

Linux – Friheden til at programmere

Version 2.5 – 25/01-2004

Jacob Sparre Andersen

Peter Toft

Hans Schou

Linux – Friheden til at programmere: Version 2.5 – 25/01-2004

af Jacob Sparre Andersen, Peter Toft og og Hans Schou

Ophavsret © 1998-2003 af Jacob Sparre Andersen, Hans Schou, Peter Toft og mange andre under "Åben dokumentlicens (ÅDL) - version 1.0".

En indføring i programmering på og til Linux. Et udvalg af de programmeringssprog der er tilgængelige til Linux, bliver præsenteret sammen med et udvalg af de programmeringsværktøjer der findes til Linux. Bogen indeholder også en introduktion til programmering efter de to hovedparadigmer; imperativ og deklarativ programmering.

Vi har begrænset os til at beskrive værktøjer der distribueres som *frit programmel*, og programmeringssprog hvis specifikation enten lever op til kravene til en *åben standard* eller har en referenceimplementation der distribueres som *frit programmel*.

Indholdsfortegnelse

Forord	i
1. Forord.....	i
2. Linux-bøgerne.....	i
3. Ophavsret	i
4. Om forfatterne og bogens historie	ii
5. Vi siger tak for hjælpen.....	iii
6. Typografi.....	iii
1. Introduktion.....	1
2. Vælg et sprog	3
2.1. Ada	3
2.2. Basic.....	4
2.3. Bourne shell.....	4
2.3.1. Brug af Bourne shell	5
2.4. C.....	5
2.5. C#.....	6
2.6. C++	6
2.7. Cobol	6
2.8. Comal	6
2.9. Csh.....	6
2.10. Eiffel.....	6
2.11. Euphoria	7
2.12. Forth	7
2.13. Fortran.....	7
2.14. Haskell	7
2.14.1. Brug af Haskell.....	7
2.14.2. Doven evaluering.....	8
2.15. Java	8
2.16. Lisp	9
2.17. ML.....	9
2.17.1. Brug af ML	9
2.17.2. Eksempel på SML	10
2.17.3. Iteration i SML.....	11
2.18. Modula-2.....	11
2.19. Modula-3.....	11
2.20. Oberon.....	11
2.21. Objective-C	11
2.22. Objective Caml.....	11
2.22.1. Nogle løse noter om OCaml.....	11
2.23. Octave.....	12
2.24. Pascal.....	12
2.25. Perl.....	12
2.26. Postscript	12
2.27. Prolog	12
2.28. Python	13
2.29. Ruby	13
2.29.1. Hej verden!.....	13
2.29.2. Et større eksempel.....	13
2.29.3. One-liners i Ruby	15
2.30. Scheme	15
2.31. Smalltalk	15
2.32. SQL	15
2.32.1. Hello, world!.....	16
2.33. Tcl/Tk.....	16
2.34. TeX	16

3. Imperativ programmering	17
3.1. Variable og konstanter	17
3.2. Tildelinger og typer	17
3.3. Funktioner og procedurer	17
3.4. Betingede strukturer	17
3.5. Løkker	18
3.6. Dynamiske kald	18
3.7. Logisk parallelisering	18
4. Deklarativ programmering	19
5. Programmørens værktøjskasse	21
5.1. Concurrent versioning system (CVS)	21
5.1.1. CVS' funktionalitet	22
5.1.2. Daglig brug af CVS	22
5.1.3. Opsætning af CVS	26
5.1.4. Forgreneinger i udviklingen	27
5.1.5. Diverse med CVS	29
5.2. diff og patch	29
5.3. Emacs og forskellige udgaver af filer	30
5.4. xxdiff	32
5.5. GNU debuggeren og DDD	33
5.6. GNU Visual Debugger	34
5.7. Glade	35
5.8. Gettext	37
5.9. Electric Fence	38
5.10. Brug af oversættelsesfiler ("makefiles")	38
5.11. Autoconf og automake	40
5.11.1. Autoconf, underkataloger	44
5.11.2. Autoconf/automake eksempel	45
5.11.3. Installation af andre filer	45
5.11.4. Tjek at en funktion er til stede	46
5.11.5. Variable i config.h	46
5.11.6. Autoconf og andre sprog	47
5.12. Anjuta	47
5.13. KDevelop	47
5.14. emacs	49
5.15. vi	49
5.16. GRASP	49
5.17. a2ps	49
A. Installationsvejledninger	51
A.1. Ada	51
A.2. Anjuta	52
A.3. Basic	53
A.4. Cobol	53
A.5. Comal	53
A.6. Eiffel	53
A.6.1. På RPM-baserede systemer	53
A.6.2. På Debian-baserede systemer	53
A.7. Forth	53
A.8. Fortran	53
A.9. GLADE	54
A.10. Haskell	54
A.10.1. På RPM-baserede systemer	54
A.10.2. På Debian-baserede systemer	54
A.11. Java	54
A.11.1. Installation af Java3D	55
A.12. ML	56
A.13. Modula-2	56
A.13.1. På RPM-baserede systemer	56
A.13.2. På Debian-baserede systemer	56
A.14. Objective Caml	56
A.15. Pascal	56
A.16. Perl	57
A.17. POSIX Shell	57

A.18. Postscript	57
A.18.1. På RPM-baserede systemer	57
A.18.2. På Debian-baserede systemer	57
A.18.3. Brug af ghostscript	57
A.19. Python	57
A.20. Smalltalk	57
A.21. SQL	57
A.22. Tcl/Tk	58
A.23. TeX	59
B. Kodebiblioteker	61
B.1. Grafiske brugergrænseflader	61
B.2. Grafik (2D og 3D)	61
B.3. Beregninger	61
B.4. Internettet	61
C. Bøger om programmering	63
C.1. Generelt om imperativ programmering	63
C.1.1. Objektorienteret programmering	63
C.1.2. Bøger om Ada	63
C.1.3. Bøger om Bash	63
C.1.4. Bøger om Basic	63
C.1.5. Bøger om C	64
C.1.6. Bøger om C++	64
C.1.7. Bøger om COMAL	64
C.1.8. Bøger om Fortran	64
C.1.9. Bøger om Haskell	65
C.1.10. Bøger om Java	65
C.1.11. Bøger om ML	65
C.1.12. Bøger om Objective Caml (OCaml)	65
C.1.13. Bøger om Perl	66
C.1.14. Bøger om POSIX Shell	66
C.1.15. Bøger om Postscript	66
C.1.16. Bøger om Ruby	66
C.1.17. Bøger om Scheme	66
C.1.18. Bøger om Tcl/Tk	67
C.1.19. Bøger om TeX	67
C.2. Generelt om deklarativ programmering	67
C.2.1. Bøger om Prolog	67
C.3. Sjove tekster om programmering	67
D. Revisionshistorie for bogen	69
Ordliste	71
Stikordsregister	73

Forord

1. Forord

Denne bog er skrevet som en indføring i programmering på og til Linux. Et udvalg af de programmeringssprog der er tilgængelige til Linux, bliver præsenteret sammen med et udvalg af de programmeringsværktøjer der findes til Linux. Bogen indeholder også en introduktion til programmering efter de to hovedparadigmer; imperativ og deklarativ programmering.

For den der har lyst til at udvikle programmer, er Linux det rene paradys. Der er et bredt udvalg af programmeringssprog og værktøjer at vælge mellem. Langt de fleste er af høj kvalitet og frit tilgængelige. Der er langt flere muligheder end dem vi kommer ind på her. Hvis dit foretrukne programmeringssprog ikke er nævnt i bogen, betyder det ikke nødvendigvis at det ikke findes til Linux.

Vi har begrænset os til at beskrive værktøjer der distribueres som *frie programmer*, og programmeringssprog hvis specifikation enten lever op til kravene til en *åben standard* eller har en referenceimplementation der distribueres som et *frit program*.

Hvis du skal til at gå i gang med at programmere, er der nogle forberedende trin du med fordel kan tage først. Du bør finde ud af hvilket programmeringssprog er bedst egnet til den opgave du vil løse. Du bør finde ud af hvilke biblioteker der findes med færdige løsninger på dele af de ting dit program skal udrette. Og du bør finde en samling værktøjer der kan hjælpe dig med programmeringsarbejdet. Kapitlet "Vælg et sprog" (Kapitel 2) skal give en overordnet idé om hvilke styrker og svagheder de sprog du kan vælge imellem, har. I appendikset "Biblioteker" (Appendiks B) har vi prøvet at lave en oversigt over hvad der findes af biblioteker til at løse bestemte opgaver. Kapitlet "Programmørens værktøjskasse" (Kapitel 5) introducerer dig til de vigtigste værktøjer du bør have kendskab til, når du programmerer. Da denne bog ikke indeholder konkrete introduktioner til at programmere i de omtalte sprog, har du nok også brug for sprogspecifikke bøger når det gælder selve det at programmere. Til det formål findes der i appendikset "Bøger om programmering" (Appendiks C) litteraturlister for alle de sprog der er omtalt i Kapitel 2. Endelig indeholder appendikset "Installation" (Appendiks A) installations- og brugsanvisninger til de forskellige sprogs oversættere/fortolkere.

2. Linux-bøgerne

Bogen er en del af en serie, som kan findes på <http://www.linuxbog.dk/>

- *Linux – Friheden til at vælge installation* – Om at installere Linux.
- *Linux – Friheden til at lære Unix* – Om hvordan man bruger Linux' (og Unix') kommandolinjeværktøjer.
- *Linux – Friheden til at vælge grafisk brugergrænseflade* – Om alle de grafiske brugergrænseflader, der findes til Linux.
- *Linux – Friheden til at vælge programmer* – Om de programmer du kan få til Linux.
- *Linux – Friheden til systemadministration* – Om at administrere sit eget linuxsystem.
- *Linux – Friheden til at programmere* – Programmering på Linux
- *Linux – Friheden til at programmere i C* – Om at programmere i sproget "C".
- *Linux – Friheden til at programmere i Java* – Om at programmere i sproget "Java".
- *Linux – Friheden til sikkerhed på internettet* – Om at sikre dit Linuxsystem mod indbrud fra internettet.
- *Linux – Friheden til egen webserver* – Om at sætte en webserver med databaser, CGI-programmer og andet godt op.
- *Linux – Friheden til at skrive dokumentation* – Om at skrive dokumentation (og andet) i SGML/DocBook, LaTeX eller andre formater.
- *Linux – Friheden til at vælge kontorprogrammer* – Kontorfunktioner på et Linux/KDE/OpenOffice.org-system.
- *Linux – Friheden til at vælge IT-løsning* – Om muligheder, fordele og ulemper ved at bruge Linux i sin IT-løsning.
- *Linux – Friheden til at vælge OpenOffice.org* – Om at bruge OpenOffice.org, både på Linux og på andre styresystemer.
- *Linux – Friheden til at vælge digital signatur* – Digital signatur på Linux.

3. Ophavsret

Denne bog er skrevet af Linux-brugere til Linux-brugere. Store dele af bogen er skrevet eller redigeret af enkelte forfattere, hvilket er nævnt i revisions-historien til bogen.

Bogen kan findes i opdateret form på SSLUG's hjemmesider: <http://www.linuxbog.dk/>, mens prøve-udgaver kan findes på <http://cvs.linuxbog.dk/>.



Figur 1. ÅDL

Bogen er udgivet under "Åben dokumentlicens (ÅDL) – version 1.0" som kan læses på <http://www.linuxbog.dk/licens.html>. Du har bl.a. herved frit lov til at kopiere dette værk uændret på ethvert medium.

Kommentarer, ris og ros og specielt fejl og mangler bedes sendt til linuxbog@sslug.dk⁵, men er du medlem af SSLUG kan du i stedet for med fordel skrive til sslug-bog@sslug.dk⁶.

4. Om forfatterne og bogens historie

Det hele startede som et bogprojekt i sommeren 1998 hvor Peter Toft, Kenneth Geissshirt og Snebjørn Andersen fik skrevet "Linux - friheden til at vælge". Bogen var i starten rettet mod at hjælpe folk med at få installeret Linux, at lidt få lidt kendskab til Unix og noget om netværk. Bogen har altid været en fri "web-bog" der kan læses enten i smådele eller som en helhed på <http://cvs.linuxbog.dk/>.

Vi, forfatterne, har anvendt Linux i flere år og har stor erfaring med det. Vi er medlemmer af Skåne Sjælland Linux User Group (SSLUG)⁸ som er en uafhængig svensk/dansk forening der søger at udbrede kendskabet til Linux og frit programmel i øvrigt. Det er gratis at være medlem af SSLUG. Foreningen findes på internettet på <http://www.sslug.dk/>. Vi har skrevet denne bog i håb om at den kan være til glæde for andre Linux-brugere.

Bogen blev udgivet i bogform på forlaget IDG i slutningen af 1998. Dengang var den et hæfte på 96 sider. Hos IDG var den en bestseller i lang tid. Cirka et år senere blev bogen i version 2.9 udgivet på forlaget GLOBE (med en del ærgerlige konverteringsfejl) – nu var det en bog på over 300 sider.

Fra version 3.6 er det Hans Schou og Peter Toft som vedligeholder bogen sammen med de mange som skriver på dele af bogen.

Siden version 1.5 (af programmeringsbogen) har Jacob Sparre Andersen været hovedredaktør på programmeringsbogen.



Figur 2. Jacob Sparre Andersen (foto: Peter Toft)

5. Vi siger tak for hjælpen

Vi har haft stor glæde af mange SSLUG-medlemmers støtte, rettelser og forslag til forbedringer – bliv ved med dette. Specielt vil vi nævne:

- Jacob Laursen har hjulpet meget med at få lavet splittet af "Linux – Friheden til at vælge" på en god måde.
- Forlaget GLOBE for en stor sproglig revision.
- Ole Vilmann for en masse om Java.
- Rolf Larsen (gentagne gange) og Steen Jensen for at have læst bogen fra ende til anden og aflevere en stribe rettelser til hele bogen.
- Katja Blankensteiner for en lang stribe sproglige rettelser og en støtte igennem hele projektet.
- Andre bidragydere er: Anna Jonna Armannsdottir, Anders Bo Rasmussen, Allan *Swoop* Andersen, Bugge T. Jensen, Erik Sørensen, Lars Kirkeskov Pedersen, Joergen Ramskov, Michael Lerskov Munk Nielsen, Morten Liljeberg, Mikkel Mastrup, Peter Makhholm og Stefan Klukowski.

Du kan i Appendiks D finde en liste over alle de ændringer som bogen har været igennem.

Hvis du støder på ord du ikke forstår, og de ikke findes i bogens ordliste, så kan <http://www.whatis.com/> være et nyttigt opslagsværk. Desværre er det kun på engelsk. Du må meget gerne også skrive til redaktionen¹¹, så vi kan få ordet med i ordlisten i næste udgave.

6. Typografi

Vi vil afslutte indledningen med at nævne den anvendte typografi.

- Navne på filer og kataloger skrives som `foo.bar`.
- Kommandoer du udfører ved at skrive dem på en kommandolinje, skrives som **help**.
- Der er flere steder i bogen hvor vi viser hvad brugeren taster, og hvad Linux svarer. Det vil se ud som:

```
hven% Dette taster brugeren  
Dette svarer Linux.
```

- Der er tilsvarende flere steder i bogen hvor vi viser hvad systemadministratoren (root) taster, og hvad Linux svarer. Det vil se ud som:

```
hven# Dette taster systemadministratoren  
Dette svarer Linux.
```

Slutbemærkning:

1. <http://www.linuxbog.dk/>
2. <http://www.linuxbog.dk/>
3. <http://cvs.linuxbog.dk/>
4. <http://www.linuxbog.dk/licens.html>
5. <mailto:linuxbog@sslug.dk>
6. <mailto:sslug-bog@sslug.dk>
7. <http://cvs.linuxbog.dk/>
8. <http://www.sslug.dk/>
9. <http://www.sslug.dk/>
10. <http://www.whatis.com/>
11. <mailto:linuxbog@sslug.dk>

Kapitel 1. Introduktion

Det specielle ved Linux som udviklingsplatform kan lettest sammenfattes som følgende tre punkter:

- Der er et stort udvalg af programmeringssprog, så det er muligt at *vælge det rigtige sprog til opgaven*.
- Der er en stor samling værktøjer der kan *sættes sammen efter smag og behov*.
- Programmer skrevet på/til Linux kan typisk uden videre *køre på alle Unix-systemer*.

Denne bog er forsøgt skrevet så den dels uddyber de to første punkter i den ovenstående liste og dels kan fungere som en generel introduktion til programmering.

Bogens første del forklarer en lang række forskellige sprogs fordele og ulemper. Bogens anden og tredje del er kurser i programmering. Og bogens fjerde del omhandler de mange værktøjer der findes til at lette programmørers arbejde i Linux. Dertil kommer et appendiks med installationsvejledninger for de i bogen omtalte programmer.

Hvor der ikke er angivet andet, går vi ud fra at du bruger **zsh** som kommandofortolker. Typisk vil det ikke være forskelligt fra **bash** eller **ksh**, mens **tcsh** er væsentligt anderledes.

Kapitel 2. Vælg et sprog

Der findes rigtig mange programmeringssprog til Linux. Af dem er der mere end tredive forskellige du kan få en "fri" oversætter eller fortolker til. De enkelte sprog er lavet med bestemte mål for øje, og er dermed velegnede til bestemte typer opgaver. En Linux-programmør benytter sig typisk af flere sprog, alt efter hvilke opgaver der skal løses. Formålet med denne del af bogen er at introducere et stort antal sprog med tilstrækkelig information til at gøre det muligt at vurdere hvilke sprog der skal bruges i forskellige sammenhænge. Sprogene bliver nævnt i alfabetisk orden.

2.1. Ada

Ada er et sprog der er udviklet til store projekter hvor det er vigtigt at der ikke er fejl i systemerne, men det kan også med fordel bruges til mindre projekter. Den nye GNU Visual Debugger er et eksempel på et "mindre" Ada-projekt. Blandt de større kan man for eksempel nævne Boeing 777-serien eller det nye franske hangarskib "Charles de Gaulle".

En af Adas styrker er systemprogrammering hvor man ofte har brug for at implementere abstraktioner af system-specifikke ting som bitmønstre og portadresser på en effektiv måde.

I det omfang man ikke direkte gør sig umage med at skrive et Ada-program til en bestemt arkitektur (processor, andet isenkram og eventuelt styresystem), kan man regne med at det fungerer ens på alle systemer. Dele af Ada er selvfølgelig nødt til at være systemspecifikke – det gælder for eksempel hvor store heltal man kan regne med – men enhver Ada-oversætter skal komme med dokumentation af hvordan de systemspecifikke dele er lavet i den.

Ada har indbygget adgang til biblioteker skrevet i andre sprog. Standarden¹ beskriver grænsesnippet til Cobol, C og Fortran, men GNU Ada kommer også med grænsesnit til Java og C++. Du har således let adgang til at genbruge gammel kode skrevet i andre sprog.

GNU Ada oversætteren (også kendt som GNAT) blev udviklet parallelt med den seneste ISO-standard for Ada (ISO/IEC 8652:1995) og var allerede da standarden udkom i februar 1995, overordentlig brugbar. I dag er det en oversætter i en kvalitet der gør at mange store firmaer vælger GNU Ada når de har brug for en Ada-oversætter. GNU Ada fås blandt andet til Linux, DOS, Digital Unix, AIX, OS/2 og Windows. Udviklerne bag GNU Ada gør meget ud af at oversætteren følger ISO-standarden for Ada. — Det at følge standarderne er i det hele taget noget man går meget op i i Ada-miljøet. Siden GCC 3.1 har GNAT været integreret i den officielle GCC-distribution.

Ada har, i modsætning til de fleste andre programmeringssprog, indbygget mulighed for at skrive programmer hvor forskellige dele af programmet kører parallelt med hinanden. Man kan for eksempel benytte parallelprogrammering hvis man vil lade sit program fordele sig på alle computerens processorer, eller hvis forskellige dele af programmet skal køre på forskellige maskiner.

Du kan finde en installationsvejledning til GNU Ada i Afsnit A.1 og henvisninger til bøger og websteder om Ada i Afsnit C.1.2.

Eksempel 2-1. »99 bottles of beer on the wall«

Som et eksempel på hvordan man bruger forskellige kontrolstrukturer i Ada, kan vi se på dette lille program der udskriver teksten til sangen »99 bottles of beer on the wall«:

```
with Ada.Text_IO, Ada.Integer_Text_IO;

procedure Bottles is
  procedure Put_Bottles (Count : in Natural) is
    use Ada.Text_IO, Ada.Integer_Text_IO;
  begin
    case Count is
      when 0 =>
        Put ("No bottles");
      when 1 =>
        Put ("1 bottle");
      when others =>
        Put (Count, Width => 0); Put (" bottles");
    end case;
  end Put_Bottles;

  use Ada.Text_IO;
begin
  for Count in reverse 1 .. 99 loop
    Put_Bottles (Count); Put_Line (" of beer on the wall,");
    Put_Bottles (Count); Put_Line (" of beer.");
  end loop;
end;
```

```
Put_Line ("Take one down and pass it around.");
Put_Bottles (Count - 1); Put_Line (" of beer on the wall.");
New_Line;
end loop;
end Bottles;
```

- Den første linje i kildeteksten (with ...) importerer de pakker vi skal bruge i programmet.
- Selve programmet er en procedure, `Bottles`, der er delt op i deklARATIONER (fra `procedure Bottles` is til `begin`) og ordrer (fra `begin` til `end Bottles`).
- I dette tilfælde er der to deklARATIONER; en procedure til at skrive antallet af flasker pænt med korrekte entals- og flertalsformer, og en use-deklARATION der gør de nævnte pakkers indhold direkte synligt i resten af proceduren (så man ikke behøver at foranstille pakkernes navn).

Hvis du gemmer programmets kildetekst² i filen `bottles.adb` og i øvrigt har installeret GCC's Ada-oversætter, så kan du oversætte det med kommandoen:

```
[tyge@hven ~]$ gnatmake bottles.adb -cargs -gnat11 -gnatf -gnato -fstack-check
og dernæst udføre det med kommandoen:
[tyge@hven ~]$ ./bottles
99 bottles
[...]
No bottles of beer on the wall.
```

Eksempel 2-2. Kommandolinjetilvalg til GCC

I eksemplet Eksempel 2-1 bruger vi kommandoen **gnatmake** med en lang række tilvalg der er nødvendige for at få GCC's Ada-oversætter til at opføre sig korrekt (som der står i Ada-standardens). Dem er der ikke nogen grund til selv at skulle skrive hver gang man skal oversætte et program. Hvis man indsætter koden:

```
function gm () {
  gnatmake ${*} -cargs -gnat11 -gnatf -gnato -fstack-check
}
```

i `${HOME}/.zshrc`, vil man i Zsh kunne nøjes med at skrive:

```
[tyge@hven ~]$ gm bottles.adb
```

når man skal oversætte programmet fra eksemplet Eksempel 2-1.

Det kan være nyttigt med lidt mere detaljerede fejlmeddelelser end »der er en fejl på linje 17«. Derfor bør man udvide definitionen ovenfor med tilvalget `-gnatv`:

```
function gm () {
  gnatmake ${*} -cargs -gnatv -gnat11 -gnatf -gnato -fstack-check
}
```

2.2. Basic

Basic var oprindeligt et fortolket sprog, men nutildags kan man også få oversættere til Basic.

2.3. Bourne shell

Bourne shell er den oprindelige unixshell som på alle unixsystemer er installeret som `/bin/sh`, som i dag er defineret af Single Unix Specification version 3 (SUSv3, tidligere POSIX).

Der er flere forskellige shells der implementerer bourne shell. På linuxmaskiner vil `/bin/sh` sædvanligvis være et link til `bash` (Bourne Again Shell) der når den bliver kaldt som `sh` kører i en speciel kompatibilitetsmode. Men også `ksh` (Korn Shell) og `zsh` implementerer bourne shell, dog med hver deres små udvidelser.

Shellscripts bliver for det meste brugt til at binde andre programmer sammen med. Ud over Bourne shell findes også `chs` og `batch`-filer i MS-DOS er egentlig også shellscripts.

De to hovedfordele ved at bruge Bourne shell er:

- At programmer skrevet i Bourne shell umiddelbart kan køres på alle systemer der overholder SUSv3-standard.
- At Bourne shell som kommandofortolkersprog er velegnet til at binde større programmer sammen, så de løser en bestemt opgave.

Medmindre man har meget gode grunde til det, bør man ikke skrive programmer der skal deles med andre, i andre kommandofortolkersprog end Bourne shell. Andre kommandofortolkersprog giver i nogle tilfælde mulighed for at løse opgaver en smule mere elegant, men det er sjældent at det opvejer fordelene ved at det er Bourne shell man kan regne med findes på stort set alle styresystemer. Den primære eksistensberettigelse for bash, ksh og zsh er dog i deres egenskaber ved interaktiv brug, hvilket er uden for denne bogs emner.

De egenskaber der gør sproget nyttigt er:

- Andre kommandoer indgår naturligt som en integreret del af sproget (grep, sort, cat, ...)
- Pipes og redirection gør det let at tage uddata fra kommandoer og føre dem videre over i andre kommandoer eller gemme dem i filer.

Desuden forøger man sin effektivitet ved brug af kommandolinjen ved at kunne det bagvedliggende scriptingsprog Bourne shell er så integreret i Unix at sproget bliver gennemgået som et selvstændigt afsnit i Friheden til at lære Unix.

2.3.1. Brug af Bourne shell

Bourne scripts gemmes normalt i filer. Disse skal så som første linje hedde `#!/bin/sh`. Følgende er et eksempel på et script

```
#!/bin/sh

while read p; do
    echo p
    if [ -z "$p" ] ; then
        exit 0
    fi
done
```

Bourne scripts kan også skrives direkte på kommandolinjen:

```
[tyge@hven ~]$ for i in foo bar baz; do echo $i; done
foo
bar
baz
[tyge@hven ~]$
```

2.4. C

Programmeringssproget C blev designet til at skrive det oprindelige Unix-styresystem i. Det er designet efter at en C-oversætter ikke må bruge særlig mange ressourcer. Den oprindelige C-oversætter gennemgik for eksempel kildeteksten tre gange og lagrede mellemresultater i andre filer for at undgå at bruge for meget hukommelse.

På grund af C's oprindelige formål er det i dag god latin – i det mindste blandt Unix-tilhængere – at styresystemer skal skrives i C.

Skal et program køre hurtigt, så er C et godt valg. C har dog en syntaks der dels er svær at forstå for begyndere og dels kræver stor opmærksomhed hvis man vil undgå fejl i sine programmer. Det er for eksempel både op til brugeren at allokerer hukommelse til datastrukturer og at sørge for at programmet kun skriver inden for den allokerede hukommelse. Dette er en egenskab ved C der i tidens løb har ledt til mange problematiske programmeringsfejl og sikkerhedshuller. Det er samtidig også en af grundene til at det kan tage lang tid at finde fejl i C-programmer. C har også en meget primitiv repræsentation af strenge. I det hele taget har C et lavt abstraktionsniveau og en alvorlig mangel på beskyttelse mod utilsigtede oversættelser mellem de forskellige datatyper.

Der findes i dag en ISO-standard for programmeringssproget C (ISO/IEC 9899:1999), men det er desværre ikke almindeligt at C-oversættere udsættes for en formel afprøvning af om de lever op til standarden. GCC har (vistnok) siden version 3.0 levet op til ISO-standarden.

2.5. C#

Da C#-standarden endnu ikke er færdig, og da der ej heller findes en fungerende fri C#-oversætter, er dette afsnit blot en smagsprøve på hvad fremtiden kan byde på.

C# (udtales på engelsk "c sharp"; på dansk måske "cis", "c-havelåge" eller "c-kryds") ligner Java (se Afsnit 2.15) overordentlig meget, men der er blandt andet tilføjet "delegates" og "egenskaber" (på engelsk "properties") som kendes fra Microsofts Visual Basic og fra Borlands Object Pascal, operator-overlæsning som kendes fra Ada (se Afsnit 2.1) og C++ (se Afsnit 2.6), og poster (på engelsk "structs" eller "records") som kendes fra praktisk taget alle andre programmeringssprog end Java, samt muligheden for at manipulere direkte med hukommelsesadresser med de risici som det medfører.

Sproget er designet af Microsoft til .NET-plattformen, men bliver standardiseret under ECMA³. Der er flere projekter som sigter på at implementere C# på andre platforme end MS Windows og .NET; DotGNU (Southern Storm i Australien), Mono (Ximian i USA) og et Microsoft-støttet projekt rettet mod FreeBSD.

2.6. C++

Praktisk taget C-kompatibelt. Objektorienteret.

Skal et program køre hurtigt, så er C og C++ gode valg af programmeringssprog. Både C og C++ har dog en syntaks der dels er svær at forstå for begyndere og dels kræver stor opmærksomhed hvis man vil undgå fejl i sine programmer. Både i C og i C++ er det op til brugeren at allokere hukommelse til datastrukturer og sørge for at programmet kun skriver inden for den allokerede hukommelse. Dette er en egenskab ved C der historisk har ledt til mange problematiske programmeringsfejl, og en grund til at det kan tage lang tid at finde fejl i C- og C++-programmer. C har også en meget primitiv repræsentation af strenge. C++ er lidt bedre på dette punkt, men for eksempel Perl er et langt stærkere sprog til behandling af store mængder tekst. I det hele taget har C (og til en vis grad også C++) et lavt abstraktionsniveau og en alvorlig mangel på beskyttelse mod utilsigtede oversættelser mellem de forskellige datatyper.

Der findes i dag en ISO-standard for programmeringssproget C++ (ISO/IEC 14882), men det er desværre ikke almindeligt at C++-oversættere udsættes for en formel efterprøvning af om de lever op til standarden. Dertil kommer at en gruppe forskere og oversætterudviklere hos IBM har sået tvivl om om det overhovedet er muligt at lave en C++-oversætter, der svarer til ISO-standarden.

2.7. Cobol

Et af Cobols særlige kendetegn er en temmelig effektiv måde at formulere betingelser der afhænger af flere parametre. Cobol bliver hovedsageligt brugt til forretningssystemer. Ifølge en rapport fra en international konsulentvirksomhed er Cobol den dag i dag det mest anvendte programmeringssprog (så er spørgsmålet bare hvad "mest anvendt" betyder).

Definitionen af Cobol foregår i den internationale standardiseringsorganisation ISO. Den seneste ISO-standard for Cobol udkom i 1985 (ISO/IEC 1989:1985).

2.8. Comal

Comal er et Basic-lignende sprog med et noget begrænset typesystem. Sproget regnes for at være velegnet til undervisningsformål.

2.9. Csh

Csh er et fortolket sprog og en interaktiv kommandofortolker. Dens syntaks skal forestille at ligne C's.

2.10. Eiffel

Eiffel ... "Programming by contract" ...

2.11. Euphoria

???

2.12. Forth

Stakorienteret? ...

2.13. Fortran

... Primært beregnet til beregninger. ... Fortran-oversættere har generelt ry for at generere meget effektive programmer.

Fortran er standardiseret i den internationale standardiseringsorganisation ISO. Den seneste ISO-standard for Fortran udkom i 1997 (ISO/IEC 1539-1:1997).

2.14. Haskell

Haskell er et funktionsorienteret programmeringssprog og minder på mange måder om SML. Mange af de ting der er beskrevet i afsnittet om SML gælder derfor også for Haskell. Der er typepolymorfi, højerordens funktioner og funktionsdefinition ved mønstre.

Den første måde hvor Haskell adskiller sig fra SML er at Haskell er et rent funktionsorienteret sprog, hvor SML har nogle imperative dele. Det har især betydning for I/O der i Haskell foregår ved hjælp af et koncept der hedder monader.

Den anden forskel er at Haskell er et dovent sprog. Dette betyder at argumenter til funktioner først bliver evalueret når den kaldte funktion benytter dem. Dette giver blandt andet mulighed for at behandle uendelige datastrukturer, for eksempel uendelige lister af tal.

Der er oversættere og fortolkere til de fleste platforme herunder *BSD, Linux, MacOS og Windows.

2.14.1. Brug af Haskell

Skal man lege med Haskell bør man tage fat i Hugs. Den implementerer en læs-beregn-udskriv løkke hvor man kan skrive et udtryk og få Hugs til at beregne udtrykket. Man kan dog ikke definere nye funktioner, de skal skrives i filer.

Hvis man for eksempel har en fil `Test.hs` med følgende indhold

```
mylength :: [t] -> Int
mylength []      = 0
mylength (x:xs)  = 1 + length xs

mysum :: [Int] -> Int
mysum []          = 0
mysum (x:xs)      = x + sum xs
```

En session med Hugs kan så se således ud

Version: November 2002

Hugs 98: Based on the Haskell 98 standard
Copyright (c) 1994-2002
World Wide Web: <http://haskell.org/hugs>
Report bugs to: hugs-bugs@haskell.org

Haskell 98 mode: Restart with command line option -98 to enable extensions

Reading file "/usr/lib/hugs/lib/Prelude.hs":

Kapitel 2. Vælg et sprog

```
Hugs session for:
/usr/lib/hugs/lib/Prelude.hs
Type :? for help
Prelude> :load Test.hs
Reading file "Test.hs":
```

```
Hugs session for:
/usr/lib/hugs/lib/Prelude.hs
Test.hs
Main> mylength [3, 6, 2, 5]
4
Main> mysum [3, 6, 2, 5]
16
Main>
```

Man kan så inde fra Hugs redigere `Test.hs` med kommandoen `:edit Test.hs` der starter en editor og efterfølgende genindlæser filen

2.14.2. Doven evaluering

Haskell er et dovent sprog. Det betyder at den udsætter beregningen af et udtryk så lang tid som det overhovedet er muligt. Normalt er man vant til at argumenterne i et funktionskald bliver beregnet før funktionen kaldes. Sådan gør Haskell ikke.

Man ville for eksempel forvente at følgende stykke kode gik i uendelig løkke:

```
foo :: Int -> Int
foo 0 = 1
foo n = foo(n+1) + foo(n-1)

switch :: Bool -> t -> t -> t
switch True n m = n
switch False n m = m

switch True (foo 0) (foo 5)
```

Men eftersom værdien af `(foo 5)` bare bliver smidt vil Haskell ikke forsøge at beregne det.

Dette giver mulighed for at definere strukture der egentligt er uendelige. For eksempel kan Fibonacci's talfølge kan let skrives som:

```
zip :: [a] -> [b] -> [(a,b)]
zip [] _ = []
zip _ [] = []
zip (x:xs) (y:ys) = (x,y) : zip xs ys

add :: (Int,Int) -> Int
add a b = a + b

fib = 0 : 1 : map add (zip fib, (tail fib))
```

`zip` tager to lister og danner en liste af par og `add` tager et par af to tal og lægger dem sammen.

`(zip fib, (tail fib))` laver en uendelig liste af par med to på hinanden følgende tal i Fibonacci's talfølge. `map add (...)` lægger disse par sammen.

2.15. Java

Java er et initiativ til at skabe et programmeringssprog der kan køre på flere styresystemer. Det er udviklet af Sun Microsystems der i 1991 arbejdede med at designe et programmeringssprog der var velegnet til at skrive programmer til fremtidens telefoner, fjernsyn, opvaskemaskiner og andre elektroniske apparater.

Det er langt fra det eneste sprog der er designet til den slags opgaver, men med en blanding af held og god markedsføring er det lykkedes at udbrede sproget.

Med udbredelsen af internettet blev Java samtidigt meget udbredt fordi Sun parallelt med selve sproget har udviklet og udbredt deres version af en *virtuel computer*, bl.a. i webbrowsere. Indlejringen af Suns virtuelle computer i webbrowsere tillader at småprogrammer (på engelsk "applets") kan lægges ind i en hjemmeside.

Fortalere for Java mener at "Java er et enkelt, objektorienteret, robust, netværksorienteret, platformuafhængigt, sikkert, fortolket, højtstående, flertrådet og dynamisk sprog" (citater fra Sun).

Samtidig mener Sun dog ikke at sproget er sikkert nok til at man kan bruge det til alvorlige ting som medicinsk udstyr, flyvemaskiner, trafikstyringssystemer og kernekraftværker. Og i virkeligheden rækker platformuafhængigheden ikke længere end at et Java-program ikke samtidig kan køre hurtigt og korrekt på en almindelig pc.

Det der ofte præsenteres som Javas hovedstyrke, har længe også kunne ses som dets vigtigste hæmsko. Indtil for nylig fandtes der ikke en oversætter der kunne oversætte programmer til maskinkode til en *virtuel* computer. Hvad angår Linux er det problem nu løst, så man ikke længere er tvunget til at oversætte sine programmer til maskinkode til en *virtuel* computer, "the Java Virtual Machine" (JVM). Programmer til JVM vil, selvom Java-programmet egentlig er oversat, skulle *fortolkes* når programmet køres. Det betyder på den ene side at det oversatte program kan køre på alle systemer der har et program der kan fortolke maskinkode til JVM, men på den anden side betyder det også at Java-programmer i al almindelighed er langsommere end andre programmer.

GCC kan oversætte Java.

C++, men uden en del af fejltagelserne i C++' design...

Et enormt udvalg af biblioteker der fungerer ens på alle styresystemer...

2.16. Lisp

Lisp er det klassiske sprog inden for funktionsorienteret programmering og kunstig intelligens. Lisp er et akronym for LISt Processing og er meget velegnet til at arbejde med symbolske data.

Sproget understøtter også metaprogrammering hvilket betyder at funktioner kan benyttes som data. Dette er praktisk til automatisk programgenerering, kodeoptimering og lignende.

Der er en del forskellige dialekter af Lisp hvoraf de tre vigtigste er:

- Common Lisp er den ANSI-standardiserede dialekt. Det er sandsynligvis den dialekt der indeholder flest features.
- Scheme er, modsat Common Lisp, et meget lille sprog. Det beskrives yderligere senere i bogen.
- Elisp er den dialekt Emacs er skrevet i.

Lisps syntaks er let genkendelig på grund af den særlige, og flittige, brug af parenteser:

```
(define increment (lambda (x) (+ x 1)))
(increment (- (* 6 7) 1))
```

2.17. ML

ML er et funktionsprogrammeringssprog der har stærkt typetjek med polymorfi, højereordensfunktioner, funktionsdefinition ved mønstergenkendelse, håndterbare undtagelser (eng. exceptions), strikt evaluering og et avanceret modulsystem. Det betyder at rigtig mange programmeringsfejl bliver opdaget allerede når man oversætter et program, ikke først når man kører det. ML blev oprindeligt (ca. 1974) udviklet som metasprog (meta-language) til logiske bevissystemer, men sproget viste sig at være særdeles velegnet til en lang række andre formål.

I dag (2001) findes der to væsentlige varianter af sproget ML: Standard ML (SML) og Objective Caml (OCaml). Der findes ganske omfattende biblioteker af nyttige funktioner til såvel Standard ML (se SML Basis Library⁴) som OCaml. Desuden har de enkelte implementationer deres egne udvidelser til f.eks. databaseadgang, generering af HTML-kode, PNG-billeder, osv.

ML er især udbredt på universiteter og forskningsinstitutioner, og benyttes i mindre grad i industriel sammenhæng. I Danmark undervises der i Standard ML ved blandt andet Danmarks Tekniske Universitet, Københavns Universitet (DIKU), Aarhus Universitet, Aalborg Universitet og IT-højskolen i København.

2.17.1. Brug af ML

Der findes flere implementationer af SML. Både SML of New Jersey og MoscowML giver mulighed for at lege med SML ved en 'læs-beregn-udskriv løkke' hvor man interagerer direkte med fortolkeren. Man skriver sine skrivers sine udtryk ind, fortolkerne beregner dem og skriver svaret ud. En session med SML kunne derfor se således ud:

```
Standard ML of New Jersey v110.42 [FLINT v1.5], October 16, 2002
- val prim = [18,43,23,10,88];
val prim = [18,43,23,10,88] : int list
- fun sum [] = 0
  | sum (x::xs) = x + sum xs;
val sum = fn : int list -> int
- val res = sum prim;
val res = 182 : int
-
```

2.17.2. Eksempel på SML

Det følgende eksempel forsøger vise en række af de ting der er specielle for SML og andre funktionsorienterede sprog. SML udleder selv typer for funktioner, og de udledte typer er her indsat over funktionsdefinitionerne i kommentarer (* ... *).

Først defineres en funktion map:

```
(* Type: ('a -> 'b) -> 'a list -> 'b list *)
fun map f nil = nil
  | map f (x::xs) = f(x) :: map f xs;
```

Typen fortæller at funktionen map som argumenter tager en anden funktion der tager et argument af typen 'a og returnerer en værdi af typen 'b. Det andet argument til map er en liste hvor elementerne har typen 'a og så returnerer map en liste af elementer af typen 'b. Men hvad er 'a og 'b?

'a er en typevariabel. Hvis man kalder map med en funktion der tager et heltal som argument vil 'a blive bundet til typen int. Dette kaldes typepolymorfi. map arbejder bare på lister og er ligeglad med hvad funktionen f gør ved elementerne.

Funktionen er defineret med mønstergenkendelse. Det vil sige at der er listet en række tilfælde af definitionen op hvor SML så vil vælge den første der passer. nil og (x::xs) er mønstre som SML sammenligner med hvad funktionen bliver kaldt med. nil er den tomme liste og (x::xs) er en liste hvor det første element bliver bundet til variabelen x og resten af listen bliver bundet til variabelen xs.

:: er en infix-operator der laver listesammensætning. map tager altså en funktion og en liste og udføre funktionen på alle elementerne i listen. Da map tager en anden funktion som argument kaldes den en højereordensfunktion.

Følgende enkle funktion tager et heltal og ganger det med 2:

```
(* Type: int -> int *)
fun double n = 2 * n;
```

Ikke noget specielt her.

Nu vil vi så anvende ovenstående to funktioner til at lave en funktion der tager en liste af heltal og fordobler dem:

```
(* Type: int list -> int list *)
val doublelist = map double;
```

Bemærk at vi her bruger val og ikke fun. Det er fordi vi egentligt ikke laver en ny funktion men tildeler navnet doublelist en værdi (funktioner er nemlig lige så meget værdier som heltal er).

Det næste man bemærker er at map kaldes med kun et argument. Men map havde da to argumenter? Dette kaldes 'currying'. Vi kalder map med det første argument, det returnerer en funktion der tager resten af argumenterne.

Vi kunne også have undladt at definere funktionen double og bruge en anonym funktion:

```
(* Type: int list -> int list *)
val doublelist = map (fn x => 2 * x);
```

Den slags anonyme funktioner bruges ganske ofte i SML.

2.17.3. Iteration i SML

SML indeholder ikke de løkkestrukturer man normalt bruger. Skal man beregne noget iterativt bliver man derfor nødt til selv at lave nogle funktioner der klarer det for sig. Man kunne for eksempel bruge følgende funktion:

```
(* Type: ('a -> 'a) -> int -> 'a -> 'a *)
fun iterate f 0 x = x
  | iterate f i x = iterate f (i-1) (f x);
```

`iterate` tager en funktion `f` og udfører den et antal gange på værdien `x`.

2.18. Modula-2

...

2.19. Modula-3

...

2.20. Oberon

...

2.21. Objective-C

...

2.22. Objective Caml

OCaml er et temmelig specielt programmeringssprog, idet det tillader at man blander flere paradigmer end sædvanligt; objektorienteret, imperativt og funktionsorienteret. ...

2.22.1. Nogle løse noter om OCaml

Fra 1987.

Statisk typet, (men) vælger automatisk den mest generelle type (betydning) af et udtryk.

Funktioner er en slags data.

Heltal regnes "modulo" 2^{30} på 32-bit-systemer og modulo 2^{62} på 64-bit-systemer $(2^{30}-1)+1 = 2^{30}$.

Skelner mellem "strukturel" og "fysisk" lighed ("=" og "=="). Med "fysisk" lighed menes der at det rent faktisk er de samme data, der ligger det samme sted i hukommelsen, mens "strukturel" lighed blot er to forskellige variable der tilfældigvis har samme værdi. Det svarer til om man ser på "A=B" eller "A'Access=B'Access" i Ada.

Underforstået parallelisering i syntaksen:

```
let navn_1 = udtryk_1 and navn_2 = udtryk_2 and ... ;;
```

beregnes principielt parallelt.

Prioritet: "f a b" svarer til "(f a) b". Der er desværre mange alternative formuleringsformer. :-)

Bortset fra parametrene evalueres en funktion når den defineres.

Polymorfe funktioner...

Har en flerdimensionel "case"-struktur i form af "match ... with ...".

2.23. Octave

Matematikværktøj ...

2.24. Pascal

Pascal blev originalt udviklet som et undervisningssprog engang i 1960'erne og har siden dannet basis for mange andre sprog såsom Modula. Pascal kendes nok mest fra Borlands Turbo Pascal (der i øvrigt var en yderst mangelfuld implementation) hvor det tilbage i "DOS-tiden" var grundlag for mange projekter. I Pascal-verdenen er det stadig Borlands implementation, nu kaldet "Object Pascal" (også kendt som "Delphi" og "Kylix"), der primært benyttes.

Der findes i dag en stribe Pascal-oversættere dog vil jeg sige de mest interessante er Borlands Kylix og Free Pascal som begge understøtter "Object Pascal". Den generelle opfattelse har været at Pascal ikke er meget andet end et undervisningssprog uden nogen reel praktisk udnyttelse. Pascals styrke (og problem siger nogle) har været at det har et fungerende typesystem der gør at man ikke bare uden videre kan oversætte mellem heltal, pseudo-reelle tal og andre typer. Meget af dette er ændret i "Object Pascal".

Pascals syntaks er forholdsvis enkel at lære og minder en smule om "almindeligt" engelsk. Du har ikke den samme frihed til selv at finde på en programmeringsstil, som man for eksempel finder i sprog som C og C++. Det kan ses som en ulempe eller som en fordel, men det betyder under alle omstændigheder at det er sværere at skrive kode som andre Pascal-programmører ikke kan læse. Pascal er velegnet til de fleste opgaver, dog undtaget systemprogrammering, og med Kylix er det, sammen med Ada, klart et sprog en nybegynder bør overveje hvis der skal skrives procedurale programmer.

Selve Object Pascal er rimelig veldefineret og understøttes ganske glimrende af Free Pascal, der er dog nogen undtagelser når det kommer til oversætttermakroer. Men sådan er det vel med de fleste sprog. "Object Pascal" er dog ikke en standard, så det er Borland der styrer den med hård hånd.

Kylix er en såkaldt Rapid Application Development (RAD) Platform. I ms-windows-udgaven kendes den som Delphi. Det gør det nemt at flytte sine programmer mellem Linux- og MS-Windows-systemer til x86-arkitekturen. Free Pascal er en mere traditionel pakke og inkluderer ikke et grafisk udviklingsmiljø, men der findes bindinger til GTK så det er muligt at udvikle grafiske applikationer.

Pascal er standardiseret af ISO m.fl. som ISO-####. GNU Pascal⁵ kan oversætte både "Standard Pascal", "Extended Pascal" og Borland Pascal 7.0 med visse udvidelser fra Delphi og Kylix.

2.25. Perl

Perl er et spændende og meget anvendeligt sprog. C og C++ er i mange tekniske sammenhænge en de facto standard, men disse programmeringssprog er ikke ret velegnede til tekst-manipulationer. Der skal allokeres den rette mængde hukommelse, og det er ofte svært at kunne beregne hvor meget man skal bruge. Perl er skabt til at behandle tekst.

2.26. Postscript

Postscript er et sprog beregnet til at beskrive udseendet af en side uafhængigt af det egentlige medie. Det var oprindeligt kun tænkt til udskrifter, men senere er det også blevet brugt til den grafiske brugergrænseflade på NeXT-systemet.

Postscript er et stakorieret sprog hvilket principielt betyder at alle data gemmes på en *stak* hvor man kun kan få fat i dem i den modsatte rækkefølge af den de er lagt på stakken i.

Den typiske måde at "skrive" Postscript-programmer på er at skrive ud fra andre programmer. Det program der skal skrive noget ud, laver så et Postscript-program med de sider der skal skrives ud, og sender det til en printer med en indbygget Postscript-fortolker.

2.27. Prolog

Prolog er et deklarativt sprog med understøttelse for metaprogrammering. Det er hovedsageligt brugt i kunstig intelligens, specielt inden for naturlige sprog og logik.

Programmeringen foregår via en formel logisk beskrivelse af hvad man ønsker programmet skal opnå.

2.28. Python

Python er et objektorienteret sprog der byder på flere spændende koncepter. En genial idé som man enten hader eller elsker, er at koden struktureres ved hjælp af indrykninger frem for tuborg-klammer som i C (og C++ og Perl).

Python er i lighed med Perl et fortolket sprog, og hastigheden er bestemt ikke et af Pythons fortrin. Til gengæld er det forholdsvis nemt at lave udvidelser i C eller C++ der så kan bruges til de dele af et program der udgør en hastighedsmæssig flaskehals. Python kan også indlejres i andre programmer hvorved du kan stille et fuldt programmeringssprog til rådighed for brugerne af programmet.

Til at konstruere grafiske brugergrænseflader bruges i Python normalt et modul kaldet Tkinter der er en grænseflade til det ovenfor nævnte Tk. Medmindre du bruger specielle Linux-faciliteter, kan Python-programmer med eller uden Tkinter også køres under andre styresystemer – f.eks. Windows NT.

Læs mere om Python på <http://www.python.org/>.

2.29. Ruby

Ruby tilhører, sammen med Perl og Python, gruppen af moderne fortolkede sprog. Fra begyndelsen er Ruby designet som et objektorienteret sprog og giver derfor en mere elegant oplevelse end Perls objekter, men derudover minder Ruby langt hen af vejen om en pænere version af Perl.

Ruby er hovedsagligt udviklet i Japan og meget af dokumentationen er skrevet på japansk. Denne mangel på engelsk dokumentation har været en hindring for Rubys internationale udbredelse. Dette er der ved at blive rådet bod på, det kan derfor kun anbefales at se nærmere på Ruby.

2.29.1. Hej verden!

Vi begynder med det sædvanlige eksempel. Et lille program der ikke gør andet end at skrive "Hello World!" ud på skærmen. Det giver ikke meget indtryk af selve sproget Ruby, men viser hvordan man bruger det. Gem følgende tekst som `hello.rb`

```
#!/usr/bin/ruby

print "Hello World!\n"
```

Programmet kan nu køres ved enten at kalde **ruby**:

```
linus% ruby hello.rb
Hello World!
linus%
```

eller ved at gøre scriptet udførbart og køre det direkte:

```
linus% chmod a+x hello.rb
linus% ./hello.rb
Hello World!
linus%
```

2.29.2. Et større eksempel

For at vise et lidt større eksempel som giver en lidt større indsigt i hvordan Ruby ser ud er her et simpelt program til afsendelse af email. Programmet er ret simpelt. Det beder om en afsender, en modtager og et emne. Derefter kan man skrive sin mail.

I sin fulde længde ser programmet således ud:

Kapitel 2. Vælg et sprog

```
#!/usr/bin/ruby

require 'net/smtp'

mail = ""

begin
  print "From: "
  from = readline
  mail << "From: " << from
  from.chomp!
  print "To: "
  to = readline
  mail << "To: " << to
  to.chomp!
  print "Subject: "
  mail << "Subject: " << readline
rescue EOFError
  print "\nAborted\n"
  exit 1
end

mail << "\n"

print "Write mail end with . or EOF\n"
begin
  line = readline
  break if (line == ".\n")
  mail << line
rescue EOFError
  break
end while true

smtp = Net::SMTP::new("localhost")
smtp.start
smtp.sendmail(mail, from, to)
smtp.finish
```

Den første linje

```
require 'net/smtp'
```

Fortæller at vi har brug for biblioteket der implementerer SMTP så vi kan sende mail.

Selvom Ruby er utyppet, så skal vi have et objekt for at kunne lave metodekald. Derfor bliver vi nødt til at initialiserer variabelen mail med et streng-objekt:

```
mail = String::new("")
```

Det næste vi skal gøre er at spørge brugeren om afsenderen, modtageren og emnet for mailen. Hvis brugeren trykker Ctrl-D (End Of File) skal programmet stoppe:

```
begin
  print "From: "
  from = readline
  mail << "From: " << from
  from.chomp
  print "To: "
  to = readline
  mail << "To: " << to
  to.chomp
  print "Subject: "
  mail << "Subject: " << readline
rescue EOFError
  print "\nAborted\n"
  exit 1
end

mail << "\n"
```

readline læser en linje fra standard-ind. Hvis brugeren sender en EOF (trykker Ctrl-D) rejses end EOFError-undtagelse, der derfor fanges tilsidst i blokken (fra **rescue EOFError**).

<< er en metode på et streng-objekt. Den tilføjer en streng til objektet. **chomp**-metoden fjerner ny linje-tegnet sidste fra en streng.

Så skal vi læse selve mailen ind. Dette gøres i en uendelig **while**-løkke:

```
print "Write mail end with . or EOF\n"
begin
  line = readline
  break if (line == ".\n")
  mail << line
rescue EOFError
  break
end while true
```

Hvis brugeren sender en EOF hopper vi ud af løkken i rescue-blokken og hvis brugeren skriver en linje kun indeholdende et punktum hopper vi også ud af løkken.

Tilbage er bare at sende mailen:

```
smtp = Net::SMTP::new("localhost")
smtp.start
smtp.sendmail(mail, from, to)
smtp.finish
```

Vi laver et smtp-objekt og sender mailen med **sendmail**-metoden

2.29.3. One-liners i Ruby

ruby understøtter en række argumenter der gør det lettere at skrive one-liners, altså små programmer der kan skrives på en linje. Disse argumenter kendes også fra perl

2.30. Scheme

Scheme er en dialekt af Lisp og i øvrigt et af de mindste sprog der findes. Et af designmålene for sproget var at have en enkel semantik. Det understøtter flere paradigmer, herunder det imperative og funktionsorienterede. Scheme har dynamisk type-tjek (modsat for eksempel Ada og ML der har statisk type-tjek).

Det har god understøttelse for metaprogrammering, samt mulighed for undnyttelse af "continuations" hvilket blandt andet kan benyttes til undtagelseshåndtering.

Der er mange implementationer (både oversættere og fortolkere) til mange platforme, heriblandt JVM, MacOS, PalmOS, Unix, WinCE og Windows. Der findes også en udgave specifikt rettet mod undervisning inklusive udviklingsmiljø.

Scheme bliver også ofte brugt som grundlag for domænespecifikke sprog, for eksempel script-fu (i tegneprogrammet Gimp), DSSSL (benyttet til manipulation af SGML-dokumenter) og Guile ("GNU's extension language", bl.a. benyttet i kontorprogrammer).

2.31. Smalltalk

Smalltalk er et typefrit sprog. Det betyder ikke at der ikke er nogle datatyper eller at objekter ikke har nogen type. Det betyder blot at de navne (variable, parametre m.m.) man refererer til objekter med, ikke er bundet til en bestemt type. De betegner altså objekter hvis typer ikke er kendt. Det eneste man kan gøre ved et objekt, er at "sende det en besked" (kalde en metode der opererer på objekter) og grundlæggende er den eneste fejl der kan opstå i den sammenhæng at objektet "ikke forstår beskeden" (metoden ikke findes), og der dermed opstår en fejl under kørslen af programmet.

Smalltalk er et egentligt objektorienteret sprog hvilket gør det meget anvendeligt til implementation af rent objektorienterede systemer. Til gengæld gør det at Smalltalk udelukkende arbejder med referencer til objekter og ikke giver adgang til "direkte" håndtering af objekter, at sproget ikke er velegnet til udvikling af lavniveausystemer hvor det for eksempel typisk vil være nødvendigt med langt mere direkte styring af hvad der sker i hukommelsen.

2.32. SQL

PostgreSQL er en database og som sådan ikke noget programmeringssprog. Som i mange andre databaser kan man lave funktioner i PostgreSQL der køres direkte i databasen. Et af de programmeringssprog man kan bruge i PostgreSQL, er PLpgSQL, som er en slags standard SQL programmeringssprog. Funktioner kan også skrives i Perl og C.

Det er specielt hvis man bruger flere sprog til at bearbejde data i en database, at det er en fordel at bruge funktioner. Det kunne være en kompliceret opdatering af en række tabeller som gør det vanskeligt både at holde kildetekst i Perl og C opdateret samtidigt.

Her er et lille "Hello, world!" eksempel i denne bog, men der er mere om PostgreSQL i Webbogen <http://cvs.linuxbog.dk/web/bog/db-pgsql.html>. Se også eksemplerne i <http://cvs.linuxbog.dk/program/eksempler/>. De færreste har brug for at fremstille Mandelbrot fraktaler med PostgreSQL, men det er også muligt <http://www.sslug.dk/~chlor/mandel/>.

2.32.1. Hello, world!

```
create function hello() returns text as '  
begin  
    return "Hello, world!";  
end;  
' language 'PLpgSQL';
```

Programmet kan afprøves med kommandoen:

```
SELECT hello();
```

2.33. Tcl/Tk

Hvis du gerne vil programmere grafik under Linux, er Tcl/Tk et godt sprog. Det er et forholdsvis nemt sprog at lære. Tcl er et makrosprog som har fået en søster Tk der er en grafisk del med samme nemme syntaks. Det er et meget velegnet sprog til hurtigt at få en grafisk brugergrænseflade til et (tekstbaseret) program.

2.34. TeX

TeX er mest kendt som tekstformatteringsværktøj, og mest i skikkelse af LaTeX, men teoretisk set er TeX et imperativt programmeringssprog.

Slutbemærkning:

1. <http://www.adaic.com/standards/ada95.html>
2. <http://cvs.linuxbog.dk/program/eksempler/bottles.ada>
3. <http://www.ecma.ch/>
4. <http://www.dina.kvl.dk/~sestoft/sml/sml-std-basis.html>
5. <http://www.gnu-pascal.de/>
6. <http://www.python.org/>
7. <http://cvs.linuxbog.dk/web/bog/db-pgsql.html>
8. <http://cvs.linuxbog.dk/program/eksempler/>
9. <http://www.sslug.dk/~chlor/mandel/>

Kapitel 3. Imperativ programmering

Imperativ programmering – eller "programmering i bydemåde" – er nok den mest anvendte form for programmering. Helt kort beskrevet betyder det at man fortæller computeren hvad den skal gøre – I modsætning til deklarativ programmering hvor man blot beskriver problemet.

Et computerprogram er et model af en del af den virkelige verden som du kan bruge til at efterligne en del af den virkelige verdens opførsel. Jo mere detaljeret modellen er, des mere præcist kan du forvente efterligningen er. Klimamodeller tager for eksempel så mange data som muligt med for at forudse klimaudviklingen længere ud i fremtiden og mere præcist.

Inden i computeren bliver alle disse informationer (også selve programmerne) repræsenteret som nuller og et-taller, men programmerne behandler de forskellige informationer forskelligt alt efter hvilke typer oplysninger de repræsenterer. Et computerprogram er ikke bare en simpel opskrift; det behandler informationer forskelligt, alt efter hvilke virkelige objekter de repræsenterer, og hvilke aspekter af disse objekter vi er interesserede i.

De eksempler der præsenteres i dette kapitel, kan også findes oversat til forskellige programmeringssprog i bogens eksempelsamling der kan hentes fra <http://cvs.linuxbog.dk/program/eksempler/>.

3.1. Variable og konstanter

Når du programmerer, vil du ofte have brug for at lagre mellemresultater undervejs i programmet. Det kan for eksempel være navnet på en person eller det tidspunkt der sidst blev trykket på en tast. Hvis disse mellemresultater skal kunne opdateres undervejs mens programmet kører, vil man opbevare dem i et objekt hvis indhold kan variere – en variabel.

Resultater der blot findes en gang for alle, og dernæst blot skal bruges, opbevares i et objekt hvis indhold ikke kan ændres – en konstant.

Bemærk at hvis du programmerer i et rent funktionsorienteret sprog, vil der ikke som sådan være nogle variable, da alt er resultater fra eller argumenter til funktioner.

3.2. Tildelinger og typer

3.3. Funktioner og procedurer

Hvis man forsøger at skrive hele sit program ud i et, bliver det meget let uoverskueligt. Og hvis der oven i købet er nogle ting man skal gøre flere steder i samme program – eller måske endda i flere forskellige programmer – er det også lidt spild af tid at skrive dem igen og igen. Et middel til at undgå disse problemer er underroutriner. Underroutriner er programstumper der udfører en eller anden opgave og kan bruges (kaldes) fra flere forskellige steder i programmet. Når man kalder en underroutine, giver man den typisk nogle data (argumenter) som den skal bruge til at løse sin opgave, med i kaldet. Underroutriner findes i to varianter, funktioner og procedurer. Funktioner fungerer ganske som man lærer det i matematik. De regner et resultat ud ud fra de argumenter man giver dem. Procedurer fungerer lidt anderledes idet de ikke direkte giver et resultat. I stedet for kan de ændre på værdien af deres argumenter.

Principielt er der ingen forskel på funktioner og procedurer, men hvis man for eksempel med det samme skal regne videre på noget en underroutine finder ud af, er det praktisk at det er en funktion, så man bare kan sætte kaldet af den ind der hvor man skal bruge resultatet. Hvis det var en procedur, ville man først være nødt til at kalde den med en variabel som et af argumenterne og dernæst sætte variabelen ind på det sted hvor man skal bruge resultatet. Tilsvarende findes der også eksempler hvor det er mest hensigtsmæssigt at bruge en procedure frem for en funktion.

3.4. Betingede strukturer

Hvis et program skal kunne handle forskelligt alt efter hvilke data det får ind, er det nødvendigt med betingede strukturer hvor det gør forskellige ting alt efter hvilken værdi dataene har. Et simpelt eksempel kan være at undersøge hvor lang tids barselsorlov en person har ret til ud fra vedkommendes CPR-nummer.

Eksempel 3-1. Barselsorlovs længde

Hvis personens CPR-nummer har et lige løbenummer, så har hun ret til seks måneders barselsorlov, ellers har han kun ret til ugers barselsorlov.

Eksemplet findes også skrevet i: Ada²

Dette er et eksempel på en simpel "*hvis udsagn så gør noget ellers gør noget andet*"-konstruktion som man vil finde i en eller anden form i ethvert imperativt programmeringssprog. I princippet er denne konstruktion sammen med muligheden for at hoppe til vilkårlige steder i programmet tilstrækkeligt til at man kan skrive et vilkårligt computerprogram. I praksis er det dog langt fra hensigtsmæssigt kun at arbejde med så primitive redskaber.

En del sprog indeholder – i en eller anden variant – en generalisering af ovenstående sand/falsk-baserede konstruktion til vilkårlige *diskrete* datatyper. Det kan for eksempel bruges til at undersøge hvor længe det lokale supermarked har åbent afhængigt af hvilken dag i ugen det er.

Eksempel 3-2. Supermarkedsåbningstider

I tilfælde af at ugedagen er:

- mandag, tirsdag, onsdag eller fredag *så* er der åbent fra 9 til 18.
- torsdag *så* er der åbent fra 9 til 20.
- lørdag *så* er der åbent fra 10 til 17.
- søndag *så* er der lukket hele dagen.

Eksemplet findes også skrevet i: Ada³

3.5. Løkker

gentagelser ...

3.6. Dynamiske kald

3.7. Logisk parallelisering

Slutbemærkning:

1. <http://cvs.linuxbog.dk/program/eksempler/>
2. <http://cvs.linuxbog.dk/program/eksempler/barselsorlovslængde.ada>
3. <http://cvs.linuxbog.dk/program/eksempler/supermarkedsåbningstider.ada>

Kapitel 4. Deklarativ programmering

Ideen med deklarativ programmering er at man nøjes med at beskrive problemet og så overlader det til computeren at finde ud af hvordan det skal løses.

Kapitel 5. Programmørens værktøjskasse

En programmørs værktøjskasse bør indeholde:

- Et versionsstyringssystem. For eksempel CVS (Afsnit 5.1). I virkeligheden bør man bruge et versionsstyringssystem til alle projekter der varer længere end et par dage, eller har flere deltagende forfattere.
- Et tekstbehandlingsprogram. For eksempel vi (Afsnit 5.15), Emacs (Afsnit 5.14), Grasp (Afsnit 5.16, mest til Ada, C og C++), Anjuta (Afsnit 5.12, bl.a. til C, C++, Perl, PHP, Python og SQL) eller KDevelop (Afsnit 5.13, kun til C++).
- Et udskriftssystem. Det kan for eksempel være **a2ps** (Afsnit 5.17) eller det der er indbygget i Grasp (Afsnit 5.16).
- Et fejlsøgningsværktøj (GVD Afsnit 5.6 eller gdb Afsnit 5.5)

Derudover kan det være praktisk med:

- En brugergrænsefladebygger (GLADE Afsnit 5.7).
- En kontrolstrukturplotter. Det kan for eksempel være Grasp (Afsnit 5.16) der kan bruges til at få en grafisk oversigt over kontrolstrukturene i Ada-, C- og C++-programmer.
- Et oversættelseskontrolsystem. Det kan for eksempel være GNU Make (Afsnit 5.10) der kan bruges til at holde styr på hvilke filer det er på tide at opdatere (automatisk).

Og hvis du programmerer i C, C++ eller C#, vil du temmelig sikkert også få brug for en læksøger. Det kan for eksempel være **efence** (Afsnit 5.9).

Og det ideelle ville selvfølgelig være hvis man kunne få hele *sin* værktøjskasse integreret i ét udviklingsmiljø. Emacs (Afsnit 5.14) er nok det mest populære integrerede udviklingsmiljø til Linux. I standardopsætningen snakker Emacs fint med CVS og en utroligt bred samling oversættere, samtidig med at det giver syntaksfremhævning og sprogspecifikke genvejstaster til langt de fleste programmeringssprog. Det at Emacs grundlæggende er et tekstbaseret program, generer vist en del mennesker, men på den anden side kan Emacs bruges til så meget at man hurtigt kommer til at undre sig over hvordan man kan klare sig uden. Hvis du har valgt at programmere i C++, er KDevelop (Afsnit 5.13) et andet muligt valg af integreret udviklingsmiljø. KDevelop er et decideret grafisk udviklingsmiljø og ser en del smartere ud end Emacs. Desværre er det begrænset til programmering i C++. Anjuta (Afsnit 5.12) er et nyt integreret grafisk udviklingsmiljø. Anjuta kan håndtere adskillige programmeringssprog, men dog ikke nær så mange som Emacs (det er vist også en svær opgave).

5.1. Concurrent versioning system (CVS)

Concurrent versioning system (CVS) er et versionsstyringssystem. Dette er godt nok en programmeringsbog, men i virkeligheden bør man bruge et versionsstyringssystem til alle projekter der varer længere end et par dage eller har flere end en enkelt deltagende forfattere. Versionsstyringssystemets opgave er at holde styr på de forskellige udgaver en fil har eksisteret i, hvilke ændringer der er foretaget, hvorfor, og af hvem. Der holdes styr på dette i et centralt fælles *CVS-arkiv*, hvor alle ændringer registreres. En af de ting der gør CVS til noget særligt blandt versionsstyringssystemer, er at det tillader at flere forfattere samtidig arbejder på de samme filer. Så længe der ikke er flere der retter i de samme linjer, vil CVS flette de forskellige ændringer sammen efterhånden som de bliver lagt ind i det fælles arkiv.

Versionsstyringssystemer som CVS er en nødvendighed for udviklingsprojekter der – som for eksempel Linux – foregår spredt rundt omkring i verden. Antag at et firma eller universitet har behov for at kunne arbejde med udvikling af programmer på kryds og tværs af geografi hvor al kommunikation skal ske via internettet. To spørgsmål melder sig hurtigt til systemadministratoren: Hvad med sikkerheden, og hvordan kan man styre softwareudviklingsprocessen, når der er mange mennesker involveret, på en sådan måde at man kan lave udgivelser som altid kan genskabes (til serviceformål), mens man samtidigt videreudvikler, og dermed har behov for at kunne bakke tilbage gennem opdateringerne i tilfælde af fejl.

De fleste danske og udenlandske softwarefirmaer har foretaget deres udvikling baseret på udvikling i lokale netværk. Med den fortsatte ekspansion af udviklingen af internettet og med den fortsatte udvikling hvor begreber som "hjemmearbejdsplads" og "distancearbejde" dukker op oftere og oftere, må det forventes at softwareudviklingsfirmaer allerede er i gang med og i endnu højere grad vil foretage udviklingen af software baseret på utroligt mange computere bundet sammen af internettet. Dette stiller store krav til de versionsstyringssystemer der benyttes.

Styring af softwareprojekter er svært! Specielt svært bliver det når mange arbejder på den samme kildetekst, og hvor samtidigt kommunikationen mellem projektdeltagerne er ringe eller besværlig. De krav der må stilles til versionsstyringssystemerne i sådanne projekter, er at alle ændringer let skal kunne tilgå andre på projektet, alle ændringer skal kunne hives ud af kildeteksten igen, og man skal kunne se hvem der har lavet hvad på hvilket tidspunkt.

Når man en gang har prøvet at være to eller flere som fysisk retter i samme kildetekst uden at man bruger et versionsstyringssystem, er det indlysende at denne form for samarbejde er uproduktiv. På sådanne præmisser er megen kommunikation og dermed megen uproduktiv tid nødvendig hvis det skal være muligt at opnå et tilfredsstillende resultat. En mulighed for at øge produktiviteten er at gøre brug af et versionsstyringssystem.

CVS er et multi-platform versionsstyringssystem som har alt det man kan ønske sig af et sådant system. Det er frit tilgængeligt under GNU-licensen (GPL). Det distribueres i køreklar form til de fleste unixsystemer (se <http://prep.ai.mit.edu/pub/gnu> og afspejlinger), såvel som til MS Windows (se <http://www.cygus.com/misc/gnu-win32>). Du kan læse mere om CVS på <http://www.linux.ie/articles/tutorials/managingaccesswithcvs.php>, <http://www.cvshome.org/docs/blandy.html>, <http://www.ibiblio.org/mdw/REF/ CVS-BestPractices/html/index.html> eller <http://cvsbook.red-bean.com/cvsbook.html>.

CVS er et forholdsvis nyt versionsstyringssystem, og dets håndtering af konflikter mellem forskellige udvikleres arbejde regnes for at være unik.

5.1.1. CVS' funktionalitet

CVS bruges til at gemme den historie som ens kildetekster (af den ene eller den anden slags) gennemløber. Grundprincippet i CVS er at alle brugere har deres egen lokale kopi af kildeteksten som kan modificeres uafhængigt af andre brugere. Dette kan ske med brugere på samme maskine eller på maskiner placeret vilkårligt langt fra hinanden. Når ens personlige kopi af kildeteksten er stabil og fungerende, kan alle de opdaterede filer lægges ind i CVS-arkivet igen, eventuelt først efter at være blevet flettet sammen med andre udvikleres ændringer af de samme filer. På denne måde kan man uden at begrænse andre i deres arbejde udføre meget arbejde (lang tids arbejde og/eller mange ændringer) på sin lokale kopi af kildeteksten, uden alvorlig risiko for at der opstår problemer når rettelserne lægges ind i det fælles arkiv.

CVS understøtter at arkivet (samlingen med versionsstyringssystemets information, inklusive kildeteksterne) kan gøres tilgængelig på nettet uden at man behøver at montere diske fra andre maskiner. Almindeligvis anvendes `ssh` til at håndtere sikker udveksling af data mellem programmørernes maskiner og det centrale arkiv.

CVS udmærker sig ved at have et sæt af kommandoer til at pakke distributionen ud, at opdatere ens egen distribution med andres ændringer, at lægge ens egne ændringer i arkivet, at lave sideløbende grene af udviklingen af produktet, at samle sideløbende grene af udviklingen, at hente information om historien af ændringer osv.

Det er værd at bemærke at tekstbehandlingsprogrammet Emacs automatisk vil bemærke det hvis du redigerer filer der ligger i et CVS-arkiv. Det vil så vise en ekstra menu der giver adgang langt de fleste af de daglige CVS-funktioner. Fra Emacs kan man bl.a. lægge filer ind i CVS, hente senere udgaver af de enkelte filer, og få sammenlignet forskellige udgaver af samme fil.

5.1.2. Daglig brug af CVS

Hvis vi går ud fra at andre har taget sig af at lægge filerne fra et projekt du skal til at arbejde på i et CVS-arkiv (se Afsnit 5.1.3, hvis du selv skal gøre det), er det forholdsvis enkelt at komme i gang. Du skal blot kende CVS-arkivets adresse og projektets navn i arkivet. De første tre af de følgende eksempler viser hvordan du går i gang med at arbejde med filer der ligger i et CVS-arkiv, afhængigt af hvilken form for adgang du har til det (Eksempel 5-1, Eksempel 5-2 og Eksempel 5-3). Dernæst følger eksempler der viser hvordan du opdaterer din personlige kopi af arkivet (Eksempel 5-4), hvordan du lægger rettede filer tilbage i det fælles arkiv (Eksempel 5-5), hvordan du tilføjer filer (Eksempel 5-6), hvordan du sletter filer (Eksempel 5-8), og hvordan du lægger en rettet fil tilbage i det fælles arkiv direkte inde fra Emacs (Eksempel 5-9).

Eksempel 5-1. Gå i gang med at bruge et lokalt CVS-arkiv

```
hven% cvs \  
-d /usr/local/CVSR00T checkout webspell
```

- "-d /usr/local/CVSR00T" betyder at det CVS-arkiv projektet skal hentes fra, ligger i kataloget /usr/local/CVSR00T på den maskine du arbejder på.
- checkout betyder at du vil have oprettet en kopi af et projekt der ligger i det førnævnte CVS-arkiv.
- Og endelig er "webspell" navnet på det projekt vi vil have en kopi af.

Eksempel 5-2. Anonym brug af et CVS-arkiv på nettet

```
hven% echo "update -dP" >> ~/.cvsrc
hven% cvs \
-d :pserver:anonymous@anoncvs.kde.org:/home/kde checkout kde-i18n
```

Den første kommando sikrer blot at `cvs` kan forstå KDE-projektets CVS-tjener. Betragt det i denne sammenhæng blot som hvid magi.

- `-d :pserver:anonymous@anoncvs.kde.org:/home/kde` betyder at det CVS-arkiv projektet skal hentes fra, ligger i kataloget `/home/kde` på maskinen "anoncvs.kde.org", og at du vil hente det som en anonym bruger.
- `checkout` betyder at du vil have oprettet en kopi af et projekt der ligger i det førnævnte CVS-arkiv.
- Og endelig er "kde-i18n" navnet på det projekt vi vil have en kopi af. I dette konkrete tilfælde er det oversættelser af KDE (menutekster og brugsanvisninger) til forskellige sprog.

Eksempel 5-3. Brug af et CVS-arkiv på nettet over SSH

```
hven% export CVS_RSH=ssh
```

Denne kommando sætter en variabel der fortæller `cvs` at kommunikation over nettet skal kunne ske med SSH-protokollen (der er krypteret og dermed sikrer mod aflytning af adgangskoder og data). Bemærk at variabelen også skal være sat når du senere vil opdatere din kopi af projektet eller vil lægge en rettet fil ind i CVS-arkivet. Det kan du hurtigt sikre med kommandoen `echo "export CVS_RSH=ssh" >> ~/.zshrc`.

```
hven% cvs \
-d tyge@cvs.sslug.dk:/usr/local/CVSR00T checkout linuxbog
```

- `-d tyge@cvs.sslug.dk:/usr/local/CVSR00T` betyder at det CVS-arkiv projektet skal hentes fra, ligger i kataloget `/usr/local/CVSR00T` på maskinen "cvs.sslug.dk", og at du vil hente det som brugeren "tyge".
- `checkout` betyder at du vil have oprettet en kopi af et projekt der ligger i det førnævnte CVS-arkiv.
- Og endelig er "linuxbog" navnet på det projekt vi vil have en kopi af. I dette konkrete tilfælde er det hele bogserien "Linux – Friheden til at vælge".

Eksempel 5-4. Opdatér din personlige kopi af et CVS-arkiv

```
hven% cd webspell/
hven% cvs update -d
```

- Du skal stå i kataloget med din lokale kopi af CVS-arkivet for at kunne opdatere filerne. Ovenfor bliver det gjort med kommandoen `cd webspell`.
- Argumentet "update" betyder at du vil have *opdateret* den personlige kopi af et CVS-arkiv der allerede ligger i det katalog du står i. Argumentet "-d" gør at også nye underkataloger bliver opdateret. Tilføj yderligere "-C" hvis eventuelle egne rettelser skal overskrives af CVS-arkivets version.

Eksempel 5-5. Læg en rettet fil tilbage i det fælles CVS-arkiv

Vi antager nu at vi som efter forrige eksempel står i kataloget med den personlige kopi af CVS-arkivet, og at vi har rettet en HTML-syntaksfejl i filen `tjek.php`.

```
hven% cvs commit \
-m 'validator.w3.org brokkede sig' tjek.php
```

- `"commit"` betyder at du vil kopiere din rettede udgave af en fil ind i det fælles CVS-arkiv.
- `-m 'validator.w3.org brokkede sig'` betyder at rettelserne i CVS-arkivets logbog skal have kommentaren "validator.w3.org brokkede sig".
- Og endelig er "tjek.php" navnet på den fil du har rettet og vil kopiere ind i det fælles arkiv.

Eksempel 5-6. Tilføj en ny fil til et CVS-arkiv

Hvis vi nu har skrevet en ny fil ved navn "tak.html", og den ligger sammen med vores personlige kopi af projektet, kan vi tilføje den med kommandoerne:

```
hven% cvs add tak.html
hven% cvs commit -m 'lidt almindelig høflighed' tak.html
```

- "add" betyder at du vil føje din rettede udgave af en fil til det fælles CVS-arkiv.
- Og endelig er "tak.html" navnet på den fil du har skrevet og vil føje til det fælles arkiv.
- Efter at "tak.html" er blevet føjet til arkivet, bruges "cvs commit" til at aktivere tilføjjelsen.

Eksempel 5-7. Fjern et helt katalog i et CVS-arkiv

CVS er ikke ret god til at slette kataloger. Måden at håndtere sletning af hele underkataloger er at slette alle filer i kataloget og derefter køre **cvs update -d -P**. Her vil -d gøre at kataloger opdateres med, og -P er at fjerne ("prune") tomme kataloger.

I nedenstående eksempel fjerner vi mit_katalog som er et underkatalog af projekt.

```
hven% cd mit_katalog
hven% ls
CVS      Makefile    prg.c      prg.h
hven% rm prg.c prg.h Makefile
hven% cvs remove prg.c prg.h Makefile
hven% cvs commit -m "Farvel til gamle filer"
hven% cd ..
hven% cvs update -d -P
```

Eksempel 5-8. Fjern en fil fra et CVS-arkiv

Vi har nu eksperimenteret lidt med en fil "tjek-eksp.php" og fundet ud af at der alligevel ikke er brug for den i projektet, så vi vil fjerne den fra den synlige del af CVS-arkivet (arkivet husker alle ændringer, så du vil stadig kunne gå tilbage til en udgave hvor filen fandtes). Det klares med kommandoerne:

```
hven% rm -f tjek-eksp.php
hven% cvs remove tjek-eksp.php
hven% cvs commit -m 'det gik ikke' tjek-eksp.php
```

- CVS vil ikke fjerne en fil fra arkivet hvis du stadig har en kopi af den i din personlige kopi af CVS-arkivet, så vi starter med kommandoen **rm -f tjek-eksp.php** der fjerner filen fra kataloget.
- "remove" betyder at du vil fjerne filen fra det fælles CVS-arkiv.
- Og endelig er "tjek-eksp.php" navnet på den fil der skal fjernes fra det fælles arkiv.
- Efter at "tjek-eksp.php" er blevet slettet fra arkivet, bruges "cvs commit" til at aktivere sletningen.

Eksempel 5-9. Læg en rettet fil tilbage i det fælles arkiv direkte inde fra Emacs

Hvis du har rettet i en fil fra et CVS-arkiv i Emacs, kan du lægge den rettede fil tilbage i arkivet direkte inde fra Emacs. Det kan for eksempel være praktisk hvis du løbende vil lave sikkerhedskopier af en rapport du er i gang med at skrive, så du uden problemer kan gå tilbage til tidligere udgaver af filen. For at lægge filen tilbage i arkivet taster du først Ctrl-X V V. Hvis filen ikke allerede er gemt, får du nu mulighed for det. Dernæst kommer der et vindue i Emacs hvor du kan indtaste en bemærkning om de seneste ændringer. Når du har indtastet ændringerne, taster du Ctrl-C Ctrl-C, og så er filen lagt tilbage i CVS-arkivet.

Hvis forbindelsen til CVS-arkivet går over en SSH-forbindelse, vil du blive bedt om løsen eller adgangskode både før og efter du indtaster bemærkningerne til ændringerne.

Hvis det er en ny fil du vil oprette i CVS-arkivet, er det først anden gang du taster Ctrl-X V V at filen rent faktisk bliver lagt ind i arkivet. I første omgang bliver filen blot registreret (svarer til **cvs add**).

5.1.2.1. Konfliktløsning ved "dobbel-opdatering"

Hvis der er konflikter mellem en fil i din personlige kopi af CVS-arkivet og den samme fil i det fælles CVS-arkiv fordi en anden bruger i mellemtiden også har opdateret filen, så vil du skulle flette de to opdateringer sammen før du kan lægge dine rettelser ind i det fælles arkiv. Det gøres ved at du først "opdaterer" din lokale kopi af filen (**cvs update <filnavn>**). CVS vil så markere forskellene mellem de to udgaver, så du i dit tekstbehandlingsprogram kan vælge hvilke udgaver af rettelserne du foretrækker. Når du har valgt mellem rettelserne og i øvrigt tjekket at filen er i orden, kan du lægge den sammenflettede opdatering ind i det fælles CVS-arkiv på sædvanlig vis (**cvs commit <filnavn>**).

5.1.2.2. Gamle udgaver af filer i CVS-arkivet

Hvis en fil er blevet rettet i en helt forkert retning, er det muligt at hente en gammel udgave af filen ud af det fælles CVS-arkiv og arbejde videre ud fra den. Det gøres med kommandoen **cvs update -r <udgavenummer> <filnavn>**.

5.1.2.3. Sætte huskemærker (tags) i CVS

Det kan være meget praktisk at kunne sætte et mærke på en fil for at man til hver en tid kan gå tilbage til netop den version af filen. For at sætte et mærke (engelsk "tag") kører man **cvs tag MÆRKE fil_liste** hvor MÆRKE er et navn (desværre må man ikke anvende mellemrum) a la `efter_indfletning_af_SSLUG_kode` eller `foer_Peters_sommerferie`.

Hvis man har brug for at gå tilbage til et givet mærke køres **cvs update -r MÆRKE**. Samme teknik anvendes også til at håndtere forskellige grene af koden.

5.1.2.4. Filers status

Enhver kan og bør principielt tjekke status for de enkelte filer før opdateringer bliver lagt ind i det fælles arkiv. Det gøres med kommandoen **cvs status <filnavn>**. CVS vil så vise udgavenummeret for den fil du har liggende, såvel som udgavenummeret for den fil der ligger i det fælles arkiv.

Anvend tilsvarende **cvs status -v <filnavn>** til at få endnu mere information om filen inklusive forgreninger.

Tip: En meget effektiv måde at få styr på status for alle filer er at anvende **cvsstat** eller **cvschk** det kan findes på <http://cvs.sslug.dk/cvs2html/>.

5.1.2.5. Logbogen

Hver gang du lægger ændringer ind i det fælles CVS-arkiv, kan du skrive en besked i logbogen om *hvorfor* du har lavet de ændringer. I Eksempel 5-5 bruger vi kommandolinje-flaget "-m" til at angive teksten til logbogen. Hvis du ikke bruger "-m"-flaget, vil CVS i stedet starte det tekstbehandlingsprogram systemvariablen "EDITOR" peger på, så du kan skrive teksten der.

Hvis du på et tidspunkt vil se hvad der for eksempel står i logbogen om filen `tjek.php`, bruger du kommandoen:

```
hven% cvs log tjek.php
```

Den vil give dig en liste med de forskellige udgaver af filen, hvem der har lavet ændringerne, og hvad der står om dem i logbogen. Selve ændringerne bliver dog ikke vist.

5.1.2.6. Forskelle mellem forskellige udgaver af en fil

Logbogen (**cvs log**) viser ikke selve ændringerne i en fil. For at se dem skal man bruge tilvalget **diff**. Den helt enkle brug af tilvalget **diff** er:

```
hven% cvs diff tjek.php
```

der viser forskellene mellem den udgave af `tjek.php` der ligger i CVS-arkivet og den der arbejdes med.

Hvis man vil se forskellene mellem to udgaver af en fil i CVS-arkivet, bruger man tilvalget `-r` til at identificere de to udgaver:

```
hven% cvs diff -r 1.21 -r 1.22 vaerktoej-cvs.shtml
```

5.1.2.7. CVS med SSH på en alternativ port

Hvis CVS-arkivet ligger på en maskine der står bag en dørvogter, eller hvis SSH af andre mystiske omstændigheder ikke kører på port 22, er det lidt sværere at få `cvs` til at kontakte arkivet. Problemet kan klares ved at man fortæller SSH hvilken port SSH kører på *uden* at fortælle CVS det. Det gør man i SSH's opsætningsfil `$HOME/.ssh/config` hvor man for eksempel kan tilføje:

```
Host cvs.saltholm.oresund.dk
  Hostname skatkammeret.saltholm.oresund.dk
  Port 4444
```

hvilket betyder at når CVS -- og dermed SSH -- bliver bedt om at kontakte "cvs.saltholm.oresund.dk" (underforstået at det er på port 22), så vil SSH i virkeligheden kontakte "skatkammeret.saltholm.oresund.dk" på port 4444.

Nu kan du altså kontakte CVS-arkivet bag dørvogteren ganske som i eksempel Eksempel 5-3:

```
hven% export CVS_RSH=ssh
hven% cvs \
-d cvs.saltholm.oresund.dk:/usr/local/CVSR00T checkout hemmeligt_projekt
```

5.1.3. Opsætning af CVS

Endelig skal nævnes hvordan man opretter et fælles CVS-arkiv, og hvordan man lægger en samling filer ind i det fælles arkiv.

5.1.3.1. Oprettelse af et CVS-arkiv

Hvis du ikke allerede har adgang til et CVS-arkiv, du kan lægge dine filer ind i, må du oprette et. Hvis du har systemadministratorrettigheder bør du oprette et katalog med navnet `/usr/share/CVSR00T` og lægge det der. Alternativt kan du for eksempel lægge det i `$HOME/.CVSR00T`. Det er vigtigt at alle der skal kunne lægge filer ind i og rette filer i CVS-arkivet, har adgang til at skrive i kataloget. Vi går her ud fra at de alle er med i brugergruppen "cvsusers" (se systemadministratorbogen for information om brugergrupper). CVS-arkivet oprettes med kommandoen:

```
hven% cvs -d $HOME/.CVSR00T init
```

Når en bruger vil oprette sin egen arbejdskopi af CVS-arkivet, skal han/hun blot kende navnet på det katalog du valgte at lægge det fælles centrale CVS-arkiv i (husk at "`$HOME`" refererer til *dit* hjemmekatalog; **echo** `$HOME/.CVSR00T` vil give dig det fulde navn på det katalog der blev brugt i ovenstående eksempel). Se i øvrigt eksemplerne Eksempel 5-1, Eksempel 5-2 og Eksempel 5-3 for flere detaljer. Vi kommer ikke ind på hvordan man giver anonym adgang til et CVS-arkiv her.

5.1.3.2. Opret et projekt i CVS-arkivet

Lad os komme videre. For at du kan lægge dine filer i CVS for første gang, skal du flytte dig til hovedbiblioteket for kildeteksten (vi antager at det – og projektet – hedder `webspell`, og at du oprettede CVS-arkivet som beskrevet i forrige afsnit) og importere filerne, fjerne filtræet og til sidst trække det ud af CVS-arkivet igen (denne gang *med* udgavehåndteringsinformation):

```
hven% cd webspell
hven% cvs -d $HOME/.CVSR00T import webspell Tyge foerste-udgave-i-cvs
hven% cd ..
```

```
hven% mv webspell webspell-før_cvs
hven% cvs -d $HOME/.CVSROOT checkout webspell
```

Tyge er et felt der af CVS kaldes "vendor tag" (leverandørmærke), og **foerste-udgave-i-cvs** er et felt der af CVS kaldes "release tag" (udgavemærke).

Det kan også nævnes at en række ekstra værktøjer fås til CVS, se <http://freshmeat.net/> under CVS. Et af disse er cvs2html der findes fra <http://cvs.sslug.dk/cvs2html/>. cvs2html anvendes til at reformattere den loginformation der er skrevet ind for hver fil ved "cvs commit", til HTML-filer. For internetbaserede samarbejdsprojekter er dette virkeligt smart.

Det tager et par timer at komme ind i denne måde at arbejde på, men ved alle programmerings- eller skriveprojekter i øvrigt med mere end en person vil CVS tjene dette overhead ind i hundredefold.

5.1.3.3. Få breve med log-meddelelser ved cvs commit

Når flere folk arbejder sammen på et software-projekt, er det nødvendigt at man nemt kan se hvad andre personer har ændret i filerne. Med **cvs watch add FILNAVN** vil du modtage breve når andre kører **cvs commit FILNAVN**, men her vil du ikke se hvad de har ændret. Du kan få en bedre føling med ændringerne ved at bruge et lille program log der følger med cvs. Med det program kan du få et brev om hvilke filer som er ændret, og du får også den log-meddelelse brugeren skrev, med i brevet. Hvis man inden for projektet holder en god stil med at skrive ordentlige log-meddelelser, så er dette super.

Lad os se på opsætningen af dette. Først skal du lave en logfil i CVSROOT-kataloget som alle deltagere i dit projekt kan læse og skrive. *I det følgende skal du erstatte STI_TIL_CVS-RODEN med den fulde sti til dit CVSROOT-katalog.*

```
hven% touch STI_TIL_CVS-RODEN/CVSROOT/commitlog
hven% chgrp PROJEKTGRUPPE STI_TIL_CVS-RODEN/CVSROOT/commitlog
hven% chmod g+rw STI_TIL_CVS-RODEN/CVSROOT/commitlog
```

Dernæst skal vi rode i noget af de "helligste" af cvs-opsætningen.

```
hven% cvs -d STI_TIL_CVS-RODEN checkout CVSROOT
hven% vi CVSROOT/logininfo
```

Indsæt følgende linje til sidst hvis tyge@hven.oresund.dk skal modtage breve. Er der flere som skal modtage breve, så tilføj gerne flere "-m modtager@maskine" efter hinanden:

```
DEFAULT /usr/share/cvs/contrib/log -m tyge@hven.oresund.dk -f STI_TIL_CVS-RODEN/CVSROOT/commitlog
```

```
hven% cvs commit -m "Man kan modtage logbeskeder per post" CVSROOT/logininfo
hven% cvs release -d CVSROOT
```

De sidste linjer vil regenerere CVSROOT (sted hvor CVS-opsætningen er gemt), og derefter sletter man den CVSROOT-kopi man har tjekket ud.

5.1.4. Forgreninger i udviklingen

Lad os se lidt mere på starten, da vi lavede projektet.

```
hven% cd MODUL_DIR
hven% cvs -d STI_TIL_CVS-RODEN import MODUL VENDOR_TAG RELEASE_TAG
```

Dette genererer med det samme to grene (engelsk: "branches") som man altid kan komme tilbage til.

- 1.1.1 – VENDOR_TAG
- 1.1.1.1 – RELEASE_TAG

Går man i gang med at tjekke filerne ud, rette og køre **cvs commit**, så kommer de enkelte versioner til at hedde version 1.1, 1.2, 1.3 osv.

Ofte vil alle bruge samme gren af filerne, men hvad gør man hvis man nu har brug for at kode et stykke tid på noget som lammer resten af udviklerne på samme projekt. I dette tilfælde kan man oprette en gren til dette – de andre udviklere kommer ikke til at se noget anderledes, og to adskilte udviklingsgrene kan køre.

Lad os antage at vi arbejder med et modul i CVS med navn "sslugkode" som er importeret i CVS ved at skrive **cvs import sslugkode start version0**. Da der ikke er nogen naturlig "vendor" sættes leverandør-tag blot til "start" og start-version til "version0".

For at skabe "gren1" ud fra hovedgrenen "version0" skriver man (uden at man nødvendigvis har koden tjekket ud):

```
hven% cvs rtag -b -r version0 gren1 sslugkode
```

Tilsvarende kunne man måske have en anden udvikler som skal arbejde samtidig på en anden gren, "gren2":

```
hven% cvs rtag -b -r version0 gren2 sslugkode
```

For at isolere sig selv i grenen "gren1" skrives:

```
hven% cvs -d STI_TIL_CVS-RODEN checkout sslugkode
hven% cd sslugkode
hven% cvs update -r gren1
```

eller direkte

```
hven% cvs -d STI_TIL_CVS-RODEN checkout -r gren1 sslugkode
```

Det normale er derefter at udvikleren retter, laver **cvs commit** som vanligt. Fordelen er at de andre udviklere ikke ser dette. Grenene er totalt afkoblede indtil der laves en "merge"

Hvis det skulle ske at vores udvikler fra "gren1" skal hjælpe ham som arbejder i "gren2"-grenen, så kan han lave **cvs commit** på egne filer og derfter skifte direkte over til "gren2":

```
hven% cvs update -r gren2
```

Endelig kan man skifte ud til hoved-sporet (komme ud af henholdsvis "gren1" og "gren2" ved at skrive:

```
hven% cvs update -A
```

5.1.4.1. Flette grene sammen igen

Antag at to udviklingsgrene som hver for sig har udviklet sig, på et tidspunkt skal flettes sammen. Du har været så smart at du kørte **cvs tag TAG1** da du startede med din "gren1", og du har også gemt et mærke TAG2 ved slutningen af "gren1" – dvs. du har lige kørt **cvs tag TAG2** i "gren1".

I "gren2"-grenen kan du opdatere samtlige ændringer fra "gren1"-grenen mellem TAG1 og TAG2 ved at skrive:

```
hven% cvs update -j TAG1 -j TAG2
```

Hvis man ikke var i "gren2", skal man først sikre at man er i "gren2"-grenen:

```
hven% cvs update -r gren2
hven% cvs update -j TAG1 -j TAG2
```

Der er lidt mere information om emnet på <http://groups.yahoo.com/group/info-cvs/message/12246>.

5.1.5. Diverse med CVS

Det kan nævnes at hvis Emacs indlæser en fil der er koblet ind i CVS, vil Emacs automatisk genkende dette og indsætte en ekstra menu, hvorfra du på enkel vis kan sende tilføjelser til den fælles database (selve ændringerne skal du ikke selv styre – det klarer Emacs og CVS).

Skal man anvende en "export" version til distribution, eller blot ikke ønsker at lave en "checkout", så er **cvs export PROJEKT** også en mulighed. Den henter kun filerne, men ikke CVS-informationsfilerne. Meget praktisk til at skelne mellem udgivelse af filer ("export") i forhold til udvikler-version med fuld CVS-information ("checkout").

Vi har tidligere vist at projekt/kildetekst hentes med:

```
hven% cvs STI_TIL_CVS-RODEN checkout projektnavn
```

Hvis man ser efter i projekt-katalog/CVS, så finder man der tre filer som tilhører CVS:

- Repository indeholder projekt-navn på CVS-serveren.
- Root indeholder CVSR00T for det pågældende projekt.
- Entries indeholder listen over de filer og kataloger som er registreret i CVS.

Næste trick er for dem som anvender SSH til at kommunikere med CVS. Man kan få ssh til at understøtte komprimering i ~/.ssh/config:

```
Compression      yes
CompressionLevel 6
```

Men vær opmærksom på at visse servere ikke kan finde ud af at anvende komprimering, så er der uforklarlige problemer med at få forbindelse, så slå det fra.

5.2. diff og patch

Ofte vil programmøren skulle se hvad der er ændret mellem to udgaver af en fil. Er der måske 2000 linjer kode er dette ofte umuligt at gøre manuelt, og derfor bør du se nærmere på et par værktøjer der kan hjælpe dig meget. Lad os tage et eksempel på hvordan dette gøres:

Eksempel 5-10. Anvendelse af diff

Først ser vi på en program-stump vi kan kalde A_org.c.

```
typedef struct {
    unsigned int R;
    unsigned int G;
    unsigned int B;
} RGB_Image;
```

```
typedef struct {
    int *m,*n;
    int No;
} pixar;
```

Denne skal vi sammenligne med A.c, der ser således ud:

```
typedef struct {
    unsigned int R;
    unsigned int G;
    unsigned int B;
} RGB_Image;
```

```
typedef struct {
    int *m,*p;
    int No;
} pixar;
```

```
int a,b,c;
```

De ser jo meget ens ud, de to stumper kode. For at se forskelle kører vi

```
hven% diff A_org.c A.c
```

```
8c8
<  int *m, *n;
---
>  int *m, *p;
10a11,13
>
>  int a, b, c;
```

Vi kan se at linje 8 er ændret (8c8 betyder at linje 8 i A_org.c er ændret i forhold til linje 8 i A.c). Næste information (10a11,13) er at linje 11 til 13 i A.c er appended, dvs. tilføjet, i forhold til linje 10 i A_org.c.

En lille huskeregel mht. > og <: < er filen til venstre i argumentlisten til **diff**, og tilsvarende er > filen til højre (2. argument).

Det skal også nævnes at man ofte laver en "unified diff" med

```
hven% diff -u A_org.c A.c
```

```
--- A_org.c      Tue Dec 19 21:ec 19 21:56:43 2000
@@ -5,6 +5,8 @@
 } RGB_Image;
```

```
typedef struct {
-  int *m, *n;
+  int *m, *p;
   int No;
} pixar;
+
+int a, b, c;
```

Her er samme information gemt, men med en anden syntaks. Med - og + vises hvilke linjer som er ændret, men også et par af de foregående og efterfølgende linjer kommer også med. Det gør det lidt nemmere at finde stedet for ændringer efterfølgende.

Ofte er der kun små ændringer mellem to udgaver af en fil, og derfor ses det ofte at man distribuerer kildetekst til en basisudgave og derefter en eller flere "lapper" til programmet. Lapperne er filer lavet ved at finde forskellen mellem basisudgaven og den nye udgave af filen med **diff**. Lad os se dette i et eksempel.

Eksempel 5-11. Anvendelse af patch

Normalt anvendes lapper til at sende ændringer/opdateringer fra programmøren til andre der programmerer på samme fil(er). Derfor er filnavne A_org.c og A.c som var anvendt i eksemplet ovenfor normalt anvendt som den originale fil i forhold til den nye. Ofte har programmøren to fil-strukturer liggende - den originale og den nye med ændringer.

Først lader vi programmøren af programmet danne en lap svarende til forskellen mellem filerne A_org.c og A.c hvor de to filer er nævnt ovenfor:

```
hven% diff -u A_org.c A.c > A_patch
```

Bemærk at jeg her vender de to filer så den nye er sidst!

Nu kan programmøren sende lappen A_patch til andre programmører som kan opdatere deres A.c. Lad os prøve dette. Vi antager nu at vi er brugeren "tyge" som har en fil A.c som er lig med programmørens fil A_org.c.

```
hven% patch < A_patch
patching file A.c
```

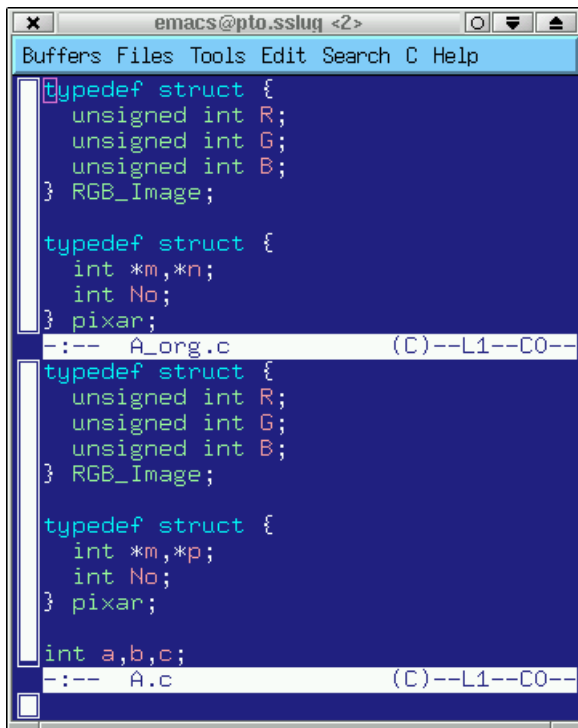
Så nemt er det at opdatere filen. En lap kan indeholde rettelser til mange filer som så automatisk opdateres.

5.3. Emacs og forskellige udgaver af filer

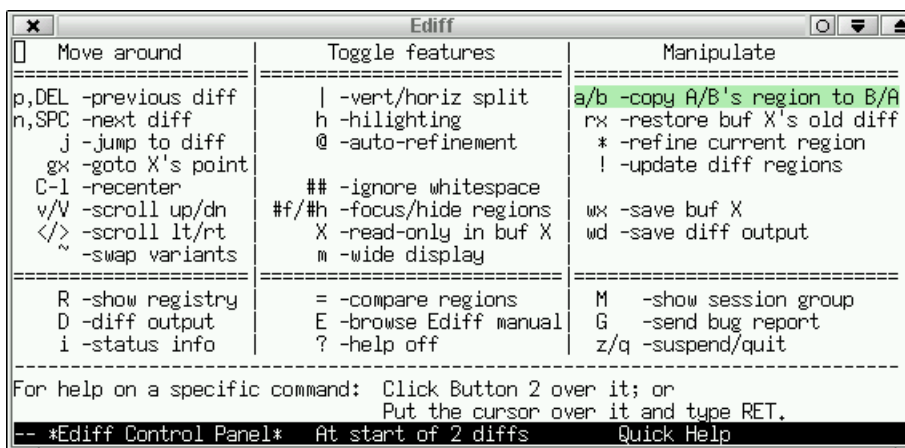
Som programmør (eller skribent af Linux-bøger) sker det ofte at en person sender en ny og opdateret udgave af en eller flere filer, og det er så programmøren, der skal læse rettelserne igennem og vurdere hvilke som skal accepteres til næste udgivelse af kildeteksten.

Hvis man skal sammenligne forskellige udgaver af filer, så er tekst-editoren **emacs** eller **xemacs** et program som virkelig er godt. Det er bare svært at forstå som en nybegynder. Men hæng i – det betaler sig. Lad os tage et eksempel. Start **emacs A_org.c A.c** – som er de to filer som blev vist ovenfor. Resultatet er vist på Figur 5-1.

Vælg nu menuen **Tools->Compare->Two Buffers...**, og accepter at buffer A er A_org.c og buffer B er A.c. Der kommer nu en lille menu-boks frem. Prøv at trykke ? i denne (før musen hen til boksen – tryk venstre museknap hvis boksen ikke er valgt – derefter ?). Her er alle kommandoer man kan anvende – se dette på Figur 5-2. Tryk ? igen for at komme tilbage til den lille ramme. Vi skal nu blive her i den lille ramme og kun anvende det store emacs-vindue til at se hvad der sker.

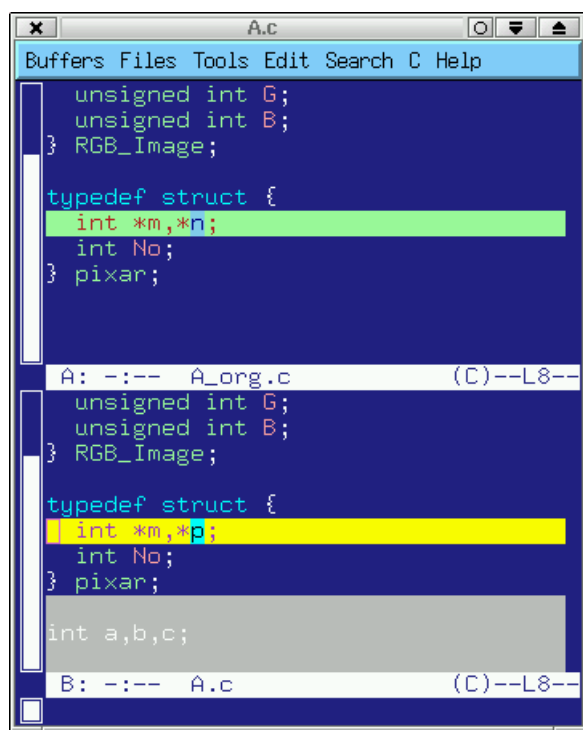


Figur 5-1. Emacs med to filer indlæst



Figur 5-2. Emacs' diff-kommandoer

Lad os komme i gang med at se hvordan vores `A_org.c` kan opdateres med rettelserne i `A.c`. Tryk **n** for at gå til første rettelse. Som det kan ses på Figur 5-3, så er der et "n" fra `A_org.c` som er ændret til et "p" i `A.c`. Hvis man nu vil acceptere denne rettelse og føre den fra `A.c` til `A_org.c` så trykker vi "b" for at føre rettelsen fra buffer B til buffer A. (Modsat så anvendes "a" for at kopiere fra A til B). Tryk "n" for næste rettelse og "p" for forrige (previous). Man kan også trykke "!" løbende for at opdatere tælleren i den lille boks med antal forskelle. Skriv til sidst "q" for at slutte denne sammenlignings-session. Svar dernæst "y" i bunden af det store Emacs-vindue. Gem filerne passende, og du er færdig.

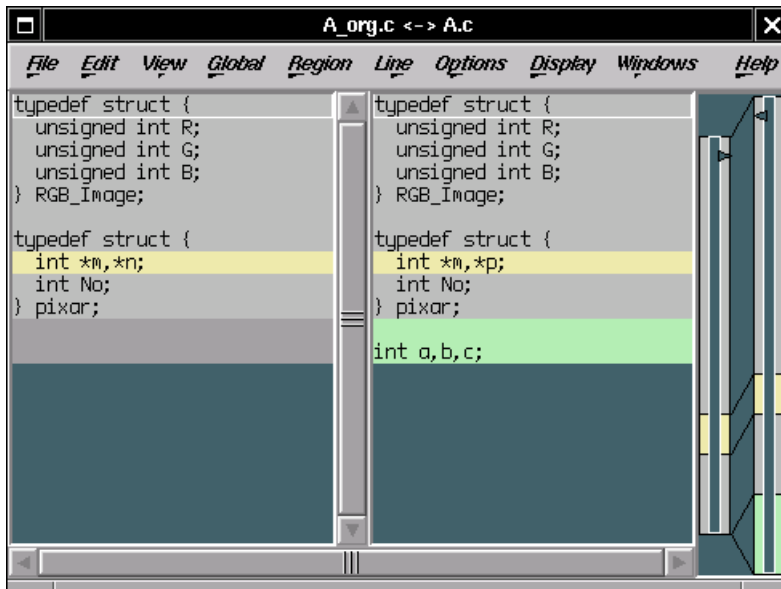


Figur 5-3. Emacs viser første rettelse

5.4. xxdiff

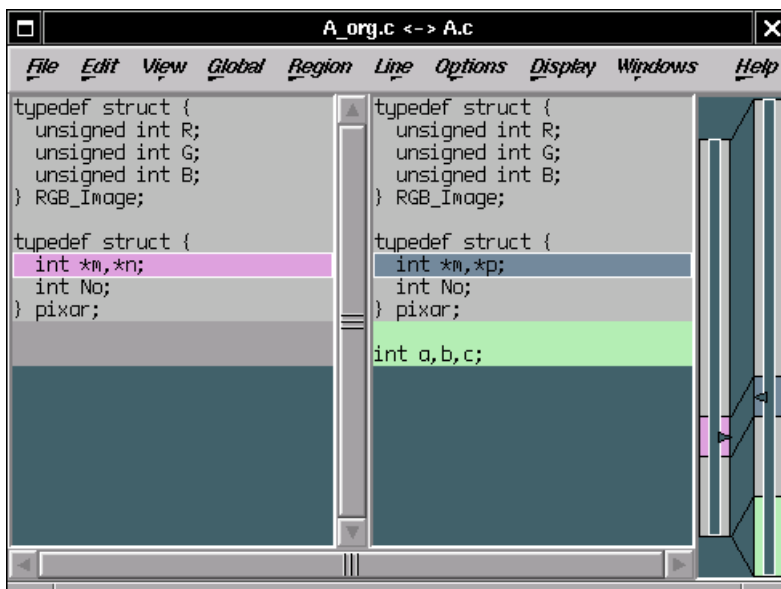
Hvis man ikke bruger emacs, kan man få samme funktionalitet ved at bruge programmet **xxdiff** <http://xxdiff.sourceforge.net/>.

Programmet startes med **xxdiff A_org.c A.c**.



Figur 5-4. xxdiff med to filer indlæst

For at gå til den første forskel trykkes "n". Hvis man vil bruge udgaven til venstre, trykker man "h", og hvis man vil bruge den til højre, trykker man "k". Bemærk at man har en grafisk oversigt over sine valg i hele filen ude i højre margen.



Figur 5-5. Den venstre udgave er valgt ved første forskel

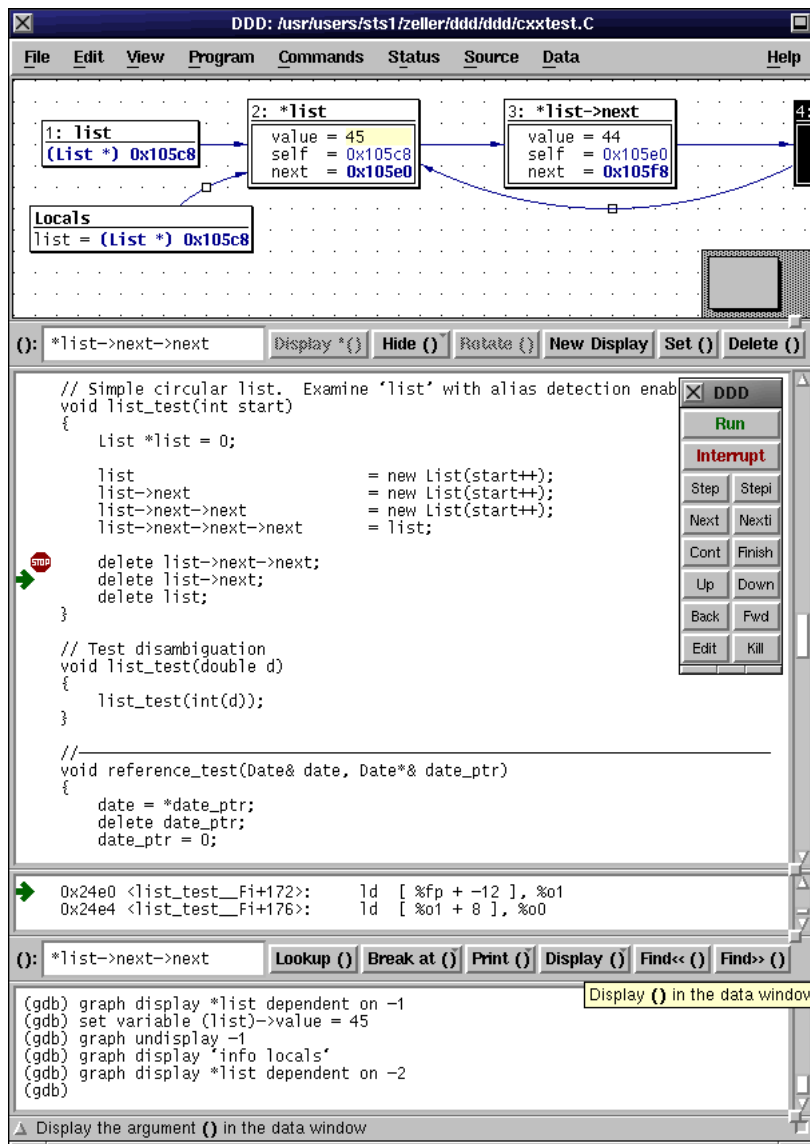
For at komme til næste forskel trykkes "n", og for den forrige "p". Når man er færdig med at vælge, skal man huske at gemme filen.

5.5. GNU debuggeren og DDD

Til GNU C/C++ findes en tekst-baseret debugger (gdb) som virker fint sammen med oversætteren. Hvis alle filer er oversat med flaget -ggdb, kan man kalde debuggeren ved at skrive **gdb filnavn** hvor **filnavn** er navnet på den kørbare fil.

Man kan vise (display) enkelte variabler eller strukturer. Du kan sætte "breakpoints" og meget andet, men det hele er tekstbaseret. Programmet **gdb** kan med fordel kaldes fra editoren Emacs ved at skrive **Meta-x gdb** (Meta=Alt). Så fås en delt skærm med debugger og kildetekst der kører sammen. Du kan i et terminalvindue skrive **man gdb** for at få muligheder for debuggeren. Når et program skal debugges så er det en meget god idé at oversætte det uden optimering, dvs. fjern -O2 ved oversættelse.

Ønsker du at anvende en grafisk debugger, kan du med fordel installere **ddd** (Data-Display-Debugger) som giver en fuldt professionel grafisk brugerflade til fejlfinding i C-, C++-, Perl-, Python-, Java- og Fortran-kildetekst. Specielt lækkert er at man nemt kan følge indholdet af strukturer, tabeller og simple variabler. DDD kan hentes fra <http://www.cs.tu-bs.de/softech/ddd>.



Figur 5-6. DDD

5.6. GNU Visual Debugger

Hvis der er en fejl i et program som du ikke kan finde årsagen til, så kan et fejlfindingsværktøj som GNU Visual Debugger (GVD) være nyttigt. Fidusen er at man kan køre programmet inde fra GVD der så holder øje med værdien af variable, status for forskellige parallelt kørende dele af programmet, hvilke funktioner og procedurer der er kaldt, og meget andet nyttigt. Ved at køre programmet i små skridt, og indimellem tjekke relevante data, kan man måske finde tydeligere spor efter fejlen.

Hvis GVD skal kunne hjælpe dig med at forstå hvad et program gør, er det nødvendigt at bede oversætteren om at inkludere fejlfindingsinformation i det oversatte program. Hvis det for eksempel drejer sig om Ada-programmet `fejlfindingseksempel.adb`, kan det gøres med følgende kommando:

```
hven% gnatmake fejlfindingseksempel -cargs -g
```

For alle oversættere der bygger på GCC er det flaget `-g` der gør at der bliver inkluderet fejlfindingsinformation i det oversatte program.

Når du har et program der indeholder fejlfindingsinformation, kan du starte GVD og indlæse programmet (her `fejlfindingseksempel` fra før) med kommandoen:

```
hven% gvd fejlfindingseksempel
```

Når GVD kører vil dets vindue være delt op i fire dele:

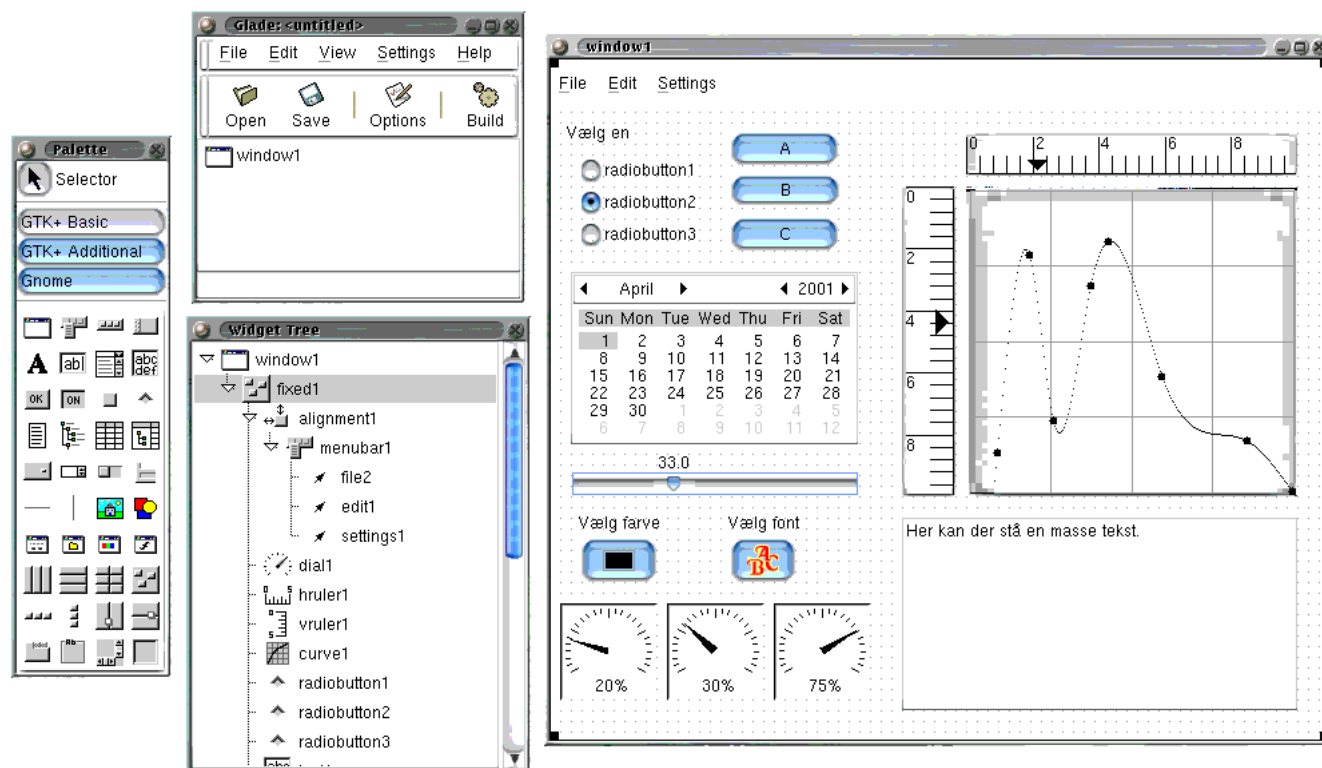
- Nederst er der en log over hvad der sker med programmet.
- I midten til venstre er der en oversigt over de kildetekstfiler programmet er sammensat af.
- I midten til højre vises de kildetekstfiler du vælger (desværre kun en ad gangen).
- Øverst er der en opslagstavle hvor du kan se udvalgte variables værdier. Når GVD starter er opslagstavlen tom. Du vælger variablene ved at klikke på dem med højre musetast nede i kildeteksten. Der fremkommer så en menu hvor et af punkterne er "Vis <variablens navn>". Når du vælger det, kommer der et lille vindue med variablen oppe på opslagstavlen.

Den vigtigste fidus ved et fejlfindingsprogram er nok at du kan vælge at køre programmet én linje ad gangen. Hvis du har en anelse mere travlt eller allerede ved hvilken del af programmet du vil studere, kan du indsætte *stopklodser* i programmet. Det gør du ved at højreklikke på den linje i kildeteksten du vil stoppe ved, og vælge "Sæt stopklods på linje <linjenummer>" på menuen. Hvis du nu starter programmet (ved at taste F2), vil programmet stoppe – men ikke afslutte – første gang det kommer til denne linje. Du kan nu gå en linje frem ad gangen med F5 eller lade programmet køre videre – indtil det kommer til en anden stopklods – med F8.

5.7. Glade

Glade er et program der kan bruges til at designe grafiske brugergrænseflader der bruger GTK- og Gnome-bibliotekerne. Glade kan generere grafiske brugergrænseflader og gemme dem som Ada, C-, C++-, Eiffel og Perl-kildetekst. Glade genererer automatisk oversættelsesfiler til at oversætte programmet.

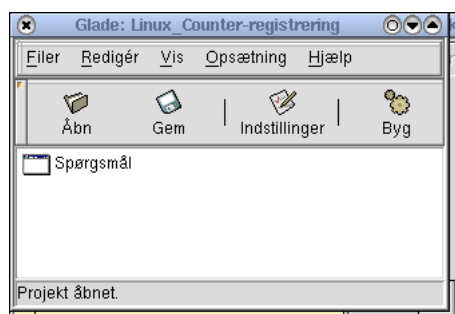
I Glade kan man for eksempel definere at der skal kaldes en bestemt funktion når der, for eksempel, trykkes på en knap. Glade gemmer så kildeteksten med en tom funktionsdefinition. Det er så op til programmøren at indsætte kode i denne funktion så programmet rent faktisk gør noget.



Figur 5-7. Design af grafisk brugergrænseflade i Glade (her vist med Aqua-temaet).

En specielt smart egenskab ved Glade er at den kan gemme hele definitionen af en grafisk brugergrænseflade i en fil. Ved hjælp af biblioteket "libglade" kan et program automatisk genskabe den grafiske brugergrænseflade, så man kan distribuere et program med en redigerbar grafisk brugergrænseflade.

I bogens eksempelsamling er der et eksempel på en grafisk brugergrænseflade lavet i Glade (http://cvs.linuxbog.dk/program/eksempler/linux_counter_registrering.glade). Hvis man starter Glade og indlæser denne fil, vil hovedvinduet komme til at se ud som Figur 5-8.



Figur 5-8. Glades hovedvindue med filen "linux_counter_registrering.glade" indlæst.

Hvis man så dobbeltklikker på teksten "Spørgsmål", vil Glade vise et præsentationsvindue med en dialog (se Figur 5-9) hvor man så kan tilføje, rette eller slette elementer.

Linux Counter-registrering

E-postadresse:

Navn:

Synlig e-postadresse: ☐ Ja ☒ Nej

Land: Brug venligst ISO-3166-landekoden (DK, SE, NO, ...).

Delstat:

By:

Brugt Linux siden:

søn	man	tir	ons	tor	fre	lør
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5
6	7	8	9	10	11	12

Vælg bare en omtrentlig dato, hvis du ikke kan huske det præcise tidspunkt for hvornår du begyndte at bruge Linux.

Bruger Linux: ☐ Hjemme ☐ I skolen ☐ På arbejdet

Hemmeligholdelse: ☒ Jeg vil være anonym ☐ Mit navn må gerne stå i listen

Indsend data Afbrød registrering

Figur 5-9. Glades præsentationsvindue med dialogen "Spørgsmål" fra filen "linux_counter_registrering.glade".

5.8. Gettext

Gettext er et værktøj du bør bruge til at gøre det enkelt for andre at oversætte menutekster m.m. i dine programmer til andre sprog. Princippet i Gettext er at man skriver sit program med alle teksterne til brugergrænsefladen på ét sprog (typisk amerikansk), men indkapsler dem i et funktionskald til Gettext-biblioteket der slår oversættelsen af teksten op i en database for det sprog brugeren har valgt.

Eksempel 5-12. Eksempel på brug af Gettext fra et POSIX Shell-program

Selv i et POSIX Shell-program kan vi bruge Gettext til at håndtere oversættelsen af de tekster brugeren skal se. Vi bruger et program, 'hej', der bare skriver "Hej Verden." som eksempel:

```
#!/bin/sh
echo "Hej Verden."
```

Det er gængs praksis at bruge amerikansk engelsk som det grundlæggende sprog i programmer der skal kunne oversættes til alle mulige sprog, så nu bruger vi i stedet for teksten "Hello World." samtidig med at vi erstatter 'echo' med 'gettext --domain eksempel -s', hvor "eksempel" er navnet på en samling oversættelser og "-s" får 'gettext' til at efterligne 'echo':

```
#!/bin/sh
gettext --domain eksempel -s "Hello World."
```

Hvis man bare uden videre kører dette program, skulle det gerne skrive "Hello World.", da 'gettext' når den ikke kan finde filen med oversættelserne (her "/usr/share/locale/da/LC_MESSAGES/eksempel.mo") eller ikke kan finde teksten der skal oversættes (her "Hello World.") i filen med oversættelserne, bare udskriver teksten på originalsproget (som den står i programmet).

Nu laver vi så en oversættelse af programmet til dansk. Kildeteksten er filen eksempel.da.po:

```
msgid "Hello World."
msgstr "Hej Verden."
```

Den kan vi oversætte til Gettexts interne format (".mo") med programmet 'msgfmt':

```
hven% msgfmt --output-file=eksempel.da.mo eksempel.da.po
```

For at vi kan bruge den skal den helst flyttes til systemets standardkatalog for danske oversættelser, /usr/share/locale/da/LC_MESSAGES/:

```
hven% su -c 'cp eksempel.da.mo /usr/share/locale/da/LC_MESSAGES/eksempel.mo'
```

Password:

Hvis du nu stiller om til at dine programmer skal være på dansk (kun i den aktive kommandofortolker) vil programmet skrive "Hej Verden." i stedet for "Hello World.":

```
hven% ./hej
Hello World.
```

```
hven% export LANG=da_DK.IS08859-1
hven% ./hej
Hej Verden.
Vi kan selvfølgelig også stille sproget om til færøsk, men da vi ikke har oversat programmet endnu, vil vi bare få
teksten ud på originalsproget:
hven% export LANG=fo_F0.IS08859-1
hven% ./hej
Hello World.
```

Eksempel 5-13. Eksempel på brug af Gettext fra et Ada-program

(kommer senere)

Eksempel 5-14. Eksempel på brug af Gettext fra et C-program

(kommer senere)

Når et program er forberedt til oversættelse med "gettext", er det næste skridt at oversætte det. Til det formål er KBabel (<http://www.klid.dk/dansk/kbabel.html>) nok det bedste værktøj.

5.9. Electric Fence

Electric Fence er et bibliotek der er uvurderligt til hurtigt finde bufferoverløb eller -underløb. Et typisk problem i C og C++ (men praktisk taget ikke i nogle andre sprog) er at man allokerer en tabel, men forsøger at skrive efter tabellens slut eller før dens start. En anden typisk fejl er at allokere en streng der er for lille, som i dette eksempel:

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char* p=(char*)malloc(16*sizeof(char));
    char* q=(char*)malloc(16*sizeof(char));

    // test for NULL

    strcpy(q,"efence test");
    strcpy(p,"efence test buffer overflow"); // 27 tegn lang!!

    printf("q:%s\n",q);
    printf("p:%s\n",p);
}
```

Oversættes dette program med **gcc -o test test.c**, så kan der ske alt muligt når det køres, men typisk vil det skrive:

```
q:low
p:efence test buffer overflow
```

En streng kopieres til strengpegeren p, men strengen er længere end den hukommelsesblok der er allokeret. Strcpy overskriver derfor delvist strengen q "efence test". Et program kan sagtens køre videre efter et bufferoverløb, men måske er dette bufferoverløb årsagen til at programmet fejler katastrofalt en halv time senere. Det kan derfor være uhyre svært at finde denne type fejl da de to ting tilsyneladende vil være helt urelaterede.

Oversættes programmet i stedet med **gcc -o test test.c -ggdb -lefence** vil programmet fejle og gemme en core-fil når det udfører den anden strcpy() linje. Køres **ddd test core** så vil debuggeren fortælle i hvilken linje bufferoverløbet skete. **man efence** giver en masse information om hvordan efence virker, og hvordan det kan bruges til at teste for bufferunderløb og meget andet.

5.10. Brug af oversættelsesfiler ("makefiles")

Næste eksempel er to filer hvor den ene kalder funktioner i den anden. I eksemplet der er skrevet i C, er det igen det største af to tal der skal findes og skrives ud.

Følgende fil kan gemmes som `ceks2.c`.

```
#include <stdio.h>
#include <math.h>

float v1,v2,retval;

float max(float, float);

int main(void)
{
    v1=sin(2.34);
    v2=tan(4.4);

    retval=max(v1,v2);

    printf("Af %f og %f er %f stoerst\n",v1,v2,retval);
    return 0;
}
```

Følgende fil kan gemmes som `cfkt.c`.

```
float max(float f1, float f2)
{
    float f3;

    if (f1>f2)
        f3=f1;
    else
        f3=f2;

    return f3;
}
```

hven% **gcc -o ceks2 ceks2.c cfkt.c -lm**

En langt mere elegant metode er at lave en oversættelsesfil. Herefter skrives blot **make**, og C-oversætteren vil kun oversætte de filer der er nyere end de oversatte filer. Oversættelsesfiler kan også bruges til at styre oversættelse af andet end regulære programmer. For eksempel LaTeX-kode. En oversættelsesfil (med navnet `Makefile`) kunne være følgende.

```
# Makefile til GNU C-oversætteren - dette er en kommentar
# I dette eksempel oversættes to filer.
# ceks2.c og cfkt.c oversættes til den kørbare fil ceks2

# Kommandoen til at oversætte C-programmer er "gcc":
CC = gcc

# Navn paa koerbar fil
OUTPUT = ceks2

# Kildefilnavne, bemærk der er ingen mellemrum efter \ tegnet.
sources = ceks2.c \
          cfkt.c

# Automatisk navngivning af objektfiler hvor .c laves om til .o
OBJS = $(sources:.c=.o)

# Oring niveau 2
CFLAGS = -O2

#Linkerflag: Inkluderer matematik-bibliotek
LDFLAGS = -lm

# Følgende linje tjekker om en .o fil er nyere end den koerbare fil. Hvis
# dette er tilfaeldet, linkes disse. Dvs. kun nye elementer oversættes.
```

```
$(OUTPUT): $(OBJS)
    $(CC) $(CFLAGS) -o $(OUTPUT) $(OBJS) $(LDFLAGS)
```

Bemærk at indrykninger i oversættelsesfiler skal være tabulatortegn! Det fungerer ikke med mellemrum.

Oversættelsesfilen består af en masse definitioner af variable og en eller flere opskrifter. Opskriften fortæller at \$(OUTPUT) (ceks2) afhænger af objektkodefilerne \$(OBJS) (ceks2.o og cfkt.o), og at ceks2-programmet kan laves med opskriften **gcc -O2 -o ceks2 ceks2.o cfkt.o -lm**. Make-programmet gætter selv hvordan .c-filer skal oversættes for at danne objektfiler .o.

Når oversættelsesfilen er skrevet, kan programmet bare oversættes ved at skrive

hven% **make**

i det katalog hvor oversættelsesfilen findes. En oversættelsesfil kan være meget lang og indeholde mange opskrifter der indbyrdes kan afhænge af hinanden. Make-programmet undersøger datoen på alle filer, og hvis datoen på kildeteksten er nyere end datoen på den tilsvarende objektfil, så er objektfilen forældet, og make oversætter automatisk kildeteksten igen og samler programmet med den nye objektfil.

5.11. Autoconf og automake

Makefiles er en stor hjælp til selv små projekter. Et problem der hurtigt opstår er at projektet begynder at udvide sig og blive mere og mere komplekst. Det får så din Makefile til at blive voldsomt stor, fyldt med fejl og uoverskuelig. Mange af de operationer der er i en Makefile, er trivielle gentagelser fra projekt til projekt. Når der skal ske det samme så mange gange, ville det være en fordel at indbygge dette i **make** eller lave noget udenom **make** der kan håndtere dette. Resultatet blev **autoconf**.

Formålet med **autoconf** er få oprettet en "make"-fil der passer til slutbrugerens linux-version og eventuelt med brugerens specielle ønsker. Hele **autoconf**-systemet er lidt af en rodebunke og ret komplekst. Der findes flere mindre gode instruktioner til **autoconf**, og dette er nok endnu et eksempel – vi forsøger alligevel.

Følgende programpakker skal være installeret inden man kan gå i gang: make, autoconf og automake.

For at gøre det hele så simpelt som muligt, laver vi det klassiske "Hello, world!" eksempel og bygger videre på dette. I et tomt underkatalog har vi så det lille simple program hello.c med følgende indhold.

```
/* Filnavn: hello.c */
#include <stdio.h>
int main(void)
{
    printf("Hello, world!\n");
    return 0;
}
```

For at komme i gang med selve **autoconf** mangler vi configure.in og Makefile.am. Makefile.am er en fil til programmet **automake**, og den vil så oprette Makefile.in. Makefile.am er ofte en lille fil og ser således ud i en minimal udgave for et C-program.

```
## Filnavn: Makefile.am
AUTOMAKE_OPTIONS = foreign
bin_PROGRAMS = hello
hello_SOURCES = hello.c
```

Næste fil er configure.in hvilket er en fil der kan gå hen og blive ret stor. I sin minimale udgave ser den således ud:

```
# Filnavn: configure.in
AC_INIT(hello.c)
AM_INIT_AUTOMAKE( demo-hello, 1.0 )
AC_PROG_CC
AC_STDC_HEADERS
AC_OUTPUT(Makefile)
```

Med filerne hello.c, Makefile.am og configure.in kan vi gå i gang med at oprette de filer vi senere skal have distribueret til brugeren.

aclocal.m4 er en samling af alle de shell-macroer som bruges i configure.in. Filen aclocal.m4 dannes af programmet **aclocal** ved at **aclocal** skanner configure.in for hvilke macroer der bliver brugt. Macroerne finder den i .m4-filer som dem i /usr/share/aclocal/ (Den kigger i nuværende bibliotek og /usr/share/aclocal/ automatisk,

man kan angive yderligere biblioteker). **aclocal** skal køres hvis der anvendes nye makroer i `configure.in`. Når man da kalder **autoconf** for at danne **configure** ud fra `configure.in`, vil **autoconf** erstatte brug af macroer med koden fra `aclocal.m4`.

```
hven% aclocal
```

For at få oprettet `Makefile.in` – det er den der senere skal læses af **configure** – køres kommandoen **automake -a**.

```
hven% automake -a
```

Der manglede nogle filer til **automake** hvilket vi automatisk får med, med option `'-a'` (`--add-missing`).

```
hven% autoconf
hven% ./configure
hven% make
```

Ved oversættelse ses at der er brugt flaget `'-g'` til `'gcc'`, det betyder at der kommer debug-information med i programmet. Dette kan slås fra ved at sætte `CFLAGS` i `configure.in` til for eksempel `"-O2"`. Dertil kommer at der er to oversætterdirektiver, `PACKAGE` og `VERSION`, med på kommandolinjen. Der kan komme flere direktiver, så det vil være smart hvis disse blev lagt i en fil for sig, så kommandolinjen ikke bliver for lang. Dette kan gøres ved at tilføje `AM_CONFIG_HEADER(config.h)` i `configure.in`.

```
# Filnavn: configure.in
AC_INIT(hello.c)
AM_INIT_AUTOMAKE( demo-hello, 1.0 )
AM_CONFIG_HEADER(config.h)
CFLAGS="-O2"
AC_PROG_CC
AC_STDC_HEADERS
AC_OUTPUT(Makefile)
```

Efter tilføjelse af `AM_CONFIG_HEADER` kan systemet drille lidt ved at komme med nogle sære fejlmeddelelser. Dette skyldes at der er nogle afhængigheder i `aclocal.m4`, så denne fil skal opdateres med kommandoen **aclocal**. For at få oprettet header-filen `config.h.in` køres **autoheader**. **./configure** vil så oprette filen `config.h` som evt. kan tilføjes i `hello.c`. Dernæst kan **./configure** køres igen, og der oprettes en `Makefile`.

```
hven% aclocal
hven% autoheader ; autoconf
hven% ./configure
hven% make clean ; make
```

Bemærk at der nu under oversættelse tilføjes `-DHAVE_CONFIG_H` på kommandolinjen. Vi kan nu teste i `hello.c` om der er en `config.h` og inkludere denne. Dernæst tjekker vi om `PACKAGE` og `VERSION` er defineret, og udskriver disse værdier. Tag et kig på `config.h`, og se hvad den indeholder.

```
/* Filnavn: hello.c */
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
#include <stdio.h>

int main(void)
{
#ifdef PACKAGE
#ifdef VERSION
    printf("Bygget med pakken: %s-%s\n\n", PACKAGE, VERSION);
#endif
#endif
    printf("Hello, world!\n");
    return 0;
}
```

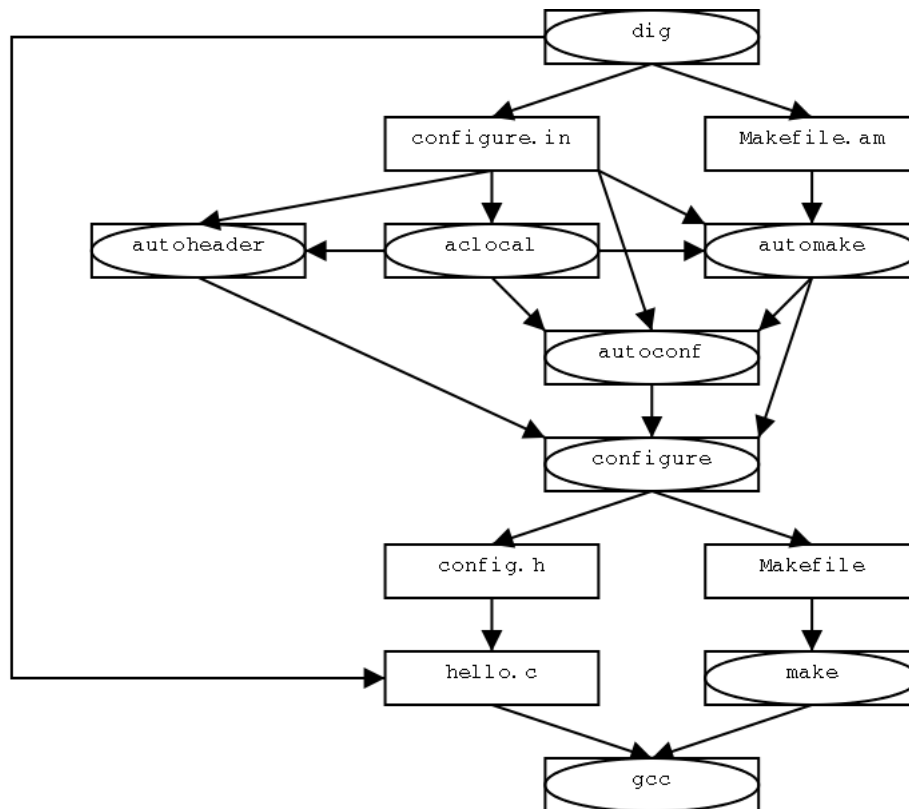
Der er en del filer i et `autoconf`-system og som nybegynder kan det være noget svært at få et overblik. Listen herunder giver en beskrivelse af de mest almindelige filer.

Filnavn	Skrives af	Læses af
---------	------------	----------

Tabel 5-1. Filer i et autoconf system

Filnavn	Skrives af	Læses af
hello.c	*dig*	gcc
Makefile.am	*dig*	automake
configure.in	*dig*	automake autoconf aclocal
config.h.in	autoheader	automake configure
aclocal.m4	aclocal	autoconf
Makefile.in	automake	configure
missing	automake	make??
install-sh	automake	"make install"
mkinstalldirs	automake	configure
stamp-h.in	automake	configure
configure	autoconf	*script*
Makefile	configure	make
config.h	configure	hello.c(gcc)
stamp-h	configure	configure
config.cache	configure	configure
config.status	configure	configure
config.log	configure	*dig*

I figuren herunder ses en skematisk tegning af de vigtigste filer i et autoconf system.



Figur 5-10. Autoconf road map

Du har nu fået oprettet de to mest basale filer `Makefile.am` og `configure.in` som fint kan danne basis for et projekt. Lad os lige stoppe op og kikke på hvad der er oprettet af filer og hvorfor. `Makefile.am` er den mest overordnede fil og også den mest avancerede m.h.t. til brug af makroer. Heldigvis for forståelsen er der ikke så mange kommandoer i denne fil, de er bare ret avancerede. `Makefile.am` indeholder informationer om hvordan projektet overordnet ser ud. Hvilke er selve programmerne, hvor skal disse installeres, hvilke kildefiler kræver disse programmer osv. En `Makefile.am` skal oprettes i hvert underkatalog, så dem kan man nemt få nogle stykker af.

Den `Makefile.am` der ligger i roden af projektet, har kommandoen **AUTOMAKE_OPTIONS** som overordnet bestemmer hvilken type af projekt man kører. En option man kan bruge til frie programmer er fx. **gnu** som er default. Med **gnu**-optionen opretter **automake** en del ekstra filer såsom `COPYING` og `INSTALL`. Dertil forventer **automake** at der er oprettet fire ekstra filer (`NEWS` `README` `AUTHORS` `ChangeLog`). Den nye `Makefile.am` ser således ud:

```
## Filnavn: Makefile.am
AUTOMAKE_OPTIONS = gnu
bin_PROGRAMS = hello
hello_SOURCES = hello.c
```

Inden **automake** køres igen, skal vi lige have tilføjet de ekstra filer som en **gnu**-installation kræver, og så kan **automake** køres. En ny `Makefile.in` oprettes, og den indeholder ny oplysninger om flere filer der skal med i en distribution når man senere kører kommandoen **make dist**.

```
hven% touch NEWS README AUTHORS ChangeLog
hven% automake
```

Kommandoen **bin_PROGRAMS** er en noget mere avanceret makro der fortæller hvilke binære filer der skal oversættes, og hvor disse skal installeres. Flere programmer kan indeholdes i en pakke, og de skrives på samme linje med mellemrum imellem. *bin* betyder at programmerne installeres i det underkatalog der er defineret i variabelen **bindir**. Slutbrugeren kan så senere under installation bestemme hvor binære filer skal installeres. Default vil programmer blive installeret i `/usr/local/bin/`, men for nogle programmer gælder at disse skal installeres i `/bin/`. Vi kan på forhånd selv bestemme **bindir**, hvilket vi kommer tilbage til. **bin_PROGRAMS** har en fætter der hedder **sbin_PROGRAMS**, og den har default `/usr/local/sbin/`.

hello_SOURCES er en noget speciel variable der fortæller hvilke kildefiler der anvendes til et enkelt program. **automake** tager hvert program der er listet i fx. **bin_PROGRAMS** og leder efter variabelen **??_SOURCES**. **automake** forventer altså at denne variabel er oprettet, og der kommer selvfølgelig en fejl hvis den ikke er.

`configure.in` er filen der fortæller hvordan en konfiguration skal foregå. Her kikkedes der ikke på kildetekster etc., men alene på hvordan konfigurationen skal forløbe for slutbrugeren. Til et projekt er der kun én `configure.in`, og denne er placeret i roden af projektet.

Der kan komme mange underlige fejl undervejs, hvilket kan afhjælpes ved at køre de forskellige programmer igen. For alle programmer gælder at det ikke skader at køre dem igen, og det tager heller ikke ret lang tid at gøre det. Så går noget galt, så kørs denne stribe programmer igen:

```
hven% aclocal; automake -a; autoheader; autoconf; ./configure
```

Læs mere: <http://sources.redhat.com/automake/automake.html>, http://sources.redhat.com/autoconf/autoconf_toc.html, <http://www.andamooka.org/reader.pl?section=autobook> og "info automake" lokalt på dit system. Se også `autoconf`-filerne fra `distribution fileutils`.

5.11.1. Autoconf, underkataloger

Til større projekter er det en fordel at lægge afgrænsede dele af projektet i underkataloger. Som tidligere nævnt består et `autoconf`-system af en `configure.in` og evt. flere `Makefile.am`. I roden af projektet lægges en `Makefile.am` der fortæller hvilke underkataloger der skal med, og i hvert katalog en `Makefile.am` der beskriver hvilke kildetekster der skal med i dette katalog. I dette eksempel er der et katalog med en C-fil og et katalog med en man-fil. Her er listen af filerne før vi går i gang med at køre `autoconf`-programmerne.

```
./Makefile.am
./configure.in
man/Makefile.am
man/hello.1
src/Makefile.am
src/hello.c
```

De enkelte filer ser således ud:

Filnavn: `./Makefile.am`

```
AUTOMAKE_OPTIONS = foreign
SUBDIRS = man src
```

Filnavn: `./configure.in`

```
AC_INIT(src/hello.c)
AM_INIT_AUTOMAKE( katalog-hello, 1.0 )
AC_PROG_CC
AC_STDC_HEADERS
AC_OUTPUT(Makefile man/Makefile src/Makefile)
```

Filnavn: `man/Makefile.am`

```
man_MANS = hello.1
man_aux = $(man_MANS:.1=.x)
EXTRA_DIST = $(man_MANS)
```

Filnavn: `man/hello.1`

```
.TH HELLO 1 "1 August 2001" "Hello" "Friheden til at programmere"
.SH NAME
hello \- en hilsen
.SH SYNOPSIS
hello
.SH DESCRIPTION
.B hello
skrives "Hello, world!" til standard out.
.SH USAGE
hello
.SH COPYRIGHT
Copyright \ (c) 2001 Friheden til at programmere.
.SH SEE ALSO
goodbye(1)
```

Filnavn: src/Makefile.am

```
bin_PROGRAMS = hello
hello_SOURCES = hello.c
```

Filnavn: src/hello.c

```
#include <stdio.h>
int main(void)
{
    printf("Hello, world!\n");
    return 0;
}
```

Med alle filer på plads er det blot at køre autoconf-programmerne:

```
hven% aclocal; automake -a; autoconf
hven% ./configure
hven% make
```

Anvendelse af underkataloger kræver altså ikke meget mere end at have alle filer i samme katalog. `./Makefile.am` har **AUTOMAKE_OPTIONS** som så ikke er i de andre `Makefile.am`-filer, **SUBDIRS** angiver i hvilke underkataloger der ligger flere `Makefile.am`'s. Specielt for `./configure.in` skal det bemærkes at alle `Makefiler` skal angives i **AC_OUTPUT()**.

5.11.2. Autoconf/automake eksempel

I eksemplet i Afsnit 5.10 er vist hvordan man manuelt opbygger en `Makefile`. Skal samme eksempel laves med autoconf/automake, skal der skrives to filer:

```
## Filnavn: Makefile.am
AUTOMAKE_OPTIONS = foreign
bin_PROGRAMS = ceks2
ceks2_SOURCES = ceks2.c cfkt.c

# Filnavn: configure.in
AC_INIT(ceks2.c)
AM_INIT_AUTOMAKE( demo-ceks , 1.0 )
AC_PROG_CC
AC_STDC_HEADERS
AC_OUTPUT(Makefile)
```

De to filer kan se meget anderledes ud, men dette er noget af det simpleste man kan have til et C-program. Herefter er det blot at køre nogle kommandoer:

```
hven% automake -a
hven% aclocal
hven% autoconf
hven% ./configure
hven% make
hven% make install
hven% make uninstall
hven% make dist # o.s.v...
```

Det er lidt mere besværligt end det oprindelige eksempel, men vi fik også en **make install** plus et par andre gode ting for besværet.

5.11.3. Installation af andre filer

I de andre eksempler er vist hvordan C-programmer kan oversættes og installeres i `/usr/local/bin` ved at bruge makroen **bin_PROGRAMS**. Har man programmer der skal lægges i `/usr/local/sbin` er det makroen **sbin_PROGRAMS** man skal bruge. HTML-filer, kommandofortolker og Perl-programmer skal håndteres anderledes. Her er et lille eksempel på hvad der skal tilføjes for at klare disse fil-typer.

```
## Filnavn: Makefile.am
...
bin_SCRIPTS = hello.pl hello.sh
```

```
sbin_SCRIPTS = minbackup.sh
sysconfdir_DATA = hello.conf
htmldir = /var/www/html/hello
html_DATA = index.html
...
```

I ovennævnte eksempel er de første tre makroer prædefinerede. Til **bin_PROGRAMS** er allerede defineret **bindir**, hvilket kan ændres af den bruger der installere programmet. Hvis bin-filerne skal ned i /bin vil brugeren skrive:

```
hven% ./configure --bindir=/bin
```

Listen over prædefinerede kataloger fås nemmest med:

```
hven% ./configure --help | grep DIR
```

I førnævnte eksempel er opfundet et nyt katalog med navnet htmldir. Skal brugeren selv have lov til at bestemme hvilket katalog det er, skal der lidt mere til, og vi skal over og rette i `configure.in`. Der skal ske det at der kommer en ny option som bliver listet sammen med **./configure --help**, og en variabel der kan sættes.

```
# Filnavn: configure.in
...
htmldir=/var/www/html/hello
AC_MSG_CHECKING(htmldir)
AC_ARG_WITH(htmldir,
[ --with-htmldir=DIR      Where to put html/php files],
[ case "$withval" in
  yes|no)
    ;;
  * )
    htmldir="$withval"
    ;;
  esac])
AC_SUBST(htmldir)
AC_MSG_RESULT(${htmldir})
...
```

Prøv herefter følgende kommandoer:

```
hven% ./configure --help
hven% ./configure --htmldir=/home/www
hven% make -n install | more
```

Eksemplet med htmldir er inspireret af automake-manualen og `configure.in` fra Samba.

5.11.4. Tjek at en funktion er til stede

En god ting at have med i sin `configure.in` er at tjekke om de funktioner man bruger i C-programmerne er til stede i det pågældende system hvor **./configure** køres. I `hello.c` bruges kun **printf()** og den tjekkes således:

```
# Filnavn: configure.in
...
AC_CHECK_FUNCS(printf)
...
```

Kør **autoconf** og **./configure** for at se resultatet. Flere funktioner kan skrives som parameter med mellemrum imellem. Eksemplet er fundet i Sambas `configure.in`.

5.11.5. Variable i config.h

En spændende ting man kan lave med `configure.in`, er at tilføje variable til `config.h` med dynamisk input på det tidspunkt `./configure` køres. For ikke at kommandolinjen skal blive for lang, er det bedst at have `AM_CONFIG_HEADER` sat i `configure.in`. I følgende eksempel hentes den aktuelle dato og tid når `./configure` køres. Den kan herefter findes i `config.h` og bruges i for eksempel `hello.c`.

```
# Filnavn: configure.in
...
CONFDATE='date "+%Y-%m-%d %H:%M:%S"'
AC_DEFINE_UNQUOTED(CONFDATE, "$CONFDATE", [Dato for kørsel af ./configure])
...
```

For at få hele systemet opdateret kræves at **autoheader** køres for at få opdateret `config.h.in`, **autoconf** skal køres for at få opdateret `configure`, og `./configure` skal køres for at opdatere `config.h`. Herefter kan man se resultatet i `config.h`.

```
hven% autoheader ; autoconf ; ./configure
hven% cat config.h
```

Oversætterdirektivet kan nu ses i `config.h`, og det kan så bruges i `hello.c`.

```
/* Filnavn: hello.c */
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
#include <stdio.h>

int main(void)
{
#ifdef CONFDATE
    printf("./configure er kørt d.: %s\n", CONFDATE );
#endif
    printf("Hello, world!\n");
    return 0;
}
```

Herefter er det blot at køre **make** og `./hello`, så vises dato og tid for hvornår `./configure` er kørt. Alternativt til dato kunne være kommandoen **uname -a** for at få oplysninger om det system programmet blev oversat på, eller en option som brugeren der kører `./configure`, selv kan definere. Eksemplet er fundet i PHP's `configure.in`.

5.11.6. Autoconf og andre sprog

Hidtil er sproget C blevet brugt i eksemplerne, men `autoconf/automake` kan også bruges til både C++ og Fortran. Det er blot et spørgsmål om at `AC_PROG_CC` udskiftes med henholdsvis `AC_PROG_CXX` eller `AC_PROG_F77`. Se følgende eksempel:

```
# Filnavn: configure.in
AC_INIT(hello.c++)
AM_INIT_AUTOMAKE( demo-hello, 1.0 )
AC_PROG_CXX
AC_OUTPUT(Makefile)
```

5.12. Anjuta

Hvis du har lyst til at lave programmer i Gtk eller Gnome, bør du se lidt på Anjuta. Anjuta er et integreret udviklingsmiljø lavet i Gnome, og som har en masse funktioner. Blandt andet er der syntaksfremhævnning til en lang række sprog. Blandt andre C, C++, Perl, PHP, Python og SQL.

Den har også en troldmand som hjælper en med at starte et projekt. Anjuta klarer selv alt det med "make"-filer (medmindre man slår det fra) og arbejder sammen med Glade. Derudover har den alle de "normale" funktioner som automatisk færdiggøring af kode, integreret fejlsøgningsværktøj og den slags. Dermed skulle kunne den kunne bruges af både begyndere og erfarne programmører.

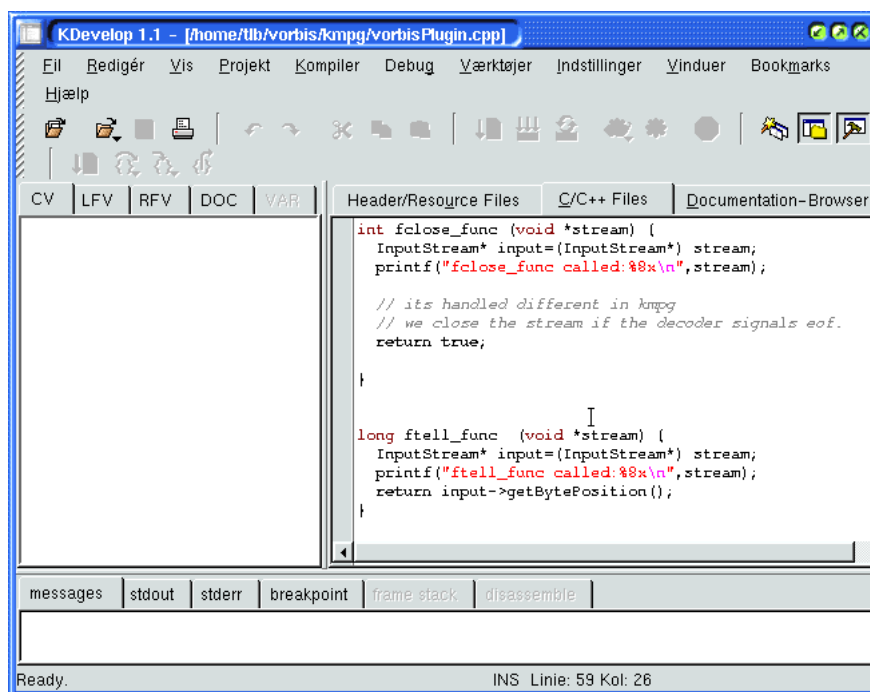
5.13. KDevelop

Der er ved at komme integrerede udviklingsmiljøer til Linux som det kendes fra f.eks. Visual C++ til Windows. Man kan købe Code Warrior til Linux, Code Crusader og Delphi er også godt på vej. Lige nu ser Code Warrior og Delphi mest lovende ud.

Et spændende helt nyt program er KDevelop til KDE.

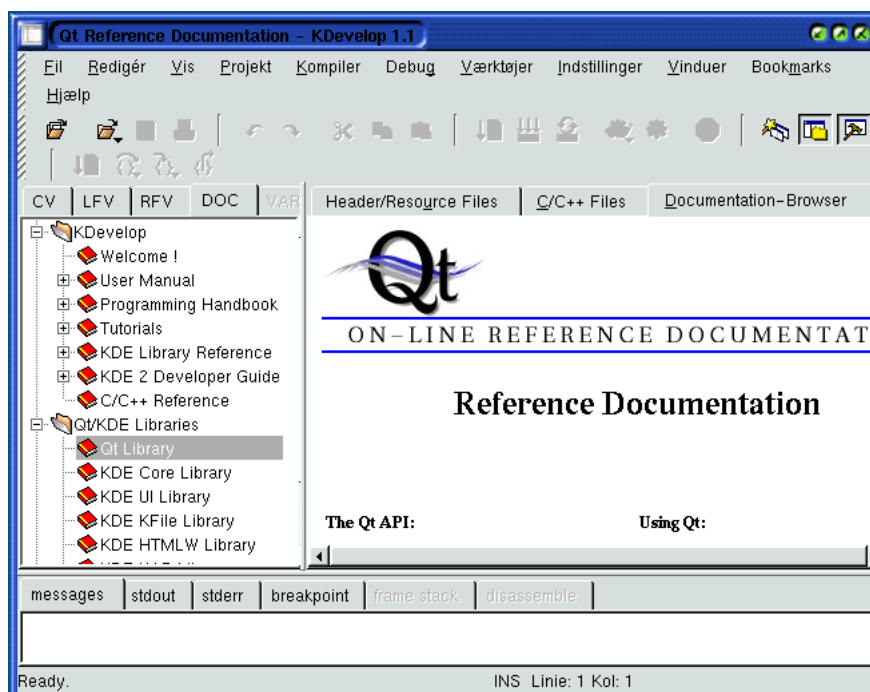
Ambitionsniveauet er meget højt, og til trods for at det er den første beta-udgave som er blevet afprøvet her, så ser det positivt ud.

Som det ses på Figur 5-11, så er der ligheder med Visual C++ fra Microsoft. Man kan nemt overskue alle filer, klasser, strukturer og variable i projektet. Der er god understøttelse for at oversætte og søge fejl (ser det ud til), og revisionskontrol er direkte integreret. Det er baseret på CVS. Der er mulighed for integreret dokumentation baseret på SGML.



Figur 5-11. KDevelop

En ting som er et meget stort plus ved KDevelop i forhold til Visual C++, er at alle projektfiler er tekstbaserede, dvs. man kan se alt med en almindelig teksteditor og rette hvis man har noget specielt, der skal ind. KDevelop laver i øvrigt selv standard-oversættelsesfiler, så man kan oversætte programmerne uden for det grafiske miljø. Med andre ord så er KDevelop en naturlig overbygning på GNU-værktøjerne, uden at disse erstattes. KDevelop følger med f.eks. Red Hat 7.0 og kan hentes fra projektets websted, <http://www.kdevelop.org/>.



Figur 5-12. KDevelop har naturligvis også indbygget dokumentation for at lave grafiske programmer.

5.14. emacs

emacs kaldes af mange "et helt styresystem". ...

5.15. vi

vi er et generelt tekstredigeringsprogram der findes på stort set alle Unix-systemer. (Er det krævet for at et system kan betegnes som et Unix-system?) Det kan ikke på nogen måde kaldes begyndervenligt, men det at det altid er der, sammen med den effektivitet hvormed øvede brugere af programmet kan arbejde i det, gør at det har en betydelig skare af brugere. Et betydeligt brugervenligheds-mæssigt kritikpunkt ved vi er at det arbejder med tre forskellige modi – og i tidlige udgaver af programmet endda uden visuel markering af hvilken modus man befandt sig i.

En fortæller for vi gør opmærksom på at det fungerer som det skal, med det samme – i modsætning til Emacs der skal have tilrettet sin opsætningsfil for at fungere fornuftigt.

5.16. GRASP

Syntaksfremhævnning... Flowdiagrammer i marginen... Godt til at forstå kildetekst andre har skrevet...

5.17. a2ps

Programmet a2ps er egentlig blot beregnet til at oversætte tekst ("a" står for "ASCII") til et format printeren kan forstå ("ps" står for "Postscript"), men det er med tiden blevet udvidet til også at lave syntaksfremhævnning af kildetekst.

Eksempel 5-15. Udskrift af en enkelt fil

Hvis du for eksempel vil udskrive filen `program.adb` på printerens "pscats" med en side per A4-ark, kan det gøres med kommandoen **a2ps -1 -Ppscats program.adb**.

- "-1" står for "en side per A4-ark" og kan for eksempel erstattes med "-2" eller "-4".
- "-Ppscats" står for at udskriften skal sendes til printerens "pscats". Med en typisk opsætning af **a2ps** kan du helt undlade flaget "-P" hvis du vil have udskriften sendt til din primære printer.
- "program.adb" er navnet på filen der skal skrives ud. Der er ikke noget i vejen for at bede **a2ps** om at skrive flere filer ud på en gang ved at give det en liste med filnavne i stedet for et enkelt filnavn.

Eksempel 5-16. Konvertering af en fil til Postscript

Hvis du har brug for at oversætte filen `program.adb` til Postscript med en side per A4-ark, kan det gøres med kommandoen **a2ps -1 --output=program-body.ps program.adb**.

- "-1" står for "en side per A4-ark" og kan for eksempel erstattes med "-2" eller "-4".
- "--output=program-body.ps" står for at udskriften skal sendes til filen `program-body.ps`.
- "program.adb" er navnet på filen der skal skrives ud. Der er ikke noget i vejen for at bede **a2ps** om at skrive flere filer ud på en gang ved at give det en liste med filnavne i stedet for et enkelt filnavn.

Slutbemærkning:

1. <http://prep.ai.mit.edu/pub/gnu>
2. <http://www.cygnum.com/misc/gnu-win32>
3. <http://www.linux.ie/articles/tutorials/managingaccesswithcvs.php>
4. <http://www.cvshome.org/docs/blandly.html>
5. <http://www.ibiblio.org/mdw/REF/CVS-BestPractices/html/index.html>
6. <http://cvsbook.red-bean.com/cvsbook.html>
7. <http://cvs.sslug.dk/cvs2html/>
8. <http://freshmeat.net/>
9. <http://cvs.sslug.dk/cvs2html/>
10. <http://groups.yahoo.com/group/info-cvs/message/12246>
11. <http://xxdiff.sourceforge.net/>
12. <http://www.cs.tu-bs.de/softech/ddd>
13. http://cvs.linuxbog.dk/program/eksempler/linux_counter_registrering.glade
14. <http://www.klid.dk/dansk/kbabel.html>
15. <http://sources.redhat.com/automake/automake.html>
16. http://sources.redhat.com/autoconf/autoconf_toc.html
17. <http://www.andamooka.org/reader.pl?section=autobook>
18. <http://www.kdevelop.org/>

Appendiks A. Installationsvejledninger

Hvis du arbejder på en pakkebaseret Linux-installation (for eksempel Debian, Mandrake, Red Hat eller SuSE), bør du så vidt muligt bruge systemets pakkehåndtering til at installere programmerne.

Eksempel A-1. Installation af pakker på et Debian-baseret system

Hvis du kører et Debian-baseret system, og der i en installationsvejledning her i bogen står at du på Debian-systemer har brug for en pakke ved navn "eksempel", skulle de følgende to kommandoer installere pakken på dit system.

```
hven% su -
Password:
hven# apt-get update
hven# apt-get install eksempel
hven# exit
hven%
```

Eksempel A-2. Installation af pakker på et RPM-baseret system

Hvis du kører et RPM-baseret system med en 32-bit Intel-kompatibel processor, og der i en installationsvejledning her i bogen står at du på RPM-baserede systemer har brug for en pakke ved navn "eksempel", skal du ud og lede efter en fil ved navn "eksempel-*<tal>.i386.rpm*" (hvis du kører på en anden processortype, erstattes "i386" med en tilsvarende streng for den processor). Hvis du er heldig, findes filen på en af dine Linux-cd-rom'er, og ellers må du ud på nettet og lede. Når du har fundet filen, så skulle den følgende kommando installere pakken på dit system.

```
hven% su -
Password:
hven# rpm --upgrade eksempel-<tal>.i386.rpm
hven# exit
hven%
```

Det kan her ske at systemet brokker sig over at der mangler nogle pakker som denne pakke har brug for. Dem bliver du i så fald nødt til at installere først.

Eksempel A-3. Installation af ".tar.gz"-filer

Hvis du kun kan finde en pakke med et program i form af en ".tar.gz"-fil, bør du strengt taget pakke det om til en pakke af den type dit system bygger på. Du kan se hvordan det gøres i afsnit ... (Debian-pakker) eller afsnit ... (RPM-pakker). Hvis vi vil springe den ideelle løsning over og bare installere programmet i en fart, kan det normalt klares med den følgende række kommandoer (vi antager at pakken hedder "eksempel-0.3.1.tar.gz").

```
hven% tar xzvf eksempel-0.3.1.tar.gz
eksempel-0.3.1/COPYRIGHT
eksempel-0.3.1/configure
...
hven% cd eksempel-0.3.1/
hven% ./configure && make && echo 0.k.
...
0.k.
hven% su root -c 'make install && echo 0.k.'
Password:
...
0.k.
hven% make clean
...
hven% cd ../
hven% rm -rf eksempel-0.3.1/
```

Kort fortalt sker der følgende: Pakken pakkes ud (**tar**), sættes op (**./configure**), oversættes (**make**), installeres (**make install**), og til sidst ryddes der op.

A.1. Ada

For at komme i gang med at programmere i Ada på Linux skal du blot installere GNU Ada oversætteren (GNAT), men der er et bredt udvalg af andre pakker man med fordel kan installere til forskellige specielle formål. Tabel A-1 indeholder en oversigt over de vigtigste pakker fordelt på de opgaver de er relevante til. Hvis du kører SuSE, Red Hat eller en lignende RPM-baseret distribution, bør du kunne finde alle de relevante pakker hos "Ada for Linux Team" (ALT) på <http://dk.gnuada.org/alt>.

Bemærk at de aktuelle ALT-pakker ikke kan bruges sammen med Red Hat 8.0-pakkerne, så hvis man bruger "gcc-gnat" og "libgnat" fra Red Hat 8.0, vil man være nødt til selv at oversætte alt det tilhører som ALT ellers har pakket. Desuden er der (blandt andet) fejl i matematikbiblioteket der kommer med Red Hat 8.0-pakkerne.

Tabel A-1. Pakker til programmering i Ada

Opgave	Debian-pakker	Red Hat 8.0-pakker	RPM-pakker fra ALT
Generel Ada-programmering.	gnat, gnat-runtime	gcc-gnat, libgnat	gnat, gnat-runtime
Distribuerede programmer.	gnat-glade	?	gnat-glade
Programmer der kan køre på en "Java Virtual Machine" (JVM). For eksempel småprogrammer (engelsk: "applets") der skal køre i en webside.	jgnat (samt en JVM, se afsnit ...)	? (samt en JVM, se afsnit ...)	jgnat (samt en JVM, se afsnit ...)
Grafiske programmer der også umiddelbart kan oversættes til andre styresystemer.	GtkAda, GtkAda-runtime (samt GLADE, se Afsnit A.9)	? (samt GLADE, se Afsnit A.9)	GtkAda, GtkAda-runtime (samt GLADE, se Afsnit A.9)

Eksempel A-4. Oversætte og køre "smart.adb"

Kildeteksten til programeksemplet "smart.adb" kan findes i bogens eksempelsamling (<http://cvs.linuxbog.dk/program/eksempler/smart.adb>). Det kan oversættes med kommandoen:

```
linux% gnatmake -m smart -cargs -gnatv -gnati1 -gnatf -gnato -fstack-check
gnatgcc -c -gnatv -gnati1 -gnatf -gnato -fstack-check smart.adb
```

GNAT 3.13p (20000509) Copyright 1992-2000 Free Software Foundation, Inc.

Compiling: smart.adb (source file time stamp: 2003-01-10 20:43:30)

```
130 lines: No errors
gnatbind -x smart.ali
gnatlink smart.ali
linux%
```

Man kan også nøjes med **gnatmake smart**, men det er ikke ideelt, da der så er en del nyttige tjek der bliver fjernet fra programmet. Det oversatte program skulle nu gerne findes som **smart** i samme katalog, så du kan køre det sådan her:

```
linux% ./smart
```

```
-- Sætter porten til 2#1000000# --
Hej
```

```
-- Sætter porten til 2#1100000# --
```

```
-- Sætter porten til 2#10000# --
```

```
Ver
```

```
-- Sætter porten til 2#100000# --
```

```
den
```

```
linux%
```

Da programmets to tråde ikke snakker med hinanden undervejs, kan du ikke være sikker på at teksterne bliver blandet præcis som ovenfor.

A.2. Anjuta

Anjuta kan hentes på <http://anjuta.sourceforge.net/download.html>.

A.3. Basic

For at komme i gang med at programmere i Basic på Linux skal du blot hente og installere Basic-fortolkeren Yabasic⁴. Hvis du kører SuSE, Red Hat eller en lignende RPM-baseret distribution på en IA32-maskine (386, 486, Pentium, Athlon, m.fl.), bør det være nok at hente og installere RPM-pakken "yabasic" fra <http://www.yabasic.de/download.htm#binlin>. Hvis du kører et andet system, eller der er pakkeafhængigheder du ikke kan få til at passe, kan du i stedet for hente kildeteksten og oversætte "yabasic" selv. Det gøres som følger:

- Start med at sikre dig at "X" og "ncurses" er installeret på systemet.
- Hent filen `yabasic-<et-eller-andet>.tar.gz` fra <http://www.yabasic.de/download.htm#sourceunix>, og gem den i kataloget `/tmp/`.
- Pak filen ud:

```
hven% tar xzvf yabasic-<et-eller-andet>.tar.gz
```

- Skift til kataloget med kildeteksten:

```
hven% cd yabasic-<et-eller-andet>
```

- Oversæt og installér Yabasic:

```
hven% ./configure && make && make check && su root -c 'make install' && echo O.k.
```

Hvis alt går godt, bliver der til sidst skrevet "O.k.".

Andre BASIC-fortolkere og -oversættere:

- Bas⁷

A.4. Cobol

Til dette eksempel er anvendt tinycobol, se <http://tiny-cobol.sourceforge.net/>.

A.5. Comal

For at komme i gang med at programmere i Comal på Linux skal du blot installere OpenComal der kan hentes fra <http://www.josvisser.nl/opencomal/>.

A.6. Eiffel

For at komme i gang med at programmere i Eiffel på Linux skal du blot installere SmallEiffel¹⁰.

A.6.1. På RPM-baserede systemer

Hentes på <http://SmallEiffel.loria.fr/general/download.html>.

A.6.2. På Debian-baserede systemer

`apt-get install smalleiffel` skulle klare det. Ellers kan du hente kildeteksten på <http://SmallEiffel.loria.fr/general/download.html>.

A.7. Forth

Stakororienteret a la Postscript?

A.8. Fortran

...

A.9. GLADE

Tabel A-2 viser hvilke pakker der skal installeres for at man kan arbejde med GUI-byggeren GLADE.

Tabel A-2. Pakker med GLADE

Debian-pakker	RPM-pakker
glade	glade

A.10. Haskell

For at komme i gang med at programmere i Haskell på Linux skal du blot installere en Haskell-oversætter eller fortolker.

A.10.1. På RPM-baserede systemer

Du kan hente RPM-pakker med Haskell-fortolkeren Hugs på <http://www.cse.ogi.edu/PacSoft/projects/Hugs/pages/downloading.htm>.

A.10.2. På Debian-baserede systemer

`apt-get update && apt-get install hugs hugs-doc` skulle klare det. Ellers kan du hente kildeteksten på <http://www.cse.ogi.edu/PacSoft/projects/Hugs/pages/downloading.htm>.

A.11. Java

Javaprogrammer kan oversættes til almindelig maskinkode under Linux med værktøjet **gcc-java**.

Det første skridt er at hente `jdk-1.2.2-RC4-linux-i386-glibc-2.1.2.sh`. Denne pakke kan findes de sædvanlige steder såsom <http://sunsite.dk>, på <http://www.blackdown.org> og <http://java.sun.com>. JDK'en som du skal hente, skal passe til din Linux-distribution. Der er forskellige pakker til netop den version af glibc som du benytter i din distribution. Du kan se hvilken version af glibc du benytter, ved at se hvilken symbolsk henvisning `libc.so.6` peger på (enten `libc-2.1.x` eller `libc-2.0.x`). Pakken fylder godt 20 Mb.

Du kan installere kommandofortolkerinstallationspakken (ender på `.sh`), men der findes også almindelige tar og **gzip**'ede filer. Installation af kommandofortolkerinstallationen foregår ved (se nedenfor vedrørende hvor det kan betale sig at installere det):

```
hven# cd /usr/local
hven# chmod +x DIN_STI/jdk-1.2.2-RC4-linux-i386-glibc-2.1.2.sh
hven# DIN_STI/jdk-1.2.2-RC4-linux-i386-glibc-2.1.2.sh
```

Hermed er udviklings- og kørselsmiljøet til Java installeret! Du skal dog gennem et par ekstra skridt for at det virker tilfredsstillende.

Man mangler Zapf Dingbats-skriftnit i installationen. Disse hentes på <http://www.gimp.org/fonts.html>. Installation af disse er beskrevet på webstedet. Man skal i korthed hente URW-skriftnittene `urw-fonts.tar.gz`. Disse skal **gunzip**'pes og af-tar'es i X Window systemets skriftnitskatalog:

```
[root@beta /root]# cd /usr/X11R6/lib/X11/fonts
[root@beta /usr/X11R6/lib/X11/fonts]# tar xzvf /mnt/cdrom/Java/java3d/linux/urw-fonts.tar.gz
[root@beta /usr/X11R6/lib/X11/fonts]# chown -R root.root
```

Dernæst skal man ændre sin X Window opsætningsfil. Hvis ens distribution kører med X Window-skriftnitstjener, så hedder filen `/etc/X11/fs/config` (Red Hat), ellers skal man rette i `/etc/X11/XF86Config` (Red Hat) som beskrevet nedenfor (Red Hat 6.x bruger skriftnitstjener):

```
/usr/X11R6/lib/X11/fonts/misc:unscaled,
```

```
/usr/X11R6/lib/X11/fonts/75dpi:unscaled,  
/usr/X11R6/lib/X11/fonts/100dpi:unscaled,  
    /usr/X11R6/lib/X11/fonts/URW,  
/usr/X11R6/lib/X11/fonts/misc,  
/usr/X11R6/lib/X11/fonts/Type1,  
/usr/X11R6/lib/X11/fonts/Speedo,  
/usr/share/fonts/default/Type1
```

Ændringer til X Window Server XF86Config filen er beskrevet på <http://www.gimp.org/fonts>, og for skriftsnitstjeneren filen skal fjerde linje med URW-skriftsnittene føjes til filen `/etc/X11/fs/config`. For at genstarte skriftsnitstjeneren køres programmet `xfs` med parametren **restart** i `/etc/rc.d/init.d` (som brugeren "root"). Næste gang din Linux-maskine starter startes X-skriftsnitstjeneren automatisk med de nye typer.

Nu mangler vi bare at sørge for at dine systemvariable er sat korrekt. Det er specielt stien der skal være sat så den kan inkludere dine kørbare Java-programmer. Normalt behøver man ikke sætte andre systemvariable hvis man har installeret i `/usr/local/jdk1.2.2`. Kataloget der skal føjes til stien (systemvariablen PATH eller path), er `/usr/local/jdk1.2.2/bin`. Det er dog en god idé også at sætte JAVA_HOME til `/usr/local/jdk1.2.2`.

De to vigtigste kørbare filer er **javac** (oversætteren) og **java** (som er Javas virtuelle maskine, JVM).

A.11.1. Installation af Java3D

Der findes masser af andre pakker der er interessant til udvikling af Java-programmer. Det er et spørgsmål om hvad man har brug for, og så ellers komme i gang med den. Et interessant valg er Java3D. Det er et meget elegant design af et visualiserings API, så derfor vil jeg kort beskrive hvordan man installerer det, og hvad det kan.

For det første hører Java3D til media-pakkerne som indeholder alt til udvikling af multimedie og Virtual Reality, spil osv. Java3D er en pakke der stiller klasser til rådighed til udvikling af 3-d-visualiseringssystemer. Java3D er et elegant værktøj der skulle stille de fleste 3-d-visualiseringssystem-udviklere tilfreds. Det baserer sig på scenegrafprincippet. Dvs. alle visuelle objekter i en virtuel verden er hængt op i en scenegraf. Der er knuder der kan gruppere dele af det virtuelle univers, knuder der kan lave transformationer på underliggende knuder, knuder der repræsenterer egentlige virtuelle objekter, lys og kameraet (øjepunktet) osv.

Der skal en hel del til for at installere Java3D på din Linux-maskine. Først og fremmest bygger Java3D på OpenGL. På Linux er den frie udgave af OpenGL et bibliotek kaldet Mesa (<http://www.mesa3d.org/>). Mesa kan også findes som RPM- og deb-pakke, og Java3D har brug for version 3.1-3 (eller senere). Hvis du kører Red Hat 6.1, er du ude efter filen Mesa-3.1-3rh61.i386.rpm. Du kan vælge også at lægge glut (et bibliotek til at lave menuer og deslige) og Mesas eksempelsamling ind (Mesa-glut-3.1-3rh61.i386.rpm og Mesa-demos-3.1-3rh61.i386.rpm på et Red Hat 6.1-system).

Desværre er det sådan at Mesa lægger OpenGL lib-filerne som `libGL.so.xxx`, mens Java3d forventer at de hedder `libMesaGL.so.xxx`. Derfor skal du i `/usr/lib` lave et ekstra link:

```
hven# cd /usr/lib  
hven# ln -s libGL.so.1.2.0 libMesaGL.so.3
```

Nu er det så tid til at installere Java3D. Det er ikke så svært. Det vigtigste er at hvis man lægger Java3D samme sted som Java 2 SDK, er det problemfrit, ellers skal man sørge for at sætte systemvariablen CLASSPATH til at inkludere stien til Java3D-filerne.

Filen du skal hente fra <http://java.sun.com> er `java3d1.1.1pre-v1-linux-sdk.tar.bz2`. Du kan kun bruge java3D sammen med Java 2 (den der er beskrevet i det foregående).

Dernæst skal du installere den ved hjælp af **tar**. Inden du gør det, skal du gennem et par skridt for ikke at løbe ind i alt for mange problemer sidenhen. Du vil gerne have Java3D til at ligge sammen med din jdk1.2.2-installation. Problemet er at Java3D-installationen er pakket i et katalog der hedder jdk1.2, derfor skal du sikre dig at `/usr/local/jdk1.2/` peger på `/usr/local/jdk1.2.2/` før du udpakker Java3D-filen:

```
hven# cd /usr/local/  
hven# ln -s jdk1.2.2 jdk1.2  
hven# tar xjvf java3d1.1.1pre-v1-linux-sdk.tar.bz2  
...
```

(vi er her gået ud fra at du lagde Java3D i kataloget `/usr/local/`). Nu er du kørende med Java3d.

Det er specielt to ting der er lagt ned i din Java-installation. Det er delvist Java-biblioteker i `/usr/local/jdk1.2.2/jre/lib/ext` og Java3D demo-filer.

Inden vi beskriver et lille programeksempel, vil jeg omtale indlæsningsroutiner til java3d. Det er sådan at det kan være et meget omfattende arbejde og er egentlig imod 3-d-visualiseringsparadigmet at programmere sig til

den visuelle verden. Det man bør gøre, er at definere den virtuelle verden i en separat fil i et eller andet format. Der findes til Java3D en lang række indlæsningsroutiner til at læse sådanne filer. Et meget kendt format er VRML som på en måde er internettets foretrukne 3-d virtuelle modelformat. Java kommer ikke med en indlæsningsroutine til VRML, men heldigvis findes den gratis på <http://www.vrml.org/>. Du kan også hente java3d vrml indlæsningseksempler i kildetekst. Den Java-jar (Java biblioteksfil) som hedder `vrml97.jar`, skal lægges ned i `/usr/local/jdk1.2.2/jre/lib/ext`, så kan man bruge den i forbindelse med udvikling af Java-3-d-programmer der benytter indlæsning af VRML filer.

For eksempler på Java-3-d-programmer henvises der til installationen af Java3d, hvor der er inkluderet 37 programseksempler inkluderende vrml-eksempelprogrammer.

På internettet kan der desuden findes et utal af modeller af visuelle objekter, i alle mulige typer af formatter, herunder VRML.

A.12. ML

For at komme i gang med at programmere i ML på Linux skal du installere en af de frie implementationer af Standard ML. Der er et meget stort udvalg at vælge imellem, blandt andre:

- Moscow ML (udviklet i Cambridge, København og Moskva). Interaktivt system plus batch-oversætter. Genererer bytekode, ikke maskinkode. Se <http://www.dina.kvl.dk/~sestoft/mosml.html>.
- Standard ML of New Jersey (udviklet på Bell Labs og Princeton i New Jersey, USA). Interaktivt system plus batch-oversætter. Genererer maskinkode. Se <http://cm.bell-labs.com/cm/cs/what/smlnj/>.
- MLton (oprindelig udviklet ved NEC Research, New Jersey, USA). Kun batch-oversætter. Genererer C-kode som så oversættes til maskinkode. Se <http://www.sourcelight.com/MLton/>.
- Poly/ML (udviklet i Edinburgh og Cambridge). Interaktivt system plus batch-oversætter. Genererer maskinkode. Se <http://www.polymml.org/>.
- ML Kit (udviklet i København). Kun batch-oversætter. Genererer bytekode eller maskinkode. På basis af ML Kit er der udviklet en særlig effektiv ????. Se <http://www.it-c.dk/research/mlkit/> og <http://www.smlserver.org/>.
- MLj (udviklet i Cambridge og Edinburgh). Kun batch-oversætter. Genererer Java-bytekode. Se <http://www.dcs.ed.ac.uk/~mlj/>.

A.13. Modula-2

For at komme i gang med at programmere i Modula-2 på Linux skal du blot installere en Modula-2-oversætter.

A.13.1. På RPM-baserede systemer

Du kan hente RPM-pakker med Modula-2-oversætteren Mocka på <http://dk.gnuada.org/alt>.

A.13.2. På Debian-baserede systemer

`apt-get update && apt-get install mocka` skulle klare det. Ellers kan du hente kildeteksten på ???.

A.14. Objective Caml

Der findes kun en (men god og fri) implementation af OCaml. Den har både et interaktivt system og en batch-oversætter. Og den genererer både bytekode og maskinkode. Den kan findes på <http://pauillac.inria.fr/ocaml/>.

A.15. Pascal

Når du skal installere en Pascal-oversætter, har du to muligheder: en åben implementation og en lukket.

Free Pascal er en fri Pascal-oversætter. Den kan hentes fra <http://www.freepascal.org/>.

Kylix er lukket, men kan fås gratis hvis man kun vil udgive sine programmer under GPL. Der er en installationsvejledning på <http://www.borland.com/kylix/openedition/>.

A.16. Perl

På de fleste systemer kommer Perl-fortolkeren i en pakke ved navn "perl".

A.17. POSIX Shell

Det bør ikke kræve nogle særlige forberedelser for at komme i gang med at programmere i POSIX Shell på Linux da en af de grundlæggende ideer med Linux netop er at det skal overholde POSIX-standarden. POSIX-kommandofortolkeren er programmet **/bin/sh**.

Første linje i et program skrevet i POSIX Shell bør være:

```
#!/bin/sh
```

Mellemrummet mellem udråbstegnet og skråstregen er ikke obligatorisk, men nogle Unix-varianter forventer at kommandofortolkerprogrammer starter med de fire bytes 35, 33, 32, 47 (svarende til strengen "#! /" kodet i ISO-646) og ikke bare de to bytes 35, 33 (der svarer til strengen "#!" kodet i ISO-646). Så hvis dit program skal vidt omkring, er det værd at bruge udgaven med mellemrum.

A.18. Postscript

ghostscript er en fri Postscript-fortolker der fungerer på praktisk taget alle styresystemer. Den kan både bruges til at vise Postscript-filer på skærmen og som printerfilter der oversætter Postscript-filer til filformater forskellige printere kan forstå.

A.18.1. På RPM-baserede systemer

Der vil typisk følge RPM-pakker med Postscript-fortolkeren Ghostscript med din Linux-distribution. På mit system kommer den som pakkerne **ghostscript**, **ghostscript-fonts** og **gv**. Ellers kan du hente kildeteksten på ???.

A.18.2. På Debian-baserede systemer

apt-get update && apt-get install ghostscript gv skulle klare det. Ellers kan du hente kildeteksten på ???.

A.18.3. Brug af ghostscript

lpr -Ppscats eksempel.ps sender Postscript-programmet **eksempel.ps** til printerens "pscats".

gv eksempel.ps sætter Postscript-fortolkeren til at køre programmet **eksempel.ps** (og vise resultatet i et vindue).

A.19. Python

...

A.20. Smalltalk

For at komme i gang med at programmere i Smalltalk på Linux skal du blot installere et Smalltalk-udviklingsmiljø. Det kan du hente på webstedet <http://www.squeak.org/>.

A.21. SQL

Se i "Linux – Friheden til egen webserver".

A.22. Tcl/Tk

Hvis du gerne vil programmere grafik under Linux, er Tcl/Tk et godt sprog. Det er et forholdsvis nemt sprog at lære. Tcl er et makrosprog, som har fået en søster Tk der er en grafisk del med samme nemme syntaks. Det er et meget velegnet sprog til hurtigt at få en grafisk brugergrænseflade til et (tekstbaseret) program.

Prøv at indtaste følgende program (kald det `editor.tcl`) efter at have installeret Tcl og Tk (**`rpm -i tcl-8.0.4-29.rpm; rpm -i tk-8.0.4-29.i386.rpm`** – eller måske lidt anderledes, alt efter hvilke versioner du netop har fået). Den øverste linje svarer til at du kører Red Hat; SuSE har installeret **wish** i `/usr/X11R6/bin`.

```
#!/usr/bin/wish -f

label .l -text "Filename:"
label .l2 -text "Editor:"

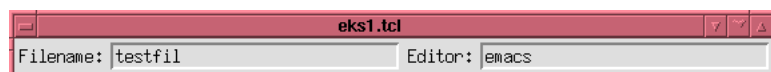
set fname testfil
set editor emacs
button .b -text "OK" -command "exec $editor $fname"
entry .e -relief sunken -width 30 -textvariable fname
entry .e2 -relief sunken -width 30 -textvariable editor
pack .l -side left
pack .e -side left -padx 1m -pady 1m
bind .e <Return> {
    exec $editor $fname
}

pack .l2 -side left
pack .e2 -side left -padx 1m -pady 1m
pack .b -side left -padx 1m -pady 1m
bind .e2 <Return> {
    exec $editor $fname
}
```

Gør programmet kørbart ved at skrive:

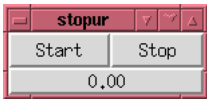
```
hven% chmod +x editor.tcl
```

og kørs det ved at skrive `./editor.tcl`. Ret i tekstfelterne, og se hvad der sker når du trykker return i hvert af tekstfelterne. Programmet er vist på Figur A-1.



Figur A-1. Tcl/Tk programmer `tcl1.tcl`

Et andet Tcl/Tk eksempel der laver et simpelt stopur med start- og stop-funktion. Programmet bliver ikke forklaret, men skal illustrere hvor lidt kode der skal til at lave et program som med andre programmeringssprog ville være meget længere. Programmet er vist på Figur A-2.



Figur A-2. Et stopur i Tcl/Tk

```
#!/usr/bin/wish -f

label .counter -text 0.00 -relief raised -width 20
button .start -text Start -command {
    if $stopped {
        set stopped 0
        tick
    }
}
button .stop -text Stop -command {set stopped 1}
pack .counter -side bottom -fill both
pack .start -side left -fill both -expand yes
pack .stop -side right -fill both -expand yes

set seconds 0
set hundredths 0
set stopped 1

proc tick {} {
    global seconds hundredths stopped
    if $stopped return
    after 50 tick
    set hundredths [expr $hundredths+5]
    if {$hundredths >= 100} {
        set hundredths 0
        set seconds [expr $seconds+1]
    }
    .counter config -text [format "%d.%02d" $seconds $hundredths]
}

bind . <Control-c> {destroy .}
bind . <Control-q> {destroy .}
focus .
```

En variant af stopur programmet er et nyttigt program som viser status på batteriet på en bærbar. I dette eksempel har vi erstattet den første linje, svarende til SuSE 6.1 placering af **wish**, og der kræves at kommandoen **apm** er installeret, dvs. **apmd-2.4-57.rpm** eller lignende.

```
#!/usr/X11R6/bin/wish -f

label .counter -text 0 -relief raised -width 50
button .start -text Start -command {
    tick
}
button .stop -text Stop -command { destroy .}
pack .counter -side left -fill both
pack .stop -side right -fill both

proc tick {} {
    after 1000 tick
    set tot [eval exec apm]
    .counter config -text [format "%s" $tot]
}

tick

bind . <Control-c> {destroy .}
bind . <Control-q> {destroy .}
focus .
```

A.23. TeX

Hvis du skulle have lyst til at programmere i TeX, skal du blot installere din distributions TeX-pakke.

Slutbemærkning:

1. <http://dk.gnuada.org/alt>
2. <http://cvs.linuxbog.dk/program/eksempler/smart.adb>
3. <http://anjuta.sourceforge.net/download.html>
4. <http://www.yabasic.de/>
5. <http://www.yabasic.de/download.htm#binlin>
6. <http://www.yabasic.de/download.htm#sourceunix>
7. <http://www.moria.de/~michael/bas/>
8. <http://tiny-cobol.sourceforge.net/>
9. <http://www.josvisser.nl/opencomal/>
10. <http://SmallEiffel.loria.fr/>
11. <http://SmallEiffel.loria.fr/general/download.html>
12. <http://SmallEiffel.loria.fr/general/download.html>
13. <http://www.cse.ogi.edu/PacSoft/projects/Hugs/pages/downloading.htm#Linux>
14. <http://www.cse.ogi.edu/PacSoft/projects/Hugs/pages/downloading.htm>
15. <http://sunsite.dk>
16. <http://www.blackdown.org>
17. <http://java.sun.com>
18. <http://www.gimp.org/fonts.html>
19. <http://www.gimp.org/fonts>
20. <http://www.mesa3d.org/>
21. <http://java.sun.com>
22. <http://www.vrml.org/>
23. <http://www.dina.kvl.dk/~sestoft/mosml.html>
24. <http://cm.bell-labs.com/cm/cs/what/smlnj/>
25. <http://www.sourcelight.com/MLton/>
26. <http://www.polymml.org/>
27. <http://www.it-c.dk/research/mlkit/>
28. <http://www.smlserver.org/>
29. <http://www.dcs.ed.ac.uk/~mlj/>
30. <http://dk.gnuada.org/alt>
31. <http://pauillac.inria.fr/ocaml/>
32. <http://www.freepascal.org/>
33. <http://www.borland.com/kylix/openedition/>
34. <http://www.squeak.org/>

Appendiks B. Kodebiblioteker

Foreløbigt indeholder dette appendiks lister med biblioteker til forskellige formål. For hvert bibliotek nævnes hvilke sprog det kan bruges fra, samt hvor det kan findes på internettet.

B.1. Grafiske brugergrænseflader

Gtk+ (Ada, C, m.fl.), Swing (Ada, Eiffel, Java) Qt (C++), Tk (Tcl)

B.2. Grafik (2D og 3D)

Java3D (Ada, Eiffel, Java), Mesa (C), Ygl (C, Fortran, Pascal)

B.3. Beregninger

"Jacobs fysikbiblioteker" (Ada), "Jacobs matematikbiblioteker" (Ada)

B.4. Internettet

AdaSockets (Ada), Ada Web Server (Ada), OpenSSL (Ada, C)

Appendiks C. Bøger om programmering

C.1. Generelt om imperativ programmering

C.1.1. Objektorienteret programmering

- Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides: *Design Patterns – Elements of Reusable Object-Oriented Software*, 1995, Addison Wesley, ISBN 0-201-63361-2, 395 sider.

Derudover er der i Ada-boglisten nogle titler om objektorienteret programmering specifikt i Ada.

C.1.2. Bøger om Ada

Der findes mange bøger om Ada-programmering. Vi har udvalgt to fra listen vi henviser til nedenfor:

- David J. Naiditch: *Rendezvous With Ada 95* (second edition), John Wiley & Sons, Inc., ISBN 0-471-01276-9, 598 sider.
En grundig introduktion til Ada 95, men ikke nødvendigvis sagen for folk der ikke har programmeret før.
- John English: *Ada 95: The Craft of Object-Oriented Programming*, oprindeligt udgivet af Prentice Hall, 1997.
En grundig introduktion til objektorienteret programmering i Ada. Den er hovedsageligt rettet mod læsere der ikke allerede har erfaring med programmering.
Bogen kan nu også læses på/hentes ned fra nettet: <http://www.it.bton.ac.uk/staff/je/adacraft/>.
- Michael A. Smith: *Object-Oriented Software in Ada 95*, Thomson Computer Press, ISBN 1-85032-185-X, 460 sider.
Bogen er skrevet som en introduktion for studerende og programmører der ønsker at lære om objektorienteret programmering.
- Ken O. Burtch: *The big online book of Linux Ada Programming*, <http://www.vaxxine.com/pegasoft/homes/book.html>
En masse nyttig information om Ada-programmering til Linux. Og det er ikke uden grund at der står "big" i titlen.

Du kan finde anmeldelser af bøger om Ada på <http://www.seas.gwu.edu/faculty/mfeldman/ada95books.html>.

Hvis du har brug for Ada-referencemanualen, kan du hente den på internettet: <http://www.adaic.com/standards/ada95.html>

SSLUG's nyhedsgruppe sslug.prog⁵ lægger plads til snak om Ada i SSLUG-regi. Og på <http://www.adapower.com/learn/> er der nyttige henvisninger for dem der vil i gang med at programmere i Ada.

C.1.3. Bøger om Bash

- Daniel Gilly and the staff of O'Reilly & Associates, Inc.: *UNIX in a Nutshell*, O'Reilly & Associates, Inc., ISBN 1-56592-001-5.
Glimrende generel introduktion til de almindeligste Unix- og Linux-kommandofortolkere og -kommandoer. Findes også i en senere udgave med titlen "Linux in a Nutshell".
- *Advanced Bash Scripting Guide*, udgivet under Open Publication License, kan fås fra The Linux Documentation Project.

C.1.4. Bøger om Basic

SSLUG's nyhedsgruppe sslug.prog7 lægger plads til snak om Basic i SSLUG-regi.

C.1.5. Bøger om C

- Brian W. Kernighan og Dennis M. Ritchie: *The C Programming Language*, 2nd edition, Prentice Hall 1988, ISBN 0-13-110362-8, 272 sider.

Det klassiske værk om C. Den omtales normalt blot som "K&R". Skrevet af C's opfindere. Der findes også en dansk oversættelse udgivet af Teknisk Forlag.

C.1.6. Bøger om C++

- Rick Decker & Stuart Hirshfield: *The Object Concept – An Introduction to Computer Programming Using C++*, 1995, PWS Publishing Company, ISBN 0-534-20496-1, 454 sider.

God introduktion til C++ (beregnet til det første år på programmeringsuddannelser) der lægger vægt på at introducere objektorientering programmering fra starten af.

- Michael Main & Walter Savitch: *Data Structures and other Objects using C++*, 2001, Addison Wesley, ISBN 0-201-70297-5, 783 sider.

Bogen er en god introduktion til datastrukturer (CS2). 2. udgave er opdateret, så den overholder ANSI/ISO C++ Standard Library classes.

- Scott Meyers: *Effective C++ – 50 Specific Ways to Improve Your Programs and Designs*, 1997, Addison Wesley, ISBN 0-201-92488-9, 256 sider.
- Herbert Schildt: *C/C++ – Programmer's Reference*, 2000, Osborne/McGraw-Hill, ISBN 0-07-212706-6, 393 sider.
- Bruce Eckel: *Thinking in C++*, 1999, ISBN 0-13979809-9, 814 sider.

Bruce Eckel stiller den elektroniske version af sine bøger gratis til rådighed. Du kan hente bogen på <http://64.78.49.204/>.

- Andrew Koenig og Barbare E. Moo: *Accelerated C++: practical programming by example*, Addison-Wesley 2000, ISBN 0-201-70353-X, 336 sider.

En god begynderbog som fra første kapitel bruger højniveaunkonstruktioner fra STL.

- Bjarne Stroustrup: *The C++ Programming Language*, 3rd edition (eller special edition), Addison-Wesley 1997, ISBN 0-201-70073-5 (special edition), 1020 sider.

"Bogen" om C++.

C.1.7. Bøger om COMAL

- Leo Højsholt-Poulsen & Frank Bason: *COMAL for Commodore 64*, 1985, Commodore Data A/S, 316 sider.

Den danske brugsanvisning til COMAL-modulet til Commodore 64'eren er en overordentlig god introduktion til programmering i COMAL. Spørgsmålet er blot om det er muligt at få fat i den i dag.

- Len Lindsay: *COMAL HÅNDBOGEN*, 1985, Teknisk Forlag A/S.

Dette er den formelle definition af COMAL (oversat til dansk).

C.1.8. Bøger om Fortran

C.1.9. Bøger om Haskell

Man kan læse om Haskell på webstedet <http://www.haskell.org/>.

C.1.10. Bøger om Java

- Jacob Nordfalk: *Objektorienteret programmering i Java*, 1. udgave, Forlaget Globe, ISBN ..., # sider.
(vi mangler en uvildig omtale, men Jacob Nordfalk er en dygtig programmør og underviser, så der er grund til at forvente at det er en rigtig god bog)
Det meste af bogen kan læses gratis på <http://javabog.dk/>.
- Peter Sestoft: *Java Precisely*, 2001, 68 sider.
En kort introduktion til Java. En gratis udgave af bogen kan hentes fra <http://www.dina.kvl.dk/~sestoft/javaprecisely/>, og der kommer en trykt udgave i starten af 2002.
- Bruce Eckel: *Thinking in Java*, 2000, Prentice-Hall, ISBN 0-13027363-5, 1128 sider.
Bruce Eckel stiller den elektroniske version af sine bøger gratis til rådighed. Du kan hente bogen på <http://64.78.49.204/>.
- Herbert Schildt & Joe O'Neil: *Java 2 – Programmer's Reference*, 2000, ISBN 0-07-212354-0, 447 sider.
SSLUG's nyhedsgruppe sslug.prog¹³ lægger plads til snak om Java i SSLUG-regi.

C.1.11. Bøger om ML

Se Moscow ML's hjemmeside¹⁴ for referencer både til udgivne bøger og til frit tilgængeligt materiale (alt sammen på engelsk). Følgende kan særlig anbefales:

- Michael R. Hansen & Hans Rischel: *Introduction to Programming using SML*, Addison-Wesley, 1999, ISBN 0-201-39820-6.
Hansen og Rischel er fra Informatik og Matematisk Modellering, Danmarks Tekniske Universitet.
- Stephen Gilmore: *Programming in Standard ML'97: A Tutorial Introduction*, 1997, University of Edinburgh Report ECS-LFCS-97-364.
Fås elektronisk som <http://www.dcs.ed.ac.uk/home/stg/NOTES/>.
- Mads Tofte: *Tips for Computer Scientists on Standard ML*, <http://www.diku.dk/users/tofte/publ/tips.ps.gz>.
Mads Tofte er fra IT-højskolen i København.

SSLUG's nyhedsgruppe sslug.prog¹⁷ lægger plads til snak om ML i SSLUG-regi.

C.1.12. Bøger om Objective Caml (OCaml)

- Emmanuel Chailloux, Pascal Manoury & Bruno Pagano: *Développement d'applications avec Objective Caml*, ISBN 2-84177-121-0, 686 sider+cd-rom.
En fornuftig introduktion til Objective Caml, hvis man stadig kan huske en smule fransk fra gymnasiet.

C.1.13. Bøger om Perl

Følgende to bøger betragtes som standardværkerne om Perl. Den ene til at komme i gang, den anden for viderekomne:

- Randal L. Schwartz & Tom Christiansen: *Learning Perl (2nd Edition)*, O'Reilly & Associates, Inc., ISBN 1-56592-284-0, 271 sider.
- Larry Wall, Tom Christiansen & Randal L. Schwartz: *Programming Perl (2nd Edition)*, O'Reilly & Associates, Inc., ISBN 1-56592-149-6, 645 sider.

SSLUG's nyhedsgruppe `sslug.perl`¹⁸ lægger plads til snak om Perl-programmering i SSLUG-regi.

C.1.14. Bøger om POSIX Shell

- Daniel Gilly and the staff of O'Reilly & Associates, Inc.: *UNIX in a Nutshell*, O'Reilly & Associates, Inc., ISBN 1-56592-001-5.

Glimrende generel introduktion til de almindeligste Unix- og Linux-kommandofortolkere og -kommandoer. Findes også i en senere udgave med titlen "Linux in a Nutshell". Afsnittet om Bash svarer grundlæggende til POSIX Shell.

C.1.15. Bøger om Postscript

- Adobe Systems Incorporated: *PostScript Language Reference Manual*, 1985, Addison-Wesley Publishing Company, Inc., ISBN 0-201-10174-2, 321 sider.

Dette er den formelle definition af Postscript. Et overordentligt nyttigt værk at have ved hånden når man programmerer i Postscript (eller skriver programmer der gør det).

- Peter Weingartner: *A First Guide to PostScript*, <http://www.gkss.de/W3/PS/postscript.html>.

En introduktion til programmering i Postscript.

C.1.16. Bøger om Ruby

Udvalget af bøger om Ruby er ikke så stort. Vi vil dog nævne to bøger, dels en decideret lærebog og dels en referencebog:

- David Thomas og Andrew Hunt: *Programming Ruby*, Addison-Wesley, ISBN: 0201710897

En lærebog

Bogen kan nu også læses på/hentes ned fra nettet: <http://www.rubycentral.com/book/index.html>.

- Yukihiro Matsumoto: *Ruby in a Nutshell*, O'Reilly & Associates Inc., ISBN: 0-596-00214-9

En referenceguide skrevet af hovedmanden bag ruby og oversat af David L. Reynolds.

C.1.17. Bøger om Scheme

Du kan finde information om Scheme på webstedet <http://www.schemers.org/>.

C.1.18. Bøger om Tcl/Tk

- Brent B. Welch: *Practical Programming in Tcl and Tk*, Prentice Hall, ISBN 0-13-616830-2, 630 sider + cd-rom.
En god og omfattende introduktion til Tcl/Tk.

C.1.19. Bøger om TeX

En bog om TeX der kan være god at have er:

- Paul W. Abrahams et al.: *TeX for the Impatient* Addison-Wesley, ISBN 0-201-51375-7, 360 sider.

Den ultimative reference er dog:

- Donald E. Knuth: *The TeXbook* Addison-Wesley, ISBN 0-201-13448-9, 496 sider.

Hvis du har spørgsmål om TeX, er den bedste af SSLUG's nyhedsgrupper sslug.dok²².

C.2. Generelt om deklarativ programmering

C.2.1. Bøger om Prolog

- J. McAllister: *Artificial Intelligence and PROLOG on Microcomputers*, 1987, Edward Arnold (Publishers) Ltd., ISBN 0-7131-3611-1, 168 sider.

C.3. Sjove tekster om programmering

Programmer til at udskrive teksten til sangen "99 bottles of beer on the wall" i alle mulige og umulige programmeringssprog: <http://internet.ls-la.net/mirrors/99bottles/>

Der hører også med en lang række eksempler til denne bog. De kan findes på adressen: <http://www.linuxbog.dk/program/eksempler/>

Slutbemærkning:

1. <http://www.it.bton.ac.uk/staff/je/adacraft/>
2. <http://www.vaxxine.com/pegasoft/homes/book.html>
3. <http://www.seas.gwu.edu/faculty/mfeldman/ada95books.html>
4. <http://www.adaic.com/standards/ada95.html>
5. news://news.sslug.dk/sslug.prog
6. <http://www.adapower.com/learn/>
7. news://news.sslug.dk/sslug.prog
8. <http://64.78.49.204/>

9. <http://www.haskell.org/>
10. <http://javabog.dk/>
11. <http://www.dina.kvl.dk/~sestoft/javaprecisely/>
12. <http://64.78.49.204/>
13. <news://news.sslug.dk/sslug.prog>
14. <http://www.dina.kvl.dk/~sestoft/mosml.html>
15. <http://www.dcs.ed.ac.uk/home/stg/NOTES/>
16. <http://www.diku.dk/users/tofte/publ/tips.ps.gz>
17. <news://news.sslug.dk/sslug.prog>
18. <news://news.sslug.dk/sslug.perl>
19. <http://www.gkss.de/W3/PS/postscript.html>
20. <http://www.rubycentral.com/book/index.html>
21. <http://www.schemers.org/>
22. <news://news.sslug.dk/sslug.dok>
23. <http://internet.ls-la.net/mirrors/99bottles/>
24. <http://www.linuxbog.dk/program/eksempler/>

Appendiks D. Revisionshistorie for bogen

Igennem tiden har bogen "Linux – friheden til programmere" udviklet sig meget. Vi frigiver ofte nye versioner, når der er kommet en del rettelser ind, eller nye afsnit er blevet skrevet. Kommentarer, ris og ros, og specielt fejl og mangler bedes sendt til linuxbog@sslug.dk¹, men er du medlem af SSLUG så skriv til sslug-bog@sslug.dk². Her er en liste over, hvad der er ændret i bogen.

Denne bog blev skabt ved en opsplitning af version 3.9 af "Linux - friheden til at vælge". Arbejdet på bogen koordineres via SSLUG's postliste sslug-bog@sslug.dk³. Du kan tilmelde dig postlisten på <http://www.sslug.dk/tilmeld#bog> (bemærk at du samtidig bliver medlem af SSLUG hvis du ikke allerede er det).

- Version 2.5 - 25. januar 2004: Jacob Sparre Andersen: »99 bottles of beer on the wall« i Ada. Lidt om kommandolinjetilvalg til GCC, når man oversætter Ada-programmer. Retter sprog.
- Version 2.4 - 7. oktober 2003: Jacob Sparre Andersen har rettet i brugen af `&linuxbogurl`. Jacob Sparre Andersen har sat Peter Makhholm på listen over andre bidragydere. Peter Makhholm samler alle afsnit der basalt set handler om bourne shell og skriver en smule mere. Peter Makhholm skriver om SML, føjer `smart.sml` til eksemplerne og begynder på at skrive om Haskell Mads Sejersens har rettet nogle syntaksfejl i SMTP-eksemplet i Ruby-afsnittet. Christian Tredal finder død henvisning. Jacob Sparre Andersen retter døde henvisninger og sproglige fejl.
- Version 2.3 - 6. april 2003: Jacob Sparre Andersen - Byrial Jensen har påpeget mangler i teksterne om Ada, Perl, Python, Pascal, Java og POSIX Shell. Johan Munk har påpeget mangler i teksterne om Scheme og **a2ps**. Byrial Jensen - Flere bøger om Bash, C og C++. Byrial Jensen - Korrekturlæsning af *hele* bogen. David List - Fanget fejl i 'patch'-kommando. Jacob Sparre Andersen - Vi er nu gået over til (forhåbentlig konsekvent) at bruge brugernavnet "tyge" og maskinnavnene "hven.oresund.dk", "saltholm.oresund.dk" og "peberholm.oresund.dk" i eksemplerne. Peter Makhholm har skrevet om Ruby.
- Version 2.3 - 1. december 2002: Torben H. Nielsen - brug af andre sprog med autoconf. Jacob Sparre Andersen - Tilføjet eksempel på oversættelse af et Ada-program. Sørget for at introafsnittet om Ada henviser til appendiksene med installationsvejledning og litteraturliste. Tilføjet Red Hat 8.0-pakkenavne i installationsafsnittet for Ada og bemærket at ALT's og Red Hat 8.0's Ada-pakker er inkompatible. Søren Enemark - Rettet fejl i CVS-afsnittene. Peter Toft retter links om CVS til.
- Version 2.2 - 1. september 2002: Torben H. Nielsen - C++ bruger `AC_PROG_CXX` i autoconf. Hans Schou - mere overskuelig listning af autoconfeksempler. Jacob Sparre Andersen - Sørget for at bogen kan oversættes både som SGML Docbook og som XML Docbook (håber han). Harmoniseret brugen af **tar**. Rettet sproglige småfejl.
- Version 2.1 - 14. juni 2002: Jacob Sparre Andersen - stikord, bedre henvisning til eksemplerne, prøvet at forbedre sammenhængen mellem de forskellige kapitler, rettet enkelte sproglige fejl. Anna Jonna Armannsdottir retter afsnit om CVS. Henrik Christian Grove har skrevet om TeX.
- Version 2.0 - 10. marts 2002: Ny licens for bogen - Åben dokumentlicens. Jesper Harder retter sprog og detaljer mht. Fortran. Rasmus Erik Voel Jensen skriver om Haskell, Lisp, Prolog og Scheme. Jacob Sparre Andersen - stikord, linjebrud i eksempler, skrevet om at bruge CVS, når SSH kører på en alternativ port, skrevet om at bruge CVS fra Emacs, udkommenteret effektivt tomme afsnit, tilføjer en bog om Objective Caml. Peter Toft - retter en stribe SGML-fejl, tilføjer stikord om CVS, og tilføjer mere om merge af grene i CVS. Tilføjer nyt eksempel om at slette kataloger i CVS. Michael Lerskov Munk Nielsen retter en dum fejl i CVS-afsnittet.
- Version 2.0alpha2 - 29. december 2001: Jacob Sparre Andersen: Indsat Jesper Laisens tilføjelser til litteraturlisterne for C++ og Java. Strøget minusserne i versionsnummeret af hensyn til RPM. RCS findes til Windows skriver Jan Eggert Kofoed. Indsat en del materiale som Peter Sestoft m.fl. har skrevet om Standard ML, Objective Caml og C# (c-havelåge). Opdateret CVS-afsnittet, så der ikke er nogle direkte henvisninger til systemvariablen "CVSROOT" (det forvirrede læserne). Jan Eggert Kofoed har bidraget med forslag og korrekturlæsning af CVS-afsnittet. Prøvet at gøre forordet mere pædagogisk. Oprettet et (forholdsvis tomt) afsnit om Emacs. Oprettet et appendiks med nyttige kodebiblioteker. Skrevet en indledning til "Programmørens værktøjskasse". Ryddet op i indledningen i CVS-afsnittet. Indsat henvisninger til bøger om Ada fra Björn Lundin. Trykfejl fra Svend Erik Venstrup.
- Version 2.0-alpha-1 - 21. oktober 2001: Jacob Sparre Andersen: Tilføjet en minimal omtale af Gettext (og KBabel). Skrevet et par ord om variable og konstanter. Indsat (og tilrettet) Mads Sejersens introduktion til Anjuta. Skrevet om betingede strukturer. Skrevet lidt om procedurer og funktioner. Oversat et af eksemplerne på betingede strukturer til Ada. Ryddet op i afsnittene om forgreninger i CVS. Henviser til eksempelsamlingen i introduktionen til kapitlet om imperativ programmering. Brygget videre på introduktionen til imperativ programmering. Føjte "Ada 95: The Craft of Object-Oriented Programming" til boglisten. Skrevet fire ekstra eksempler på brug af CVS. Indsat Jacob Nordfalks tilføjelser og rettelser til omtalen af Java. Henvisning til "99 bottles of beer on the wall"-siden. Indsat Jesper Laisens litteraturliste for OOP og C++.

- Version 2.0-alpha-0 - 11. august 2001: Jacob Sparre Andersen: Tilføjet lidt om Smalltalk og GVD. Udvidet ordlisten. Opdateret og udvidet abstract og forord. Rettet diverse sproglige fejl. Skrevet generelt om pakkeinstallation. Sat numre på ISO-standarder ind. Ryddet lidt op. Hans Schou: autoconf og automake. Ole Tange: road map diagrammer til autoconf. Thue Janus Kristensen: aclocal/autoconf.
- Version 1.9 - 9. juli 2001: Erik Søb Sørensen: sproglige rettelser i Postscript-afsnittet. Peter Toft: Tilføjet en henvisning til nye bøger. Jacob Sparre Andersen: Oprettet en udviklingsgren for at omstrukturere bogen. Tilføjet en del sprog og værktøjer (foreløbig uden noget videre indhold om dem). Gået i gang med at skrive eksempler til CVS-afsnittet. Erik Søb Sørensen: URL til en bog om Postscript samt rettelsesforslag til omtalen af Postscript. Peter Toft: Nye afsnit om grene (branches) i CVS.
- Version 1.8 - 24. maj 2001: Peter Toft har tilføjet nyt afsnit om hvordan man får cvs commit log-beskeder per e-post til at fungere. Stor tak til Lars Kirkeskov Pedersen. Kim Schulz har rettet en stribe fejl. Erik Søb Sørensen har rettet en enkelt fejl.
- Version 1.7 - 15. april 2001: Jacob Sparre Andersen har rettet nogle formuleringer og stavfejl og rettet lidt op på SGML-koden. Der er også blevet føjet lidt til introduktionen til C- og C++-afsnittet. Rasmus Ory Nielsen har rapporteret en tastefejl i et URL. Carsten Svaneborg har lavet en større omskrivning/udvidelse af C- og C++-afsnittet. Blandt andet med omtale af Electric Fence og Glade. Christian Mikkelsen har gennemlæst og rettet hele C- og C++-afsnittet.
- Version 1.6 - 12. marts 2001: Erik Søb Sørensen retter et par fejl. Henrik Christian Grove fandt en dum trykfejl i oversigt over bøger. Jacob Sparre Andersen har rettet nogle formuleringer og føjet lidt til stikordsregistret. Peter Toft rettede lidt SGML-fejl.
- Version 1.5 - 4. februar 2001: Jacob Sparre Andersen har skrevet om Ada, er nu medredaktør på programmeringsbogen, har rettet diverse sproglige fejl og har pillet lidt ved Java-afsnittet. Erik Søb Sørensen har lavet en række gode tilføjelser og rettelser i shell-afsnittet. Ny bog om Docbook nævnt i serien. Tommy Mogensen og Erik Søb Sørensen fandt hver en sprogbøf.
- Version 1.4 - 29. december 2000: En masse 'hello' og 'udskriv' eksempler af Zaphod Beeblebrox (Peter Stubbe) <stubbe@bitnisse.dk>. Cobol-, PostgreSQL og Python eksempler af Hans Schou. udskriv.sh shell-eksempel af David Axmark <david@mysql.com>. Donald Axel fandt en grim fejl i det første C++-program - at anvende echo med gæseøjne til at skabe kommando-filer. Her vil dollar-variable blive ekspanderet. Peter Toft har skrevet mere om diff, patch og emacs - dette er så rettet af Jørgen Ramskov. Anders Bo Rasmussen har skrevet om xxdiff. Link til den nye bog "Linux - friheden til at programmere i C". Nyt afsnit om CVS (baseret på en tidligere artikel af Ole Vilmann og Peter Toft om CVS). Frank har givet tilføjelser til CVS-afsnittet og Jørgen Ramskov har rettet CVS-teksten til.
- Version 1.3 - 10. november 2000: Program eksempler er med.
- Version 1.2 - 26. september 2000: Sproglige rettelser fra Frank Damgaard. Morten Liljeberg havde et par rettelser om anvendelse af root.
- Version 1.1 - 30. august 2000: Allan *Swoop* Andersen har lavet sproglige rettelser og et par nye gode referencer til Java-kapitlet. Mindre rettelser fra Bugge T. Jensen.
- Version 1.0 - 30. juli 2000: Poul Petersen har fundet en stribe fejl. Mindre rettelser fra Mikkel Mastrup.

Slutbemærkning:

1. <mailto:linuxbog@sslug.dk>
2. <mailto:sslug-bog@sslug.dk>
3. <mailto:sslug-bog@sslug.dk>
4. <http://www.sslug.dk/tilmeld#bog>

Ordliste

Deklarative sprog

En gruppe sprog bestående af funktionsorienterede sprog og logikorienterede sprog, hvor man beskriver forholdet mellem variable istedet for at opstille en række beregningsskridt computeren skal gennemføre. Et eksempel på logikprogrammering er prolog og eksempler på funktionorienteret programmering er Scheme og Haskell. (Sammenlign med *imperative sprog*).

fri standard

En fri standard overholder følgende krav:

1. Ingen restriktion på anvendelse.
2. Ingen restriktion på implementation.
3. Bevarelse af standardens integritet.
4. Fri tilgængelighed til standarden.

ad 1) En standard må ikke sætte begrænsninger på hvor standarden kan finde anvendelse, og heller ingen begrænsninger på hvad man må gøre ud over hvad standarden foreskriver. (cirka open source-definitionen punkt 5 og 6)

ad 2) Der må ikke være restriktioner på hvem der må implementere standarden, hvordan den implementeres, eller hvordan implementationen licensieres. Dette hindrer blandt andet at åbne standarder anvender patenterede algoritmer. (cirka open source-definitionen punkt 5 og 6, giver mulighed for at anvende OSD-licenser på implementationen)

ad 3) En standard må kræve at der tydeligt skal gøres opmærksom på hvor implementationen går ud over hvad standarden implementerer. (cirka open source-definitionens punkt 4)

ad 4) Det skal være tilladt af videredistribuere hele eller dele af standarden. (cirka open source-definitionens punkt 1)

Se i øvrigt Peter Makholms skriblerier om åbne og frie standarder (<http://peter.makholm.net/skriblerier/openstandards>) for en uddybning af denne definition.

frie programmer

Programmer der distribueres efter "Open Source"-reglerne². Kort fortalt går det ud på at hvis du har fået et *frit program*, så:

- Må du frit sælge eller videregive programmet.
- Har du ret til at få kildeteksten til programmet.
- Har du ret til at videreudvikle programmet.

GNU Compiler Collection (GCC)

GNU-projektets oversættersystem. Det kan oversætte adskillige programmeringssprog til nok endnu flere forskellige platforme. Fidusen ved GCC er at oversættersystemet er delt i to dele; "forenden", hvor der findes en til hvert programmeringssprog, og "bagenden", hvor der findes en til hver platform. Og kommunikationen mellem for- og bagende sker på en form der er uafhængig af både programmeringssprog og platform.

hoballokering (eng: heap allocation)

Når allokering af data foregår fra en på forhånd afgrænset del af hukommelsen. Hvor i hukommelsen data bliver langt har intet med programmets yderligere struktur at gøre.

I C får bruger man hoballokeret hukommelse når man bruger funktionen **malloc**. Se også *stakallokering*.

Imperative sprog

Sprog hvor man beskriver en række trin der skal udføres for at foretage en beregning. Eksempler på imperative sprog er Ada og C

PID

Forkortelse af det engelske udtryk "process identification". Se *procesnummer*.

procesnummer

Et tal som styresystemet bruger til at identificere en proces, dvs. et startet program. Hvis du starter flere eksemplarer af det samme program, vil hvert eksemplar have sit procesnummer. Så længe en proces bruger ressourcer, for eksempel hukommelse, på systemet, vil man kunne referere til den ved dens procesnummer, uanset om den egentlig er aktiv.

stak (datastruktur)

En måde at organisere data på der svarer til en stak papirer på et skrivebord. Du kan kun se det øverste objekt på stakken, og for at få adgang til et objekt længere nede i stakken, må du først fjerne alle dem der ligger ovenpå.

stakallokering

Når variable bliver allokeret i en stak. Man kan godt tilgå variable der er allokeret længere nede i stakken, men variable bliver deallokeret i streng modsat rækkefølge end de bliver allokeret.

Stakallokering bliver ofte brugt til lokale variable i funktioner. Se også *hoballokering*

åben standard

En åben standard overholder de første tre krav til en *fri standard*, men behøver ikke nødvendigvis at kunne distribueres frit.

Slutbemærkning:

1. <http://peter.makholm.net/skriblerier/openstandards>
2. <http://www.opensource.dk/docs/definition.html>

Stikordsregister

Symboler

2D-grafik, 61
3D-grafik, 61
3D-grafik i Java
 OpenGL, 55
ÅDL, ii

A

a2ps, 49
Ada, 3
 bøger, 63
 imperativ programmering, 17
 installation, 52
Anjuta, 47
 installation, 53
arkitekturaafhængighed, 3
autoconf, 40
 andre sprog end C, 47
 installation af andre filer, 45
 man filer, 44
 underkataloger, 44
automake, 40

B

Bash
 bøger, 63
 imperativ programmering, 17
Basic, 4
 imperativ programmering, 17
 installation, 53
betingede strukturer, 17
biblioteker, 61
Bourne shell, 4
brugergrænsefladebygger, 35
brugergrænseflader
 grafiske, 61
bufferoverløb
 find, 38
bøger, 63
 om Ada, 63
 om Bash, 63
 om C++, 64
 om Java, 65
 om ML, 65
 om Perl, 66
 om POSIX Shell, 66
 om Postscript, 66
 om Ruby, 66
 om Tcl/Tk, 67
 om TeX, 67

C

C, 5
 bøger, 64
 imperativ programmering, 17
 kalde fra Ada, 3
 Makefile til C-filer, 45
C#, 6
C++, 6
 bøger, 64
 imperativ programmering, 17
 kalde fra Ada, 3
Cobol, 6
 imperativ programmering, 17
 installation af, 53
 kalde fra Ada, 3
Comal, 6
 bøger, 64
 imperativ programmering, 17
 installation, 53
Concurrent versioning system, 21
configure
 autoconf, 40
 installation i andet katalog, 45
copyright, ii
Csh, 6
 imperativ programmering, 17
CVS, 21
 add, 24
 checkout, 23, 26
 commit, 23
 CVSROOT, 27
 diff, 25
 fjern en fil, 24
 fjerne katalog, 24
 flette grene sammen igen, 28
 Forskelle mellem forskellige udgaver af en fil, 25
 gennem en dørvogter, 26
 import, 26
 init, 26
 kom i gang med at bruge et arkiv på nettet, 23
 kom i gang med at bruge et arkiv på nettet over SSH, 23
 kom i gang med at bruge et lokalt arkiv, 22
 log, 25
 læg en rettelse tilbage i det fælles arkiv, 23
 med SSH på en alternativ port, 26
 og Emacs, 24
 opdatér personlig kopi, 23
 opsætning af, 26
 remove, 24
 rtag, 28
 status, 25
 tag, 25
 tilføj en ny fil, 24
 update, 23
 update -j, 28
 update -r, 25

D

- DDD, 34
- deklarativ programmering
 - oversigt, 19
- deklarative sprog
 - Prolog, 13
- diff, 29
 - CVS, 25
- dynamiske kald, 18
- dørvogter
 - CVS gennem, 26

E

- e-post, 61
- Eiffel, 7
 - installation, 53
- eksempler
 - "99 bottles of beer on the wall", 67
 - "Hej Verden" og andet godt, 67
- Electric Fence, 38
- emacs, 49
 - forskellige udgaver af filer, 30
 - og CVS, 24
- Euphoria, 7

F

- fejlfinding
 - med GNU Visual Debugger, 35
- flowdiagrammer
 - med GRASP, 49
- flytbarhed, 3
- forberedelse af oversættelse
 - med gettext, 37
- forfatterne, ii
- forretningssystemer, 6
- forskelle mellem filer, finde, 29
- Forskelle mellem forskellige udgaver af en fil i et CVS-arkiv, 25
- Forth, 7
 - installation af, 53
- fortolkede sprog
 - Basic, 4
 - Bourne shell, 4
 - Comal, 6
 - Csh, 6
 - Haskell, 7
 - Lisp, 9
 - Octave, 12
 - Postscript, 12
 - Ruby, 13
 - Scheme, 15
 - Smalltalk, 15
 - SQL, 16
 - Tcl/Tk, 16
 - TeX, 16
- Fortran, 7
 - bøger, 65
 - imperativ programmering, 17
 - installation af, 54
 - kalde fra Ada, 3

- funktioner, 17
- funktionsorienterede sprog
 - Objective Caml, 11
 - Scheme, 15
- funktionsprogrammeringssprog
 - ML, 9

G

- GDB, 34
- gettext
 - forberedelse af oversættelse med, 37
- Glade, 35
 - installation, 54
- GNAT, 3
- GNU Ada, 3
- GNU Visual Debugger, 35
- grafik
 - 2D, 61
 - 3D, 61
- grafisk udviklingsmiljø
 - Anjuta, 47
- grafiske
 - brugergrænseflader, 61
- GRASP, 49
- GVD, 35

H

- Haskell, 7
 - bøger, 65
 - installation, 54
- Hello, world!
 - PostgreSQL - PLpgSQL, 16
- historie
 - bogens, ii
- HTTP, 61
- hvis-så, 17

I

- imperativ programmering
 - oversigt, 17
- imperative sprog
 - Ada, 3
 - Basic, 4
 - Bourne shell, 4
 - C, 5
 - C#, 6
 - C++, 6
 - Cobol, 6
 - Comal, 6
 - Csh, 6
 - Eiffel, 7
 - Euphoria, 7
 - Forth, 7
 - Fortran, 7
 - Java, 8
 - Lisp, 9
 - Modula-2, 11
 - Modula-3, 11
 - Oberon, 11

- Objective-C, 11
- Octave, 12
- Pascal, 12
- Perl, 12
- Postscript, 12
- Python, 13
- Ruby, 13
- Scheme, 15
- Smalltalk, 15
- Tcl/Tk, 16
- TeX, 16
- installationsvejledning, 51
- ISO-standard
 - Ada, 3
 - C, 5
 - C++, 6
 - Cobol, 6
 - Fortran, 7
- ISO/IEC
 - 14882, C++, 6
 - 1539-1:1997, Fortran, 7
 - 8652:1995, Ada, 3
- iteration
 - i SML, 11

J

- Java, 8
 - bøger, 65
 - imperativ programmering, 17
 - installation af, 54
 - kalde fra Ada, 3
- Java3D
 - 3D-grafik, 55

K

- KDevelop, 48
- kodebiblioteker, 61
- konstanter, 17

L

- lapper, 30
- Lisp, 9
- logisk parallelisering, 18
- løkker, 18
 - i SML, 11

M

- make, 39
 - autoconf, 40
- Makefile, 39
 - autoconf, 40
- man-filer
 - eksempel på indhold, 44
- matrixregning, 61
- meget store tal, 61
- ML, 9
 - bøger, 65
 - installation, 56

- Modula-2, 11
 - installation, 56
- Modula-3, 11

O

- Oberon, 11
- Objective Caml, 11
- Objective Caml (OCaml)
 - bøger om, 65
 - installation, 56
- Objective-C, 11
- objektorienterede sprog
 - Ada, 3
 - C++, 6
 - Eiffel, 7
 - Java, 8
 - Objective-C, 11
 - Pascal, 12
 - Ruby, 13
 - Smalltalk, 15
- objektorientering, 18
- OCaml (Objective Caml), 11
- Octave, 12
- OpenGL
 - Java3D, 55
- ophavsret, ii
- opsætning af
 - CVS, 26
- oversatte sprog
 - Ada, 3
 - Basic, 4
 - C, 5
 - C#, 6
 - C++, 6
 - Cobol, 6
 - Comal, 6
 - Eiffel, 7
 - Fortran, 7
 - Haskell, 7
 - Java, 8
 - ML, 9
 - Modula-2, 11
 - Modula-3, 11
 - Oberon, 11
 - Objective Caml, 11
 - Objective-C, 11
 - Pascal, 12
 - Prolog, 13
 - Scheme, 15
- oversættelse
 - med gettext, forberedelse af, 37
- oversættelsesfiler, 39

P

- parallelisering
 - logisk, 18
- parallelprogrammering
 - Ada, 3
- Pascal, 12
 - imperativ programmering, 17
 - installation, 56
- patches, 30
- Perl, 12
 - bøger, 66
 - imperativ programmering, 17
 - installation af, 57
- POSIX Shell
 - bøger, 66
 - installation, 57
- Postscript, 12
 - bøger, 66
 - installation af, 57
- procedurer, 17
- Programmering under Linux, i
- Prolog, 13
 - bøger, 67
 - deklarativ programmering, 19
- Python, 13
 - imperativ programmering, 17
 - installation af, 57

R

- Revisionshistorie, 69
- revisionskontrollsystemer
 - CVS, 21
- Ruby, 13
 - bøger, 66

S

- Scheme, 15
 - bøger, 67
- Shellscript
 - bash, 4
 - ksh, 4
 - zsh, 4
- SI-enheder, 61
- Smalltalk, 15
 - installation, 57
- sprog
 - oversigt, 3
- SQL, 16
 - imperativ programmering, 17
 - installation af, 58
- SSH på en alternativ port
 - CVS gennem, 26
- stakorienterede sprog
 - Euphoria, 7
 - Forth, 7
 - Postscript, 12
- syntaksfremhævning
 - med a2ps, 49
 - med GRASP, 49
- systemprogrammering, 3

T

- tal
 - meget store, 61
- Tcl/Tk, 16
 - bøger, 67
 - imperativ programmering, 17
 - installation af, 58
- tekstredigering
 - med emacs, 49
 - med vi, 49
- TeX, 16
 - bøger, 67
 - installation, 60
- tildelinger, 17
- typer, 17

U

- underrountiner, 17

V

- variable, 17
- vektorregning, 61
- versionsstyring
 - med CVS, 21
- vi, 49
- Visuelt programmeringsmiljø
 - debug, 34
 - KDevelop, 48

W

- webserver, 61

Z

- Zsh
 - imperativ programmering, 17