

Programmering i Pascal

Kurt Nørmark ©

Afdeling for Datalogi
Aalborg Universitet

1997

1. Maskiner, programmering og Internettet

Datalogikurset: Form og indhold.

Maskiner, operativsystem og DOS

Programmering og Turbo Pascal

Internet og Netscape

Kursusoversigt.

Formål:

- At give en grundlæggende introduktion til datamaskiners og programsystemers opbygning,
- at opøve en grundlæggende programmeringsfærdighed,
- at give en forståelse af datalogiske grundbegreber.

Indhold:

- Introduktion til maskine, programmering og Internettet.
- Introduktion til regneark.
- Basal programmering
- Abstraktion med procedurer og funktioner.
- Kontrolstrukturer.
- Procedurer og funktioner med parametre.
- Datatyper og filer.
- Datastrukturer: Arrays og records.
- Dataabstraktion.
- Rekursion.

Evaluering:

- Mundtlig eksamen på grundlag af et miniprojekt med karakter efter 13-skalaen.

Informationsteknologi: En fri studieaktivitet.

Formål:

- At orientere om og give færdighed i væsentlige nyere informationsteknologier på tværs af forskellige anvendelsesområder.
- At orientere om problemløsnings- og informationssøgningsteknikker knyttet til disse teknologier.

Indhold:

- Netværk & Internet I: World Wide Web og HTML (8/10/97)
- Netværk & Internet II: Design og World Wide Web (15/10/97)
- Problemløsningssteknikker: Diagrammering (22/10/97)
- Databaser: Introduktion til relationsdatabaser (29/10/97)
- Datastrukturer og informationssøgning (5/11/97)

Omfang: 1 modul.

Evaluering:

- Aktiv deltagelse, herunder deltagelse i opgaveregning.

Forløbet af et minimodul.

Forløb af minimoduler	Formiddage	Eftermiddage
Forelæsning	8.15 - 10.00	12.30 - 14.15
Øvelser	10.15- 12.00	14.30 - 16.15

Øvelser: Opgaveløsning med kursusholder og hjælpelærere i grupperum.

Repetition: En forelæsning starter med en kort repetition af stoffet fra forrige lektion.

Kursusforudsætninger.

- Ingen forudsætninger i datalogi.
- Almene forudsætninger fra gymnasiet.
- I virkeligheden har studerende på den Teknisk Naturvidenskabelige Basisuddannelse meget blandede forudsætninger:
 - Datalogi i gymnasiet.
 - PC-freak.
 - Lejlighedsvis PC-bruger.
 - Har aldrig brugt en computer før.
 - Ved intet om computere.

Motivation.

Typiske anvendelser af computere

- Fleksibel skrivemaskine.
- Kraftig regnemaskine.
- Data administrationsmaskine (EDB-maskine).
- Kommunikationsmedium.

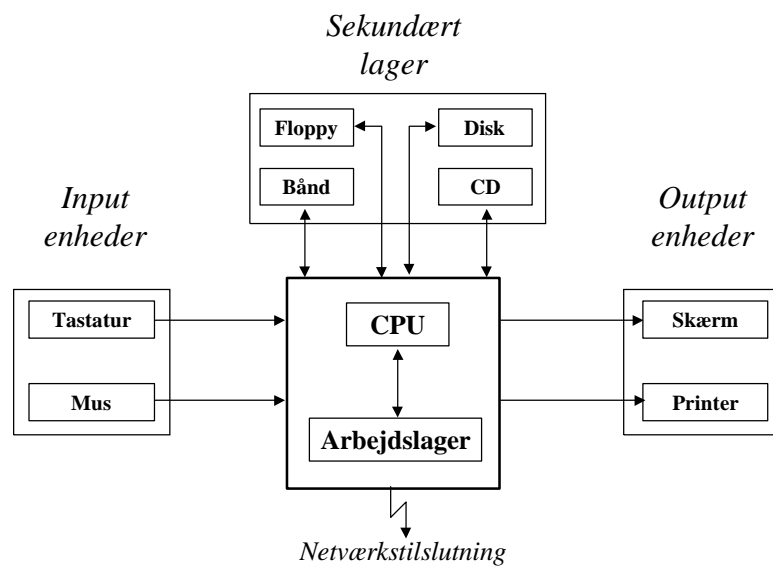
Observation

- Programmérbarhed er den unikke egenskab ved computere, som adskiller disse fra næsten alle andre apparater.

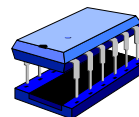
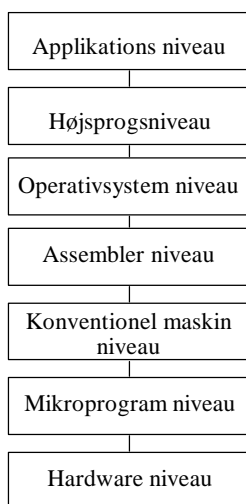
Motivation

- Studerende i teknik eller naturvidenskab har brug for computere til *problemløsning*.
- *Programmeringsforståelse* er en vigtig forudsætning for generel problemløsning på computere.

Maskinens opbygning.



Den lagdelte maskine.

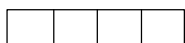


Operativsystemet.

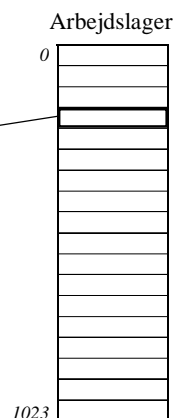
- *Operativsystemet* er det niveau i maskinen, som administrerer maskinens forskellige ressourcer.
 - Eksempler på ressourcer:
 - Harddisken og floppydisks.
 - Filer og kataloger.
 - Printer og skærmen.
 - Tastaturet og musen.
 - Eksempler på operativsystemer:
 - DOS og UNIX
- Et operativsystem viser sig gennem et *kommandosprog*, som tillader os at operere på maskinens ressourcer.
- I moderne systemer er operativsystemets rolle overtaget af *vinduessystemet*.
 - Eksempler på vinduessystemer:
 - Windows95, Windows NT og OS2

Arbejdslageret.

En celle i maskinens arbejdslager



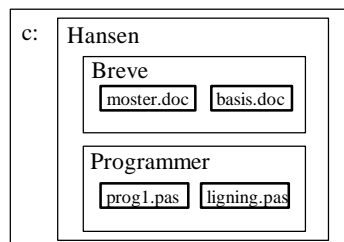
- En celle i arbejdslageret består af typisk 4 bytes.
- En *byte* består af 8 bits.
 - En byte kan indeholde ét tegn (bogstav, ciffer mv.)
- En *bit* indholder enten 0 eller 1.
 - En bit er den mindste lagerenhed, som maskinen kan håndtere.



Betydningen af lagerindholdet afhænger af programmernes fortolkning

Det sekundære lager.

- Det sekundære lager struktureres i
 - *Filer* (files)
 - Den enhed, hvori *programmer* og *data* opbevares af operativsystemet.
 - *Kataloger* (directories)
 - Et katalog indeholder filer og andre kataloger.
 - Kataloger tillader en hierarkisk organisering af programmer og data på de sekundære lagre.
 - *Drev*
 - Lager eller lagerdelen, hvori filer og kataloger befinder sig.



Navne af filer i kataloger:

- Fornavn: filens egentlige navn (*moster*)
- Efternavn: Angiver typisk filens type.
 - Kaldes ofte *fil-extension* (*doc*)
- Sti: Et navn som angiver filens placering i katalogerne
(*c:\Hansen\Breve\moster.doc*).

Kommandoer i DOS.

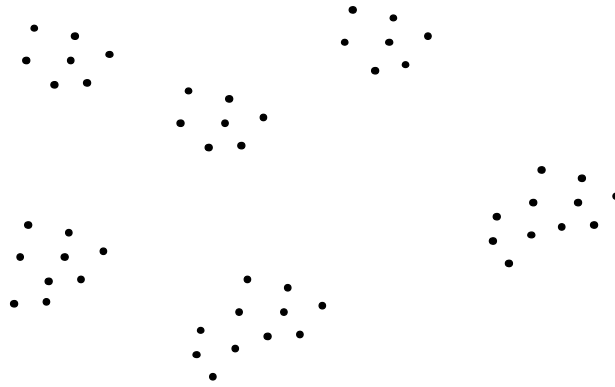
- Kataloger:
 - `cd katalog` Skifter fra det nuværende katalog til *katalog*
 - `dir` Udskriver filer og kataloger i det nuværende katalog
 - `mkdir katalog` Laver nyt *katalog* i det nuværende katalog
 - `rmdir katalog` Fjerner et eksisterende katalog fra nuværende katalog
 - Særlige kataloger . = nuværende katalog .. = forældre katalog
- Filer
 - `type fil` Udskriver *fil*'s indhold på skærmen
 - `print filnavn` Udskriver *fil*'s indhold på en printer
 - `copy fil1 fil2` Kopierer *fil1*'s indhold over i *fil2*
 - `move fil katalog` Flytter en *fil* fra det nuværende katalog til *katalog*
 - `rename fil1 fil2` Ændrer navnet af *fil1* til *fil2*.
 - `del fil`
- Drev
 - `format drev` Forbereder *drev* på at kunne indholde filer og kataloger. Sletter alle eksisterende filer og kataloger på drevet. Eksempel: `format a:`

Programmering i Pascal.

- Dette kursus er baseret på programmeringssproget Pascal.
 - Pascal er et højniveau sprog, som er udviklet i '70erne specielt med henblik på undervisning i struktureret programmering.
 - Vi benytter Turbo Pascal 7.0 i DOS operativsystemet.
- Idag: Anvendelse af Turbo Pascal på allerede eksisterende programmer.
 - Start af Turbo Pascal.
 - Menuer og funktionstaster.
 - Muligheder for hjælp.
 - Indskrivning af et nyt program.
 - Oversættelse af et program.
 - Kørsel af et program.
 - Ændring af arbejdskatalog.
 - Lagring af et program.
 - Åbning af et eksisterende program.
 - Afslutning af Turbo Pascal.

Maskine, net, Internet (1).

Verden består af isolerede computere.



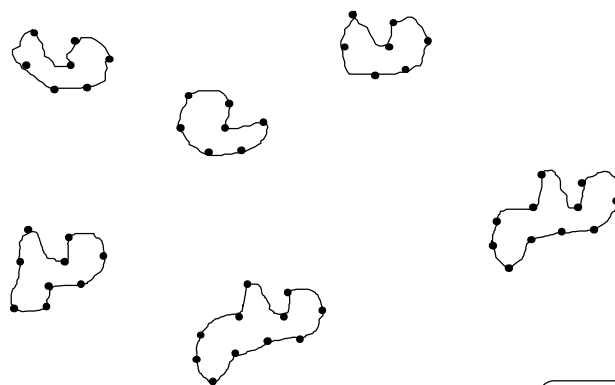
• Isoleret computer

Basis 97: Programmering i Pascal

15

Maskine, net, Internet (2).

Computere tæt på hinanden knyttes sammen i lokale netværk.



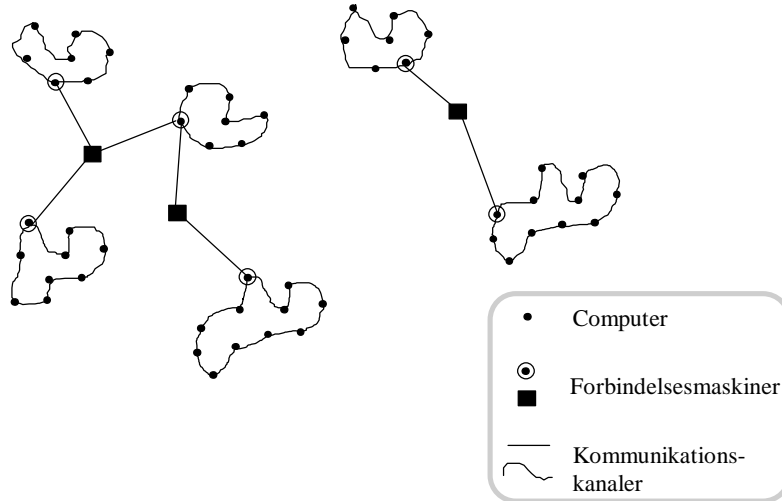
• Computer
— Kommunikationskanal

Basis 97: Programmering i Pascal

16

Maskine, net, Internet (3).

Lokale net knyttes sammen i lang-distance netværk.

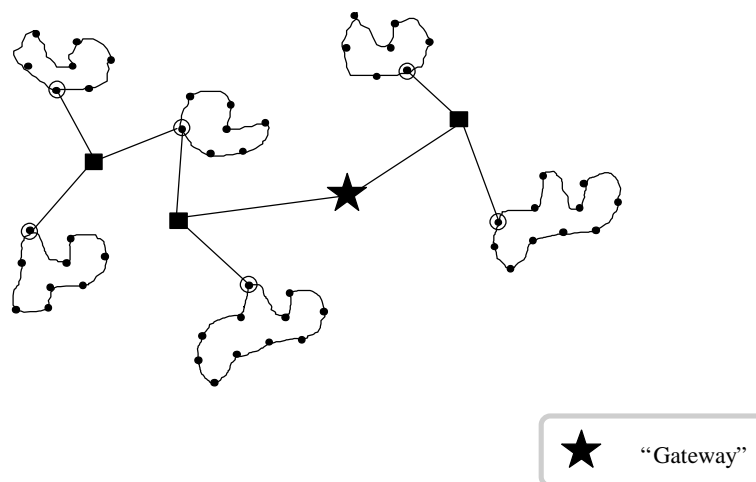


Basis 97: Programmering i Pascal

17

Maskine, net, Internet (4).

Netværk kobles sammen i et globalt Internet.



Basis 97: Programmering i Pascal

18

World Wide Web.

- Den mest interessante del af Internettet hedder *World Wide Web* (WWW).
 - World Wide Web er et stort hypertext/hypermedie netværk.
- *Hypertekst* er ikke-lineær tekst, som sammenkæder *knuder* med *links*.
- World Wide Web tilgås via et værktøj, som kaldes en *browser*.
 - Eksempelvis: Mosaic, Netscape, Internet Navigator.
- Information om kurset i datalogi udsendes over World Wide Web.
 - Kursets hjemmeside: <http://www.cs.auc.dk/education/datbasis/>
 - Idag: Anvendelse af Netscape's browser.
 - Grundliggende navigering på WWW.
 - Baglæns og forlæns navigering samt "hjem".
 - Bogmærker.
 - Søgning - lokalt og globalt.
 - E-mail via Netscape.
 - Hvad er der bagved den pæne overflade?

2. Regneark.

Idéen bag regneark.
Regnearksbegreber.
Formler.
Eksempler.
 Karaktertabel.
 Lønregnskab.
Afhængigheder i regneark.
Udvikling af talrækker i regneark.
Arbejde med sandhedstabeller i regneark.
Selektive udtryk.

Regnearksidéen.

- I et regneark organiseres maskinens lager som en stor to-dimensional tabel (en matrix) af celler.
- Cellerne kan indeholde tekster, tal eller formler.
- På ethvert tidspunkt vises en del af tabellen på skærmen.
- Beregninger foretages via formler, som kan knyttes til tabellens celler.
- Tabellen holdes på ethvert tidspunkt konsistent og opdateret.

Regnearksbegreber.

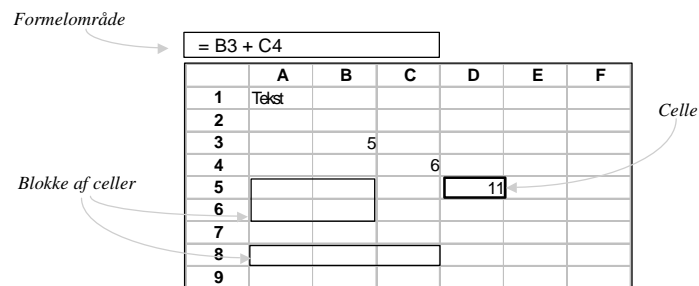
Celle

- Adresse
- Navn
- Værdi

Hvor er cellen placeret i regnearket?
Hvad er det logiske navn af cellen?
Hvilken værdi og datatype indeholder cellen?

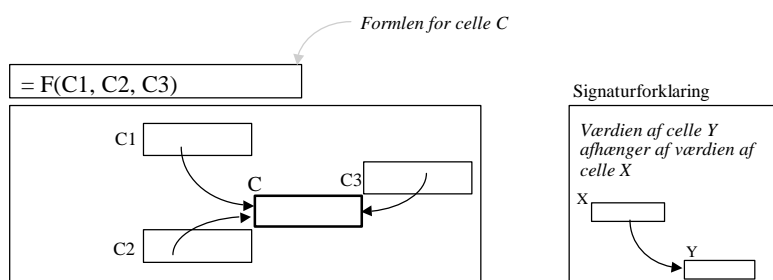
- Formel
- Formatering.

Hvordan beregnes cellens værdi ud fra andre værdier?
Hvordan fremtræder cellens værdi på skærmen?



Formler.

- En formel kan knyttes til en celle i et regneark.
- En formel i cellen C angiver hvorledes C's værdi afhænger af andre cellers værdier.
 - C's værdi er en *funktion* F af andre cellers værdier.
- I det øjeblik en værdi i C1, C2, eller C3 ændres, udregnes en ny værdien af C ud fra formelen F.



Eksempel: Karaktertabel.

	B	C	D	E	F	G	H
3	Antal fag:	5					
4	Antal elever:	6					
5							
6							
7		Dansk	Matematik	Fransk	Græsk	Historie	Gn. snit
8	Per	6	8	7	11	6	7.6
9	Hans	5	9	3	11	8	7.2
10	Pia	3	10	6	7	7	6.6
11	Poul	5	7	8	3	5	5.6
12	Arne	13	6	11	10	3	8.6
13	Frederik	10	3	7	5	13	7.6
14	Gn. snit	7.00	7.17	7.00	7.83	7.00	7.20

@SUM(C8..C13)/C4

@SUM(C8..G13)/(C3*C4)

@SUM(C8..G8)/C3

Alternativer:

- Navngiv celle C3 og C4 som hhv. 'antalfag' og 'antalelever'.
- Brug en gennemsnitsfunktion @AVG i stedet for @SUM.

Eksempel: Lønregnskab.

		$+C2-C3$	$+F12$	$+C3/C2$	$+C4/C2$	$+D7*E7$
	B	C	D	E	F	
2	Lønbudget	1,000,000.00		Dags dato:	November 11, 1995	
3	Forbrugt	792,525.00	79.25%			
4	Rest	207,475.00	20.75%			
5						
6	Person	Ansættelsesdato	Dagløn	Dage	Løn ialt	
7	Lars	February 1, 1995	700.00	202	141,400.00	
8	Gitte	May 5, 1994	500.00	396	198,000.00	
9	Astrid	June 6, 1994	700.00	374	261,800.00	
10	Gertrud	November 1, 1995	975.00	7	6,825.00	
11	Hans	March 3, 1995	1,025.00	180	184,500.00	
12	Total		3,900.00	1159	792,525.00	

$@SUM(D7..D11)$ $@SUM(E7..E11)$ $@SUM(F7..F11)$
 $@BDAYS(C7,F2)$

Basis 97: Programmering i Pascal

25

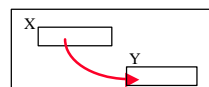
Afhængigheder i et regneark.

Afhængighederne mellem cellerne i et regneark kan anskueliggøres med en 'afhængighedsgraf'.

Cykliske afhængigheder kan ikke håndteres.

	B	C	D	E	F
2	Lønbudget	1,000,000.00		Dags dato:	November 11, 1995
3	Forbrugt	792,525.00	79.25%		
4	Rest	207,475.00	20.75%		
5					
6	Person	Ansættelsesdato	Dagløn	Dage	Løn ialt
7	Lars	February 1, 1995	700.00	202	141,400.00
8	Gitte	May 5, 1994	500.00	396	198,000.00
9	Astrid	June 6, 1994	700.00	374	261,800.00
10	Gertrud	November 1, 1995	975.00	7	6,825.00
11	Hans	March 3, 1995	1,025.00	180	184,500.00
12	Total		3,900.00	1159	792,525.00

Værdien i celle Y
afhænger af værdien i
celle X



Basis 97: Programmering i Pascal

26

Udvikling af talrækker med regneark.

Via meget simpel formelkopiering kan det lade sig gøre af udvikle talrækker ala Fibonacci og talrækken i Euclid's algoritme.

Forudsætningen for dette er at celle adresserne 'opfører sig relativt' under formelkopieringen.

	B	C	D	E	F	G	H	I
2	Fib:							
3	0	1	1	2	3	5	8	13
4								
5	Euclid GCD							
6	24	9	6	3	0	ERR	ERR	ERR

+B3+C3 +D3+E3 +F3+G3
 +C3+D3 +E3+F3 +G3+H3
 @MOD(B6,C6) @MOD(D6,E6) @MOD(G6,H6)
 @MOD(C6,D6) @MOD(F6,G6)

Basis 97: Programmering i Pascal

27

Arbejde med sandhedstabeller i regneark.

Ved brug af et regneark er det let at tabellægge funktioner med få elementer i definitionsmængden.

Bekræftelse på en af Demorgan's love:

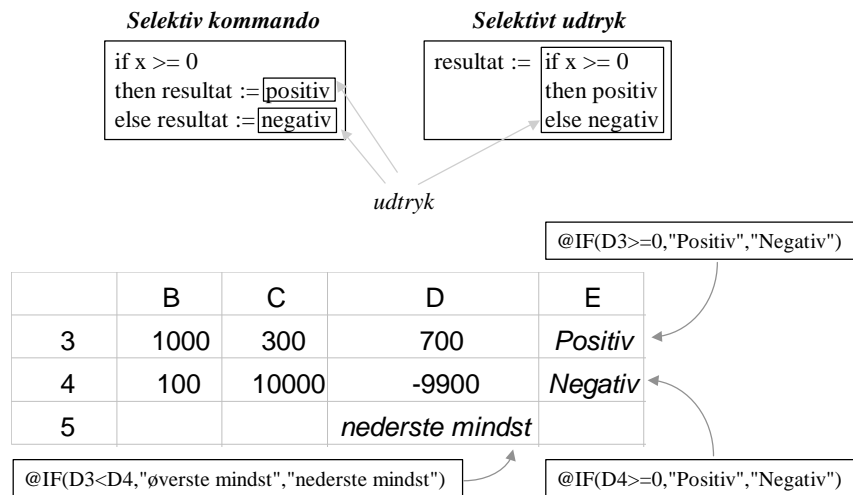
$$\text{not}(a \text{ and } b) = (\text{not } a) \text{ or } (\text{not } b)$$

a	b	a and b	not a	not b	not(a and b)	(not a) or (not b)
0	0	0	1	1	1	1
0	1	0	1	0	1	1
1	0	0	0	1	1	1
1	1	1	0	0	0	0

Basis 97: Programmering i Pascal

28

Selektive udtryk.



3. Basal Programmering

Pascal: et højniveausprog.
Grundliggende elementer i Pascal
Basale datatyper
Variable og assignments
Indlæsning og udskrivning
Aritmetiske udtryk
Programkonventioner
Introduktion til 'udviklingsmetode'

Pascal: Et højniveau programmeringssprog.

- Et *højniveau programmeringssprog* som
 - understøtter *generel problemløsning* på en computer
 - tillader formulering af et program på programmøren's snarere end maskinens præmisser
 - kræver punktlig overholdelse af en række forskellige regler
- *Pascal* er et sprog fra 1970'erne udviklet i Schweiz af Niklaus Wirth.
 - populært til undervisning
 - anvendes også i praktisk programudvikling

Der findes mange forskellige slags programmeringssprog, som ofte opdeles i

- *Generationer* (1. - 5. generation)
- *Paradigmer*
 - Imperativ
 - Funktionsorienteret
 - Logik
 - Objekt-orienteret

Grundlæggende elementer i et Pascal program.

```
program cirkel;
{Et program som beregner en cirkel's areal}
const pi = 3.14159;
var radius, areal: Real;
begin
  {Indlæsning af data}
  Writeln('Indtast cirkelens radius:');
  Readln(radius);

  {Programmets beregningdel}
  areal := radius * radius * pi;

  {Udskrivning af resultatet}
  Writeln('Cirklen har følgende areal: ', areal:6:2)
end.
```

← *Programhoved og -navn*

← *Kommentar*

← *Erklæringsdel*

← *Programkrop*

Navne:
Reserverede ord:

- Eks.: begin og end

Navne (identifiers):

- Standardnavne
 - Eks.: writeln
- Brugerdefinerede
 - Eks.: radius

Oversigt over datatyper.

En datatype

- er en *mængde af værdier*.
- angiver en *fortolkning* af et lav-niveau bit-mønster
- er karakteriseret af en *mængde af operationer* på datatypens værdier

Datatyper i Pascal:

- Reelle tal: `Real`.
- Ordinal typer:
 - Heltal: `Integer`
 - Tegn: `Char`
 - Sandhedsværdier: `Boolean` (*true* og *false*)
- Sammensatte typer:
 - Tabeller: `Array`
 - Tekster: `String`
 - Poster: `Record`

Typen real.

Typen `real` repræsenterer udvalgte reelle tal i et nærmere bestemt interval. Mange reelle tal har kun en tilnærmet repræsentation i typen `Real`.

• Operationer:

- Fortegnsoperationer: + -
- Aritmetiske operationer: + - * /

• Notationer:

- Floating point (videnskabelig notation) *fortegn heltalsdel .brøkdel E exponentialdel*
-3.456E2

- Fix point *fortegn heltalsdel .brøkdel*

-345.6

Decimalpunkt er påkrævet.

Mindst ét ciffer før og efter decimal punktet.

Typerne integer, char og boolean.

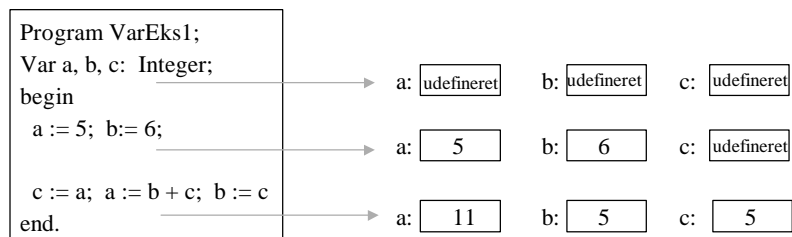
	Værdier	Typiske operationer
Integer	Alle heltal mellem -32768 og 32767	Fortegnsoperationer: + - Andre operationer: + - * div mod
Char	128 forskellige tegn, herunder små og store bogstaver samt cifre	
Boolean	<i>true</i> og <i>false</i>	Not and or

Værdierne i ovennævnte typer er ordnede indenbyrdes. Det er således meningsfuldt at tale om forgænger $pred(v)$ og efterfølger $succ(v)$ af en værdi v (pånær af yderværdierne).

Variable og assignment (1).

- En variabel er en *navngiven celle* i maskinens arbejdslager, som kan indeholde en *værdi* af en bestemt *type*.
 - Værdien af en variabel er udefineret indtil første værditilskrivning.
- Værdien af en variabel kan ændres ved udførelse af en *assignment kommando*.
 - En assignment kommando “tilskriver en værdi” til en variabel.

Eksempel:



Variable og assignments (2).

Den generelle *form* (syntaks) af en assignmentkommando:

Variabel := Udtryk

Et udtryk er en formel, hvori der kan indgå operatore og operander.

Betydning og krav:

- Udtrykket beregnes.
- Typen af udtrykket skal være den samme som typen af variabelen.
 - Dog kan et heltal tilskrives til en reel variabel.
- Den hidtidige værdi af variabelen overskrives af udtrykkets værdi.

Eksempel:

```
Program VarEks2;
Var a: Integer;
    r: Real;
begin
  a := 3;
  a := a + 1;
  r := sqrt(a)
end.
```

→ a og r er udefinerede
→ a har værdien 3, r er udefineret
→ a er talt op med 1 og er nu 4
→ r er tilskrevet værdien af kvadratroden af a, altså 2.0

Basis 97: Programmering i Pascal

37

Navngivne konstanter

En navngiven konstant refererer til en bestemt værdi gennem et navn.

Det er ikke muligt at ændre værdien af konstanten under udførelsen af programmet.

Navngivne konstanter øger programlæseligheden, og de gør det let at ændre værdien af sådanne *i forbindelse med programmodifikation*.

Eksempel på konstant erklæringer:

```
const pi = 3.1415927;
      MaxInt = 32767;
      FoersteBogstav = 'a';
      SidsteBogstav = 'å'
```

Basis 97: Programmering i Pascal

38

Indlæsning og udskrivning.

- **Write** er Pascal proceduren hvormed værdien af én eller flere udtryk kan udskrives som tekst på skærmen.
- **Read** er Pascal proceduren hvormed en eller flere værdier, angivet som tekst, kan indlæses i et antal programvariable.
- Der findes følgende varianter
 - **Writeln** som udskriver en eller flere værdier afsluttet med et linesskift.
 - **Readln** som læser en eller flere værdier til og med et linesskift.

```
var a, b, c: real;
begin
  a := 5; b := 6;
  writeln('Værdien af a, b og c er: ', a, b, c);
  writeln('Indlæs a og b på een linie');
  readln(a, b);
  writeln('tak');
  writeln('Indlæs nu c');
  read(c);
  writeln('Værdien af a, b og c er: ', a, b, c);
end.
```

Aritmetiske udtryk.

- Et aritmetisk udtryk består af *operatorer* og *operander*.
 - Eksempler
 - $(a + 5) * (b \bmod 3)$
 - $3 - 4 - 5 * 6 \text{ div } 7$
 - $3.5 * 4 / 2$
- Mulige operatorer:
 - + - * / div mod
- Mulige operander:
 - Variable, konstanter og talværdier (literals).
- Typen af et aritmetisk udtryk:
 - Integer hvis alle operander er heltal og ingen operator er / (divisionstegn).
 - Real ellers.

Beregningsrækkefølge af deludtryk.

Problem: I hvilken rækkefølge beregnes deludtrykkene af et udtryk?

Løsning 1 (den simple):

- Alle deludtryk omkranses af parenteser.
- Udtryk beregnes "indefra og ud".

Løsning 2 (den komplicerede):

- Operatører tildeles prioriteter.
- Deludtryk med højt-prioriterede operatører beregnes før deludtryk med lavere prioriterede operatører.
- Deludtryk med operatører, der har samme prioritet, beregnes fra venstre mod højre.

Udtryk uden parenteser	Tilsvarende udtryk med parenteser
$3 + 4 * 5$	$(3 + (4 * 5))$
$6 - 3 - 2$	$((6 - 3) - 2)$
$3 - 4 - 5 * 6 \text{ div } 7$	$((3 - 4) - ((5 * 6) \text{ div } 7))$

Operator prioriteter i Pascal.

Prioritet 5	()	Parenteser
Prioritet 4	Not	Negation
Prioritet 3	* / Div Mod And	Multiplikative
Prioritet 2	+ - Or	Additive
Prioritet 1	= <> < <= > >=	Relationelle

- Operatører med det højeste prioritetstal anvendes først.
- Operatører, der har samme prioritet, anvendes fra venstre mod højre.

Eksempel:

$x < y$ or $x = y$ Meningsløst

$(x < y)$ or $(x = y)$ OK

Konventioner.

- Det er tilladt at have *ekstra mellemrum* mellem ord og symboler i et program.
- Det er tilladt at have *ekstra tomme linier* i et program.
- Det er tilladt at samle adskillige linier til én linie.
 - Hele programmet kan i princippet skrives på én lang linie.
- *Kommentarer* i { ... } eller (* ... *) kan optræde overalt i et program hvor der forekommer mellemrum.
- Der er ingen forskel på store og små bogstaver i navne og nøgleord.
- Anbefalinger:
 - Brug indrykning til at fremhæve programmets struktur.
 - Brug kommentarer til at forklare aspekter af et program, som ikke er umiddelbart forståelige for læseren.
 - Start altid programmet med en overordnet kommentar.
 - Vær konsekvent ved brug af små og store bogstaver i navne.

Hvordan udvikler man et program?

Software udviklingsmetode

- Når man udvikler software (et program) anbefales det at følge en bestemt *metode* (fremgangsmåde).
- Ofte opdeles det samlede arbejde med et program i en række *faser*, såsom:
 - Kravspecifikation
 - Problemanalyse
 - Design
 - Implementation
 - Aftestning
 - Vedligeholdelse
- Senere i kurset vil vi illustrere konkrete eksempler på disse faser.

4. Abstraktion med procedurer og funktioner

Ris à l'mande.
Eksempel på procedurer: Tændstikkvinde.
Programmering ved trinvis forfinelse.
Procedurebegrebet.
Programforløbet ved procedurekald.
Forfinet grafik.
Eksempel på funktioner: Temperaturomregning.
Funktionsbegrebet.
Fremblik: Det fulde procedure- og funktionsbegreb.

Ris à l'amande.

LAV EN PORTION RIS À L'AMANDE:

Smør en gryde med lidt margarine;
Bring 1 liter mælk i kog;
Tilsæt 110 gram grødris;
Kog det hele i 50 minutter;
Smag til med salt;
Afkøl grøden;
Tilsæt 1/2 dl Gammel Dansk;
Hæld 4 dl piskefløde i en skål;
Pisk i fløden indtil det bliver stift;
Vend skummet i grøden;
Afkøl desserten i 2 timer

Dette giver en vældig god ris à l'amande.
Men opskriften er dårlig struktureret.

Struktureret Ris à l'amande.

LAV EN PORTION RIS À L'AMANDE:

Lav en portion risengrød;
Afkøl grøden;
Tilsæt 1/2 dl Gammel Dansk;
Lav 1/2 liter flødeskum;
Vend skummet i grøden;
Afkøl desserten i 2 timer

LAV EN PORTION RISENGRØD:

Smør en gryde med lidt margarine;
Bring 1 liter mælk i kog;
Tilsæt 110 gram grødris;
Kog det hele i 50 minutter;
Smag til med salt

LAV 1/2 LITER FLØDESKUM

Hæld 4 dl piskefløde i en skål;
Pisk i fløden indtil det bliver stift

Ris à l'amande i Pascal.

```
Program RisALaMande;

  Procedure LavRisenGrød;
  begin
    ...
  end;

  Procedure LavFlødeskum;
  begin
    ...
  end;

begin
  LavRisenGrød;
  Afkøl grøden;
  Tilsæt 1/2 dl Gammel Dansk;
  LavFlødeskum;
  Vend skummet i grøden;
  Afkøl desserten i 2 timer
end.
```

Eksempel: Tegning af en tændstikkvinde med procedurer

Observation: Det er muligt at lave en række simple tegninger ved brug af et lille antal forskellige *geometriske primitiver*.

```

procedure TegnTaendstikKvinde;
begin
  TegnHoved;
  TegnArme;
  TegnKrop;
  TegnBen
end

begin { hovedprogram }
  ClrScr;
  TegnTaendstikKvinde;
  ReadKey
end.

procedure TegnHoved;
begin
  TegnCirkel
end

procedure TegnArme;
begin
  TegnVandretLinie
end

procedure TegnKrop;
begin
  TegnOmvendtV;
  TegnVandretLinie
end

procedure TegnBen;
begin
  TegnOmvendtV
end
  
```

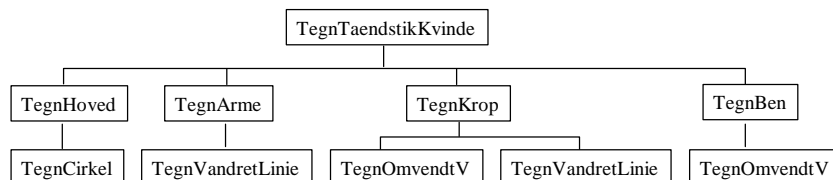
Eksempel: Geometriske primitiver.

```

procedure TegnCirkel;
begin
  WriteLn (' * ');
  WriteLn (' * * ');
  WriteLn (' * * * ');
end;

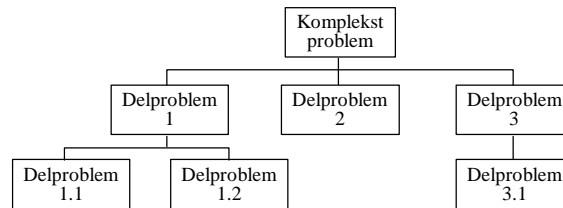
procedure TegnVandretLinie;
begin
  WriteLn ('-----')
end;

procedure TegnOmvendtV;
begin
  WriteLn (' / \ ');
  WriteLn (' / \ ');
  WriteLn (' / \ ');
end;
  
```



Programudvikling ved trinvis forfinelse.

- *Komplekse problemer* kan ofte opdeles i *delproblemer*.
- Løsningerne på delproblemerne kan ofte *sammensættes* til en løsning af det oprindelige problem.



- I et program, som løser et komplekst problem, udformes løsningen på hvert problem og delproblem som en *proceduredefinition*.
- I kroppen af proceduren for et komplekst problem angives hvordan løsningerne for delproblemerne skal kombineres til en løsning af det komplekse problem.
- Proceduredefinitioner for delproblemer indlejres ofte som lokale procedurer i proceduredefinitionen for det komplekse problem.

Basis 97: Programmering i Pascal

51

Procedurer i forhold til et program.

```
program programnavn;  
const nogle konstanter;  
var nogle variable;  
begin  
  sekvens af kommandoer  
end.
```



```
program programnavn;  
const nogle konstanter;  
var nogle variable
```

```
  procedure p1;  
  begin ... end;
```

```
  procedure p2;  
  begin ... end;
```

```
begin {hovedprogram}  
  ... P1;  
  ..... P2;  
  .... P1;  
end.
```

- Procedurer kan opfattes som “programmer i programmer”
- Procedurer erklæres lige efter variabelerklæringer i hovedprogrammet.
- I Pascal skal procedurer erklæres før brug i programteksten.

Basis 97: Programmering i Pascal

52

Procedurebegrebet (uden parametre).

Proceduredefinition:

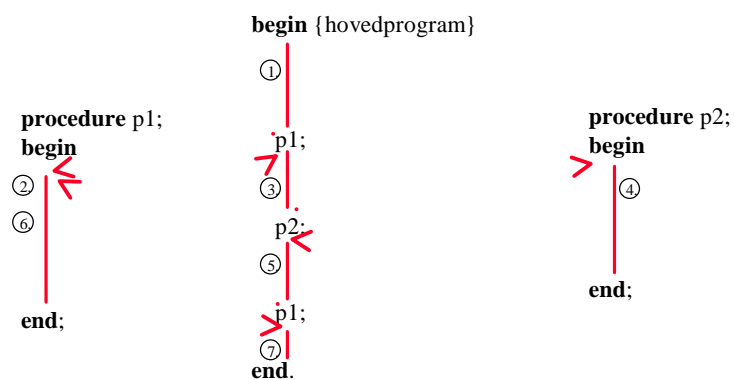
```
procedure procedurenavn;  
begin  
    kommando sekvens  
end;
```

Procedurekald: En kommando

```
procedurenavn
```

- **Abstraktion:** En procedure er en abstraktion over en sekvens af kommandoer.
- **Genbrug:** En procedure indkapsler et antal kommandoer, som kan bruges (kaldes) mange steder i et program.

Programforløbet ved procedurekald.



Eksempel: Forfinet grafik.

```
program grafikProgram;
uses graph; { benytter grafikbibliotek }
const Directory = 'c:\tp\bgi';
      { kataloget med grafik drivere }

var Driver, Mode: integer;
begin
  { opsætning af grafik: }
  initgraph(Driver, Mode, Directory);

  { Eksempel på kald af grafik procedurer: }
  Circle(cx, cy, radius);
  Line(x1, y1, x2, y2);
end.
```

Eksempel: Tændstikkvinde med forfinet grafik.

Vi udskifter nu tegneprimitiverne, således at vi opnår "ægte" grafik.
Vi er nødt til at bekymre os om den relative placering af legemsdele.

```
procedure TegnTaendstikKvinde;
begin
  TegnHoved;
  TegnArme;
  TegnKrop;
  TegnBen
end;

begin { hovedprogram }
  Initgraph(Driver, Mode, Directory);
  Clrscr;
  TegnTaendstikKvinde;
  ReadKey
end.
```

```
procedure TegnHoved;
begin
  TegnCirkel(100,100,30)
end;

procedure TegnArme;
begin
  TegnVandretLinie(25,130,150)
end;

procedure TegnKrop;
begin
  TegnOmvendtV(100,130,100,200);
  TegnVandretLinie(0,330,200)
end;

procedure TegnBen;
begin
  TegnOmvendtV(100,330,75,150)
end;
```

Eksempel: Geometriske primitiver med forfinet grafik.

```
procedure TegnCirkel(x_centrum, y_centrum, radius: integer);
begin
  Circle(x_centrum, y_centrum, radius);
end;

procedure TegnVandretLinie(x_start, y_start, laengde: integer);
begin
  Line(x_start, y_start, x_start+laengde, y_start);
end;

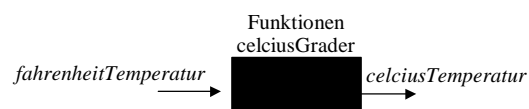
procedure TegnOmvendtV(x_top, y_top, dx, dy: integer);
begin
  line(x_top, y_top, x_top-dx, y_top+dy);
  line(x_top, y_top, x_top+dx, y_top+dy)
end;
```

Eksempel: Temperaturomregning med funktioner

Definition af en funktion der omregner fahrenheit til celcius grader:

```
Function celciusGrader (fahrenheitTemperatur: Real): Real;
{Omregn en fahrenheit temperatur til celcius grader og returner resultatet}
begin
  celciusGrader := (5.0 / 9.0) * (fahrenheitTemperatur - 32.0)
end;
```

Anvendelse af funktionen i et hovedprogram:



```
var celciusTemperatur : Real;
begin {hovedprogram}
  celciusTemperatur := celciusGrader(32.0);
  writeln('Det er koldt: ', celciusTemperatur:5:2);
  writeln('Det er varmt: ', celciusGrader(102.0):5:2)
end.
```

Funktionsbegrebet (med parametre).

Funktionsdefinition:

```
function funktionsnavn(parametre): returværdi-type;  
begin  
    funktionsnavn := udtryk  
end;
```

Funktionskald: Et udtryk

```
funktionsnavn(parametre)
```

• **Abstraktion:**

- En funktion er en abstraktion over et udtryk.
- I Pascal kan en funktion dog udføre kommandoer, blot skal der returneres en værdi (ved et assignment til funktionsnavnet).

• **Genbrug:**

- En funktion indkapsler et udtryk, som kan genbruges (kaldes) mange steder i et program.

Fremblik mod det fulde procedure- og funktionsbegreb.

Procedurer og funktioner

- kan *generaliseres* med parametre.
 - værdi- og variableparametre.
- kan have lokale variable.
- kan have lokale procedurer og funktioner.

```
procedure procedurenavn  
    (formelle-parametre);  
var lokale variable;  
lokale procedure og funktioner;  
begin  
    kommando sekvens  
end;
```

```
function funktionsnavn  
    (formelle-parametre): Værditype  
var lokale variable;  
lokale procedure og funktioner;  
begin  
    kommando sekvens;  
    funktionsnavn := udtryk  
end;
```

Mere om dette i lektion 6

5. Kontrolstrukturer.

Primitive kontrolstrukturer med hop.
Logiske udtryk.
Udvælgelse.
If og case.
Gentagelse.
While- og for-løkker
Eksempel: Rødsøgning.
Repeat løkker.

Motivation: Primitive kontrolstrukturer med hop.

En goto kommando tillader os at hoppe fra et sted i et program til et andet, markeret sted i samme program.

Programmer med goto kommandoer kan være meget vanskelige at forstå.

Det anbefales at undgå brug af goto kommandoer i normale programmeringssituationer.

Goto kommandoer er kun acceptable i undtagelsessituationer, eksempelvis hvis man ønsker at hoppe ud af programmet eller ud af en dyb løkke.

```
if x <= y then goto 1;  
pos := 1;  
res := x;  
goto 2;  
1: pos := 2;  
res := y;  
2: {næste kommando}
```

Oversigt.

- **Sekventiel rækkefølge**
 - Kommandoer følger efter hinanden i en bestemt rækkefølge.

```
kommando1 ;  
kommando2
```
- **Sammensætning**
 - Et antal kommandoer sammensættes til én enkelt kommando.

```
begin  
kommando1;  
kommando2  
end
```
- **Udvælgelse**
 - Én kommando udvælges blandt flere mulige. Udvælgelsen foretages typisk ud fra værdien af et logisk udtryk.

```
if logiskUdtryk  
then kommando1  
else kommando2
```
- **Gentagelse**
 - En kommando gentages et antal gange. Antallet af gentagelser styres typisk af et logisk udtryk.

```
while logiskUdtryk  
do kommando
```

Logiske udtryk (boolske udtryk).

Et logisk udtryk beregnes til en værdi af datatypen Boolean. Datatypen boolean indholder værdierne *true* og *false*.

Eksempler på logiske udtryk:

```
n = 5  
n >= 5  
n <> 5  
(n >= 5) and (n < 10)  
not ((n >= 5) and (n < 10))  
(n < 5) or (n >= 10)
```

Eksemplerne i en sammenhæng:

```
program logik;  
var n: integer;  
    b: boolean;  
begin  
    n := 6;  
    writeln(n=5);  
    if n >= 5 then writeln('stor') else writeln('lille');  
    b := n <> 5; writeln(b);  
    while (n >= 5) and (n < 10) do n := n + 1;  
    if not ((n >= 5) and (n < 10)) =  
        (n < 5) or (n >= 10)  
    then writeln('OK')  
    else writeln('Problemer');  
end.
```

De logiske operatører.

Idet der kun er to værdier i datatypen Boolean er det let at tabellægge de boolske operatører **and**, **or** og **not**.

AND

A	B	A and B
true	true	true
true	false	false
false	true	false
false	false	false

OR

A	B	A or B
true	true	true
true	false	true
false	true	true
false	false	false

NOT

A	not A
true	false
false	true

Prioriteter:

Prioritet 5	()
Prioritet 4	Not
Prioritet 3	* / Div Mod And
Prioritet 2	+ - Or
Prioritet 1	= <> < <= > >=

Kontrolstrukturer til udvælgelse (1).

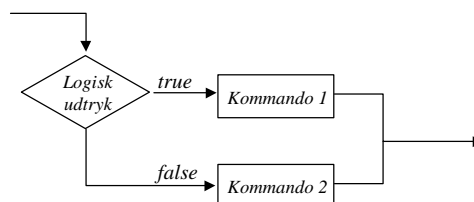
En *sektiv kontrolstruktur* udvælger én kommando til udførelse fra en mængde af muligheder.

Grundform

```
if logiskUdtryk
then kommando1
else kommando2
```

Betydning

Først beregnes *logiskUdtryk*.
Hvis værdien er sand udføres kommando 1.
Ellers udføres kommando 2.



```
if logiskUdtryk
then kommando
```



```
if logiskUdtryk
then kommando
else (* tom kommando *)
```

Kontrolstrukturer til udvælgelse (2).

Grundform

```
case udtryk of
  constant1: kommando1;
  ...
  constantn: kommandon
else
  default-kommando
end
```

Betydning

Først beregnes udtryk. Udtrykket skal være af en ordinaltype.
Hvis værdien af udtrykket er én af konstanterne constant₁ .. constant_n udføres den tilsvarende kommando, hvorefter case-strukturen afsluttes.
Ellers udføres default-kommando.

Det er muligt at sammentrække en række tilfælde i en case kontrolstruktur til ét tilfælde.

```
case n of
  2, 4: writeln('To eller fire');
  5 .. 7: writeln('Mellem fem og syv')
else
  writeln('Noget andet')
end
```



```
case n of
  2: writeln('To eller fire');
  4: writeln('To eller fire');
  5: writeln('Mellem fem og syv');
  6: writeln('Mellem fem og syv');
  7: writeln('Mellem fem og syv')
else
  writeln('Noget andet')
end
```

Basis 97: Programmering i Pascal

67

Eksempler på udvælgelse.

En funktion der beregner det største af to tal:

```
function maks(x, y: Real): Real;
begin
  if x > y then maks := x else maks := y
end;
```

En funktion der returnerer antallet af dage i en måned:

```
function antalDageIMåned(månedNr: Integer;
  skudÅr: Boolean): Integer;
begin
  case månedNr of
    1, 3, 5, 7, 8, 10, 12: antalDageIMåned := 31;
    4, 6, 9, 11:          antalDageIMåned := 30
  2: if skudÅr
    then antalDageIMåned := 29
    else antalDageIMåned := 28
  end
end;
```

Eksempel der modsvare
goto-eksemplet

```
if x > y
then begin
  pos := 1;
  res := x
end
else begin
  pos := 2;
  res := y
end
```

```
if x <= y then goto 1;
pos := 1;
res := x;
goto 2;
1: pos := 2;
res := y;
2: {næste kommando}
```

Basis 97: Programmering i Pascal

68

Kontrolstrukturer til gentagelse: While løkker.

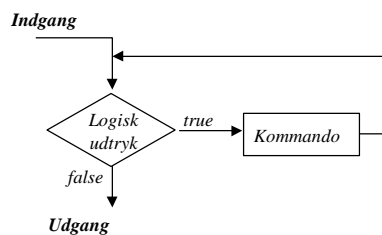
I en *løkke* er det muligt at gentage en kommando nul, én eller flere gange.
I en while-løkke er det i udgangspunktet ubestemt hvormange gange kommandoen gentages.

Form

while logiskUdtryk
do kommando

Betydning

Gentag kommando 0, 1 eller flere gange.
Udregn og test det logiske udtryk før hver gentagelse.
Afbryd løkken første gang det logiske udtryk bliver falsk.



For at sikre, at løkken standser, er det vigtigt at kommandoen i løkken ændrer på de variable, som indgår i løkkens logiske udtryk

Eksempel på brug af while-løkke (1).

- Beregning af den største fælles divisor af to heltal ved Euclid's metode.
- Et eksempel en 'hændelseskontrolleret løkke'.

```
function gcd(Input1, Input2: Integer): Integer;
var Stor, Lille, Rest: Integer;
begin
  Stor := Input1; Lille := Input2;
  while Lille > 0
  do begin
    Rest := Stor mod Lille;
    Stor := Lille;
    Lille := Rest
  end;
  gcd := Stor
end;
```

Eksempel på brug af while-løkke (2).

```
program SumScores;
const
  Sentinel = -1;    {sentinel value. På dansk: en vagt}
var
  Score,           {input - each exam score}
  Sum : Integer;   {output - sum of scores}
begin {SumScores}
  {Accumulate the sum.}
  Sum := 0;
  WriteLn ('When done, enter -1 to stop. ');
  Write ('Enter the first score> ');
  ReadLn (Score);
  while Score <> Sentinel do
  begin
    Sum := Sum + Score;
    Write ('Enter the next score> ');
    ReadLn (Score)
  end; {while}

  {Display the sum.}
  WriteLn;
  WriteLn ('Sum of exam scores is ', Sum :1)
end. {SumScores}
```

Kontrolstrukturer til gentagelse: For-løkker.

I en for-løkke gentages en kommando et antal gange, som bestemmes forud for første gentagelse.

<u>Form</u>	<u>Betydning</u>
for var := første to sidste do kommando	Gentag kommandoen for var = first, succ(first), ..., last. Afbryd løkken når kommandoen er udført for i = last.

- første og sidste er udtryk hvis værdier skal tilhøre samme ordinaltype.
 - En ordinaltype er en type, hvor det giver mening at tale om næste og foregående værdi.
- Variablen var må ikke ændres i kommandoen.
- Det er ikke muligt at ændre på udtrykket sidste i kommandoen med det formål at ændre på antallet af gentagelser i løkken.

Eksempler på brug af for-løkker.

- Udskriv en tabel over i , i^2 og kvadratroden af i , for i løbende fra 1 til 10.
- En tæller-kontrolleret løkke.

```
for i:= 1 to 10 do  
  writeln(i, i*i, sqrt(i));
```

- Udskriv de fire mulige logiske kombinationer af udtrykket:
not b or c

```
for b := false to true do  
  for c := false to true do  
    writeln(b, ' ', c, ' ', not b or c)
```

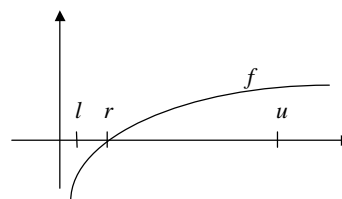
Hvornår bruges for-løkker:

- Når man på forhånd kender antallet af gentagelser, som der er behov for.
- Det er let at lave fejl ved første og sidste gentagelse i en while løkke.
 - Brug derfor for-løkker når det er muligt!

Eksempel: Søgning efter en rod i en kontinuert funktion (1).

Analyse:

Enhver kontinuert funktion f , som har negativ værdi i l og positiv værdi i u (eller omvendt) vil have mindst én rod mellem l og u .



Input:

En Pascal funktion f .
Intervallet l og u , hvor $l \leq u$.
 l og u er reelle tal (reals).

Forudsætninger:

Funktionen f har forskellig fortegn i l og u .
Funktionen er kontinuert.

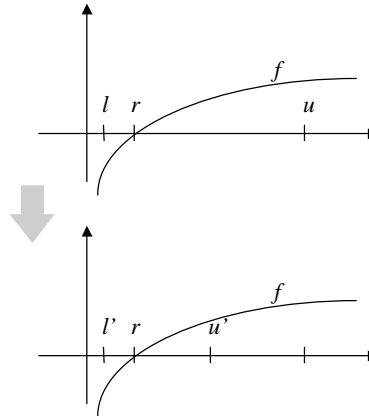
Output:

En værdi r hvorom der gælder, at $f(r) = 0$.
 $l \leq r \leq u$.

Eksempel: Søgning efter en rod i en kontinuert funktion (2).

Design:

Idéen i algoritmen er at indsnævre intervallet omkring den rod, vi ved der eksisterer mellem l og u . I hvert gennemløb af en løkke halverer vi intervallet, hvori vi med sikkerhed ved, at roden befinder sig. På et tidspunkt bliver l og u sammenfaldende.



Eksempel: Søgning efter en rod i en kontinuert funktion (3).

```
function find_rod_mellem(l, u: real): real;
{ Forudsæt: f(l) og f(u) har forskellig fortegn, f kontinuert og l < u. }
begin
  while not ( f(midpunkt(l,u)) = 0 )
  do begin
    if samme_fortegn(f(midpunkt(l, u)), f(u))
    then u := midpunkt(l,u)
    else l := midpunkt(l,u);
  end;
  find_rod_mellem := midpunkt(l,u);
end;
```

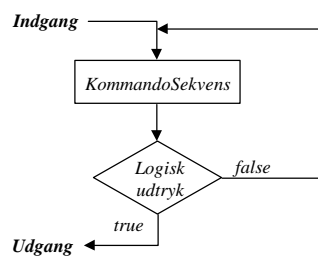
```
function midpunkt(x,y: real): real;
{ beregn og returner tallet midt mellem
  x og y. Forudsæt at x er mindre end y }
begin
  midpunkt := x + (y - x)/2;
end;
```

```
function samme_fortegn(x,y: real): boolean;
{ returner hvorvidt x og y har samme fortegn }
begin
  samme_fortegn := ((x * y) > 0);
end;
```

Kontrolstrukturer til gentagelse: Repeat-løkker.

En repeat-løkke er et alternativ til en while-løkke.
En repeat kontrolstruktur sikrer altid mindst et gennemløb af løkken.

<u>Form</u>	<u>Betydning</u>
repeat kommandoSekvens until logiskUdtryk	Gentag kommandoSekvens 1 eller flere gange. Udregn og test det logiske udtryk efter hver gentagelse. Afbryd løkken første gang boolean-expression bliver sand.



- Det logiske udtryk i en repeat kontrolstruktur er en afslutningsbetingelse.
- Det logiske udtryk i en while kontrolstruktur er en fortsættelsesbetingelse.

Eksempel på brug af repeat-løkke.

Spørg brugeren: ja eller nej

```
var svar: Char;  
...  
repeat  
  writeln('Svar ja eller nej (j/n)');  
  readln(svar)  
until (svar = 'j') or (svar = 'n')
```

Konvertering af for- og repeat-løkker til while.

Ethvert program med 'for' og 'repeat' kontrolstrukturer kan skrives om til et program med 'while'.

```
repeat  
  kommandoSekvens  
until logiskUdtryk
```



```
begin  
  kommandoSekvens  
end;  
while not logiskUdtryk  
do begin  
  kommandoSekvens  
end
```

```
for var := first to last do  
  kommando
```



```
var := first;  
while var <= last  
do begin  
  kommando;  
  var := succ(var)  
end
```

6. Procedurer og funktioner med parametre.

- Resumé af procedurebegrebet.
- Motivation for parametre til procedurer.
- Værdiparametre.
- Variabelparametre.
- Lokale variable.
- Indlejrede procedurer.
- Scopebegrebet.
- Funktioner.

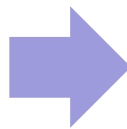
Resumé af procedurebegrebet.

- En procedure er en *abstraktion* over en sekvens af kommandoer.
 - Proceduren indkapsler en mængde detaljerede kommandoer, som efter proceduredefinitionen kan refereres gennem proceduren's navn.
- En procedure *defineres* i erklæringsdelen af et program.
- En procedure kan opfattes som et “program i programmet”: Et *subprogram*.
- Et procedure kaldes i kroppen af et program eller i kroppen af en anden procedure.
 - Et procedurekald er en kommando.
 - Et procedurekald kan optræde alle de steder hvor kommandoer er tilladt.
 - En procedure skal defineres før den kan kaldes.
 - En procedure kan kaldes mange gange.
 - Med et procedurekald slipper vi for at gentage detaljerne i kroppen for hvert kald.

Eksempel: En procedure med værdiparametre.

```
writeln('Peter Madsen');  
writeln('Badehusvej 13');  
writeln('Storgruppe 31');  
writeln('9000 Aalborg C');  
writeln;
```

```
writeln('Mads Petersen');  
writeln('Badehusvej 13');  
writeln('Storgruppe 33');  
writeln('9000 Aalborg C');  
writeln;
```



```
SkrivAdresse('Peter Madsen', 31);
```

```
SkrivAdresse('Mads Petersen', 33);
```

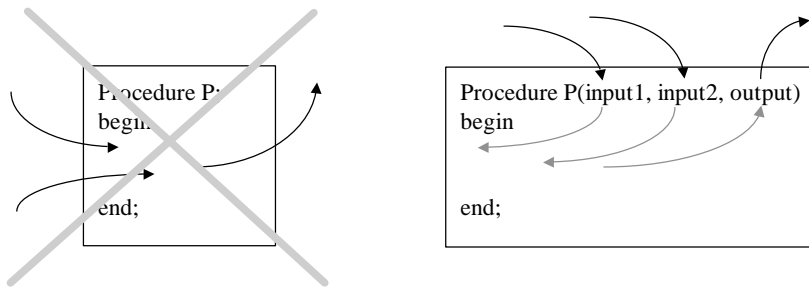
```
procedure SkrivAdresse(Navn: String; Storgruppe: Integer);  
begin  
  writeln(Navn);  
  writeln('Badehusvej 13');  
  write('Storgruppe '); writeln(Storgruppe: 2);  
  writeln('9000 Aalborg C');  
end;
```

Motivation: Hvorfor interessere sig for parametre?

De kommandoer, som indkapsles i en procedurer, bliver *mere generelt anvendelige* hvis vi kan overføre input til dem og output fra dem.

Parametrene udgør *grænsefladen* (interfacet) mellem en procedure og dens omverden (hovedprogrammet og andre procedurer).

- Alle *databevægelser* mellem en procedure og omverdenen bør foregå gennem parametre.



Basis 97: Programmering i Pascal

83

Overførsel af værdiparametre (1).

c og i er formelle værdiparametre

```
Procedure P(c: char; i: integer);
begin
  {Udfør kommandoer som
   anvender c og i som input}
end;
```

```
var j: integer;
    tegn: char;
...
begin
  j := 2; tegn := 'a';
  P(tegn, j + 3);
end
```

Udtrykkene 'a' og j+3 er aktuelle parametre

Procedure P(c: char; i: integer); {procedure krop}

```

      ↑
      |
P(tegn, j + 3);
      |
      ↑

```

Værdien af de aktuelle parameterudtryk overføres til de tilsvarende formelle parametre

Basis 97: Programmering i Pascal

84

Overførsel af værdiparametre (2).

```
Procedure P (c: char; i: integer);  
begin  
  {Udfør kommandoer som  
  anvender c og i som input }  
end;
```

```
var j: integer;  
    tegn: char;  
...  
begin  
  j := 2; tegn := 'a';  
  P(tegn, j + 3);  
end
```

Umiddelbart efter overførslen af parametrene:

Globale variable

Parametre i P

j: (Integer)

c: (Char)

tegn: (Char)

i: (Integer)

Regler for overførsel af værdiparametre.

```
Procedure P(fp1: T1; fp2: T2; ... ; fpn: Tn);  
begin  
  ...  
end;
```

```
P(ap1, ap2, ..., apn)
```

- *Værdierne* af de aktuelle parametre beregnes. Hver aktuel parameter er et *udtryk*.
- Første aktuelle parameter overføres til første formelle parameter, anden aktuelle parameter overføres til anden formelle parameter, osv.
- Antallet af aktuelle parametre skal være lig med antallet af formelle parametre.
- Værdien af en aktuel parameter skal kunne assignes til den tilsvarende formelle parameter.
- Procedurens krop udføres.
 - Et assignment til en formel parameter er lovlig, men ændrer aldrig på den tilsvarende aktuelle parameter.
 - Det er dårlig stil at lave assignments til værdiparametre.

Lokale variable.

- Variable i hovedprogrammet kaldes *globale variable*.
- Variable i en procedure kaldes *lokale variable*.
 - En lokal variabel oprettes umiddelbart efter et procedurekald og nedlægges når proceduren er færdig.

```
Program gennemsnitsProgram;
var x, y: integer; ← Globale variable.

Procedure printGennemsnit(a, b, c: integer);
var sum: integer;
    gnSnit: Real; ← Lokale variable i proceduren
begin
    sum := a + b + c; gnSnit := sum/3;
    writeln('Gennemsnittet er: ', gnSnit )
end;

begin
    x := 5; y := 7;
    printGennemsnit(x, 6, y);
    printGennemsnit(y, 8, 9)
end.
```

Eksempel: Procedure med variabelparametre.

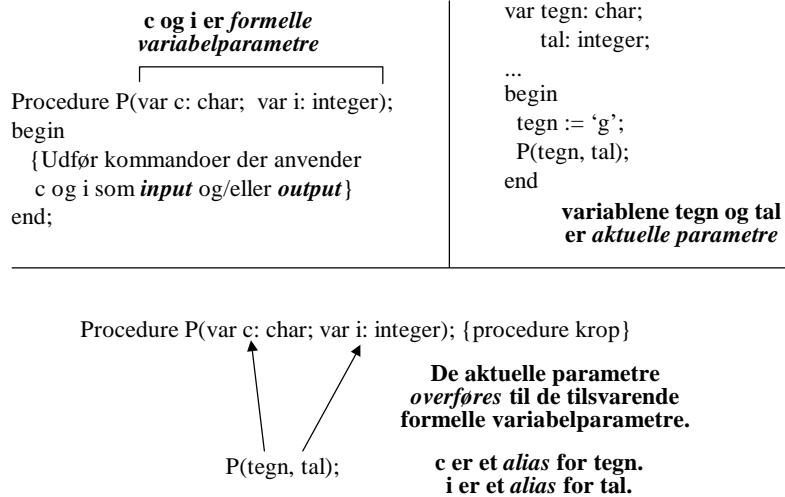
```
tmp := x;
x := y;
y := tmp;
...
ttt := a;
a := x;
x := ttt;
...
temp := første;
første := anden;
anden := temp;
```



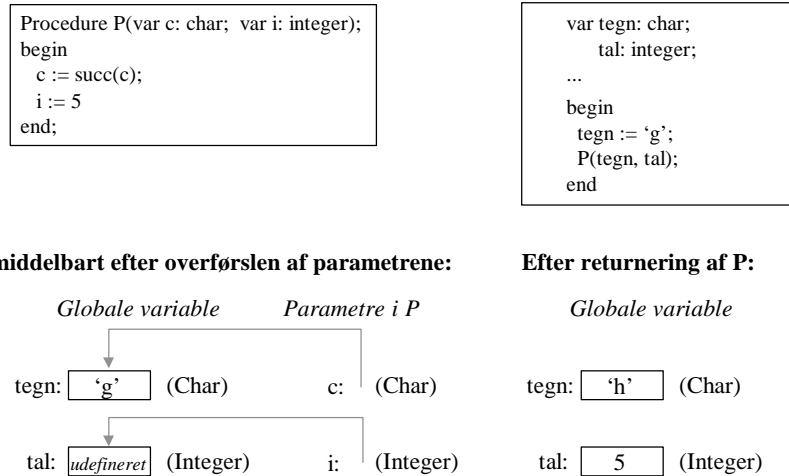
```
ombyt(x,y);
...
ombyt(a, x);
...
ombyt(første, anden);
...
```

```
Procedure ombyt(Var x, y: Real);
{Ombyt værdierne af variablene x og y}
var temp: Real;
begin
    temp := x; x := y; y := temp
end;
```

Overførsel af variabelparametre (1).



Overførsel af variabelparametre (2).



Regler for overførsel af variabelparametre.

Procedure P(var fp ₁ : T ₁ ; var fp ₂ : T ₂ ; ... ; var fp _n : T _n); begin ... end;	P(ap ₁ , ap ₂ , ..., ap _n)
---	--

- Hver aktuel parameter skal være en *variabel*.
- Første aktuelle parameter overføres til første formelle parameter, anden aktuelle parameter overføres til anden formelle parameter, osv.
- Antallet af aktuelle parametre skal være lig med antallet af formelle parametre.
- Typen af en aktuel parameter skal være identisk med typen af den tilsvarende formelle parameter.
- Procedurens krop udføres.
 - I kroppen er det formelle parameternavn fp_i et *alternativt navn* (et alias) for den tilsvarende aktuelle variabelparameter ap_i.
 - Et assignment til fp_i udgør et assignment til ap_i.

Eksempel: Løsning af en andengradslikning.

```
procedure LoesAndengradsLigning(a,b,c: Real; {input}
                               var AntalLoesninger: Integer; {output}
                               var loes1, loes2: Real {output});
var determinant: Real;
begin
  determinant := b*b-4*a*c;
  if determinant < 0
  then AntalLoesninger := 0
  else if determinant = 0
  then begin
    AntalLoesninger := 1;
    loes1 := -b/(2*a)
  end
  else begin
    AntalLoesninger := 2;
    loes1 := (-b + sqrt(determinant))/(2*a);
    loes2 := (-b - sqrt(determinant))/(2*a)
  end
end;
```

Eksempel: Sortering af tre tal.

```
program Sorter3;
  var Num1, Num2, Num3 : Real;

  procedure Ombyt(var X,Y {input/output} : Real); { ... }

  procedure Order (var X, Y {input/output} : Real);
  {Pre : X and Y er tilskrevet værdier.
  Post: X er det mindste af de to tal, og Y det største.}
  var Temp : Real;
  begin
    if X > Y then
      Ombyt(X,Y)
  end; {Order}

begin {Sort3Numbers}
  ReadLn (Num1, Num2, Num3);
  Order (Num1, Num2); Order (Num1, Num3); Order (Num2, Num3);
  WriteLn (Num1 :8:2, Num2 :8:2, Num3 :8:2);
end.
```

Basis 97: Programmering i Pascal

93

Scope regler.

Scopereglerne bruges til at svare på spørgsmålet:

I hvilke dele af programmet er et navn på en variable, en konstant eller en procedure gyldig?

For hver erklæring i et program kan der udpeges et område af programmet, som dækkes af erklæringen: *Scopet* af det erklærede navn.

Scoperegel 1: Scope af et navn er det program eller den procedure, hvori det er erklæret.

Scoperegel 2: Et navn refererer altid til den inderste erklæring af navnet.

```
Program ScopeProgram;
  var i, j, k: integer;

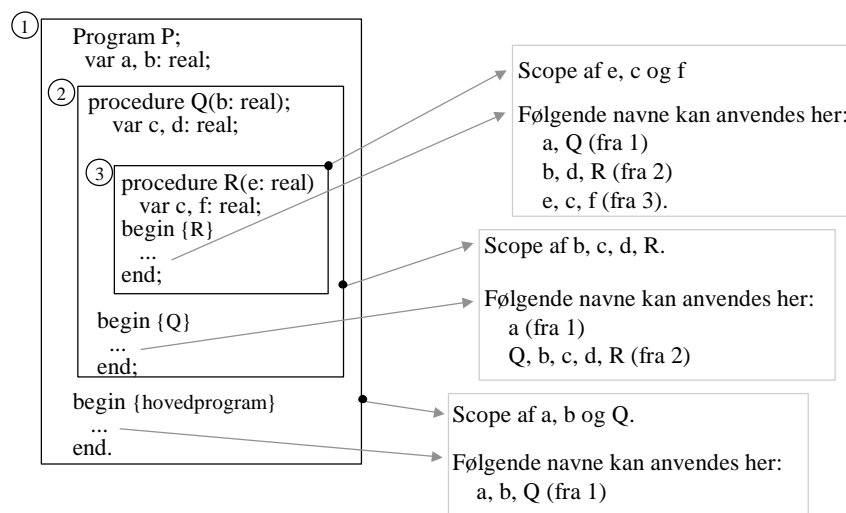
  procedure P(a, j: integer);
  var b, k: integer;
  begin {lokalt sted} end; {p}

begin
  {globalt sted}
end.
```

Basis 97: Programmering i Pascal

94

Scoperegler og indlejrede procedurer.



Basis 97: Programmering i Pascal

95

Funktioner i Pascal.

Ideelt set bør kroppen af en funktion være et udtryk.

Reelt er kroppen af en Pascal funktion en sekvens af kommandoer, som kan tilskrive funktionsnavnet en værdi (funktionens værdi).

Parameter- og scopeforhold af funktioner er præcist som for procedurer.

Eksempel:

```
function gennemsnit_af_3(a, b, c: real): real;  
(* returner gennemsnittet af a, b og c *)  
var sum: real;  
begin  
  sum := a + b + c;  
  gennemsnit_af_3 := sum/3  
end;
```

Basis 97: Programmering i Pascal

96

Programmeringsstil.

- **Undgå sideeffekter fra procedurer.**
 - En sideeffekt er ændring på en global variabel fra en procedure uden om procedurens parametre.
 - Det er dårlig stil at skrive procedurer med sideeffekter.
- **Undgå “effekter” (tilstandsforandring) fra en funktion**
 - Fasthold ideen om, at en funktion *begrebsmæssigt* er en abstraktion over et udtryk.
 - Undgå at have variabelparametre i funktioner.
- **Undgå at lave assignments til værdiparametre.**
 - Værdiparametre formidler input til procedurer og funktioner. Man bør ikke lave om på (assigne til) inputtet af en abstraktion inde fra abstraktionen selv.
 - Overfør parametrene til lokale variable, og foretag assignments på de lokale variable.
- **Undgå for dyb indlejring af procedurer i procedurer.**
 - Dyb indlejring er svær at overskue.
 - Dyb indlejring hæmmer genbrugeligheden af de indlejrede procedurer.

7. Datatyper og filer.

Ordinaltyper
Tegn og tekststreng.
Intervaltyper.
Nummereringstyper.
Filer i Pascal.

Repetition af ordinaltyper.

I en ordinaltype kan man tale om 'forrige værdi' og 'efterfølgende værdi'.
Typerne Integer, Char og Boolean er ordinaltyper.
Intervaltyper og nummereringstyper er også ordinaltyper.

Funktioner knyttet til en ordinaltype T:

```
succ: T -> T
pred: T -> T
ord: T -> Integer
chr: Integer -> Char
```

Program eksempel med funktionerne:

```
program ordinalProgram;
var ch: char;
    i: integer;
begin
  ch := 'k';
  writeln(succ(ch));
  ch := pred(ch);
  i := ord(ch);
  writeln(chr(i+1))
end.
```

Tegn og tekststreng.

- Datatypen **Char** i Turbo Pascal:
 - En ordinaltype med 128 tegn.
 - Tegnene i Char er ordnede indbyrdes.
 - Notation: 'a', '3', ' '.
 - Standardiseret alfabet: ASCII tegnsættet (jf Koffman, appendix D side 792)
 - Bogstaver, cifre, specialtegn og kontroltegn.
- Datatypen **String** i Turbo Pascal (tekststreng):
 - Et array (en tabel) af tegn fra typen Char.
 - Notation: 'En streng'.
 - Ordningen mellem tegn giver os en ordning af tekststreng.
 - Leksikografisk ordning.
 - Eksempler:
 - 'abe' < 'zebra' 'abe' < 'abekat'

Mere om tekststreng i lektion 8

Intervaltyper (subrange types).

I Pascal er det muligt at definere et interval af en ordinaltype (heltal og tegn). Der er således muligt at definere nye typer ud fra eksisterende typer.

Eksempel:

```
Type ciffer_numbers = 0 .. 9;  
    ciffer_chars = '0' .. '9'
```

- Brug af navngivne intervaltyper giver ofte mere læselige og forståelige programmer.
- Programmet kan afbrydes hvis en variabel falder uden for sin intervaltype (range error).
- Intervaltyper er nødvendige i forbindelse med definition af tabeller (arrays).

Eksempel på intervaltyper.

```
{SR+}  
Program subrange_ex;  
Type ciffer_numbers = 0 .. 9;  
    ciffer_chars = '0' .. '9';  
Var c1: ciffer_numbers;  
    c2: ciffer_chars;  
    i: integer;  
  
function ciffer_to_char(ciffer: ciffer_numbers): ciffer_chars;  
{ Returner ciffer tegn fra ciffer tal. Eksempelvis: 1 -> '1' }  
begin  
    ciffer_to_char := chr(ciffer + ord('0'));  
end;  
  
begin  
    i := 7;  
    writeln(ciffer_to_char(7));  
    writeln(ciffer_to_char(10)); { kompileringstidsfejl }  
    writeln(ciffer_to_char(i + 3)); { køretidsfejl }  
end.
```

Nummereringstyper (enumeration types).

En nummereringstype er en ordinaltype, hvor vi eksplicit opregner typens værdier. Værdierne i en nummereringstype er konstanter, som introduceres til lejligheden. Værdierne i en nummereringstype kan ikke umiddelbart indlæses (via read) eller udskrives (via write).

```
Type maaned= (januar, februar, marts, april, maj, juni, juli,
              august, september, oktober, november, december);
              maanedinterval = 1..31;

function antalDageIMaaned (m: maaned; skudaar: boolean): maanedinterval;
begin
  case m of
    januar, marts, maj, juli, august, oktober, december: antalDageIMaaned := 31;
    april, juni, september, november:                  antalDageIMaaned := 30;
    februar: if skudaar
              then antalDageIMaaned := 29
              else antalDageIMaaned := 28
    else Halt
  end
end;
```

Basis 97: Programmering i Pascal

103

Nummereringstyper: Måneder.

```
Program maanednavne;

Type maaned= (januar, februar, marts, april, maj, juni, juli,
              august, september, oktober, november, december);
              maanedinterval = 1..31;

var m: maaned;

function antalDageIMaaned (m: maaned; skudaar: boolean): maanedinterval;
begin { se forrige slide } end;

function naesteMaaned(m: maaned): maaned;
begin
  if m < december then naesteMaaned := succ(m) else naesteMaaned := januar
end;

begin
  m := december;
  writeln('December har ', antalDageIMaaned(m, false), ' dage');
  writeln('Januar har ', antalDageIMaaned(naesteMaaned(m), false), ' dage')
end.
```

Basis 97: Programmering i Pascal

104

Nummereringstyper: Karakterer.

```
Program enumEx;
Type simpel_karakter = (bestaaet, ikke_bestaaet);
   tal_karakter = (nul_nul, nul_tre, fem, seks, syv, otte, ni, ti, elleve, tretten);

function omsaet_karakter(k: tal_karakter): simpel_karakter;
begin
  if k <= fem
  then omsaet_karakter := ikke_bestaaet
  else omsaet_karakter := bestaaet
end;

procedure print_karakter(k: simpel_karakter);
begin
  case k of
    bestaaet: writeln('bestaaet');
    ikke_bestaaet: writeln('ikke_bestaaet')
  end
end;

begin
  print_karakter(omsaet_karakter(syv));
end.
```

Basis 97: Programmering i Pascal

105

Brugerdefinerede typer ift. andre definitioner.

Vi har set, at det i Pascal er muligt at definere nye navngivne typer.

Rækkefølgen af definitioner

konstanter, typer, variable, procedurer/funktioner
er den "naturlige" rækkefølge at introducere erklæringerne.

```
Program brugerTyper(output);
  Const lower = 0;
         upper = 9;
  Type  ciffers = lower .. upper;
  Var  ciffer: ciffers;

  procedure p(ciffer: ciffers);
  begin
    {...}
  end;

begin { hovedprogram }
  ciffer := 5;
  p(ciffer)
end.
```

- *Konstanter bruges i typedefinitioner.*
- *Typer bruges i variabelerklæringer.*
- *Variable kan bruges i procedurer og funktioner ifølge scopereglerne.*

Basis 97: Programmering i Pascal

106

Filer i Pascal.

- **Tekstfiler.**
 - Filer som indeholder tegn fra ASCII alfabetet.
 - Tal repræsenteres som en række cifre, der hver i sær er tegn.
 - Filrepræsentationen kan håndteres af en teksteditor.
- **Binære filer.**
 - Filer som indeholder bitmønstre for andre datatyper end tegn.
 - Tal repræsenteres direkte i det binære talsystem.
 - Kompakt repræsentation.
 - Benyttes ofte til at repræsentere filer med poster (records).

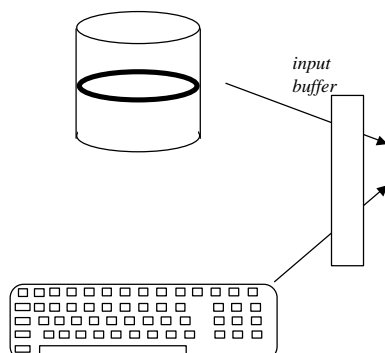
Filer i Pascal er modelleret efter egenskaberne ved magnetbånd:

- Data tilgås sekventielt.
- Man er enten i *læsetilstand* eller *skrivetilstand*.

Overordnet model for input fra filer i Pascal.

DOS

c:\katalog\inndata.dat



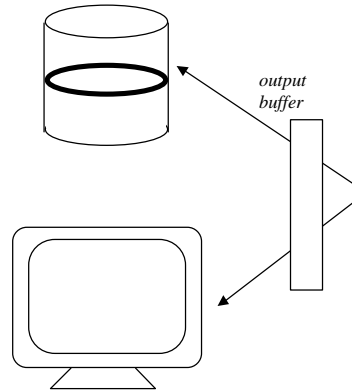
Turbo Pascal

```
var inputData: Text;  
    i: integer;  
    r: real;  
begin  
    assign(inputData, 'c:\katalog\inndata.dat');  
    reset(inputData);  
    ...  
    read(inputData, r, i);  
    readln(inputData);  
    ...  
    close(inputData)  
end
```

Overordnet model for output til filer i Pascal.

DOS

c:\katalog\uddata.dat



Turbo Pascal

```
var outputData: Text;
    i: integer;
    r: real;
begin
    assign(outputData, 'c:\katalog\uddata.dat');
    rewrite(outputData);
    ...
    write(outputData, r, i);
    writeln(outputData);
    ...
    close(outputData)
end
```

Oversigt over procedurer og funktioner, der arbejder på filer: Læsning og skrivning.

Read(*inputfil*, *input liste*)

Readln(*inputfil*, *input liste*)

Første parameter er af typen Text.

Læser fra *inputfil* ligesom Read læser fra tastaturet.

Alle parametre er variabelparametre.

Write(*outputfil*, *output liste*)

Writeln(*outputfil*, *output liste*)

Første parameter er af typen Text.

Skriver på *outputfil* ligesom Write skriver til skærmen.

Første parameter er en variabelparameter.

Filposition før reads: ▼

Her er første linie <eol>Her er anden linie<eoln>123 456.789

c1, c2, c3: Char;

Read(inputfil, c1, c2, c3); Readln(inputfil)

c1 = 'H', c2 = 'e', c3 = 'r'

Filposition efter reads: ▼

Her er første linie <eol>Her er anden linie<eoln>123 456.789

Oversigt over procedurer og funktioner, der arbejder på filer: Associering, åbning og lukning.

Assign(<i>filvariabel</i> , <i>filsti</i>)	Associerer en <i>ekstern</i> DOS fil med en <i>intern</i> Pascal fil. Hvis <i>filsti</i> er tom (‘’) associeres til enten input (tastatur) eller output (skærm).
Reset(<i>filvariabel</i>)	Forbered læsning af en fil. “Spol filen tilbage” til start.
Rewrite(<i>filvariable</i>)	Forbered skrivning på filen. Nulstil filen (slet alt).
Close(<i>filvariabel</i>)	Afbryd associeringen mellem <i>filvariabel</i> og DOS filen. Ved skrivning: Tøm output bufferen.

Oversigt over procedurer og funktioner, der arbejder på filer: Boolske funktioner.

Eof(<i>filvariabel</i>)	Returner hvorvidt det næste tegn i filen er ‘end of fil’ tegnet.
Eoln(<i>filvariable</i>)	Returner hvorvidt det næste tegn i filen er et ‘end of line’ tegn.

Eksempel.

```
program SquareAndRoot;
var InFile,           {input file}
    OutFile : Text;   {output file}
    InFileName : string; {directory name of input file}
    NextNum : Real;    {next number}

begin
  Write ('Input file name> '); ReadLn (InFileName);
  Assign (InFile, InFileName); Reset (InFile);

  Assign (OutFile, 'SQUARES.TXT'); Rewrite (OutFile);

  WriteLn (OutFile, 'N' :10, 'Square root':15,'Square' :15);

  while not EOF(InFile) do
  begin
    ReadLn (InFile, NextNum);
    WriteLn (OutFile, NextNum :10:2, Sqrt(NextNum) :15:2, Sqr(NextNum) :15:2)
  end;

  Close (InFile); Close (OutFile);
  WriteLn ('Table of roots and squares is in file SQUARES.TXT.')
end.
```

8. Datastrukturer: Arrays og Records.

Arrays (tabeller).
Tekststreng.
Arrays af flere dimensioner.
Records (poster).
Tabeller af poster.

Introduktion til arrays.

Hvis man eksempelvis skal sortere 10 tal i stigende orden er det alt for omstændeligt at erklære 10 forskellige variable til disse tal.

```
var tal1, tal2, tal3, tal4, tal5, tal6, tal7, tal8, tal9, tal10: Real;

procedure sorter(var tal1, tal2, tal3, tal4, tal5, tal6, tal7, tal8,
                 tal9, tal10: Real)
begin {sorter de 10 tal på en eller anden måde} end;

begin
  sorter(tal1, tal2, tal3, tal4, tal5, tal6, tal7, tal8, tal9, tal10)
end
```

tal1: (Real)
 tal2: (Real)
 ⋮
 tal10: (Real)

```
type talTabel = array[1..10] of real;
var tal: talTabel;

procedure sorter(var tal: talTabel)
begin {sorter de 10 tal på en eller anden måde} end;

begin
  sorter(tal)
end
```

tal: (talTabel)
 [1] (Real)
 [2] (Real)
 [3]
 ⋮
 [10] (Real)

Array begrebet (tabeller).

Et array er en tabel med et antal *nummererede elementer* af samme datatype.

*En ordinaltype,
typisk et interval*

En vilkårlig type

Type TabelType = **array**[index_type] of element_type;

Var Tabel: TabelType;

Indekser → i_1 i_2 i_3 i_4 i_5 i_6 i_7
 Elementer → el_1 el_2 el_3 el_4 el_5 el_6 el_7

Aflæsning af et element fra en tabel:

Tabel[i_3]

Udtryk

Skrivning af et element i en tabel:

Tabel[i_4] := nyt_element

Tabellægning af faktetetsfunktionen (1).

```
Program FaktetetsProgram;
{$Q+,R+} {overflow check, range check}
uses crt;

Const Sidste = 12;
Type IndexType = 1 .. Sidste;
   HeltalsTabel = array[IndexType] of Longint;
Var Faktet: HeltalsTabel;
   n: integer;

Procedure LavFakultetsTabel(indtil: IndexType; var FT: HeltalsTabel);
var i: Integer;
begin
  FT[1] := 1;
  for i := 2 to indtil do FT[i] := FT[i-1] * i
end;

begin
  LavFakultetsTabel(Sidste, Faktet);
  repeat
    write('Opslå fakultet af (0 afslutter): '); readln(n);
    if n > 0 then writeln(Fakultet[n])
  until n = 0
end.
```

Basis 97: Programmering i Pascal

117

Optælling af tegn i en tekstfil.

```
program TegnTaeller;
uses Crt;

Type TegnTaellerTabel = array[Char] of integer;

Var InputFil: Text;
   AntalForekomster: TegnTaellerTabel;
   ch: Char;

begin
  clrscr;
  Assign(InputFil, 'tegnantal.pas'); Reset(InputFil);

  for ch := chr(0) to chr(127) do AntalForekomster[ch] := 0;

  while not eof(InputFil)
  do begin
    read(InputFil, ch);
    AntalForekomster[ch] := AntalForekomster[ch] + 1
  end;

  for ch := '0' to 'z' do
    begin writeln(ch: 2, AntalForekomster[ch]:3); readkey end
end.
```

Basis 97: Programmering i Pascal

118

Tekststreng.

- En tekststreng er i princippet et array af typen `array[1..længde] of char` med
- en særlig notation som tillader at “notere en streng” direkte.
 - en ordning som tillader os at sammenligne to streng med '<' og '>'.
 - et antal nyttige operationer, som virker på streng.

```
Program strengEksempel;
uses Crt;
```

```
Type Navn = String[15];
```

```
Var fornavn: Navn;
    efternavn: Navn;
    fuldenavn: String[30];
```

```
procedure MedStort(var n: Navn);
begin
    n[1] := Uppcase(n[1]);
end;
```

```
begin {hovedprogram}
    ClrScr;
    fornavn := 'peter'; efternavn := 'petersen';
    MedStort(fornavn); MedStort(efternavn);
    fuldenavn := Concat(fornavn, ' ', efternavn);
    writeln(fuldenavn, ': ', length(fuldenavn):3, ' tegn');
    ReadKey
end.
```

Arrays af flere dimensioner.

Elementtypen af et array kan selv være en array-type.

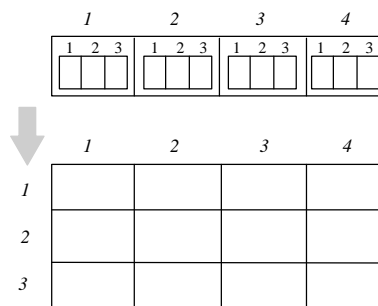
```
Type Matrix = array[1 .. 4] of
    array[1 .. 3] of Real;
```

```
Var M: Matrix;
    r: Real;
```

```
begin
    r := M[4][3];

    M[4][3] := 2 * r
end.
```

```
array[1 .. 4] of array[1 .. 3] of Real
array[1 .. 4, 1 .. 3] of Real
```



$M[i][j] \longleftrightarrow M[i, j]$

Introduktion til records.

Der er ofte behov for at binde en mængde variable af forskellige typer sammen til en *logisk helhed*.

```
var fornavn, efternavn: Navn;
    alder: integer;
    cpr: cprNr;
```

```
procedure indlaesPerson(var fornavn, efternavn: Navn;
                        var alder: integer; var cpr: CprNummer);
begin ... end;
```

fornavn: (Navn)

efternavn: (Navn)

alder: (Integer)

cpr: (cprNr)

```
type personType =
  record
    fornavn, efternavn: Navn;
    alder: integer;
    cpr: cprNr;
  end;
var person: PersonType;

procedure indlaesPerson(var p: personType);
begin ... end;
```

person: (personType)

fornavn: (Navn)

efternavn: (Navn)

alder: (Integer)

cpr: (cprNr)

Record begrebet (poster).

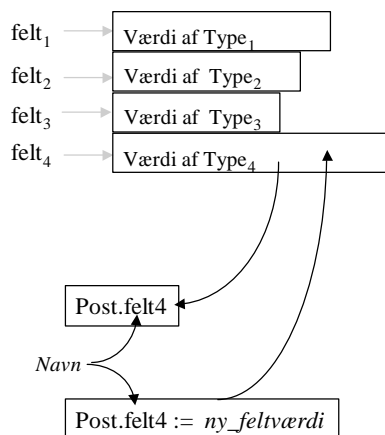
Et record er en datastruktur med et antal *navngivne felter* som kan være af forskellige datatyper.

```
Type PostType =
  record
    felt1: Type1;
    felt2: Type2;
    ...
    feltn: Typen;
  end;
```

```
Var Post: PostType;
```

Aflæsning af felt fra en post:

Skrivning af felt i en post:



Eksempel: Datoer.

```
Type
Maanedsdag = 1..31;
MaanedNavn = (januar, februar, marts, april, maj, juni, juli,
              august, september, oktober, november, december);
DagIUge = (mandag, tirsdag, onsdag, torsdag, fredag, loerdag, soendag);
Aarstal = Integer;
```

```
Dato = Record
      ugedag: DagIUge;
      dag: Maanedsdag;
      maaned: MaanedNavn;
      aar: Aarstal;
End;
```

```
function tidligere(d1,d2: Dato): boolean;
{Returner hvorvidt d1 er en tidligere dato end d2}
begin
  tidligere := (d1.aar < d2.aar) or
              ((d1.aar = d2.aar) and (d1.maaned < d2.maaned)) or
              ((d1.aar = d2.aar) and (d1.maaned = d2.maaned) and (d1.dag < d2.dag))
end;
```

Eksempel: Data om en bog (1).

```
Type Bog =
  record
    titel, forfatter, forlag: String[40];
    udgivelsesaar: Integer;
    laerebog: Boolean;
  end;
```

```
var Koffman: Bog;
```

```
procedure read_bog(var b: Bog);
begin
  write('titel '); readln(b.titel);
  write('forfatter '); readln(b.forfatter);
  write('forlag '); readln(b.forlag);
  write('udgivelsesaar '); readln(b.udgivelsesaar);
  ask_user('laerebog? ', b.laerebog);
end;
```

fortsættes...

Eksempel: Data om en bog (2).

```
procedure write_bog(b: Bog);
begin
  write('titel: '); writeln(b.titel);
  write('forfatter: '); writeln(b.forfatter);
  write('forlag: '); writeln(b.forlag);
  write('udgivelsesaar: '); writeln(b.udgivelsesaar);
  write('laerebog: '); if b.laerebog then writeln('Ja') else writeln('Nej')
end;

begin
  read_bog(Koffman); write_bog(Koffman)
end.
```

Tabeller af poster: Et bibliotek.

I mange praktiske sammenhænge er det nyttigt at lave tabeller som indeholder et antal poster.

```
Program biblioteksProgram;
const MaxAntalBoeger = 25;

type Bog =
  record
    titel, forfatter, forlag: String[40];
    udgivelsesaar: Integer;
    laerebog: Boolean;
  end;

  Bibliotek =
    array[1 .. MaxAntalBoeger] of Bog;

var boghylde: Bibliotek;
    i: integer;
    fortsaet: boolean;

begin {hovedprogram}
  fortsaet := true; i := 1;
  while fortsaet
  do begin
    read_bog(Boghylde[i]);
    i := i + 1;
    ask_user('Fortsat? ', fortsaet);
  end;
  {...}
end.
```

Tabeller af poster: En kalender (1).

```
Const Antal_dage_i_kalender = 100; { Antal dage, som kalenderen kan indeholde }

Type Dage = 1 .. Antal_dage_I_Kalender;
    Kalender = array[Dage] of Dato;

Var Kal: Kalender;
    d, start, slut: Dage;
    startDato, slutDato: dato;

procedure lav_kalender(fraDato, tilDato: Dato; var k: Kalender);
{Prebetingelse: fraDato er tidligere end tilDato.
 Der er ikke flere dage mellem fraDato og tilDato end Antal_Dage_I_Kalender.}
var d: Integer;
    denne_dato, naeste_dato: Dato;
begin
    d := 1; denne_dato := fraDato;
    while tidligere(denne_dato, tildato)
    do begin
        k[d] := denne_dato; d := succ(d);
        imorgen(denne_dato, naeste_dato); { Oevelse! }
        denne_dato := naeste_dato
    end;
end;
```

Tabeller af poster: En kalender (2).

```
begin {Hovedprogram}
    writeln('Startdato'); laesDato(startDato); {Indlæser en dato - ikke def. på slides}
    writeln('Slutdato'); laesDato(slutDato);
    tildeUgeDag(startDato); {bestemmer ugedagen af startdato - ikke def. på slides}

    lav_kalender(startDato, slutDato, Kal);

    writeln('Kalenderudskrivning, hvilken start nummer'); readln(start);
    writeln('Kalenderudskrivning, hvilken slut nummer'); readln(slut);
    for d := start to slut do
        dato_udskriv(Kal[d]); {udskriver en dato paa skaermen - ikke def. på slides}
    end.
```

9. Dataabstraktion.

Repetition af datatyper.
Abstrakte datatyper.
Fordele ved dataabstraktion.
Eksempel: Den abstrakte datatype Punkt.
Units i Turbo Pascal.
På vej mod objekt-orienteret programmering.

Repetition af abstraktion med procedurer.

- En procedure er en *abstraktion* over en sekvens af kommandoer.
 - Proceduren *indkapsler* en mængde detaljerede kommandoer, som efter proceduredefinitionen kan refereres gennem proceduren's navn.
 - Procedurens navn og parametre udgør *grænsefladen* til de dele af programmet, som kalder proceduren.
 - En *præbetingelse* er et udsagn, der udtrykker forudsætninger der skal opfyldes forud for procedurens kald.
 - En *postbetingelse* er et udsagn, der karakteriserer programtilstanden efter at proceduren er kørt til ende.

```
Procedure Navn(parametre);  
{Præbetingelse: Udsagn}  
begin  
  Krop  
{Postbetingelse: Udsagn}  
end;
```

```
Procedure squareRoot(x: real; var res: real);  
{Præbetingelse: x >= 0}  
begin  
  ...  
{Postbetingelse: res * res = x}  
end;
```

Datatyper.

En datatype beskriver en mængde af værdier.

Eksempel:

Integer = {-32768, -32767, ..., -2, -1, 0, 1, 2, ... 32767}
Boolean = {false, true}

I Pascal er det muligt at definere nye typer:

- Synonymer
- Intervaltyper
- Nummereringstyper

- Tabeller (arrays)
- Poster (records)

```
Type Årstal = Integer;  
Cifre = '0' .. '9';  
Maanednavn = (jan, feb, mar, apr, maj,  
              jun, jul, aug, sept, okt,  
              nov, dec);  
CifferTabel = array[1..10] of Cifre;  
Dato = record  
    dag: 1 .. 31;  
    maaned: Maanednavn;  
    år: Årstal;  
end;
```

Datatyper og operationer.

De indbyggede datatyper i Pascal 'er født med' en samling af operationer, som tillader os at arbejde på typens værdier.

Enhver konstruktion, forandring eller aflæsning af egenskaber sker i princippet gennem én af datatypens operationer (i bred forstand).

Vi har ikke adgang til hvorledes datatypens værdier lagres inden i maskinen.

Eksempel: Heltal af typen Integer.

Maskinens lager

1101
0101
1001

Værdi-notation og operationer.

13	+	*
5	-	
9	div	mod
	succ	pred

Abstrakte datatyper.

En abstrakt datatype (ADT) er

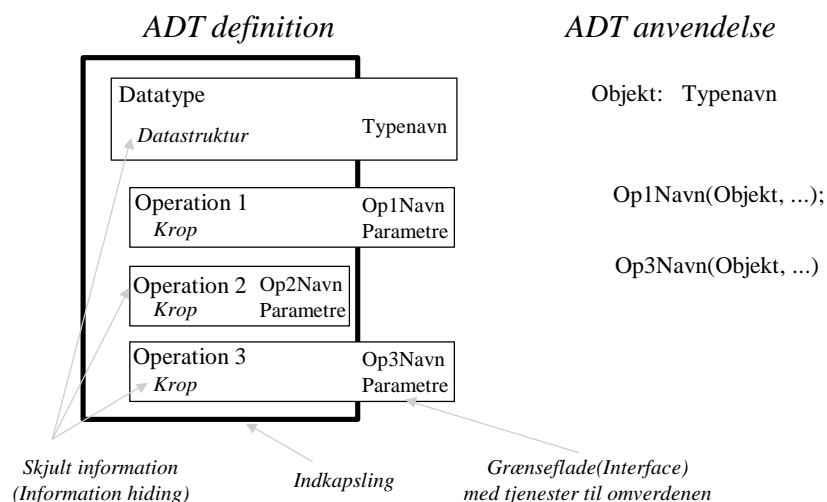
- en mængde af data-objekter.
- en samling af operationer (procedurer og funktioner), som arbejder på disse data-objekter via parametre.

Data-objekterne kan være konkrete eller abstrakte hverdags-ting, som vi ønsker at behandle i et program.

Eksempel: *Studenter i et registreringsprogram på AAU.*

ADT:	Mængden af studerende på AAU.
Et data-objekt:	En bestemt studerende.
Operationer:	NyStuderende(navn, adresse, alder): Studerende Alder(studerende): Integer StudenterId(studerende): Nummer ErFærdigUddannet(Studerende): Boolean RegistrerEksamen(Studerende, Eksamen, Resultat)

Definition og anvendelse af abstrakte datatyper.



Eksempel på definition og anvendelse af abstrakte datatyper.

ADT definition

```
Type Studerende =  
  Record  
  Navn: String;  
  Fødselsdato: Dato;  
  EksamensRes: ResultatTabel;  
End
```

```
Procedure NyStuderende  
  (var s: studerende, n: String, fd: Dato);  
begin  
  s.Navn := n; s.Fødselsdato := fd;  
end;
```

```
Function ErFærdig  
  (var S: Studerende): Boolean;  
begin  
  { gennemløb S.EksamensRes }  
end;
```

ADT anvendelse

```
Var Jens: Studerende;  
    FD: Dato;  
Begin  
  NyDato(FD, 5, 11, 95);  
  NyStuderende(Jens, 'Jens', FD);  
  
  If ErFærdig(Jens)  
  then writeln('Tillykke')  
  else writeln('Tag endnu et semester')  
  
End
```

Fordele ved dataabstraktion.

- Datastrukturen af et objekt (datarepræsentationen) er ukendt for den programmør, som anvender datatypen.
- Det er muligt at ændre på detaljer i en datastruktur uden at skulle ændre i hele programmet.
 - Det er nok at ændre lokalt i den abstrakte datatype.
- Den abstrakte datatype virker som en **brandmur** omkring datarepræsentationen.
- Objekter i den abstrakte datatype manipuleres udelukkende gennem de operationer, som udgør grænsefladen af typen.
 - Det er muligt at forstå en operation på typen uden kendskab til detaljer i kroppen af operationen.
- Ved brug af abstrakte datatyper er vi selv i stand til at lave typer, der har tilsvarende egenskaber som de typer, Pascal er 'født med'.
- ADT-er virker som **program-byggeklodser**, der kan bruges og genbruges i flere forskellige sammenhænge.
- Objekter i en ADT udbyder en samling af passive **tjenester**, der skal aktiveres af det omkringliggende program for at være nyttige.

Eksempel: Den abstrakte datatype Punkt (1).

```
{ ***** ADT Punkt ***** }

{ Et punkt i et sædvanligt 2-dimensionelt koordinatsystem }
type Punkt = record
    x, y: Real;
end;

procedure nytPunkt(var P: Punkt; x_koord, y_koord: Real);
{ tilskriv koordinater af punktet P }
begin P.x := x_koord; P.y := y_koord end;

function xKoordinat(var P: Punkt): Real;
{ aflæs og returner x-koordinaten af P }
begin xKoordinat := P.x end;

function yKoordinat(var P: Punkt): Real;
{ aflæs og returner x-koordinaten af P }
begin yKoordinat := P.y end;

procedure flytPunkt(var P: punkt; deltaX, deltaY: Real);
{ forskyd punktet P med deltaX i x-retning og deltaY i y-retning }
begin P.x := P.x + deltaX; P.y := P.y + deltaY end;
```

fortsættes ...

Eksempel: Den abstrakte datatype Punkt (2).

```
procedure drejPunkt(var P: punkt; vinkel: Real);
{ drej punktet P om nulpunktet med vinklen vinkel mod urets omløb. }
begin ... end;

function afstand(var P, Q: Punkt): Real;
{ Returner afstanden mellem P og Q }
begin
    afstand := sqrt(sqrt(abs(P.x - Q.x)) + sqrt(abs(P.y - Q.y)))
end;

function afstandTilNulpunkt(var P:Punkt): Real;
{ Returnerer afstanden mellem P og koordinatsystemets nulpunkt }
var Nulpunkt: Punkt;
begin
    nytPunkt(Nulpunkt, 0.0, 0.0);
    afstandTilNulpunkt := afstand(Nulpunkt, P);
end;

{ ***** Slut på ADT Punkt ***** }
```

Eksempel: Anvendelse af den abstrakte datatype Punkt.

```
var A, B: Punkt;

begin
  nytPunkt(A, 3.0, 1.0);
  nytPunkt(B, 3.0, 1.0);
  flytPunkt(B, 3.0, 4.0);
  writeln('B er placeret i (' , xKoordinat(B):4:1, ', ', yKoordinat(B):4:1, ')');
  writeln('A er placeret i (' , xKoordinat(A):4:1, ', ', yKoordinat(A):4:1, ')');
  writeln('Afstand mellem A og B er ', afstand(A, B):4:1);
end.
```

Eksempel: Ændring af Punkt's repræsentation (1).

```
{ ***** ADT Punkt ***** }
type Punkt = record { Et punkt i polære koordinater }
  radius, vinkel: Real;
end;

function vinkel(x, y: Real): Real;
{ Beregner vinklen, i grader, af punktet (x,y) ift. x-aksen. Intern operation. }
begin ... end;

procedure nytPunkt(var P: Punkt; x_koord, y_koord: Real);
{ tilskriv koordinater af punktet P }
begin
  P.radius := sqrt(sqrt(x_koord) + sqrt(y_koord));
  P.vinkel := vinkel(x_koord, y_koord);
end;

function xKoordinat(var P: Punkt): Real;
{ aflæs og returner x-koordinaten af P }
begin xKoordinat := P.radius * cos(P.vinkel) end;

function yKoordinat(var P: Punkt): Real;
{ aflæs og returner y-koordinaten af P }
begin yKoordinat := P.radius * sin(P.vinkel) end;
```

fortsættes ...

Eksempel: Ændring af Punkt's repræsentation (2).

```
procedure flytPunkt(var P: punkt; deltaX, deltaY: Real);
{ forskyd punktet P med deltaX i x-retning og deltaY i y-retning }
begin ... end;

procedure drejPunkt(var P: punkt; vinkel: Real);
{ drej punktet P om nulpunktet med vinklen vinkel mod urets omløb. }
begin P.vinkel := P.vinkel + vinkel end;

function afstand(var P, Q: Punkt): Real;
{ Returner afstanden mellem P og Q }
begin ... end;

function afstandTilNulpunkt(var P: Punkt): Real;
{ Returnerer afstanden mellem P og koordinatsystemets nulpunkt }
var Nulpunkt: Punkt;
begin
  afstandTilNulpunkt := P.radius;
end;

{ ***** Slut på ADT Punkt ***** }
```

Basis 97: Programmering i Pascal

141

Dataabstraktion med Turbo Pascal Units.

En **unit** i Turbo Pascal er et modul, som tillader os at indkapsle en mængde af definitioner.

Definitioner i en unit's **interface-del** er synlige udadtil.

Definitioner i en unit's **implementations-del** er usynlige udadtil.

En unit oversættes separat og uafhængig af de programmer, som anvender det.

```
Unit PunktADT;
Interface
  Type Punkt = Record
    x, y: Real;
  End;

  procedure nytPunkt(var P: Punkt; x_koord, y_koord: Real);
  function xKoordinat(var P: Punkt): Real;
  function yKoordinat(var P: Punkt): Real;
  ...

Implementation
  { Fuld definition af alle procedurer og funktioner i interface delen,
  samt definition af interne operationer }
End.
```

Basis 97: Programmering i Pascal

142

Anvendelse af Turbo Pascal units.

```
program punkt_program(input, output);
{ Dette program bruger den abstrakte datatype
  punktADT, som er programmet i en TP unit }

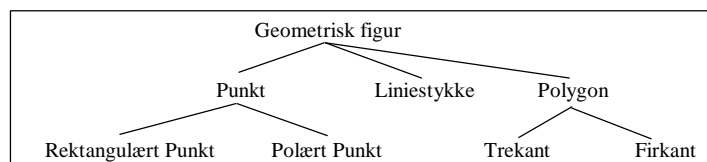
uses punktADT;

var A, B: Punkt;

begin
  nytPunkt(A, 3.0, 1.0);
  nytPunkt(B, 3.0, 1.0);
  flytPunkt(B, 3.0, 4.0);
  writeln('B er placeret i (' , xKoordinat(B):4:1, ',', yKoordinat(B):4:1, ') ');
  writeln('A er placeret i (' , xKoordinat(A):4:1, ',', yKoordinat(A):4:1, ') ');
  writeln('Afstand mellem A og B er ', afstand(A, B):4:1);
end.
```

På vej mod objekt-orienteret programmering.

- I objekt-orienterede programmeringssprog bruges klasser i stedet for moduler ala units fra Turbo Pascal.
- Abstrakte datatyper er fundamentet for ethvert objekt-orienteret programmeringssprog.
- En klasse kan som helhed opfattes som en definition af en abstrakt datatype.
- Et modul (en unit) er en ramme omkring en typedefinition og nogle operationer.
- Klasser kan organiseres i et hierarki, hvor én klasser arver fra en anden klasse.
 - Sådanne klassehierarkier kan afspejle begrebshierarkier, som vi kender dem fra vores hverdag.



Sammenligning af klasser og moduler (units).

```
class Punkt
export nytPunkt, xKoordinat,
      yKoordinat, flytPunkt;

x, y: Real;

nytPunkt(xk,yk: Real) begin ... end;
xKoordinat: Real begin ... end;
yKoordinat: Real begin ... end;
flytPunkt(deltay, deltay: Real)
      begin ... end;
end;
```

```
P: Punkt;
P.nytPunkt(3.0, 1.0);
P.flytPunkt(3.0, 4.0);
```

```
Unit PunktADT;
Interface
Type Punkt = Record
      x, y: Real;
End;

procedure nytPunkt
      (var P: Punkt; x_koord, y_koord: Real);
function xKoordinat(var P: Punkt): Real;
function yKoordinat(var P: Punkt): Real;
procedure flytPunkt(var P: Punkt; deltay, deltay: Real);
...

Implementation
...
End.
```

```
P: Punkt;

nytpunkt(P, 3.0, 1.0);
flytpunkt(P, 3.0, 4.0);
```

10. Rekursion.

Kæder af procedurekald.
Gentagelse ved brug af rekursion.
Styrken af rekursion.
Potensopløftning.
Hanoi's tårne.

En kæde af procedurekald.

```
Program KædeAfKald;

Procedure P1(i: integer);
begin writeln('P', i) end;

Procedure P2(i: integer);
begin writeln('P', i); P1(i-1) end;

Procedure P3(i: integer);
begin writeln('P', i); P2(i-1) end;

begin {hovedprogram}
  P3(3)
end.
```

Et øjebliksbillede af
programudførelsen på det tidspunkt
hvor P1 er aktiveret.

P1: i = 1

P2: i = 2

P3: i = 3

Hoved-
program

Procedurene P_i ligner hinanden så meget, at de kan
erstattes af én procedure P, som kalder sig selv.

Udskrift:

P3
P2
P1

En kæde af rekursive procedurekald.

```
Program RekursiveKald;

Procedure P(i: integer);
begin
  if i >= 1
  then begin
    writeln(i);
    P(i-1)
  end
end;

begin {hovedprogram }
  P(3)
end.
```

Et øjebliksbillede af
programudførelsen på det tidspunkt
hvor P er aktiveret på 0

P: i = 0

P: i = 1

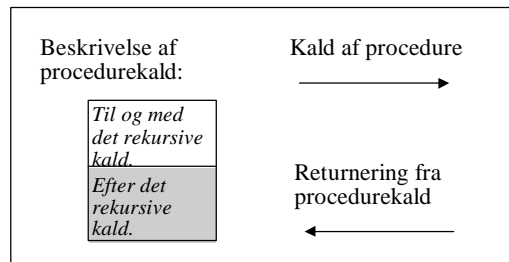
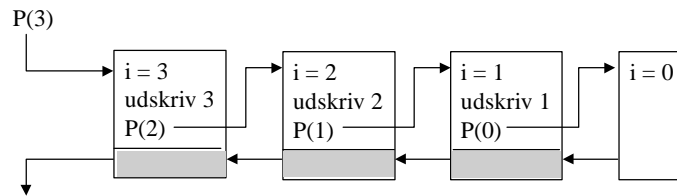
P: i = 2

P: i = 3

Hoved-
program

Proceduren P er *rekursiv* idet den for nogle programtilstande kalder sig selv.
En rekursiv procedure skal altid omfatte en programtilstand, hvor den undlader at
kalde sig selv.

Trace af P.



Gentagelse ved brug af rekursion (1).

Det er muligt at erstatte anvendelsen af gentagende kontrolstrukturer (løkker) med kald af en rekursiv procedure.

Brug af løkke

```

procedure spoerg_brugeren
  (sp: string; var svar: boolean);
var respons: char;
begin
  repeat
    writeln(sp, '. Svar j eller n ');
    readln(respons)
  until (respons = 'j') or (respons = 'n');
  svar := respons = 'j'
end;
  
```

Brug af rekursiv procedure

```

procedure spoerg_brugeren
  (sp: string; var svar: boolean);
var respons: char;
begin
  writeln(sp, '. Svar j eller n ');
  readln(respons);
  if (respons <> 'j') and (respons <> 'n')
  then spoerg_brugeren(concat('Igen:', sp),
                       svar)
  else svar := respons = 'j'
end;
  
```

Gentagelse ved brug af rekursion (2).

```
function find_rod (l, u: real): real;
begin
  while not ( f(midtpunkt(l,u)) = 0 )
  do begin
    if samme_fortegn(f(midtpunkt(l, u)), f(u))
    then u := midtpunkt(l,u)
    else l := midtpunkt(l,u);
    end;
  find_rod:= midtpunkt(l,u);
end;
```

Brug af gentagende kontrolstruktur

Brug af rekursiv procedure

```
function find_rod (l, u: real): real;
begin
  if f(midtpunkt(l,u)) = 0
  then find_rod := midtpunkt(l,u)
  else if samme_fortegn(f(midtpunkt(l, u)), f(u))
  then find_rod := find_rod(l, midtpunkt(l, u))
  else find_rod := find_rod(midtpunkt(l, u), u)
end;
```

Styrken af rekursion.

```
Program rekursion;
const antal = 3;

procedure laes_og_skriv(i: integer);
var tegn: char;
begin
  if i = 0
  then writeln('-----')
  else begin
    write('i = ', i:2, '. Indlaes tegn: ');
    readln(tegn);
    laes_og_skriv(i-1);
    writeln(tegn)
  end;
end;

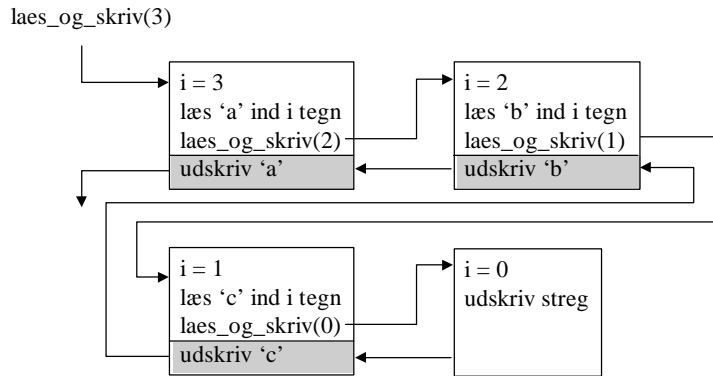
begin
  writeln('Indskriv ', antal, ' tegn');
  laes_og_skriv(antal);
end.
```

Vi læser elementer af tabellen 'på vej op ad kaldstakken'.

Vi udskriver elementer 'på vej ned ad kaldstakken'.

Samlet effekt: Elementer udskrives i modsat rækkefølge af indlæsning.

Trace af læs_og_skriv



Eksempel: Potensopløftning (1).

```

Function oploeft(tal: Real; potens: integer): Real;
begin
  if potens = 0
  then oploeft := 1.0
  else if potens > 0
  then oploeft := tal * oploeft(tal, potens - 1)
  else oploeft := 1.0/oploeft(tal, -potens)
  { Postbetingelse: oploeft = tal ** potens }
end;
  
```

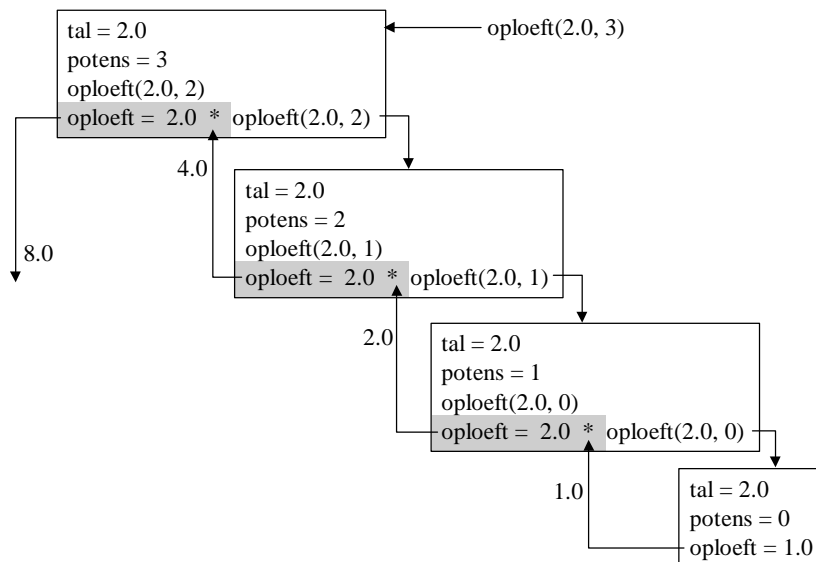
potens > 0:
 $tal^{potens} = tal^{potens - 1} \cdot tal$

potens = 0:
 $tal^0 = 1.0$

potens < 0:
 $tal^{potens} = 1.0 / tal^{-potens}$

Denne version af funktionen udfører samme antal multiplikationer som parameteren potens.

Trace af oploeft.



Basis 97: Programmering i Pascal

155

Eksempel: Potensopløftning (2).

```

Function oploeft(tal: Real; potens: integer): Real;
begin
  if potens = 0
  then oploeft := 1.0
  else if (potens > 0) and lige(potens)
  then oploeft := sqr(oploeft(tal, potens div 2))
  else if (potens > 0) and not lige(potens)
  then oploeft := tal * oploeft(tal, potens - 1)
  else oploeft := 1.0/oploeft(tal, -potens)
  { Postbetingelse: oploeft = tal ** potens }
end;
  
```

potens er lige:

$$tal^{potens} = (tal^{potens/2})^2$$

potens er ulige:

$$tal^{potens} = tal^{potens-1} \cdot tal$$

function lige(n: integer): boolean;

```

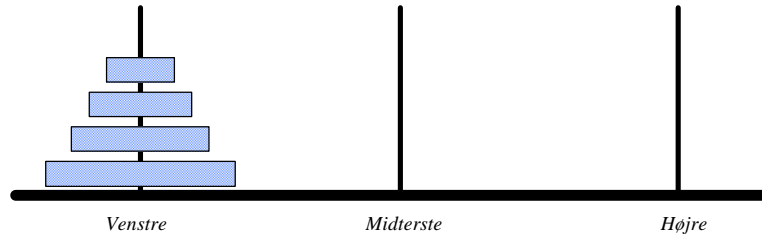
begin
  lige := n mod 2 = 0
end;
  
```

Denne version af funktionen udfører et meget mindre antal multiplikationer end versionen på forrige slide.

Basis 97: Programmering i Pascal

156

Hanoi's tårne (1).



Givet:

Et antal skiver på venstre stang, hvor mindre skiver er placeret oven på større skiver.

Problemet:

At flytte stakken af skiver fra venstre til højre stang, således at

- skiverne flyttes én ad gangen.
- en større skive må aldrig placeres oven på en mindre skive undervejs i flytteprocessen.

Hanoi's tårne (2).

Hanoi(3, Venstre, Højre, Midten):

```
type Stang = (Venstre, Midten, Højre);  
  
procedure Hanoi(n: Integer; Fra, Til, Imellem: Stang);  
{ Flyt n skiver fra 'Fra' stangen til 'Til' stangen  
  med brug af 'Imellem' stangen som mellemstation. }  
{ Prebetingelse: n >= 1.  
  Fra, Til, og Imellem er indbyrdes forskellige stænger }  
begin  
  if n = 1  
  then flyt_een_skive(Fra, Til)  
  else begin  
    Hanoi(n-1, Fra, Imellem, Til);  
    Flyt_een_skive(Fra, Til);  
    Hanoi(n-1, Imellem, Til, Fra)  
  end  
end;
```

```
Flyt fra Venstre til Højre  
Flyt fra Venstre til Midten  
Flyt fra Højre til Midten  
Flyt fra Venstre til Højre  
Flyt fra Midten til Venstre  
Flyt fra Midten til Højre  
Flyt fra Venstre til Højre
```

```
procedure Flyt_een_skive(Fra, Til: Stang);  
begin  
  writeln('Flyt fra ', out(Fra), ' til ', out(Til));  
end;
```

Hanoi's tårne (3).

Et kald af Hanoi(n, ...) foretager $2^n - 1$ flytninger.

Når antallet af skiver bliver stor kan man i praksis antage, at flytte-processen aldrig bliver færdig.

n	2^n	$10^{-9} \cdot 2^n$	$10^{-3} \cdot 2^n$
5	32	$3.2 \cdot 10^{-8}$ sek	$3.2 \cdot 10^{-2}$
25	$3.4 \cdot 10^7$	0.03 sek	9.3 timer
50	$1.1 \cdot 10^{15}$	13 dage	35702 år
65	$3.7 \cdot 10^{19}$	1170 år	$1.17 \cdot 10^9$ år
80	$1.2 \cdot 10^{24}$	$3.8 \cdot 10^7$ år	$3.8 \cdot 10^{13}$ år
100	$1.2 \cdot 10^{39}$	$4.0 \cdot 10^{13}$ år	$4.0 \cdot 10^{20}$ år

Hanoi(4, Venstre, Højre, Midten):

Flyt fra Venstre til Midten
Flyt fra Venstre til Højre
Flyt fra Midten til Højre
Flyt fra Venstre til Midten
Flyt fra Højre til Venstre
Flyt fra Højre til Midten
Flyt fra Venstre til Midten
Flyt fra Venstre til Højre
Flyt fra Midten til Højre
Flyt fra Midten til Venstre
Flyt fra Højre til Venstre
Flyt fra Midten til Højre
Flyt fra Venstre til Midten
Flyt fra Venstre til Højre
Flyt fra Midten til Højre

11. Datastrukturer og Informationssøgning

Fri studieaktivitet i Informationsteknologi

Indhold

- Lineær søgning i en tabel.
- Binær søgning i en tabel.
- Indsættelse og sletning i en tabel.
- Sammenkædede datastrukturer.
- Forgrenede, sammenkædede datastrukturer: Træer
- Søgning i søgetræer.
- Indsættelse i søgetræer.
- Informationssøgning på en disk.
- Filer og indekser.

Informationssamlinger

- En informationssamling skal tilbyde mulighed for
 - søgning efter information
 - indsættelse af ny information
 - sletning af eksisterende information.
- Effektiv søgning er næsten altid ønskelig eller påkrævet.
- Effektiv indsættelse og sletning er ikke altid væsentlig.
 - Informationssamlingen opdateres lejlighedsvis, evt. hver nat, hvor der ikke forekommer søgninger.
- Grundlaget for en abstrakt datatype for informationssamlinger er en datastruktur, som tillader tilstrækkelig effektiv udformning af søgning, indsættelse og sletning.
- Informationssamlinger manifesterer sig ofte som databaser.

Søgning

1	2	3	n
Fornavn Peter	Fornavn Lars	Fornavn Lise		Fornavn Albert
Efternavn Jensen	Efternavn Hansen	Efternavn Fredriksen		Efternavn Fisker
Cpr Nr. 030563-1131	Cpr Nr. 050633-1235	Cpr Nr. 030963-3436		Cpr Nr. 091021-3335

- Hvad er udgangspunktet for søgningen?
 - Fornavn, efternavn eller personnummer.
- Hvad søger vi efter?
 - Noget af den information, som ikke er i udgangspunktet for søgningen.
- En *nøgle* er et felt i en post, som entydigt identificerer posten i hele informationssamlingen.
 - Hvis vi søger efter et felt, som ikke er en nøgle, risikerer vi at få to eller flere poster retur.

Lineær søgning i en tabel

Ved lineær søgning testes tabellens elementer én for én, fra venstre mod højre.

Uanset elementernes indbyrdes placering vil vi finde et element, hvis det eksisterer i tabellen.

1	2	3	4	5	6	7	8	9	10
Mads	Kurt	Lars	Ib	Åse	Niels	Alberte	Mett	Albert	Bo

- Tidsforbrug:
 - I værste tilfælde skal vi gennemsøge alle elementer i tabellen.
 - I gennemsnitstilfældet skal vi gennemsøge halvdelen af tabellens elementer.

Pascal procedure til lineær søgning

```

Procedure soeg_sekventielt
  (VAR Tabel: ArrayType;
   soegeNoegle: NoegleType;
   fra, til: TabelInterval;
   VAR position: Integer;
   VAR fundet: Boolean);
  (* Soeg sekventielt i Tabel efter Noegle.
   Hvis det findes returneres fundet = true og
   Tabel[position] = Noegle. Hvis det ikke findes returneres fundet = false. *)
Begin
  fundet := False;
  position := fra;
  while (position <= til) and (not fundet) do
    (* Løkkeinvariant: fra <= position <= til + 1;
     Søgenøgle findes ikke i intervallet [ fra..position - 1 ]. *)
    if Tabel[position].Noegle = soegeNoegle
    then fundet := True
    else position := position + 1
  End;

```

```

Const Max = 100;
Type TabelInterval = 1 .. Max;
NoegleType = integer;
ArrayType =
  array[TabelInterval] of
    Record
      Noegle: NoegleType;
      (* Andre data *)
    End;

```

Binær søgning i en tabel

Hvis vi sorterer elementerne i tabellen kan det lade sig gøre at søge meget mere effektivt end ved lineær søgning.

T:

1	2	3	4	5	6	7	8	9	10 = n
Albert	Alberte	Bo	Ib	Kurt	Lars	Mads	Mett	Niels	Åse

Idéen bag binær søgning:

- Antag vi søger efter elementet x .
- Sammenlign tabellens midterste element $T[m]$ med x .
- Hvis $T[m] = x$ har vi fundet hvad vi søgte.
- Hvis $x < T[m]$ kan vi nøjes med at søge efter x i intervallet $1..(m-1)$.
- Hvis $x > T[m]$ kan vi nøjes med at søge efter x i intervallet $(m+1) .. n$

Tidsforbrug:

- I værste tilfælde halverer vi intervallet $[1.. n]$ $\log_2(n)$ gange inden vi finder x (eller indtil vi erkender, at x ikke er i tabellen).

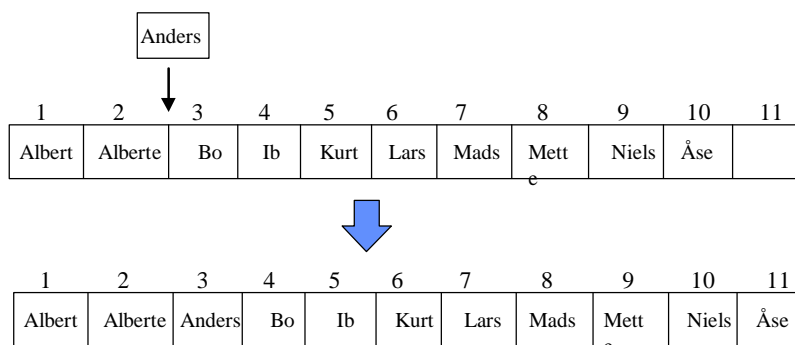
Pascal procedure til binær søgning

```
Procedure Soeg_binaert
  (VAR Tabel: ArrayType; soegeNoegle: NoegleType;
   fra, til: TabelInterval;
   VAR position: Integer; VAR fundet: Boolean);
(* Søg binaert i Tabel efter Noegle.
 Hvis det findes returneres fundet = true og Tabel[position] = Noegle.
 Hvis det ikke findes returneres fundet = false. *)
Begin
  if fra > til
  then fundet := false
  else begin
    position := (fra + til) div 2;
    if Tabel[position].noegle = soegeNoegle
    then fundet := true
    else if Tabel[position].noegle < soegeNoegle
    then Soeg_binaert(Tabel, soegeNoegle, position+1, til,
                     position, fundet)
    else Soeg_binaert(Tabel, soegeNoegle, fra, position-1,
                     position, fundet)
  end
End;
```

Basis 97: Programmering i Pascal

167

Indsættelse og sletning i en sorteret tabel



Indsættelse og sletning kræver i gennemsnit, at man skal “skubbe rundt med” halvdelen af tabellens elementer.

Det er derfor vanskeligt at opnå effektiv indsættelse og sletning såvel som effektiv søgning i en sorteret tabel.

Basis 97: Programmering i Pascal

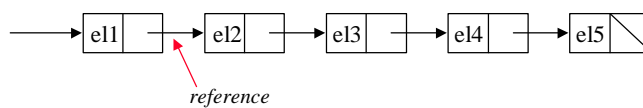
168

Sammenkædede datastrukturer

En sammenkædet datastruktur består af dataenheder, som kan indeholde en *reference* til (en adresse på) en anden dataenhed.

Dette er et alternativ til en tabel, hvor dataenhederne (tabellens elementer) er placeret i umiddelbar forlængelse af hinanden.

Kædet liste:



Tilsvarende tabel:

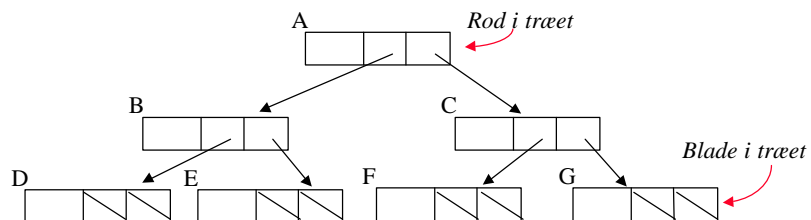
el1	el2	el3	el4	el5
-----	-----	-----	-----	-----

Forgrenede, sammenkædede datastrukturer til søgning

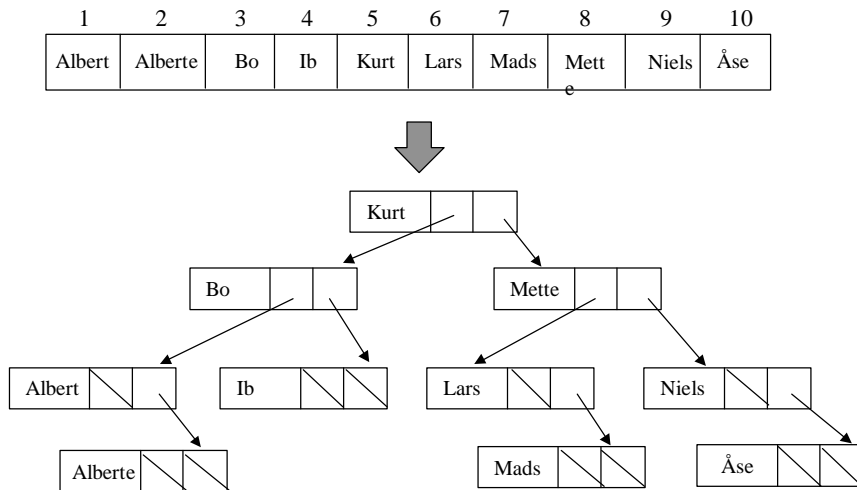
Ved brug af sammenkædning er det muligt at lave forgrenede datastrukturer, som vi kalder *træer*.

I et *søgetræ* kræver vi at alle knuder til venstre for roden er mindre end roden, og at alle knuder til højre for roden er større end roden.

Samme egenskab skal gælde i højre og venstre undertræ.



Søgetræer i forhold til binær søgning i tabel



Basis 97: Programmering i Pascal

171

Søgning i et søgetræ.

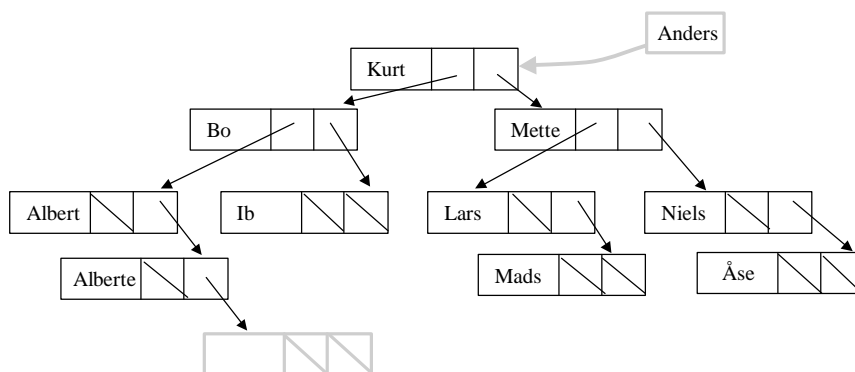
- Søgning i et (binært) søgetræ svarer nøje til binær søgning i en sorteret tabel:
 - Antag vi søger efter elementet x .
 - Sammenlign roden i søgetræet med x .
 - Hvis roden = x har vi fundet hvad vi søgte.
 - Hvis $x <$ roden kan vi nøjes med at søge efter x i det venstre undertræ.
 - Hvis $x >$ roden kan vi nøjes med at søge efter x i det højre undertræ.
- Tidsforbrug for søgning i et søgetræ afhænger af hvor mange grene vi skal følge på vej fra roden til det sted, hvor det søgte element evt. befinder sig.
- Hvis et træ med n knuder er i rimelig balance er der ca. $\log_2(n)$ grene fra roden til et blad i træet.

Basis 97: Programmering i Pascal

172

Indsættelse af data i et søgetræ.

Det er muligt at indsætte et vilkårligt nyt element som et nyt blad i et søgetræ.
Der er fare for at træet bliver "skævt" i takt med at nye elementer indsættes.
Det kan blive nødvendigt at "afbalancere" træet som følge af dette.



Basis 97: Programmering i Pascal

173

Søgning efter information på disken

- Søgning i tabeller retter sig primært mod søgning i maskinens interne arbejdslager.
- I mange realistiske tilfælde er information lagret på maskinens eksterne lager: på en disk.
 - Vi ønsker at information skal overleve, at maskinen slukkes.
 - Informationen kan ikke rummes i arbejdslageret.
- En 'informationsenhed' på disken kaldes for en *fil*.

Basis 97: Programmering i Pascal

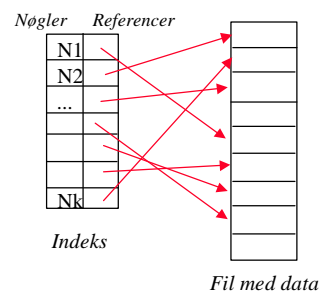
174

Lidt om filer

- En fil er en tabel-lignende datastruktur, som vedbliver med at eksistere selv efter at maskinen slukkes.
 - Filer bor typisk på en disk eller et magnetbånd.
- Forskellige slags filer:
 - Tekstfiler: Kan behandles af en teksteditor.
 - “Binære filer”:
 - Filer med en tabel af poster, som organiserer en mængde af information.
 - Mange andre slags ...
 - I nogle filer er det kun muligt at læse post n hvis man først læser post 1, 2, ..., $n-1$: *sekventielle filer*.
 - I andre filer kan man læse post nummer n direkte: *direkte filer*.
- Filer i forhold til arrays.
 - Tilgang til information på filer er meget langsommere end tilgang til information i en tabel.

Søgning efter information på filer via et indeks

- Et *indeks* er en tabel i arbejdslageret, som kun indeholder nøgler og referencer til poster på en fil.
- Et indeks svarer til et stikordsregister bag i en bog.
- Hvis indekset er sorteret er det muligt at lave binær søgning efter nøglen i indekset, og dermed kan posten findes meget hurtigt på filen.



Eksempel på et indeks til en datafil

- Indekset på personnumre er i arbejdslageret.
- Indekset er sorteret efter årstal, måned, dag og løbenummer i personnummeret.

010133-4513	
050633-1235	
050557-4522	
030957-3436	
030563-1131	
030563-1133	
091071-3335	

- Posterne i filen kan placeres på en vilkårlig måde hvis blot indekset holdes sorteret efter personnummeret.
- Søgetid efter en post ud fra personnummer: $\log_2(n)$, hvor n er antallet af poster i filen.

Fornavn	Albert
Efternavn	Fisker
Cpr Nr.	091071-3335
Fornavn	Lise
Efternavn	Fredriksen
Cpr Nr.	030957-3436
Fornavn	Lars
Efternavn	Hansen
Cpr Nr.	050633-1235
Fornavn	Peter
Efternavn	Jensen
Cpr Nr.	030563-1131
Fornavn	Lars
Efternavn	Jensen
Cpr Nr.	030563-1133
Fornavn	Thøger
Efternavn	Ellemand
Cpr Nr.	010133-4513
Fornavn	Stine
Efternavn	Anker
Cpr Nr.	050557-4522

12. Eksempler på Problemløsning med Datamater.

Sortering ved udvælgelse.
Quick sort.
Life.

Sorteringsproblemet.

Udgangspunkt: En tabel.

Forudsætning: Der findes en ordning af tabellens elementer.

Problem: Ombyt tabellens elementer således at element 1 er mindre end eller lig med element 2, som er mindre end eller lig med element 3, ..., som er mindre end eller lig med element *Max*.

T:

1	2	3	Max = 4
Fornavn 'Mads'	Fornavn 'Lars'	Fornavn 'Kurt'	Fornavn 'Bo'
Efternavn 'Andersen'	Efternavn 'Hansen'	Efternavn 'Nørmark'	Efternavn 'Bramsen'
Cpr Nr. 030563-1131	Cpr Nr. 030963-4321	Cpr Nr. 050557-1235	Cpr Nr. 091021-3335

Sorter(T) efter *fornavn*.

T:

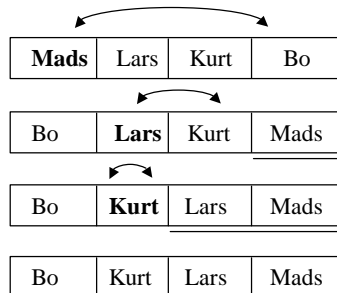
1	2	3	Max = 4
Fornavn 'Bo'	Fornavn 'Kurt'	Fornavn 'Lars'	Fornavn 'Mads'
Efternavn 'Bramsen'	Efternavn 'Nørmark'	Efternavn 'Hansen'	Efternavn 'Andersen'
Cpr Nr. 091021-3335	Cpr Nr. 050557-1235	Cpr Nr. 030963-4321	Cpr Nr. 030563-1131

Sortering ved udvælgelse.

Idé:

1. Gennemløb hele tabellen og find placeringen af det største element.
2. Ombyt det dermed fundne elementet med det sidste, usorterede element.
3. Gentag trin 1 og 2 på gradvist mindre forender af tabellen indtil hele tabellen er sorteret.

Eksempel:



Sortering ved udvælgelse: Konstanter og typer.

```
Const Max = 10;

Type  TabelInterval = 1 .. Max;

      TabelElement = Record
          Fornavn: String;
          Efternavn: String;
          { Andre person data }
      End;

      ArrayType = array[TabelInterval] of TabelElement;
```

Sortering ved udvælgelse: Procedurestruktur.

```
Program sortering;
{Typer og konstanter fra forrige slide}
Var T: ArrayType;

procedure Initialiser(var T: ArrayType);

procedure Udskriv(var T: ArrayType; fra, til: TabelInterval);

procedure Sorter(var T: ArrayType);

      function FindMaxIndex(var T: ArrayType;
          fra, til: TabelInterval): TabelInterval;

      procedure Ombyt(var T: ArrayType; i,j: TabelInterval);

begin { Sorter }
...
end; { Sorter }

begin { hovedprogram }
  Initialiser(T);
  Sorter(T);
  Udskriv(T,1,max)
end.
```

Sortering ved udvælgelse: Kroppen af 'sorter'.

```
procedure Sorter(var T: ArrayType);
  {Sorter T i stigende orden. Sorteringen foregår ved at bytte om på elementerne i T}
  var i,m: TabelInterval;

  begin
    for i := max downto 2 do
      {T[i+1..max] er sorteret i stigende orden.
       T[1..i] <= T[i+1..max]}
      begin
        m := FindMaxIndex(T,i);
        Ombyt(T,i,m)
      end
    {Postbetingelse: T[1] <= T[2] <= ... <= T[max]}
  end
```

Sortering ved udvælgelse: FindMaxIndex.

```
function FindMaxIndex(var T: ArrayType; fra, til: TabelInterval): TabelInterval;
  { Find elementet med den største værdi i T[fra..til], og returner dets indeks }
  { Prebetingelse: 1 <= fra <= til <= max }
  var i, MaxIndex: TabelInterval;
      MaxElement: String;
  begin
    MaxIndex := fra; MaxElement := T[fra].Fornavn;
    for i := fra+1 to til do
      if T[i].Fornavn > MaxElement
      then begin
        MaxElement := T[i].Fornavn;
        MaxIndex := i
      end;
    FindMaxIndex := MaxIndex
  { Postbetingelse: For alle j i intervallet [fra .. til]: T[j] <= T[MaxIndex]}
  end; { FindMaxIndex }
```

Eksempel på tabel.

1	2	3	4	5	6	7	8	9	10
Mads	Kurt	Lars	Ib	Åse	Niels	Alberte	Mette	Albert	Bo

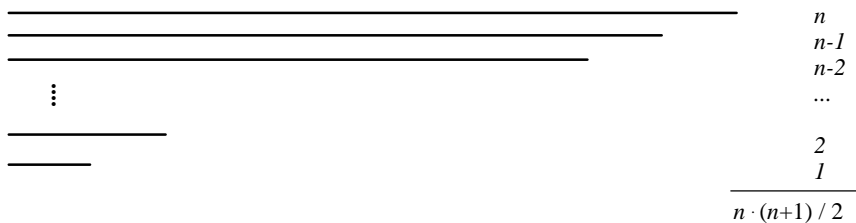


1	2	3	4	5	6	7	8	9	10
Albert	Alberte	Bo	Ib	Kurt	Lars	Mads	Mette	Niels	Åse

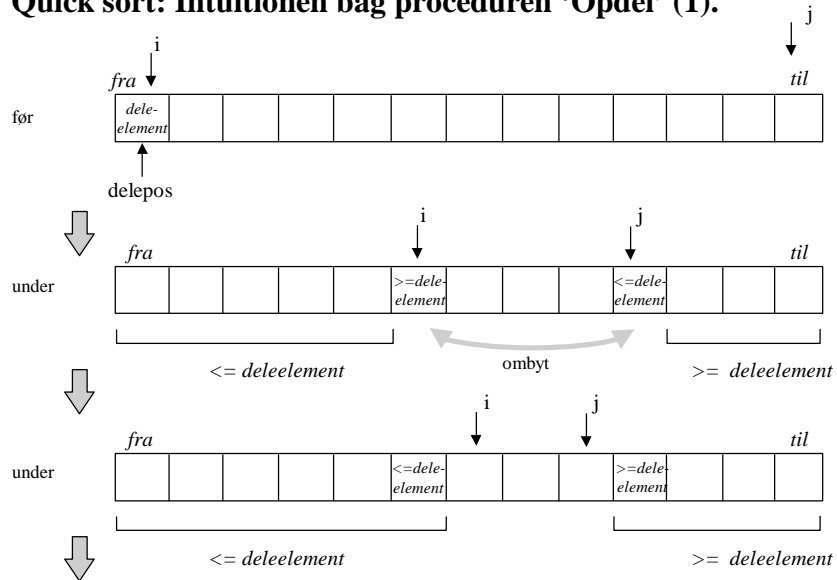
Vurdering af sortering ved udvælgelse.

- Sortering af n elementer i en tabel kræver i størrelsesordenen $n \cdot (n+1) / 2$ 'skridt' (sammenligninger/ombytninger).
- Sortering ved udvælgelse er ikke særlig effektiv.

1	2	3	4	5	6	7	8	9	$n = 10$
Mads	Kurt	Lars	Ib	Åse	Niels	Alberte	Mette	Albert	Bo



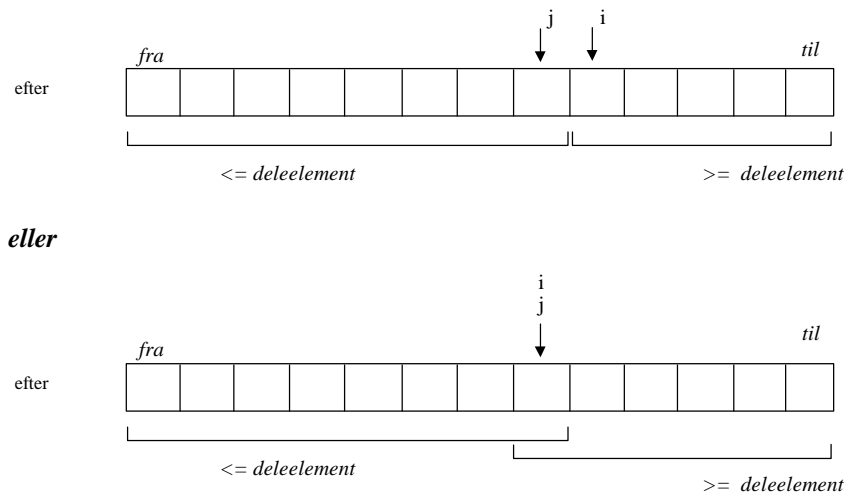
Quick sort: Intuitionen bag proceduren 'Opdel' (1).



Basis 97: Programmering i Pascal

189

Quick sort: Intuitionen bag proceduren 'Opdel' (2).



Basis 97: Programmering i Pascal

190

Quick sort: Proceduren opdelt.

```

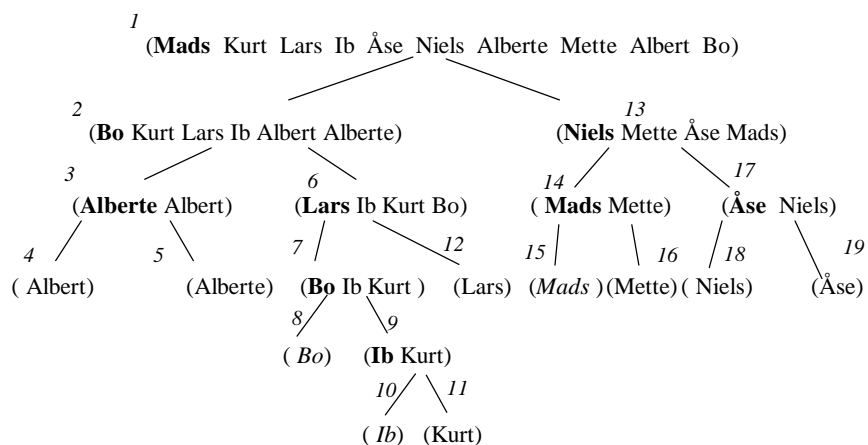
Procedure Opdel(var T: ArrayType; fra, til: TabelInterval;
               var venstreTil, hoejreFra: TabelInterval);
{Opdel T [fra .. til] i to dele T[fra .. venstreTil] og T[hoejreFra .. til].
 Brug T[fra] som deelelement.}
{ Prebetingelse: fra < til }
var deleElement: TabelElement;
    i,j: Integer;

begin
  i := fra - 1; j := til + 1;
  deleElement := T[fra];
  repeat
    repeat i := i+1 until T[i].fornavn >= deleElement.fornavn;
    repeat j := j-1 until T[j].fornavn <= deleElement.fornavn;
    if i < j then Ombyt(T,i,j);
  until i >= j;

  if i > j then begin venstreTil := j; hoejreFra := i end
  else begin venstreTil := j-1; hoejreFra := i+1 end;
{ Postbetingelse: T[fra .. venstreTil] <= T[hoejreFra .. til] }
end;

```

Forløbet af quick sort på eksemplet.



Vurdering af quick sort.

- Sotering af n elementer i en tabel kræver i størrelsesordenen $n \cdot \log_2(n)$ 'skridt' (sammenligninger/ombytninger).
- Quick sort er således meget effektiv, dog forudsat at opdelningen er hensigtsmæssig.
- I uheldige tilfælde kan Quick sort være lige så dårlig (eller dårligere) end sortering ved udvælgelse.
 - Sortering af en allerede sorteret tabel er et sådant tilfælde.
 - Et bedre valg af opdelningselement kan gøre de uheldige tilfælde meget sjældne.
- Det kan bevises at køretiden i størrelsesordenen $n \cdot \log_2(n)$ er optimal hvad angår sorteringsteknikker, der udelukkende er basert på sammenligning af elementer.

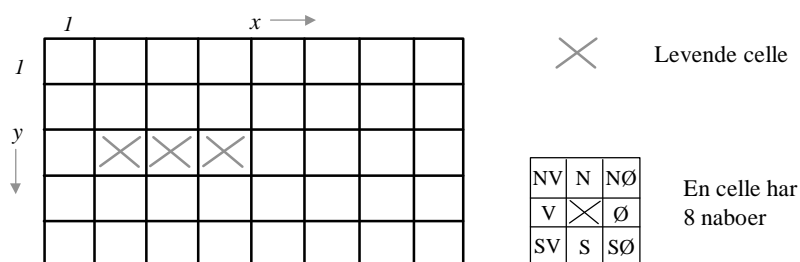
n	n^2	$n \cdot \log_2(n)$
100	10.000	664
1.000	1.000.000	9.966
10.000	100.000.000	132.887
100.000	10.000.000.000	1.660.964

Basis 97: Programmering i Pascal

193

Life.

Life er en simulering af, hvordan mønstre af 'levende celler' udvikler sig givet tre simple regler for, hvornår nyt liv opstår, og hvornår levende celler dør.



Regler:

1. En levende celle med mindre end to naboer dør af isolation.
2. En levende celle med mere end tre naboer dør af overbefolkning.
3. Nyt liv opstår i en tom celle, som har netop tre levende naboer.

Basis 97: Programmering i Pascal

194

En abstrakt datatype: LifeVerden.

```
Type CelleStatus = (I_live,Doed);
  LifeVerden = record { Datarepræsentation } end;
  Retning = (N, NO, O, SO, S, SV,V, NV);

Procedure NulstilVerden(Var V: LifeVerden; xStr, yStr: Integer);

Function Levende(var V: LifeVerden; i,j: Integer): Boolean;
{ Returner hvorvidt felt (i,j) i verden V er levende. (i,j) kan angive
et felt uden for verden, i hvilket tilfælde resultatet er falsk }

Procedure SaetStatus(var V: LifeVerden; i,j: integer; s: CelleStatus);
{ Definer indholdet af celle (i,j) i verden V til status s. Celler uden for V forbliver døde }

Function NaboLevende(var VR: LifeVerden; i,j: Integer; r: Retning): Boolean;
{ Returner hvorvidt feltet i retning r i forhold til feltet (i,j) er levende.}

Function AntalLevendeNaboer(var VR: LifeVerden; i,j: Integer): Integer;

Procedure PrintVerden(Var V: LifeVerden);

Procedure NyGeneration(var Gammel, Ny: LifeVerden);
{ På basis af den gamle verden laves en ny verden }
```

Basis 97: Programmering i Pascal

195

Forløbet af en Life simulation.

```
Var V1,V2: LifeVerden;
begin { hovedprogram }
  Lav en start situation i V1;
  repeat
    PrintVerden(V1);
    NyGeneration(V1,V2);
    V1 := V2
  until færdig
end.

Procedure NyGeneration(var Gammel, Ny: LifeVerden);
{ På basis af den gamle verden laves en ny verden }
var i,j: integer;
begin
  NulstilVerden(Ny, Gammel xmax, Gammel ymax);
  for j:= 1 to Gammel ymax
  do for i := 1 to Gammel xmax
  do if Levende(Gammel,i,j) then
    if (AntalLevendeNaboer(Gammel,i,j) < 2) or
      (AntalLevendeNaboer(Gammel,i,j) > 3)
    then SaetStatus(Ny,i,j,Doed)
    else SaetStatus(Ny,i,j,I_live)
  else { (i,j) i Gammel er død }
    if AntalLevendeNaboer(Gammel,i,j) = 3
    then SaetStatus(Ny,i,j,I_live)
    else SaetStatus(Ny,i,j,Doed)
  end; { NyGeneration }
```

Basis 97: Programmering i Pascal

196

Datadrepræsentation af 'LifeVerden'.

```
Const  xVerdenMax = 74;  { Øvre grænser      }
       yVerdenMax = 24;  { for størrelsen af verden }

Type   CelleStatus = (I_live,Doed);
       LifeVerden = record
           matrix: array[1..xVerdenMax,1..yVerdenMax]
                of CelleStatus;
           xmax: 1..xVerdenMax;
           ymax: 1..yVerdenMax;
       end;
```

Funktioner der undersøger om celler er levende.

```
Function Levende(var V: LifeVerden; i,j: Integer): Boolean;
begin
    if (i < 1) or (i > V.xmax) or (j < 1) or (j > V.ymax)
    then Levende := false
    else Levende := V.matrix[i,j] = I_live
end; { Levende }
```

```
Function NaboLevende(var VR: LifeVerden;
                    i,j: Integer; r: Retning): Boolean;
begin
    case r of
        N: NaboLevende := Levende(VR,i,j-1);
        NO: NaboLevende := Levende(VR,i+1,j-1);
        O: NaboLevende := Levende(VR,i+1,j);
        SO: NaboLevende := Levende(VR,i+1,j+1);
        S: NaboLevende := Levende(VR,i,j+1);
        SV: NaboLevende := Levende(VR,i-1,j+1);
        V: NaboLevende := Levende(VR,i-1,j);
        NV: NaboLevende := Levende(VR,i-1,j-1);
    end { case }
end; { NaboLevende }
```

```
Function AntalLevendeNaboer
    (var VR: LifeVerden; i,j: Integer): Integer;
var r: Retning;
    antal: integer;
begin
    antal := 0;
    for r := N to NV do
        if NaboLevende(VR, i, j, r)
        then antal := antal + 1;
    AntalLevendeNaboer := antal
end; { AntalLevendeNaboer }
```