ^ "\n"
      ^ streg (nbrd + sbrd + mbrd);
fun hale (nbrd,sbrd,mbrd,sTotal,mTotal)
    = streg (nbrd + sbrd + mbrd) ^ venstretving (,nbrd)
      ^ hoejrestil (Int.toString sTotal,sbrd)
      ^ hoejrestil (Int.toString mTotal,mbrd) ^ "\n";

(* visFacitlinje (nbrd,sbrd,mbrd) viser hvert element (løbenummer,
   navn,stemmetal,mandattal,rest) som navn, stemmetal og mandattal
   i felter med bredde henholdsvis nbrd, sbrd og mbrd positioner *)
fun visFacitlinje (nbrd,sbrd,mbrd) (_,navn,s,m,_)
    = venstretving (navn,nbrd) ^ hoejrestil (Int.toString s,sbrd)
      ^ hoejrestil (Int.toString m,mbrd) ^ "\n";

(* visFacit viser efter et hoved med overskriften
   hvert element i bData og til sidst en sammentællingslinje *)
fun visFacit (ovskr,nbrd,sbrd,mbrd,bData,sTotal,mTotal)
    = hoved (nbrd,sbrd,mbrd,ovskr)
      ^ String.concat (map (visFacitlinje (nbrd,sbrd,mbrd)) bData)
      ^ hale (nbrd,sbrd,mbrd,sTotal,mTotal);

(**********************************************************************
 Ind- og udlæsning (funktioner med bivirkning)
**********************************************************************)
open TextIO;

(* laesMedStikord s
   præsenterer teksten s (afsluttet med linjeskift) på stdErr
   og har den efterfølgende linje fra stdIn som sin funktionsværdi *)
```

```
fun laesMedStikord s
    = (output (stdErr,s ^ "\n"); inputLine stdIn);

(* hentTekstliste danner en liste af tekster,
   hver svarende til en linje fra den angivne indgangsstrøm *)
fun hentTekstliste (indstrm : instream) : string list
    = if endOfStream indstrm then []
      else inputLine indstrm :: hentTekstliste indstrm;

(* gemTekst skriver teksten til den angivne udgangsstrøm
   og tømmer bufferen, men lukker ikke strømmen *)
fun gemTekst ((udstrm,t) : outstream * string) : unit
    = (output (udstrm,t); flushOut udstrm);

(*************************************************************************
 Hovedfunktion
*************************************************************************)
fun main ()
    = let
          val stikordInd
= "Angiv navnet på filen med afstemningsresultaterne; afslut med\n\
  \linjeskift (en blank linje vil blive fortolket som inddata fra\n\
  \tastatur):"
          val stikordUd
= "Angiv navnet på den fil, hvori beregningsresultaterne skal gemmes;\n\
  \afslut med linjeskift (en blank linje vil blive fortolket som\n\
  \uddata til skærm):"
          val indstrm
= let val indfil = fjernAfslutningsblanke (laesMedStikord stikordInd)
  in if indfil =  then stdIn else openIn indfil end
          val udstrm
= let val udfil = fjernAfslutningsblanke (laesMedStikord stikordUd)
  in if udfil =  then stdOut else openOut udfil end
          val ts = rensTekstliste (hentTekstliste indstrm
                                   before closeIn indstrm)
          val (ovskr,afstemnres,n,sTotal,mTotal)
              = hentOverskrAfstemnresNbrdSTotalMTotal ts
          val nbrd = if n > maxnbrd then maxnbrd else n
          val facit = beregnMandattal (afstemnres,sTotal,mTotal)
          val udtekst
              = visFacit (ovskr,nbrd,sbrd,mbrd,facit,sTotal,mTotal)
      in
         gemTekst (udstrm,udtekst)
      end;
```

3.6.7 Bilag: testkørsler

```
Moscow ML version 1.43 (April 1998)
Mangled by e & eMake 06 May 1998
Use the Enter key to evaluate an input expression.
- use "nils-hd1:Applications:mosml:DatOekspl:SumbevAfr.sml";
[opening file "nils-hd1:Applications:mosml:DatOekspl:SumbevAfr.sml"]
> val splitAt = fn : 'a list * int -> 'a list * 'a list
> val splitOn = fn : 'a list * ('a -> bool) -> 'a list * 'a list
> val repeat = fn : 'a * int -> 'a list
> val blanke = fn : int -> string
> val venstretving = fn : string * int -> string
> val hoejrestil = fn : string * int -> string
> val streg = fn : int -> string
> val flet = fn : ('a * 'a -> bool) -> 'a list * 'a list -> 'a list
> val fsort = fn : ('a * 'a -> bool) -> 'a list -> 'a list
> val it = () : unit
> val fjernAfslutningsblanke = fn : string -> string
> val talFraSpejlcifliste = fn : char list -> int
> val splitITekstOgTal = fn : string -> string * int
> val rensTekstliste = fn : string list -> string list
> val hentAfstemnresNbrdSTotal =
    fn
    : string list -> (string * int) list * int * int
File "nils-hd1:Applications:mosml:DatOekspl:SumbevAfr.sml", line 121, characters 19-27:
!       else let val ([t],ts) = splitAt (tekstliste,1)
!                   ^^^^^^^^
! Warning: pattern matching is not exhaustive

> val hentOverskrAfstemnresNbrdSTotalMTotal =
    fn
    : string list -> string * (string * int) list * int * int * int
> val tilBData =
    fn
    : ('a * int) list * int * int -> (int * 'a * int * int * int) list * int
> val forhoejMandattal =
    fn
    : ('a * 'b * 'c * int * 'd) list * int * int ->
      ('a * 'b * 'c * int * 'd) list
> val prioritOrd =
    fn
    : (int * 'a * 'b * int * int) * (int * 'c * 'd * int * int) -> bool
> val opstilOrd =
    fn
    : (int * 'a * 'b * 'c * 'd) * (int * 'e * 'f * 'g * 'h) -> bool
> val beregnMandattal =
```

```
    fn
    : ('a * int) list * int * int -> (int * 'a * int * int * int) list
> val sbrd = 10 : int
> val mbrd = 9 : int
> val maxnbrd = 53 : int
> val hoved = fn : int * int * int * string -> string
> val hale = fn : int * int * int * int * int -> string
> val visFacitlinje =
    fn
    : int * int * int -> 'a * string * int * int * 'b -> string
> val visFacit =
    fn
    : string * int * int * int * ('a * string * int * int * 'b) list * int *
      int -> string
> type elem = char
  type vector = string
  type cs
  type instream
  type outstream
  val stdErr = <outstream> : outstream
  val openAppend = fn : string -> outstream
  val flushOut = fn : outstream -> unit
  val inputAll = fn : instream -> string
  val inputLine = fn : instream -> string
  val output1 = fn : outstream * char -> unit
  val closeIn = fn : instream -> unit
  val closeOut = fn : outstream -> unit
  val lookahead = fn : instream -> char option
  val stdIn = <instream> : instream
  val input = fn : instream -> string
  val print = fn : string -> unit
  val inputNoBlock = fn : instream -> string option
  val endOfStream = fn : instream -> bool
  val output = fn : outstream * string -> unit
  val inputN = fn : instream * int -> string
  val outputSubstr = fn : outstream * substring -> unit
  val openIn = fn : string -> instream
  val openOut = fn : string -> outstream
  val stdOut = <outstream> : outstream
  val input1 = fn : instream -> char option
  val scanStream =
    fn
    : ((cs -> (char * cs) option) -> cs -> ('a * cs) option) -> instream ->
      'a option
> val laesMedStikord = fn : string -> string
```

```
> val hentTekstliste = fn : instream -> string list
> val gemTekst = fn : outstream * string -> unit
> val main = fn : unit -> unit
[closing file "nils-hd1:Applications:mosml:DatOekspl:SumbevAfr.sml"]
> val it = () : unit
- val abc = "abc";
> val abc = "abc" : string
- val abcliste = explode abc;
> val abcliste = [#"a", #"b", #"c"] : char list
- val tomliste = [] : char list;
> val tomliste = [] : char list
- splitAt (tomliste,0);
> val it = ([], []) : char list * char list
- splitAt (tomliste,2);
> val it = ([], []) : char list * char list
- splitAt (abcliste,0);
> val it = ([], [#"a", #"b", #"c"]) : char list * char list
- splitAt (abcliste,2);
> val it = ([#"a", #"b"], [#"c"]) : char list * char list
- splitOn (tomliste,fn c => c = #"b");
> val it = ([], []) : char list * char list
- splitOn (abcliste,fn c => c = #"b");
> val it = ([], [#"a", #"b", #"c"]) : char list * char list
- splitOn (abcliste,fn c => c <= #"b");
> val it = ([#"a", #"b"], [#"c"]) : char list * char list
- repeat (abc,0);
> val it = [] : string list
- repeat (abc,2);
> val it = ["abc", "abc"] : string list
- blanke 0;
> val it = "" : string
- blanke 2;
> val it = "  " : string
- venstretving (abc,0);
> val it = "" : string
- venstretving (abc,2);
> val it = "ab" : string
- venstretving (abc,3);
> val it = "abc" : string
- venstretving (abc,5);
> val it = "abc  " : string
- hoejrestil (abc,0);
> val it = "abc" : string
- hoejrestil (abc,3);
> val it = "abc" : string
```

```
- hoejrestil (abc,5);
> val it = "  abc" : string
- streg 3;
> val it = "---\n" : string
- flet op <= ([],[]);
> val it = [] : int list
- flet op <= ([],abcliste);
> val it = [#"a", #"b", #"c"] : char list
- flet op <= (abcliste,[]);
> val it = [#"a", #"b", #"c"] : char list
- flet op <= ([2,5,7],[1,3,5]);
> val it = [1, 2, 3, 5, 5, 7] : int list
- fsort op <= [];
> val it = [] : int list
- fsort op <= [10];
> val it = [10] : int list
- fsort op <= [3,9,10];
> val it = [3, 9, 10] : int list
- fsort op <= [10,3,9];
> val it = [3, 9, 10] : int list
- fjernAfslutningsblanke "";
> val it = "" : string
- fjernAfslutningsblanke "abc";
> val it = "abc" : string
- fjernAfslutningsblanke "234\n  \t";
> val it = "234" : string
- talFraSpejlcifliste (explode "");
> val it = 0 : int
- talFraSpejlcifliste (explode "12");
> val it = 21 : int
- talFraSpejlcifliste (explode "1A");
> val it = 171 : int
- splitITekstOgTal "";
! Uncaught exception:
! Fail "Tal mangler efter teksten "
- splitITekstOgTal abc;
! Uncaught exception:
! Fail "Tal mangler efter teksten abc"
- splitITekstOgTal "24";
> val it = ("", 24) : string * int
- splitITekstOgTal "abc13";
> val it = ("abc", 13) : string * int
- splitITekstOgTal "abc  12      13";
> val it = ("abc  12", 13) : string * int
- rensTekstliste [];
```

```
> val it = [] : string list
- rensTekstliste ["abcd","abc","d","abcd"];
> val it = ["abcd", "abc", "d", "abcd"] : string list
- rensTekstliste ["abcd  ","abc\t \t","","d","\t\t","a b c d "];
> val it = ["abcd", "abc", "d", "a b c d"] : string list
- hentAfstemnresNbrdSTotal [];
> val it = ([], 0, 0) : (string * int) list * int * int
- hentAfstemnresNbrdSTotal ["00","0","0"];
> val it =
    ([("", 0), ("", 0), ("", 0)], 0, 0)
    : (string * int) list * int * int
- hentAfstemnresNbrdSTotal ["A0","00","CCC0"];
> val it =
    ([("A", 0), ("", 0), ("CCC", 0)], 3, 0)
    : (string * int) list * int * int
- hentAfstemnresNbrdSTotal ["A  60","30","C         10"];
> val it =
    ([("A", 60), ("", 30), ("C", 10)], 1, 100)
    : (string * int) list * int * int
- hentOverskrAfstemnresNbrdSTotalMTotal [];
! Uncaught exception:
! Fail "Helt tom indgangstekst"
- hentOverskrAfstemnresNbrdSTotalMTotal ["Eksempel et 5"];
! Uncaught exception:
! Fail "Ingen afstemningsdata"
- hentOverskrAfstemnresNbrdSTotalMTotal ["Eksempel to 5","A0","0","CC  0"];
! Uncaught exception:
! Fail "Ingen stemmer"
- hentOverskrAfstemnresNbrdSTotalMTotal ["Eks. tre 5","A60","30","C   10"];
> val it =
    ("Eks. tre", [("A", 60), ("", 30), ("C", 10)], 1, 100, 5)
    : string * (string * int) list * int * int * int
- tilBData (#2 it,#4 it,#5 it);
> val it =
    ([(1, "A", 60, 3, 0), (2, "", 30, 1, 50), (3, "C", 10, 0, 50)], 4)
    : (int * string * int * int * int) list * int
- val bData = #1 it;
> val bData =
    [(1, "A", 60, 3, 0), (2, "", 30, 1, 50), (3, "C", 10, 0, 50)]
    : (int * string * int * int * int) list
- forhoejMandattal (bData,5,5);
> val it =
    [(1, "A", 60, 3, 0), (2, "", 30, 1, 50), (3, "C", 10, 0, 50)]
    : (int * string * int * int * int) list
- forhoejMandattal (bData,4,5);
```

```
> val it =
    [(1, "A", 60, 4, 0), (2, "", 30, 1, 50), (3, "C", 10, 0, 50)]
    : (int * string * int * int * int) list
- forhoejMandattal (bData,3,5);
> val it =
    [(1, "A", 60, 4, 0), (2, "", 30, 2, 50), (3, "C", 10, 0, 50)]
    : (int * string * int * int * int) list
- forhoejMandattal (bData,3,6);
> val it =
    [(1, "A", 60, 4, 0), (2, "", 30, 2, 50), (3, "C", 10, 1, 50)]
    : (int * string * int * int * int) list
- forhoejMandattal (bData,10,14);
> val it =
    [(1, "A", 60, 4, 0), (2, "", 30, 2, 50), (3, "C", 10, 1, 50)]
    : (int * string * int * int * int) list
- print (hoved (8,10,9,"Testoverskrift"));
Testoverskrift

         Stemmetal Mandater
---------------------------
> val it = () : unit
- print (hale (8,10,9,123123,17));
---------------------------
            123123       17
> val it = () : unit
- print (visFacitlinje (8,10,9) (1,"Socdem",123123,35,319));
Socdem      123123       35
> val it = () : unit
- laesMedStikord "Giv mig et tal!";
Giv mig et tal!
tolv
> val it = "tolv\n" : string
- val indstrm = openIn "nils-hd1:Applications:mosml:DatOekspl:Eksempel1ind";
> val indstrm = <instream> : instream
- hentTekstliste indstrm;
> val it =
    ["Eksempel 1 5   \n", "\n", "A           55     \n", "\n", "B       32\n",
     "C         13\n"]
    : string list
-
```

3.6.8 Bilag: eksempler

```
- fun visfil filnavn = app print (hentTekstliste (openIn filnavn));
> val visfil = fn : string -> unit
```

```
- visfil "nils-hd1:Applications:mosml:Dat0ekspl:Fejleksempel1";


> val it = () : unit
- main ();
Angiv navnet på filen med afstemningsresultaterne; afslut med
linjeskift (en blank linje vil blive fortolket som inddata fra
tastatur):
nils-hd1:Applications:mosml:Dat0ekspl:Fejleksempel1
Angiv navnet på den fil, hvori beregningsresultaterne skal gemmes;
afslut med linjeskift (en blank linje vil blive fortolket som
uddata til skærm):
! Uncaught exception:
! Fail "Helt tom indgangstekst"
- visfil "nils-hd1:Applications:mosml:Dat0ekspl:Fejleksempel2";


Kun en enkelt ikke blank linje 12


> val it = () : unit
- main ();
Angiv navnet på filen med afstemningsresultaterne; afslut med
linjeskift (en blank linje vil blive fortolket som inddata fra
tastatur):
nils-hd1:Applications:mosml:Dat0ekspl:Fejleksempel2
Angiv navnet på den fil, hvori beregningsresultaterne skal gemmes;
afslut med linjeskift (en blank linje vil blive fortolket som
uddata til skærm):
! Uncaught exception:
! Fail "Ingen afstemningsdata"
- visfil "nils-hd1:Applications:mosml:Dat0ekspl:Fejleksempel3";
Et parti uden stemmer 12
A 13
Nulpartiet
C 14
> val it = () : unit
- main ();
Angiv navnet på filen med afstemningsresultaterne; afslut med
linjeskift (en blank linje vil blive fortolket som inddata fra
tastatur):
nils-hd1:Applications:mosml:Dat0ekspl:Fejleksempel3
Angiv navnet på den fil, hvori beregningsresultaterne skal gemmes;
afslut med linjeskift (en blank linje vil blive fortolket som
uddata til skærm):
```

```
! Uncaught exception:
! Fail "Tal mangler efter teksten Nulpartiet"
- visfil "nils-hd1:Applications:mosml:DatOekspl:Fejleksempel4";
Slet ingen stemmer 12
A 00
Nulpartiet 0
C 0
> val it = () : unit
- main ();
Angiv navnet på filen med afstemningsresultaterne; afslut med
linjeskift (en blank linje vil blive fortolket som inddata fra
tastatur):
nils-hd1:Applications:mosml:DatOekspl:Fejleksempel4
Angiv navnet på den fil, hvori beregningsresultaterne skal gemmes;
afslut med linjeskift (en blank linje vil blive fortolket som
uddata til skærm):
! Uncaught exception:
! Fail "Ingen stemmer"
- visfil "nils-hd1:Applications:mosml:DatOekspl:Eksempel1";
Eksempel ét 5

A           55

B       32
C         13
> val it = () : unit
- main ();
Angiv navnet på filen med afstemningsresultaterne; afslut med
linjeskift (en blank linje vil blive fortolket som inddata fra
tastatur):
nils-hd1:Applications:mosml:DatOekspl:Eksempel1
Angiv navnet på den fil, hvori beregningsresultaterne skal gemmes;
afslut med linjeskift (en blank linje vil blive fortolket som
uddata til skærm):
nils-hd1:Applications:mosml:DatOekspl:Eksempel1ud
> val it = () : unit
- visfil "nils-hd1:Applications:mosml:DatOekspl:Eksempel1ud";
Eksempel ét

     Stemmetal Mandater
-----------------------
A           55        3
B           32        1
C           13        1
-----------------------
           100        5
```

```
> val it = () : unit
- visfil "nils-hd1:Applications:mosml:Dat0ekspl:Eksempel2";
Eksempel to  5
P 1            60
P 2            30
P 3            10
> val it = () : unit
- main ();
Angiv navnet på filen med afstemningsresultaterne; afslut med
linjeskift (en blank linje vil blive fortolket som inddata fra
tastatur):
nils-hd1:Applications:mosml:Dat0ekspl:Eksempel2
Angiv navnet på den fil, hvori beregningsresultaterne skal gemmes;
afslut med linjeskift (en blank linje vil blive fortolket som
uddata til skærm):
nils-hd1:Applications:mosml:Dat0ekspl:Eksempel2ud
> val it = () : unit
- visfil "nils-hd1:Applications:mosml:Dat0ekspl:Eksempel2ud";
Eksempel to

     Stemmetal Mandater
-----------------------
P 1         60        3
P 2         30        1
P 3         10        1
-----------------------
           100        5
> val it = () : unit
- visfil "nils-hd1:Applications:mosml:Dat0ekspl:Eksempel3";
Eksempel tre 5
A         40
B         30
C         30
> val it = () : unit
- main ();
Angiv navnet på filen med afstemningsresultaterne; afslut med
linjeskift (en blank linje vil blive fortolket som inddata fra
tastatur):
nils-hd1:Applications:mosml:Dat0ekspl:Eksempel3
Angiv navnet på den fil, hvori beregningsresultaterne skal gemmes;
afslut med linjeskift (en blank linje vil blive fortolket som
uddata til skærm):
Eksempel tre

     Stemmetal Mandater
-----------------------
```

```
A           40        2
B           30        2
C           30        1
----------------------
           100        5
> val it = () : unit
- visfil "nils-hd1:Applications:mosml:Dat0ekspl:EUvalget1999";
EU-parlamentsvalget den 10. juni 1999 16

Socialdemokratiet           324249
Det Radikale Venstre           180116
Det Konservative Folkeparti           166518
Centrum-Demokraterne           68583
Socialistisk Folkeparti           139845

JuniBevægelsen - Mod Unionen           317051
Folkebevægelsen mod EU           143706
Dansk Folkeparti           114859
Kristeligt Folkeparti           39128

Venstre           460823
Fremskridtspartiet           14233
> val it = () : unit
- main ();
Angiv navnet på filen med afstemningsresultaterne; afslut med
linjeskift (en blank linje vil blive fortolket som inddata fra
tastatur):
nils-hd1:Applications:mosml:Dat0ekspl:EUvalget1999
Angiv navnet på den fil, hvori beregningsresultaterne skal gemmes;
afslut med linjeskift (en blank linje vil blive fortolket som
uddata til skærm):
EU-parlamentsvalget den 10. juni 1999

                            Stemmetal Mandater
-----------------------------------------------
Socialdemokratiet              324249        3
Det Radikale Venstre           180116        1
Det Konservative Folkeparti    166518        1
Centrum-Demokraterne            68583        1
Socialistisk Folkeparti        139845        1
JuniBevægelsen - Mod Unionen   317051        3
Folkebevægelsen mod EU         143706        1
Dansk Folkeparti               114859        1
Kristeligt Folkeparti           39128        0
Venstre                        460823        4
Fremskridtspartiet              14233        0
```

```
-----------------------------------------------
                              1969111       16
> val it = () : unit
- main ();
Angiv navnet på filen med afstemningsresultaterne; afslut med
linjeskift (en blank linje vil blive fortolket som inddata fra
tastatur):
Angiv navnet på den fil, hvori beregningsresultaterne skal gemmes;
afslut med linjeskift (en blank linje vil blive fortolket som
uddata til skærm):
Kun ét parti 12
Der til gengæld har et navn på mere end tooghalvfjerds positioner 43
Kun ét parti

                                                    Stemmetal Mandater
-------------------------------------------------------------------------
Der til gengæld har et navn på mere end tooghalvfjerd        43       12
-------------------------------------------------------------------------
                                                             43       12
> val it = () : unit
- I/O failure: Unknown Error (-21)
```

Opgave

Modificer programmet, så kolonnen med partiidentifikationer får betegnelsen "Navn", på samme måde som betegnelserne "Stemmetal" og "Mandater" indleder de to talkolonner. Husk at overveje, hvordan den kolonnemæssige opstilling kan bevares, hvis partiidentifikationerne er kortere end 4 tegn.

Bemærkning

Det er værd at lægge mærke til, hvordan omkring to trediedele af programteksten vedrører ind- og udlæsning, mens kun cirka en trediedel drejer sig om de egentlige numeriske beregninger — et forhold, som er karakteristisk for mange databehandlingsopgaver. Det er også almindeligt, at afprøvning af de mulige fejlsituationer mindst er lige så omfattende som test af "normale" data.

3.7 Otte forskrifter for problemløsning

Fra Ray Hyman & Barry Anderson: Solving Problems, *International Science and Technology*, September 1965.

1. Lad problemstillingens dele passere revy i hurtig følge flere gange, indtil der danner sig et mønster, som omfatter alle dele samtidig

2. Lad vurderingen vente. Drag ikke hastige slutninger.

3. Gå på opdagelse i omgivelserne. Flyt om på tingene i både tid og rum.

4. Find endnu en løsning efter den første.

5. Se kritisk på dine egne ideer. Find de værdifulde træk ved andres.

6. Er du kørt fast, prøv da at finde en anden måde at beskrive problemet på. Hvis en konkret model ikke dur, prøv da en abstrakt, og omvendt.

7. Hold en pause, når du er kørt fast.

8. Fortæl en anden om dit problem.

Kapitel 4

Kursusbeskrivelse for Datalogi 0 GA for efteråret 2003

Beskrivelsen herunder er *foreløbig*. De aktuelle beskrivelser kan hentes elektronisk; se dels Studieinformationssystemets oplysninger: `http://www.sis.ku.dk/`, dels Kursernes hjemmesider (ajourføres af undervisere): `http://www.diku.dk/undervisning/2003e/dat0ga/`.

Mandag i uge 36 Den 1. september 2003 1–2

Simulering. Funktionsprogrammering.

Fænomen, begreb, repræsentation. Databehandling som simulering med en model. Typer og problemorienteret analyse. Paradigmer for universelle maskiner. Funktionsbegrebet. Regneudtryk med operatorer og operander. Gruppering: prioritet og associeringsretning. Navnet `it`. Indføring af nye navne (mønstre uden tilpasning). Funktionsdefinition med `fun`. Konstruere en funktion ved hjælp af andre.

Litteratur: Noter om databehandling, typer og funktionsbegrebet.

Torsdag i uge 36 Den 4. september 2003 3–4

Virkefeltsregler. Sæt. Rekursion.

Kataloger, filer, programmer og vinduer. Sæt, mønstre. Redefinition. Biblioteksmoduler og `load`. Rekursion. Valg. Strikshed. Typen `unit`.

Litteratur: H&R kapitel 1.

Mandag i uge 37 Den 8. september 2003 5–6

Simple typer. Lokale erklæringer. Korrekthed. Robusthed.

Heltal, reelle tal, sandhedsværdier, tegn og tekster. Euklids algoritme. Programmerings- og opstillingsregler. Kommentarer. Terminering; nødstop. Fejlmeldinger (`raise Fail` *tekst*). Robusthed. Korrekthed. Afprøvning versus bevis. Blokke (`let` og `local`). Programkonstruktion ud fra egenskaber.

Litteratur: H&R kapitel 2, eksempel 3.2 og afsnit 3.7.

Torsdag i uge 37 Den 11. september 2003 7–8

Højereordensfunktioner.

Funktioner som funktionsresultat; sekventialiseret form. Funktioner som argument. Anonyme funktioner.

Litteratur: Noter. H&R afsnit 9.1–9.2.

Mandag i uge 38 Den 15. september 2003 9–10

Lister. Lighedstyper. Afsøgning.

Lister og operationer på dem. Mønstersynonymer (`as`). Værdipolymorfi. Lighedstyper. Lineær afsøgning.

Litteratur: H&R kapitel 5. Paulson side 96–98. Noter.

Torsdag i uge 38 Den 18. september 2003 11–12

Naïv sortering. Map.

Simple sorteringsmetoder (med kvadratisk udførelsestid): Boblesortering, indsættelsessortering og udtagelsessortering. Listefunktionalen `map`.

Litteratur: Noter. H&R afsnit 9.3–9.4.

Mandag i uge 39 Den 22. september 2003 13–14

Et større eksempel. Programfiler; moduler.

Et større eksempel (kalender). Programfiler og biblioteksmoduler. Hjælpefunktionerne `use` og `open`.

Litteratur: Noter. H&R appendiks D.

Torsdag i uge 39 Den 25. september 2003 15–16

Programdokumentation. Præsentation af G

Vejledning i problemanalyse, afprøvning og rapportskrivning. G stilles.

Litteratur: Noter om projektarbejde.

Fredag i uge 41 Den 10. oktober 2003 G

Sidste frist for aflevering af G.

Uge 42 *efterårsferie*

Mandag i uge 43 Den 20. oktober 2003 17–18

Økonomi og præcision. Talrepræsentation.

Hvor begynder talrækken? Søg størst muligt gyldighedsområde og færrest mulige specialtilfælde. Division med kvotient og rest. Repræsentation i positionssystemer med et eller flere grundtal. Repræsentation af reelle tal. Faldgruber ved regning med dem (flydende regning).

Litteratur: Noter

Torsdag i uge 43 Den 23. oktober 2003 19–20

Kombinatorisk søgning. Undtagelser.

Møntvekslingsproblemet og problemet "de otte dronninger". Generel signalering (`raise`) og behandling (`handle`) af undtagelser.

Litteratur: Noter. H&R afsnit 7.6–7.7. Paulson side 83–85 og 134–141.

Mandag i uge 44 Den 27. oktober 2003 21–22

Effektivitetsovervejelser. Del og hersk. Pseudotilfældige tal. Effektiv sortering.

$\mathcal{O}$-, Ω- og Θ-notation. Generering af pseudotilfældige tal. Flettesortering og "quicksort" (i de simple grundversioner).

Litteratur: Noter. Paulson side 108–113.

Torsdag i uge 44 Den 30. oktober 2003 23–24

Konstruerede typer. Partielle funktioner.

Data- og typekonstruktører (`datatype`.). Valgkonstruktionen `case`. Typen `order` og typegeneratoren `option`.

Litteratur: H&R kapitel 7 til og med afsnit 7.5.

Mandag i uge 45 Den 3. november 2003 25–26

Træer.

Træstrukturer.

Litteratur: H&R kapitel 8 til og med afsnit 8.5.

Torsdag i uge 45 Den 6. november 2003 27–28

Listefunktioner af højere orden.

`List.filter`, `List.exists`, `List.all`, `List.partition`. Foldningsfunktionalerne `foldr` og `foldl`.

Litteratur: H&R afsnit 9.5, 9.6 og 9.8. Paulson side 182–191.

Mandag i uge 46 Den 10. november 2003 29–30

Ræsonneren om funktionsprogrammer.

Strukturel induktion. Beviser for funktioners egenskaber.

Litteratur: Paulson side 213–237.

Torsdag i uge 46 Den 13. november 2003 31–32

Mængder.

Abstrakt type. Opretholdelse af invariant. Repræsentation af mængder.

Litteratur: H&R kapitel 10.

Mandag i uge 47 Den 17. november 2003 33–34

Moduler. Strukturer. Signaturer. Funktorer.

Strukturering af store programsystemer. Adskillelse af "hvad?" fra "hvordan?".

Litteratur: H&R kapitel 11.

Torsdag i uge 47 Den 20. november 2003 35–36

Stakke og køer. Trægennemløb. Søgetræer.

Sidst ind først ud-lister og traditionelle køer (først ind, først ud). Trægennemløb i præ-, in- og post-orden.

Litteratur: H&R afsnit 8.7 (især eksempel 8.1). Paulson side 141–147 og 257–274. Noter.

Mandag i uge 48 Den 24. november 2003 37–38

Ind- og udlæsning som bivirkning. Iterativ funktionsform.

Typerne `TextIO.instream` og `TextIO.outstream` og deres standardfunktioner. Iterativ eller "halerekursiv" form, specielt af fakultetesfunktionen, listespejlvending og Fibonaccitalberegning.

Litteratur: H&R kapitel 14 til og med afsnit 14.5 samt kapitel 17.

Torsdag i uge 48 Den 27. november 2003 39–40

Programtransformation. Præsentation af K.

Hurtig potensopløftning. Snedige versioner af flettesortering og "quicksort". K stilles.

Litteratur: Noter.

Torsdag i uge 51 Den 18. december 2003 K

Sidste frist for aflevering af K.

Kapitel 5

Diverse

Tidligere udgaver af dette hæfte indeholdt her nogle af de mange love, bekendtgørelser og cirkulærer, som er af interesse for studerende. Efter at dette stof er blevet elektronisk tilgængeligt, henviser vi til, at man søger disse informationer på nettet. Derved opnås også større sikkerhed for, at man får fat i den aktuelt gældende version.

Følgende internetadresser er særligt nyttige i denne forbindelse: Slå op på studieinformationssystemet `http://www.sis.ku.dk`, vælg Det naturvidenskabelige Fakultet og dernæst eksamensregler (det naturvidenskabelige fakultets eksamensregler) eller Love og bekendtgørelser (med Universitetsloven, Eksamensbekendtgørelsen, Karakterbekendtgørelsen, reglerne om første årsprøve, om de studerendes ophavsret over deres opgaver, om ordensregler og disciplinære foranstaltninger og meget mere).

En anden nyttig adresse er Københavns Universitets regelsamling `http://www.ku.dk/regel/index.html`. Her kan man blandt andet finde loven om røgfri miljøer i offentlige lokaler, transportmidler og lignende: `http://www.ku.dk/regel/3/3309.html`.