

# TI kursus

PASCAL, GRUNDLÆGGENDE

46/56    15416.05  
1985

28. oktober - 1. november 1985.

Mandag den 28.10.:

Kl. 9.00 - 10.20	Introduktion til programmering Introduktion til opgave Introduktion til øvelsesudstyr.
- 10.40 - 12.00	Øvelser.
- 13.00 - 14.25	Introduktion til Pascal.
- 14.40 - 16.00	Øvelser.

Tirsdag den 29.10.:

Kl. 9.00 - 10.20	Kontrolstrukturer.
- 10.40 - 12.00	Øvelser.
- 13.00 - 14.20	Kontrolstrukturer, fortsat Procedurer
- 14.40 - 16.00	Øvelser.

Onsdag den 30.10.:

Kl. 9.00 - 10.20	Datastrukturer, tabeller Variabel erklæringer.
- 10.40 - 12.00	Øvelser
- 13.00 - 14.20	Typebegreb.
- 14.40 - 16.00	Øvelser.

Torsdag den 31.10.:

Kl. 9.00 - 10.20	Scope-regler.
- 10.40 - 12.00	Øvelser.
- 13.00 - 14.20	Parameteroverførsel.
- 14.40 - 16.00	Øvelser.

# TI kursus

PASCAL, GRUNDLÆGGENDE.

46/56 15416.05  
1985

28. oktober - 1. november 1985.

Fredag den 01.11.:

Kl. 9.00 - 10.20	Orientering om funktioner, records, filer.
- 10.40 - 12.00	Øvelser.
- 13.00 - 14.20	Opsamling Kogebog i systemarbejde og programudvikling.
- 14.40 - 16.00	Øvelser Afslutning.

VARIGHED:

30 timer.

UNDERVISNINGSTED:

Teknologisk Institut  
Gregersensvej, indg. 8  
2630 Tåstrup

LOKALE:

85 A

LÆRERE:

Bjarne G. Pedersen  
Nicolaj Holm  
Flemming Bitz  
Knud Musaeus

PASCAL, GRUNDLÆGGENDE.

46/56 15416.05

1985

28. oktober - 1. november 1985.

1. Niels P. Bjerremose	Metalindustriens Fagskole
2. Bent Andersen	Kosan Teknova
3. Fritz H. Pedersen	Frederiksberg tekn. Skole
4. Ib Hingeberg	Storno
5. Bjarne Nielsen	-
6. Niels Erik Liltorp	Forsvarsstaben
7. Johnny V. Mogensen	Brüel & Kjør
8. Freddy Hansen	-
9. Lotte Ring Brodersen	-
10. Erling Frederiksen	-
11. Stig Emborg	Radiometer
12. Laurs Lursen	JT
13. Erik Sørensen	F.L. Smidth & Co.
14. Torben Hessilt	Pædagogisk Central

PASCAL GRUNDLÆGGENDE

DATAMATINTRODUKTION

KOSTPLANBESKRIVELSE/OPGAVE

PROGRAMMERINGSINTRODUKTION

PASCAL I  
GRUNDLÆGGENDE ELEMENTER

PASCAL II  
KONTROLSTRUKTURER

PASCAL III  
UNDERPROGRAMMER

PASCAL IV  
DATASTRUKTURER

EKSTRA OPGAVER

"KOGEBOG" I SYSTEMARBEJDE OG  
PROGRAMMERING.

EOA G. Schreiner

MADE IN DENMARK

NR. 18 41 11

### Datamaskinens virkemåde

Et datamaskinesystem består af to grundelementer:

Materiellet (hardware): Den fysiske datamaskine opbygget af elektroniske og mekaniske elementer.

Programmellet (software): De programmer, der bevirker, at datamaskinen kan udføre bestemte funktioner.

Fig. 1 viser en typisk datamats hardware enheder:

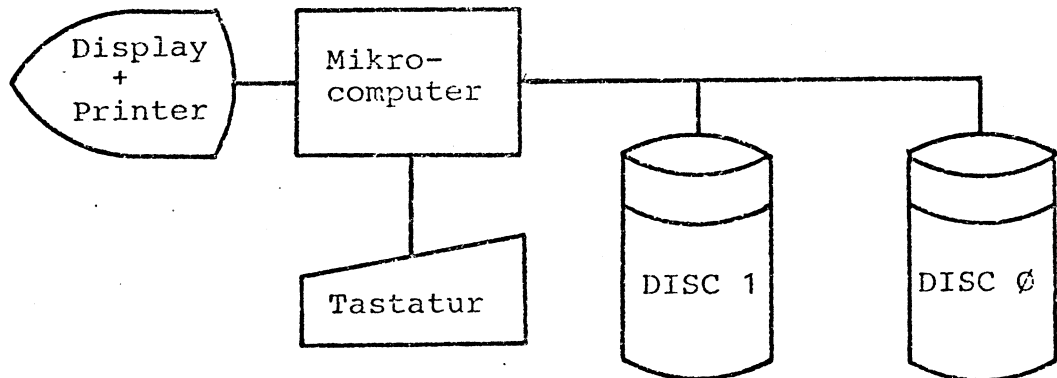


Fig. 1. Datamaskinens hardware enheder.

Materiellet (hardware) består af:

- a) Mikrocomputer
- b) Ydre enheder (f.eks. dataskærm, printer, disk).

Mikrocomputeren består af:

- a) Ind/ud-enhed (I/O)
- b) Lager-enhed (Intern Memory)
- c) Central processor-enhed (C.P.U.)

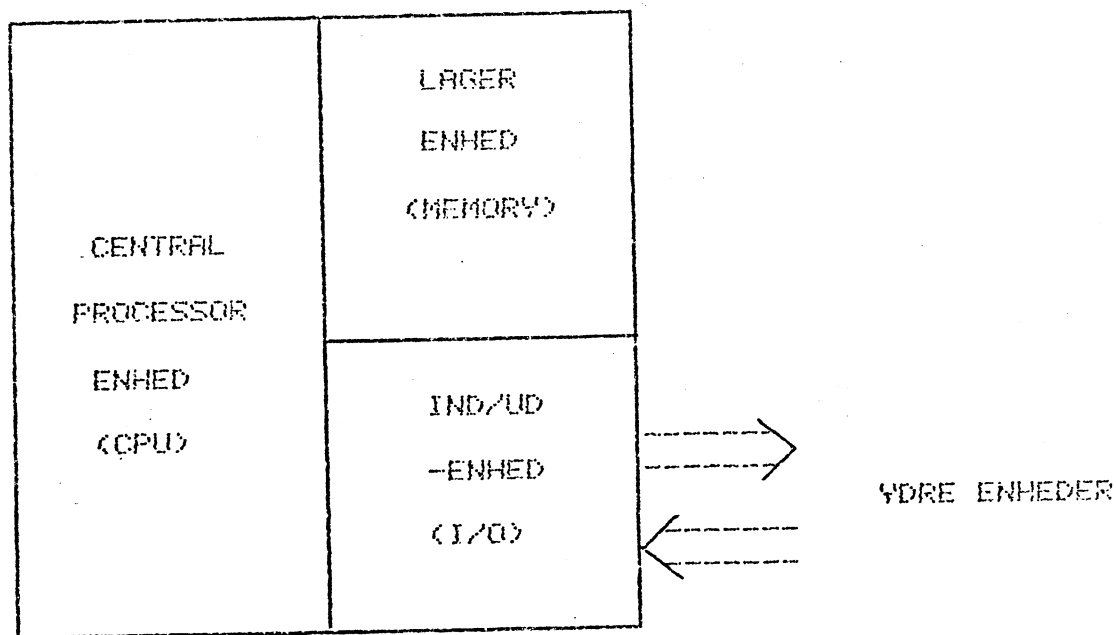


Fig. 2. Mikro computer.

Lager-enheden er det sted, hvor der kan gemmes programinstruktioner og data. Programinstruktioner er de koder som dirigerer central processor-enheds aktiviteter. Data er de informationer som skal behandles eller er blevet behandlet af central processor-enheden.

Central processor-enheden "læser" hver instruktion fra lagerenheden i en logisk rækkefølge og bruger instruktionerne som ordre til at udføre de ønskede (programmerede) aktiviteter.

Såfremt programmet er sammenstillet af instruktioner, hvis rækkefølge er logisk og sammenhængende, vil procesudførelsen give et forståeligt og brugbart resultat.

Ind-udlæseenheden anvendes ved dataoverførsel mellem datamaskinens lager og tilsluttede ydre enheder (f.eks. printer).

Fig. 3 viser de softwaremæssige hjælpemidler der er til rådighed i en datamat til programudvikling.

Operativ system sørger for at bruger kan dirigere med datamat (kalde ønskede programmer).

Fig. 3 programmel til udviklingsdatamat.

Editor til brug ved skrivning, redigering og retning i kildeprogrammer (sourcekode).

Compiler til oversætning af det skrevne kildeprogram til maskinkode (objekt kode).

Linker til sammenhægtning af programmer herunder biblioteksprogrammer.

LIB Manager til at indlægge programmer i programbiblioteker.

Print program til udskrivning af programmer.

Loader program til hentning af programmer til mikrocomputerens lager.

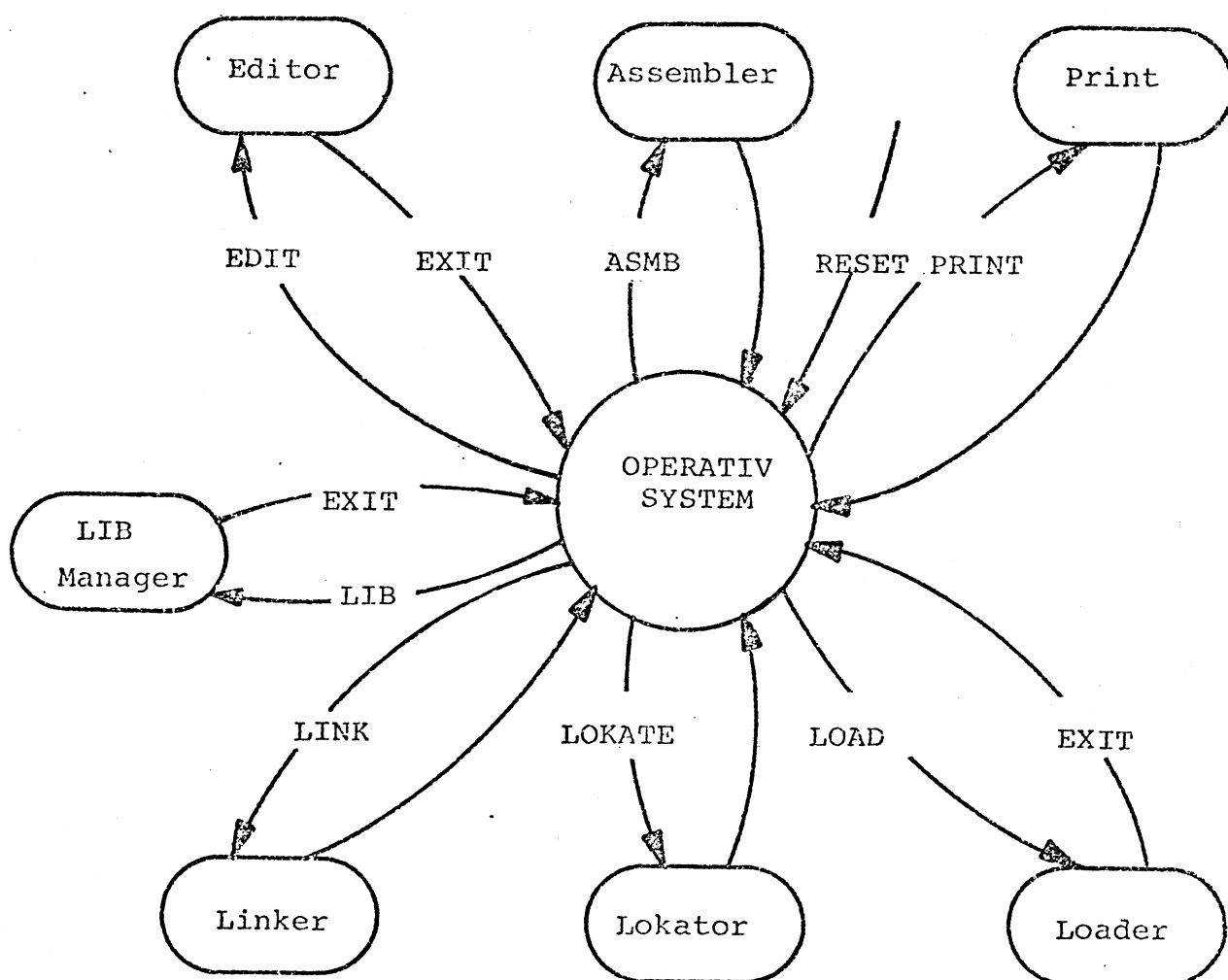


Fig. 3. Softwaremæssige hjælpemidler ved programudvikling.



1985.08.20

KM/ej 7.1

# BETJENINGSVEJLEDNING TIL SPERRYSYSTEMET

=====

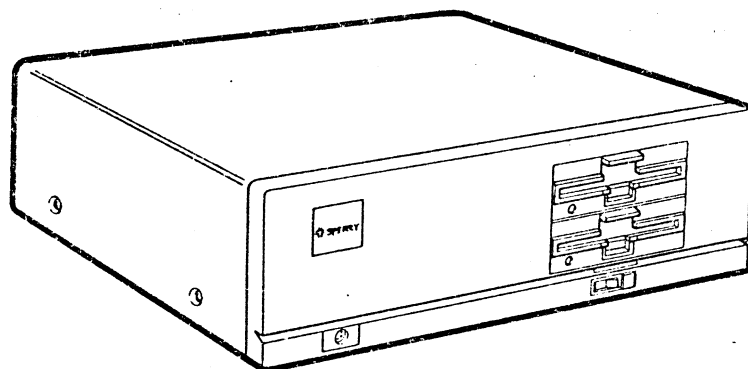
- SPERRYDATAMATEN
- MS-DOS
  - kommandoer
  - oversigtsdiagram
- TURBO-PASCAL
  - Turbo kommandoer
  - Editor kommandoer
  - oversigtsdiagram
- FUNKTIONSTAST OVERSIGT

## INTRODUKTION TIL SPERRY PC-30

=====

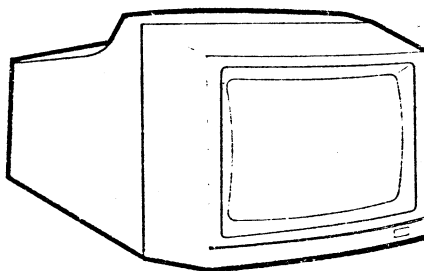
En Sperry PC-30 er en IBM kompatibel personlig computer, som består af fire komponenter:

## SYSTEMKOMPONENTEN:



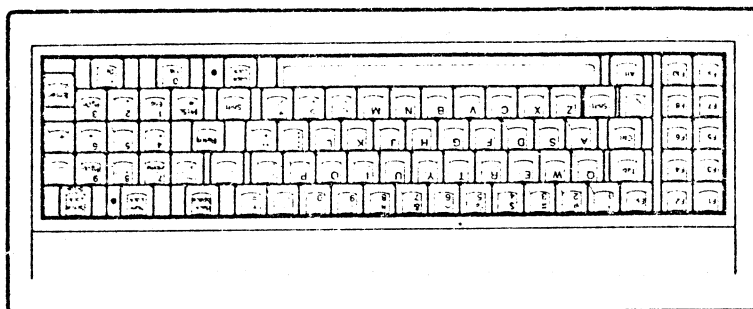
Systemkomponenten er den centrale enhed. Den indeholder regeneenhed (cpu'en), lageret (RAM 128 kB), 2 diskette-drev á 360 kByte og den styrer al kommunikation med tastatur, skærm samt printer.

## SKÆRMERN:



Skærmen er en 12 tommer farvegrafikskærm. Den kan præsentere data i op til 80 kolonner og 25 linier. I grafikmode har den en opløsning på 320 gange 200 punkter.

## TASTATURET:



Operatørens input til PC'en går via tastaturet. Sperry PC'ens tastatur inkluderer alle sædvanlige skrivemaskine-taster plus nogle forskellige taster til editering og specialfunktioner. Ude til højre på tastaturet findes et numerisk tastatur, magen til dem man kan finde på elektroniske regnemaskiner.

Sperry'ens tastatur er magen til IBM PC'ens.

## INTRODUKTION TIL MS-DOS

=====

MS-DOS er en forkortelse af Micro-Soft Disc Operating System.

MS-DOS er et styresystem, som bruges til mange forskellige PC-fabrikater, bl.a. også til Sperry. I MS-DOS er der nogle indbyggede kommandoer og funktioner, der kontrollerer den overordnede virkemåde af Sperry'en.

MS-DOS gør det også muligt at oprette og administrere egne filer samt at køre programmer skrevet i forskellige programmeringssprog, deriblandt Basic og Pascal.

MS-DOS er der ca. 45 forskellige kommandoer, men i dette kursus er det kun nødvendigt at kende følgende 3:

DIR: Viser diskettens indeholdsfortegnelse (directory).

Skriv: DIR <ret>

CTRL S stopper udskrift.

DEL: Fjerner en fil fra disketten.

Skriv: DEL <filnavn.ext> <ret>

PRINT: Udskriver en fil på en tilsluttet printer.

Skriv: PRINT <filnavn.ext> <ret> <ret>

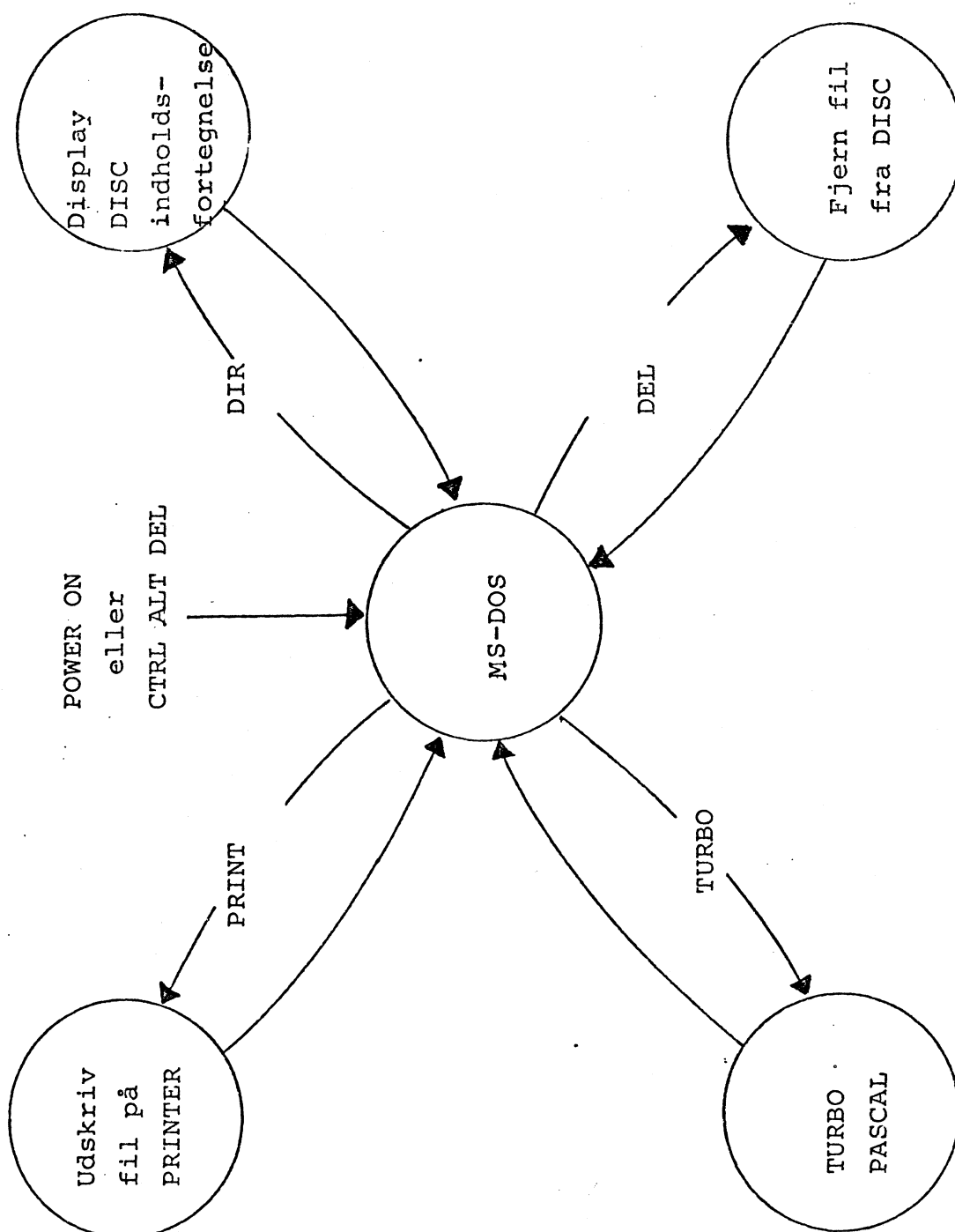
Når Sperry PC'en tændes, "vågner den op" i MS-DOS, forudsat at man har monteret den rigtige diskette i det øverste diskettedrev (a:). For at starte Pascal-systemet skal man skrive:

TURBO <ret>

Kommandoen "TURBO" er ikke en MS-DOS kommando, men MS-DOS tillader at man selv laver kommandoer.

## SYNTAKSDIAGRAM FOR MS-DOS

=====



## INTRODUKTION TIL TURBO-PASCAL

=====

TURBO-Pascal er et komplet programudviklingssystem, baseret på det blokstrukturerede programmeringssprog Pascal.

TURBO-Pascal har alle de faciliteter, der behøves for editering, oversættelse og kørsel af programmer skrevet i Pascal.

Systemet består af en runtime programdel, en skærmorienteret editor og en Pascal-compiler.

### TURBO KOMMANDOER:

L)ogged drive: A

A)ctive directory: /

W)ork file:

M)ain file:

E)dit C)ompile

R)un S)ave

D)ir Q)uit

compiler O)ptions

Text: 0 bytes

Free: 62635 bytes

>

TURBO systemet har 11 forskellige kommandoer:

L): Vælger diskette-drev	R): Kører oversat program
A): Vælger directory	S): Gennem arbejdsfil på diskette
W): Vælger arbejdsfil	D): Viser directory på skærmen
M): Vælger hovedfil	Q): Forlader TURBO-systemet
E): Editerer i arbejdsfil	(returnerer til MS-DOS)
C): Kompilerer arbejdsfil	O): Diverse Optionns

## TURBO-PASCAL EDITOR KOMMANDOER

=====

## CURSOR STYRING:

Home: Flytter Cursor til første linie, første kolonne  
End: Flytter Cursor til sidste linie, sidste karakter  
Pg Up: Flytter Cursor 23 linier tilbage i teksten  
Pg Dn: Flytter Cursor 23 linier frem i teksten

↑ Flytter Cursor 1 linie op  
↓ Flytter Cursor 1 linie ned  
← Flytter Cursor 1 position til venstre  
→ Flytter Cursor 1 position til højre.

## BLOK KOMMANDOER:

F3: Sæt start blok mærke      F6: Flyt blok  
F4: Sæt slut blok mærke      F7: Slet blok  
F5: Kopier blok én gang      CTRL/K/H: Slet blok mærker

## SLET KOMMANDOER:

F2: Slet linie

## SØGNINGS KOMMANDOER:

F9: Find streng

Tilvalg:

U: Der skelnes ikke mellem små og store bogstaver

W: Finder kun hele ord, ikke dele af andre ord.

F10: Søg og erstat streng.

Tilvalg:

G: Global søgning, fortsætter indtil hele teksten  
er gennemgået (starter fra cursor position)

N: Erstat uden at spørge

U: Se under F9

W: Se under F9.

F8: Fortsat søgning.

## EDIT-KOMMANDOER:

INS-tast: Skifter mellem Insæt- og Overskriv- mode

DEL-tast: Skifter karakteren som cursoren står på.

: Sletter karakteren til venstre for cursoren.

## FUNKTIONSTAST OVERSIGT

=====

end edit	! F1 !	! F2 !	slet linie
	-----	-----	
	-----	-----	
start blok mærke	! F3 !	! F4 !	slut blok mærke
	-----	-----	
	-----	-----	
kopier blok	! F5 !	! F6 !	flyt blok
	-----	-----	
	-----	-----	
slet blok	! F7 !	! F8 !	fortsæt søgning
	-----	-----	
	-----	-----	
find streng	! F9 !	! F10 !	find og erstat streng
	-----	-----	

OPGAVEBESKRIVELSE (Kravspecifikation).

Softwarefirmaet "Always in time" har fået til opgave at udvikle et program, som kan beregne næringsindholdet i f.eks. en middagsret.

Opgaven er kommet fra en kunde, som mener at kunne sælge dette program til dem, som har brug for/interesse i at kende til næringsindholdet i en bestemt sammensat kost. Det kunne f.eks. være sygehuse, alm. hustande, cateringsfirmaer, m.v.

Det er et krav fra kunden at programmet er menustyret således at enhver, som har fået en meget kort introduktion, skal kunne bruge programmet, ellers kan programmet ikke sælges.

Ved hjælp af programmet skal det være muligt at sammensætte en menu og derefter få en udskrift af næringsindholdet, prisen samt den relative pris pr. næringsenhed.

Desuden skal det være muligt at få at vide hvilke data, der ligger til grund for beregningerne og der skal selvfølgelig også være mulighed for at ændre disse data, da f.eks. priserne ofte ændrer sig.

Ydermere ønskes det, at det skal være muligt at lave en række specielle oversigter, f.eks. hvilke råvarer indeholder mere end 100 fibre pr. 100 g; hvilke råvarer koster mindre end 10 kr. pr. 100 g. fibre osv.

ANALYSE.

I det følgende vil denne lidt løse kravspecifikation blive nærmere analyseret og ind- og uddata vil blive fastlagt.



Det var et krav fra brugeren at programmet skulle være menustyret. Desuden skulle det være muligt at

- lave en beregning af næringsindhold og pris i en ret
- vise de data der lå til grund for beregningen
- ændre i data
- lave specielle oversigter

Af denne grund bør kostplanprogrammet starte med at give brugeren en oversigt indeholdende de 4 nævnte punkter samt en mulighed for at vælge et af de fire.

Formattet kan være:

```
HOVEDMENU.  
-----  
1. Lav kostberegning.  
2. Vis kost-data.  
3. Ændre i kost-data.  
4. Specielle oversigter.  
-----  
99. Forlad programmet.
```

Indtast et nr:

Fig. 1: Overordnet oversigt.

Brugeren har altså mulighed for at vælge et af de 4 underpunkter (underprogrammer) ved at taste 1,2,3 eller 4; eller 99 og så forlade programmet.

#### 1.

Hvis brugeren vælger at lave en kostberegning, dvs. der tastes 1, så skal der komme et nyt skærbillede frem, som viser de råvarer som det er muligt at vælge til sammensætningen af en menu. (I dette tilfælde er der kun medtaget 15 produkter for overskuelighedens skyld).



## SPISE-SEDDEL.

Nr. Produkt	Bestilt mængde.
1. Rugbrød	0.0 gram.
2. Franskbrød	0.0 gram.
3. Kartoffler	0.0 gram.
4. Ymer	0.0 gram.
5. Jordbær	0.0 gram.
6. Svinekød	0.0 gram.
7. Oksekød	0.0 gram.
8. Gulerødder	0.0 gram.
9. Mel	0.0 gram.
10. Havregryn	0.0 gram.
11. Salt	0.0 gram.
12. Wienerbrød	0.0 gram.
13. Lagkage	0.0 gram.
14. Chokolade	0.0 gram.
15. Kaffe	0.0 gram.

99. Start beregning.

Indtast et nr:

Indtast mængde i gram:

Fig. 2. Spiseseddel.

Ved at taste et af numrene har brugeren så mulighed for at vælge et produkt, f.eks. 2, som er fransbrød og derefter indtaste den mængde i gram f.eks. 200 man ønsker at anvende/spise. Herved kan man sammensætte en menu, og derefter taste 99, hvor ved rettens næringsværdi og pris udregnes og de bestilte råvarer vises på skærmen. Hver gang der er bestilt en vare skal den opdaterede spiseseddel vises på skærmen. (se fig. 3).



## SPISE-SEDDEL.

Nr. Produkt	Bestilt mængde.
1. Rugbrød	0.0 gram.
2. Franskbrød	200.0 gram.
3. Kartofler	0.0 gram.
4. Ymer	150.0 gram.
5. Jordbær	0.0 gram.
6. Svinekød	0.0 gram.
7. Oksekød	0.0 gram.
8. Gulerødder	0.0 gram.
9. Mel	0.0 gram.
10. Havregryn	0.0 gram.
11. Salt	0.0 gram.
12. Wienerbrød	0.0 gram.
13. Lagkage	0.0 gram.
14. Chokolade	0.0 gram.
15. Kaffe	100.0 gram.
-----	
99. Start beregning.	

Indtast et nr:

Fig. 3. Spiseseddel med bestillinger.

Når der tastes 99, skal rettens næringsværdi udregnes og der skal vises et billede med de bestilte varer samt næringsværdi.

## BESTILLING.

Mængde	Produkt	Antal fibre	Pris
200.0	Franskbrød	100.00	1.40
150.0	Ymer	7.50	0.39
100.0	Kaffe	0.00	36.00

Fibre i alt: 107

Pris i alt: 37.79 Kr.

Pris for 1000 fibre: 351.53 Kr.

Tryk på RETUR for at komme til hovedmenu.

Fig. 4. Kostberegning.



Ved at aktivere return kommer man så tilbage til den overordnede oversigt igen, svarende til fig. 1.

2.

Når der taster 2 kommer de data frem som ligger til grund for beregningerne.

Billedet kan se sådan ud:

KOST-DATA.

Nr.	Produkt navn	Fibre pr. 100 g	Pris pr. 100 g
1.	Rugbrød	100.00	0.10
2.	Franskbrød	50.00	0.70
3.	Kartofler	300.00	0.05
4.	Ymer	5.00	0.26
5.	Jordbær	20.00	1.00
6.	Svinekød	0.00	0.89
7.	Oksekød	0.00	5.00
8.	Gulerødder	500.00	0.25
9.	Mel	15.00	8.50
10.	Havregryn	365.00	1.20
11.	Salt	0.00	7.00
12.	Wienerbrød	0.00	6.90
13.	Lagkage	0.00	12.00
14.	Chokolade	0.00	9.50
15.	Kaffe	0.00	36.00

Tryk på RETUR for at komme tilbage til hoved-menuen.

Fig. 5. Kost-data.



Ved at aktivere RETURN kommer man tilbage til den overordnede oversigt igen.

### 3.

Hvis punkt 3 vælges bliver det muligt at ændre i de kostdata, som ligger til grund for beregningerne. Billedet som fremkommer:

#### KOST-DATA.

Nr.	Produkt navn	Fibre pr. 100 g	Pris pr. 100 g
1.	Rugbrød	100.00	0.10
2.	Franskbrød	50.00	0.70
3.	Kartofler	300.00	0.05
4.	Ymer	5.00	0.26
5.	Jordbær	20.00	1.00
6.	Svinekød	0.00	0.89
7.	Oksekød	0.00	5.00
8.	Gulerødder	500.00	0.25
9.	Mel	15.00	8.50
10.	Havregryn	365.00	1.20
11.	Salt	0.00	7.00
12.	Wienerbrød	0.00	6.90
13.	Lagkage	0.00	12.00
14.	Chokolade	0.00	9.50
15.	Kaffe	0.00	36.00

Ændring af kost-data.

Indtast Nr paa produkt, der skal ændres.

(99 = Tilbage til hoved-menuen).

Fig. 6. Kost-data, hvor det er muligt at ændre.

Ved at indtaste et nummer mellem 1 og 15 får man så mulighed for at ændre data for den pågældende råvarer. Her efter indlæses fiberindholdet pr. 100 g og så prisen pr. 100 g.

Når der indtastes 99 kommer man tilbage til den overordnede oversigt igen.

4.

Hvis der i den overordnede oversigt indlæses et 4, skal det være muligt at vælge imellem at en række specielle udskrifter. For at give et overblik er det nødvendigt at lave en oversigt over de mulige udskrifter, der kan laves.

Billedet bliver:

Specielle oversigter:

- 1: Råvarer med fiberindhold større end et angivet antal
- 2: Råvarer der koster mindre end en angivet pris pr 100 g.
- 3: 1 og 2 kombineret.

99: Tilbage til hovedmenu.

Indtast et nr:

Fig. 7: Oversigt over de specielle udskrifter.

Af figuren ses, at det skal være muligt at vælge mellem 3 udskrifter ved at indlæse 1, 2 eller 3.

4.1

Når der indlæses 1 skal det være muligt at lave en udskrift af de råvarer som indeholder mere end x antal fibre pr. 100 g. x skal indlæses.

Indlæsningen kan se sådan ud.

Indlæs grænseværdien for det mindste fiberindhold pr 100 g.: 300

Fig. 8: Indlæsning af mindste antal fibre pr. 100 g.



Herefter vises et skema indeholdende de råvarer som opfyldte den indlæste grænseværdi.

Oversigt over råvarer hvis fiberindhold pr. 100 g. er større end: 300

Nr.	Produkt navn	Fibre pr. 100 g	Pris pr. 100 g
3.	Kartofler	300.00	0.05
8.	Gulerødder	500.00	0.25
10.	Havregryn	365.00	1.20

Aktiver RETURN For at komme tilbage til udskriftsoversigt.

Fig. 9: Råvarer indeholdende flere end "x" fibre pr. 100 g.

Ved at aktivere return kommer man tilbage til oversigten svarende til fig. 7.

#### 4.2.

Ved indlæsning af et 2 få man mulighed for at lave en udskrift over de råvarer som koster mindre end x kr. pr. 100 g; x skal indlæses.



Oversigt over råvarer hvis pris pr. 100 g. er mindre end kr: 10.00

=====

Nr.	Produkt navn	Fibre pr. 100 g	Pris pr. 100 g
1.	Rugbrød	100.00	0.10
2.	Franskbrød	50.00	0.70
3.	Kartofler	300.00	0.05
4.	Ymer	5.00	0.26
5.	Jordbær	20.00	1.00
6.	Svinekød	0.00	0.89
7.	Oksekød	0.00	5.00
8.	Gulerødder	500.00	0.25
9.	Mel	15.00	8.50
10.	Havregryn	365.00	1.20
11.	Salt	0.00	7.00
12.	Wienerbrød	0.00	6.90
14.	Chokolade	0.00	9.50

-----  
Aktiver RETURN For at komme tilbage til udskriftsoversigt.

Fig. 10. Pris på mindre end x kr. pr. 100 g.

Når return aktiveres kommer oversigten vist på fig. 7 frem igen.

#### 4.3

Under punkt 3 er der mulighed for at lave en udskrift, som er en kombination af de to tidligere udskrifter.

Dvs. produkter som indeholder mere end x fibre pr. 100 g og som koster mindre end y kr. pr. 100 g.



De to grænseværdier indlæses igen. Og udskriften bliver:

Oversigt over råvarer hvis pris pr. 100 g. er mindre end kr: 6.00  
og indeholder flere fibre pr. 100 g end: 300  
=====

Nr.	Produkt navn	Fibre pr. 100 g	Pris pr. 100 g
3.	Kartofler	300.00	0.05
8.	Gulerødder	500.00	0.25
10.	Havregryn	365.00	1.20

-----  
Aktiver RETURN For at komme tilbage til udskriftsoversigt.

Fig. 11. Råvarer som indeholder mere end x fibre/100 g  
og koster mindre end y kr./100 g.

Ved at aktivere RETURN kommer man tilbage til oversigten  
svarende til fig. 7.

## PROGRAMUDVIKLING

I dette afsnit vil vi forsøge at udvikle programmet, der skal kunne beregne næringsindholdet i en middagsret. Selve programudviklingen vil tage udgangspunkt i den netop beskrevne analyse af opgaven.

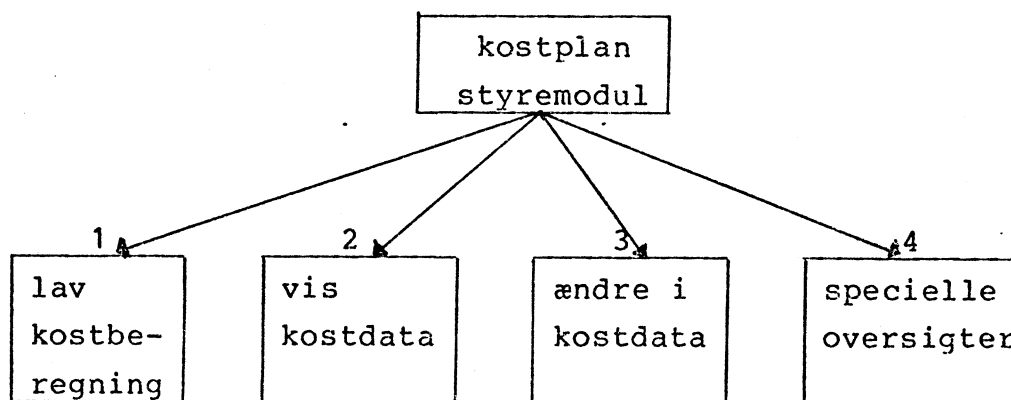
Det var som nævnt et krav fra kunden at programmet var menustyret og i analysen fandt vi ud at der var 4 underpunkter/delopgaver at vælge imellem.

### Programstruktur.

I dette afsnit skal vi fastlægge programmets helt overordnede struktur. Vi tænker slet ikke på nogen detaljer endnu. Det vil afspejle sig i programstrukturen, at der er 4 mulige delopgaver. Samtidig er det nødvendigt med en indbyrdes koordinering/styring af de 4 delopgaver. Ordet modul anvendes om en afgrænset uafhængig del af et program. Programmet vil derfor bestå af

1 hoved/styremodul  
4 moduler

### Moduloversigt.



Herover er vist en skematisk fremstilling af programstrukturen.

I sproget Pascal laves moduler som procedurer. Denne programstruktur vil derfor helt overordnet give følgende Pascalprogram:

```
PROGRAM      Kostplan;
  PROCEDURE Lav_kostberegning;
  BEGIN
  END;

  PROCEDURE Vis_kost_data;
  BEGIN
  END;

  PROCEDURE Aendre_i_kost_data;
  BEGIN
  END;

  PROCEDURE Specielle_oversigter;
  BEGIN
  END;

BEGIN      (* kostplan *)

              Lav_kostberegning;
              Vis_kost_data;
              Aendre_i_kost_data;
              Specielle_oversigter;
              } Hovedprogram

END.
```

Læg mærke til at vi i selve programmet bruger de samme navne som i analysen.

Dette letter den senere detaljerede udvikling og fejlfinding.

### Opgave 1.

- a. Indtast programmet.
- b. Oversæt programmet og få det fri for syntaksfejl.
- c. Hvad sker der når programmet køres/eksekveres.

Kørsel med programmet gøres synlig. *ingen ting*

Når vi kører/eksekverer programmet sker der ikke noget synligt på skærmen. Det kan vi sørge for vha. skrivesætninger.

### Metode.

I hver procedure indsættes 1 skrivesætning, der udskriver, at proceduren er blevet kaldt, og at den ikke virker endnu. Derefter afventes indtastning af et "RETURN" for at fortsætte.

### Eksempel.

Procedure "Lav\_kostberegning" udvides således:

```
PROCEDURE Lav_kostberegning;
BEGIN      (*Lav_kostberegning*)
            WRITELN ('Lav-kostberegning - (er ikke lavet)');
            WRITELN ('Tast RETUR for at fortsætte');
            READLN; (*Lav_kostberegning*)

END;
```

Opgave 2.

Udvid alle 4 procedurer på denne måde, og kørs programmet.

Hvad bliver programmets uddata?

Byt rundt på de 2 første procedurekald. *ingen ting*

Hvad betyder det for uddata?

Visning af skærbillede (hovedmenu).Opgave 3.

Lav en procedure, der kan udskrive hovedmenuen på skærmen.

Kald proceduren for "Vis-hovedmenu".

Indlæsning af nr. for brugerens valg.

Når hovedmenuen er vist, skal hovedprogrammet indlæse det nr. brugeren har indtastet.

Opgave 4.

Lav en variabel, som hedder "nr" af typen "Integer" og indlæs en værdi til den.

Dette skal ske i hovedprogrammet.

Kald af procedurerne afhængigt af nr.

Når nr. er blevet indlæst fra skærmen, skal hovedprogrammet sørge for at kalde den ønskede procedure.

```
Hvis nr. = 1    så kald  Lav_kostberegning
-   nr. = 2    -   -   Vis_kost_data
-   nr. = 3    -   -   Aendre_i_kost_data
-   nr. = 4    -   -   Specielle_oversigter
-   nr. = 99   -   forlad programmet.
```

Opgave 5.

Lav ovenstående ændring/udvidelse i hovedprogrammet.

Opgave 6.

Hvad sker der, når man taster 5 eller 100 ind?

Mulighed for at foretage flere valg i hoved-menuen.

Som programmet er lavet nu, har man kun mulighed for at vælge netop et nr. og derefter stopper programmet.

Opgave 7.

Udvid hovedprogrammet, så der er mulighed for at vælge et vilkårligt antal gange mellem hovedmenuens muligheder. Først, når der indtastes "99", skal programmet forlades. (Brug "While" eller "Repeat").

Nu har vi altså fået lavet den overordnede styring af programmet, og nu skal vi til at udvide de enkelte moduler.

Skærbillede til "Lav-kostberegning".

Proceduren "Lav\_kostberegning" skal udvides, så den kalder en procedure, der kan vise en spiseseddel.

Opgave 8.

Lav en procedure "Vis\_spiseseddel", der kan vise en simplificeret spiseseddel på skærmen. Senere raffinerer vi billedet.

Proceduren skal være lokal i "Lav-kostberegning".

Se evt. figur 2 i analysen.

Simplificeret spiseseddel

1	xxxxx	0 g
2	xxxxx	0 g
3	xxxxx	0 g
.	.	.
.	.	.
.	.	.
15	xxxxx	0 g

Brug følgende ide:

i: INTEGER; (\* i skal være lokal i Vis-spiseseddel\*).

.

.

FOR i = 1 TO 15 DO

BEGIN

Udskriv (en overskrift);

udskriv (i);

udskriv (xxxxxxxxxxx O<sub>g</sub>);

END;

Indlæsning af valg i lav-kostberegning.Opgave 9.

Udvid "Lav\_kostberegning" med en sløjfe/løkke, så brugeren kan indtaste et vilkårligt antal muligheder mellem 1 og 15.

Når der indtastes et tal mellem 1 og 15, skal der kaldes en procedure, som hedder "Behandl\_bestilling".

Opgave 10.

Lav proceduren "Behandl\_bestilling". Den skal være lokal i "Lav-kostberegning".

I første omgang skal den udskrive teksten: "Behandl bestilling - er ikke lavet".

"Tast <Retur> for at fortsætte".

Tabel over levnedsmiddelprodukterne.

Indtil nu har vi kigget på hvorledes programmet skal opføre sig dynamisk, dets programstruktur. Nu skal vi til at se på den anden halvdel af et program, nemlig data-strukturering som består i at organisere de data som indgår i en opgave på fornuftig vis, således at det i programmet er let at overskue og genfinde disse data.

Vi har valgt at begrænse os til 15 forskellige produkter, og for hver af disse produkter skal vi senere gemme nogle data. I første omgang vælger vi at organisere disse data i tabeller. (Senere skal vi se en anden måde at gøre det på, nemlig v.h.a. poster (records)).

Opgave 11.

Lav en tabel, der kan indeholde 15 produktnavne. Et navn må højst bestå af 20 bogstaver. Tabellen skal være global, kald den for: "Produkt\_navn".

IDE

```
CONST
```

```
    Max_antal.....
```

```
TYPE  Navne_tabel = .....
```

```
VAR   Produkt_navn: Navne_tabel;
```

Opgave 12.

Lav en procedure "Init\_produkt\_navn", der lægger navnene på produkterne ind i "Produkt\_navn" tabellen. Sørg for at denne procedure kaldes forest i hovedprogrammet, en initialiseringsdel. Dvs. på de enkelte pladser i tabellen "Produkt\_navn" skal lægges et produktnavn. (Se evt. fig. 2 i analyse hvilke produktnavne der kan indgå).  
Bør denne procedure ligge i et af de 4 moduler eller skal den ligge for sig selv uden for modulerne?

Når vi nu har fået initialiseret "Produkt\_navn" tabellen kan vi få disse data at se ved at udfylde den simplificerede spiseseddel fra opgave 10.

Opgave 13.Udskrift af de rigtige produktnavne i spisesedlen.

Den simplificerede spiseseddel skal udvides, så de rigtige produktnavne kan udskrives i stedet for 'xxxxx' - erne.

Brug følgende ide:

i : INTEGER;

.

.

.

.

.

FOR i: = 1 TO Max\_antal DO

BEGIN

udskriv (i, '.\_');

udskriv (Produkt \_ navn [i]);

udskriv ('0 g');

END;

Sørg selv for at teksten kommer til at stå pænt på skærmen, svarende til oplægget i analysen fig. 2.

### Bestilling.

I tilknytning til "Lav\_kostberegning" skal der laves en registrering af de produkter, der indgår i kostberegningen, dvs. vi skal huske hvor mange gram der er bestilt for de enkelte produkter.

Det vi ønsker at komme frem til er en spiseseddel svarende til fig. 3 i analysen, hvor der f.eks. er bestilt 200 g franskbrød, 150 g ymer og 100 g kaffe.

### Fremgangsmåde.

#### Opgave 14.

Hvilken datatype har bestillingen og hvordan defineres denne type i programmet? Lav en global variabel af den definerede type og kald den "Bestilling".

#### Opgave 15.

Lav en procedure, der sætter alle værdier i "Bestilling" til 0. dvs.: nulstil bestillingen. Kald proceduren "Init\_bestilling".

Hvornår får man brug for nulstilling?

Udskrift af de rigtige vægtangivelser i spisesedlen.

Opgave 16.

Den simplificerede spiseseddel skal udvides, så de bestilte vægtangivelser udskrives i stedet for '0 g'.

Brug følgende skitse:

FOR i: = 1 TO Max\_antal DO

BEGIN

udskriv (i, '.');  
udskriv (Produkt\_navn [i]);  
udskriv (Bestilling [i]);

END;

Indlæsning af vægtangivelse i Lav kortberegning.Opgave 17.

Når brugeren vælger et af de 15 produkter ud fra spisesedlen, skal programmet (proceduren "Behandl\_bestilling") skrive: "Indtast mængde i gram:".

Derefter skal det indlæse tallet brugeren indtaster.

Dette tal gemmes på den relevante plads i "Bestilling".

Opgave 18.

Sørg for at spisesedlen med de bestilte mængdeangivelser bliver ajourført på skærmen under hele bestillingsprocessen

Nu har vi således lavet hele kommunikationen omkring det at lave en bestilling og nu skal vi så til at lave selve kostberegningen. til dette mangler vi nogle data, nemlig fiberindholdet pr. 100 g og prisen pr. 100 g.

Tabel over fiberindhold pr. 100 g.Opgave 19.

Der skal laves en tabel over fiberindholdet i de 15 produkter (pr. 100 g).

Kald tabellen for "Fiber\_indhold", og lav en procedure, som initialiserer denne tabel. Kald proceduren for "Init\_fiber\_indhold". Værdierne kan eventuelt hentes fra fig. 5 i analysen.

Proceduren bør ligge sammen med initialiseringen af "Produkt\_navn" tabellen uden for de 4 moduler.

Tabel over priser pr. 100 g.Opgave 20.

Lav en tabel over prisen i kr. pr. 100 g for de 15 produkter.

Kald tabellen for "Pris" og lav en procedure "Init\_pris" som initialiserer tabellen. (Se evt. fig. 5 i analyse for pris eksempler).

Hvor bør denne procedure ligge i programmet?

Vis kost data på skærmen

Nu har vi fået initialiseret datastrukturene og for at kontrollere værdierne kunne vi meget passende udbygge modulet "Vis\_kost\_data".

Opgave 21.

Proceduren "Vis\_kost\_data" skal udvides, så den viser billedet svarende til fig. 5 i analysen, på skærmen. Proceduren skal afvente indtastning af et "RETURN" inden proceduren forlades.

Forslag: Lav en procedure som hedder "Vis\_produkt\_oversigt", der skriver produkterne med fiberindhold og pris ud (se evt. opg 16). Lad "Vis\_kost\_data" kalde denne procedure.

Vi har nu fået kontrolleret kost data og skulle nu være i stand til at lave proceduren "Lav\_kostberegning" færdig. Vi mangler at lave selve beregningen af bestillingens næringsværdi.

Beregning af fiberindholdet og prisen for en bestilling.Opgave 22.

Udvid proceduren "Lav\_kostberegning" så den kan beregne følgende for den afgivne bestilling:

- antal fibre i alt
- pris for bestilling
- pris for 1000 fibre.

Lav passende variable til at huske disse beregninger.

Opgave 23.

Lav en lokal procedure i "Lav\_kostberegning" som kan vise kostberegningsresultatet svarende til fig. 4 i analysen. Kald proceduren for "Vis\_kost\_beregnings\_resultat". Den kan eventuelt deles i 2 procedurer; een der viser de bestilte produkter og en der viser beregningsresultatet.

Nu er procedurerne (modulerne), "Lav\_kost\_beregning" og "Vis\_kost\_data" færdige.

Næste delopgave som skal udvides er "Aendre\_i\_kost\_data".

Aendre i kost data.

"Aendre\_i\_kost\_data" skal vise en produktoversigt og sørge for at det er muligt at ændre produkternes fiberindhold og pris. (se fig. 6).

Opgave 24.

I "Aendre\_i\_kost\_data" skal der som nævnt vises en produktoversigt.

En procedure til dette er allerede lavet i opg. 21, så den kan genbruges.

Pseudokode for "Aendre\_i\_kost\_data" bliver herefter:

BEGIN

produktnr: = 0;

WHILE produktnr <> 99 DO

BEGIN

Vis\_produkt\_oversigt;

læs produktnr;

IF (1 ≤ produktnr) AND

(produktnr ≤ max\_antal) THEN

Aendre\_data\_for\_produkt (produktnr);

END;

END;

Proceduren "Aendre\_data\_for\_produkt (Nr)" kan være lokal for "Aendre\_kost\_data". "Nr" er parameteren der overføres til proceduren. (I dette tilfælde er parameter overførsel delvis unødvendigt).

Det sidste modul vi mangler at udfylde er "Specielle\_oversigter". I analysen fremgår det at det skal være muligt at vælge mellem 3 forskellige udskrifter. Dvs. dette modul's struktur faktisk skal være den samme som hele programmets, blot i en mindre målestok.

Programstrukturen for det modul vil så blive:

```

PROCEDURE Specielle_oversigter;
  PROCEDURE Fiber_graense;
  BEGIN
  END;

  PROCEDURE Pris_graense;
  BEGIN
  END;

  PROCEDURE Pris_og_fiber_graense;
  BEGIN
  END;

BEGIN
  (*udskrifts styring*)

END;

```

Hvis vi starter med at se på selve udskriftsstyringen (det overordnede niveau) fremgår det af analysen, at der skal præsenteres en oversigt over de mulige udskrifter, samt at det skal være muligt at vælge udskrifter indtil der indlæses 99.

#### Opgave 25.

Indtast den viste programstruktur og lav programmet for udskriftsstyringen. Udskriftsoversigten kan lægges i en procedure, der hedder "Vis\_menu". Afprøv denne styring ved at indføje simple skrivesætninger i disse 4 procedurer.

Nu skal vi til at udfylde proceduren til udskriftsoversigten "Vis\_menu" og de 3 udskriftsprocedurer.

Hvis vi ser på "Vis\_menu" kan vi i analysen fig. 7 se, hvad den skal indeholde. Da vi desuden ønsker at det skal stå pænt på skærmen, får vi brug for at kunne flytte cursoren rundt til et vilkårligt sted på skærmen. Til dette formål findes der indbygget en procedure, der hedder "GOTOXY". Ved at overføre to parametre til den kan cursoren styres.

Eksempel.

```
GOTOXY (10,15).
```

Dette betyder, at cursoren stilles i kolonne 10, linie 15. Da denne procedure er systemafhængig og ikke standard er det bedst at anvende den så få steder som muligt. Af denne grund vil vi opbygge en procedure som samtidig med at flytte cursoren også skriver en tekst.

#### Opgave 26.

Opbyg procedure "Msg" (Message) som kan skrive en tekst et vilkårligt sted på skærmen.

Kaldet skal se sådan ud:

```
Msg (10,15, 'en tekststreng');
```

#### Opgave 27.

Lav proceduren "Vis-menu" v.h.a. "Msg (x,y,str)";

Hvis vi ser på de 3 udskrifter i analysen fig. 9-11 ser vi at de er opbygget omkring et standardskema.

Opgave 28.

Opbyg et standardskema, der kan anvendes i alle 3 udskrifter. Kald proceduren "Lav\_st\_skema".

Desuden fremgår det også af analysen at de enkelte produkter også har et standard format, når de udskrives i standard skemaet.

Opgave 29.

Lav en procedure, der kan udskrive et enkelt produkt i det standard format, der passer i standard skemaet. Kald proceduren: "Udskriv\_st\_format". Husk proceduren skal have overført produktnummeret som parameter.

Nu har vi fået opbygget en række værktøjer, som kan lette den videre programmering af udskrifterne.

Opgave 30.

Lav proceduren "Fiber\_graense", således at det er muligt at indlæse et fiberantal, og derefter få en udskrift af de produkter, der indeholder dette antal eller mere pr. 100 g.

Pseudokode for proceduren "Fiber\_graense":

```

    Udskriv ledetekst
    Læs fiberantal
    Lav overskrift til skema
    Lav_st_skema
    FOR I:= 1 TO Max_antal DO
        IF fiberindhold [I]>= Fiber_antal THEN
            Udskriv_st_format [I];

    READLN ;
  
```

Opgave 31.

Lav proceduren "Pris\_graense" efter samme koncept som "Fiber\_graense". Der ønskes en udskrift af de produkter, der koster mindre end den indlæste pris pr. 100g.

Opgave 32.

Lav proceduren "Pris\_og\_Fiber\_grænse" der er en kombination af opg. 30 og 31.

VED PROGRAMMERING FORSTÅS ALLE DE FASER DER INDGÅR I LØSNINGEN AF ET PROBLEM MED EN DATAMAT.

TIDSMÆSSIGT KAN PROGRAMUDVIKLING/PROGRAMMERING DELES OP I FØLGENDE FASER:

1. PROBLEMFASEN. HVOR IDEER OG PROBLEMER FORMULERES OG EN KRAVSPECIFIKATION UDARBEJDES. TIDSRAMMER FOR OG ØKONOMIEN BAG PROJEKTET FASTSÆTTES.
2. ANALYSEFASEN. HVOR EN NÆRMERE ANALYSE AF SPECIFIKATIONEN EVT. ALTERNATIVE LØSNINGER MED TILHØRENDE ØKONOMIVURDERINGER. HER VÆLGES DEN ENDELIGE LØSNING, OG IND- OG UDDATA SPECIFICERES NÆRMERE.
3. DESIGNFASEN. HVOR DEN VALGTE LØSNING DESIGNES OG PROGRAMMERINGSGRUNDLAGET UDARBEJDES.
4. GENNEMFØRELSEFASEN. I DENNE FASE SKRIVES OG AFPRØVES PROGRAMMER. BRUGER- OG SERVICE-VEJLEDNINGER UDARBEJDES.
5. DRIFTSFASEN. HVOR DEN ENDELIGE PRAKTISKE AFPRØVNING AF SYSTEMET SOM HELHED FORETAGES. HER FOREGÅR ENDVIDERE VEDLIGEHOLDELSE AF PROGRAMMER OG MASKINEL.

I DEN GENNEMGÅENDE OPGAVE "KOSTPLANPROGRAM" ER FASERNE 1-3 UDARBEJDET PÅ FORHÅND, DVS. FASE 4 GENNEMFØRELSEFASEN SKAL UDFØRES I KURSET.

Det er nødvendigt med en systematisk metode til fastlægning af et programs overordnede struktur.

Det vil ikke være en fordel med eet stort program da det er vanskeligt at dokumentere og vedligeholde.

Derfor skal programmet opdeles i mindre afsnit (moduler). denne opdeling skal ikke foretages på tværs af funktionsmæssige forhold i programmet, men netop tage hensyn til de enkelte del-funktioner. en metode til en sådan opdeling i moduler (del-funktioner) kan være funktionsnedbrydning.

Systemdesigneren kan i designfasen foretage en funktionsnedbrydning af projektets hovedfunktion, hvorved der fremkommer programstruktur med et topmodul samt et hierarki af underliggende moduler. Detaljeringsgraden stiger ned igennem niveauerne.

Funktionsnedbrydning er ikke een bestemt teknik og der findes ikke een bestemt løsning. Resultatet afhænger helt af designeren.

Når der er foretaget en funktionsnedbrydning vil opdelingen i programmoduler kunne foretages, og ud fra detaljerede beskrivelser af hvert enkelt modul kan programmet skrives.

Det er denne metode, der er benyttet ved udvikling af den gennemgående opgave. dette vil også fremgå af opgavebeskrivelsen, hvor forløbet vises.

For at en datamat skal kunne udføre et program, skal dette foreligge må maskinkode, dvs. Det "sprog" som den pågældende datamat direkte forstår.

Programmering i maskinkode er imidlertid yderst kompliceret. Hver eneste instruktion har en talkode, som man altså skal meddele maskinen på en eller anden form. For at lette denne kodning har man indført "symbolsk maskinsprog" er et sæt bogstavkoder, der angiver hvilken instruktion der er tale om, og som typisk er lettere at huske end den til instruktionen hørende talkode.

Når der foreligger et program skrevet i symbolsk maskinsprog, skal det oversættes til maskinkode (til dette formål har man et program, der kaldes en "indsætter" eller "assembler"), hvorpå det kan køre på datamaskinen.

Men selv symbolsk maskinsprog kan være yderst kompliceret at programmere i, og for at råde bod på dette opstod der i løbet af 1950'erne og 60'erne et antal "højere programmeringssprog" ("højniveausprog"), der skulle gøre programmering lettere.

Programmer skrevet i højniveausprog har typisk følgende fordele frem for programmer skrevet i symbolsk maskinsprog.

- hurtigere udviklingstid
- lettere at rette og modificere
- lettere at forstå for mennesker
- lettere at indføre andre personer i programmernes virkemåde
- mindre dokumentation nødvendig
- lettere at overføre fra en maskine til en anden

Programmer skrevet i højniveausprog har typisk følgende ulemper frem for programmer skrevet i symbolsk maskinsprog:

- Langsommere eksekveringstid
- Større lagerkrav.

Der findes et stort antal højere programmeringssprog og dialekter heraf. Af de vigtigste kan nævnes: FORTRAN, der er et af de ældste sprog, men netop derfor er meget udbredt trods dets store mangler; ALGOL og de dermed beslægtede sprog PL/1, Pascal og Ada, der gør skrivning af smukke, velstrukturerede programmer let; COBOL, der er specielt velegnet til administrative opgaver; BASIC, der er yderst simpelt og derfor let at lære for nybegyndere, men som har visse meget store mangler, der gør dets praktiske værdi yderst tvivlsom.

## Introduktion til programopbygning

Vi benytter programmeringssprog for at fortælle hvad datamaskinen skal udføre.

Men det er ikke nok at maskinen er i stand til at udføre programmet. Mennesket må også kunne forstå programmerne.

Den mest omkostningskrævende funktion omkring datamaskinen er programudvikling og vedligeholdelse af programmerne.

Der er sædvanligvis mere økonomi i at skrive læsbare og forståelige programmer, fremfor programmer, der "udmærker" sig ved at de afvikles hurtigt på kørselstidspunktet, samt fylder lidt i datamatens lager:

Programmer bør skrives for mennesker. Derfor må der også ved programskrivningen tages hensyn til menneskelige faktorer.

Her resumeres et par menneskelige "svagheder":

Vi kan ikke udføre flere ting samtidig.

Vi er kun i stand til at sammenholde 5-7 ting samtidig.

Som "kompensation" på disse "svagheder" har den menneskelige hjerne et par bemærkelsesværdige evner:

Vi er i stand til at håndtere begreber på en abstrakt og generaliserende måde.

Vi er i stand til at sætte ting i relation til hinanden.

Desuden er vi i stand til at associere (danne forbindelser mellem ting).

For at skrive programmer som er så klare og forståelige som mulig må vi undgå overskridelser af den menneskelige "kapacitet", og i stedet anvende teknikker, der udnytter vores evner til at generalisere, modularisere, abstrahere og associere.

Sædvanligvis er der ikke problemer med små programmer.

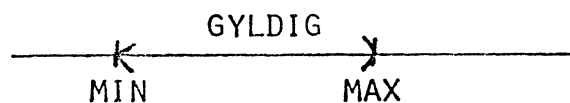
Programmet PROGRAM NO-1 synes dog at indikere at selv små programmer kan være svære at overskue, hvis ikke der ved programmeringsarbejdet tages hensyn til nogle få og simple regler.

De regler der resumeres i det efterfølgende er udelukkende af hensyn til programmøren og en senere læser af programmet. (Oversætteren er ligeglad).

# UOVERSKUELIGT PROGRAM

---

INTERVAL\_TEST:



PROGRAM NO\_1;

VAR DATA\_1, DATA\_2, DATA\_3: INTEGER; OK: BOOLEAN;

BEGIN

READ (DATA\_1, DATA\_2);

OK := TRUE;

WHILE OK <> FALSE

DO BEGIN READ (DATA\_3);

IF (DATA\_3 < DATA\_1) OR (DATA\_3 > DATA\_2) THEN

WRITELN ('FEJL') ELSE

OK := FALSE END;

IF OK THEN; ELSE WRITELN ('OK')

END.

ER SLØJFESTYRINGEN I ORDEN

KAN DU SE FEJLEN I ALGORITMEN

## Kommentarer til PROGRAM NO 1

Programnavn No\_1 indikerer åbenbart at dette er personens første program. Programmet giver derimod ingen antydning af hvad programmet gør.

De variable DATA1, DATA2, DATA3 indikerer ikke noget om deres funktion i programmet.

I programmets krop er det svært at se afgrænsningen af programmets logiske opdeling i separate sekvenser (Programmet består af tre sekvenser?).

Programmets kontrolstruktur fremstår særdeles uklar (der er tre, hvoraf to er flettet).

PROGRAM INTERVALTEST;

```
VAR      MIN:      INTEGER; (*MIN, GRÆNSE*)
        MAX:      INTEGER; (*MAX, GRÆNSE*)
        OKFLAG:    BOOLEAN; (*SLØJFESTYRING*)
        TESTDATA:  INTEGER;
```

BEGIN

```
        (*INDLÆS MAX OG MIN GRÆNSE*)
        WRITELN ('INDTAST 2 HELTAL FOR MIN OG MAX GRÆNSEVÆRDI');
        READLN (MIN, MAX);
```

```
        (*INDLÆS INDTIL TALLET FALDER I DET INDLÆSTE
        INTERVAL*)
```

```
        OKFLAG: = FALSE;
        WHILE NOT OKFLAG DO
        BEGIN WRITE ('INDLÆS TESTDATA');
        READLN (TESTDATA);
```

```
        IF (TESTDATA < MIN)OR (TESTDATA > MAX)THEN
        WRITELN ('FEJL I INDDATA')
        ELSE
        OKFLAG:=TRUE
        END; (*WHILE*)
```

```
        (*UDSKRIV RESULTATET AF TESTEN*)
        WRITELN ('TALLET', TESTDATA:4, 'ER INDENFOR INTERVALLET')
```

END. (\*INTERVALTEST\*)

## Kommentarer til programmet INTERVALTEST

Med udgangspunkt i dette forholdsvis simple program opstilles nogle få og effektfulde programmeringsregler.

### Programstørrelse:

Programmet er af en sådan størrelse at det kan forstås som een enhed. En almindelig regel siger at et program bør bestå af maksimalt 60 linier.

60 linier er valgt da hele programmet derved kan være på een A4 side.

### Læsbarhed:

#### Programnavn:

Brug navne for programmerne således at der allerede heri gives information om programmets funktion.

#### Erklæringer:

Definering af konstanter og typer, samt erklæring af variable kan yderligere bidrage til programmets læsbarhed. Ved valg af navne bør man sørge for at navne så entydigt som muligt angiver funktionen af de enkelte navne.

#### Programstruktur:

Ethvert program vil bestå af et antal programsekvenser.

Den enkelte programsekvens kan ofte yderligere bestå af flettede sekvenser omgivet af en kontrolstruktur. (Valg/gentagelse).

Ved hjælp af linieafstand og eventuelle kommentarer bør de enkelte hovedsekvenser i programmet angives.

#### Programkontrol:

Basale kontrolstrukturer som WHILE, REPEAT-UNTIL, IF-THEN-ELSE bør angives således at læseren umiddelbart kan se, hvor kontrolstrukturen begynder og hvor den slutter.

De programsekvenser, der ligger inden i en kontrolstruktur kan tydeliggøres ved at benytte indrykninger af programteksten.

Det er vigtigt, at den benyttede teknik tydeligt angiver hvad der er den udførende del af programmet, og hvad der er styring (kontrol) af programmet.

#### Kommentarer:

Kommentarer er en forklarende information, der benyttes til at hjælpe læseren til at forstå hvad der foregår i programmet. Gode kommentarer bør være reglen og ikke undtagelsen.

Brug i starten af programmet kommentarer, der angiver det generelle formål med programmet, samt data vedrørende programmør og dato. (Se nærmere under programdokumentation).

Brug kommentarer til at opdele programmet i logiske programsegmenter.

Brug kommentarer til at fortælle hvad programmet gør og hvorfor det gøres. Lad ikke kommentaren angive hvorledes programmet udføres.

Undgå at "overkommentere" programmet: Programmet drukner i kommentarer.

Undgå brugen af kommentarer, der blot angiver hvad man umiddelbart kan læse ud af selve programteksten.

Undgå at "underkommentere" programmet. Kommentarer skal kunne hjælpe andre til at forstå programmet. Tilføj kommentarer når programmet skrives og ikke bagefter.

## Teknik ved nedbrydning af et større program

Et af de vigtigste hjælpemidler ved konstruktion af store og komplekse programmer er vores evne til at behandle problemer på en generel og overordnet måde. Der startes med at fastslå hvad, der skal gøres. Hvorledes det gøres gemmes til en senere detailbearbejdning.

Ved yderligere at opdele problemstillingen i få separate delproblemer, er vi i stand til at overskue hele programmets funktion; men dog kun på en overordnet måde.

De enkelte separate delproblemer kan derefter hver for sig yderligere opdeles i separate delproblemer, der kan behandles uafhængigt af hinanden.

Den ovenfor skitserede problemløsningsteknik kaldes ofte:  
Trinvis raffinering.

Metoden giver en del umiddelbare fordele i programmeringsarbejdet fremfor blot at starte programmeringen på en tilfældig og uorganiseret måde.

Overordnede problemstillinger ses i sammenhæng på et overordnet niveau. Detaljer "gemmes" derimod i de underliggende niveauer.

Ved trinvis raffinering udskydes beslutninger vedrørende delproblemer, hvis løsninger ikke er umiddelbar synlige på "top-niveauet".

Ved opdeling af overordnede problemstillinger i et antal separate underproblemer er det nemmere at realisere programmet ved at flere programmører udvikler hver sit delprogram.

Raffineringsmetoden giver dog ofte problemer, idet det er svært at se hvad der er "toppen" af programmet, d.v.s. det er svært at se hvor problemnedbrydningen skal starte.

For at illustrere metoden vil der i det efterfølgende blive opstillet en problematik vedrørende redigering af tekst. Efter en beskrivelse af hvad tekstredigeringsprogrammet skal gøre, gives der et eksempel på, hvorledes programmet trinvis raffineres, således at det totale program kommer til at bestå af separate funktionelle programafsnit.

Ved funktionelle programafsnit forstås en samling af programkoder (statement), der udfører en bestemt og entydig databasehandling.

Ved den trinvis nedbrydning af programmet benyttes både pseudo-kode og struktogrammer til at beskrive programfunktionen.

## OPERATORER I PASCAL

---

OPERATORERNE BRUGES TIL AT SAMMENSÆTTE VARIABLE TIL UD-  
TRYK.

DER FINDES 3 TYPER OPERATORER

- REGNE OPERATORER
- RELATIONS OPERATORER
- LOGISKE OPERATORER

---

## REGNE OPERATORER

### DEFINITION:

+ : ADDITION  
- : SUBTRAKTION  
\* : MULTIPLIKATION  
/ : REELTALDIVISION  
DIV : "HELTALSDIVISION" (DIVISION OG TRUNKERING)  
MOD : REST EFTER HELTALSDIVISION

### PRIORITET:

HØJEST       \* / DIV MOD  
LAVEST       + -

### EKSEMPLER:

3 \* 5       = INTEGER (15)  
3 - 2       = INTEGER (1)  
3 + 2.0     = REAL       (5.0)  
14/4       = REAL       (3.5)  
14 DIV 4    = INTEGER (3)  
14 MOD 4    = INTEGER (2)  
REAL       := 5        (5.0)

## RELATIONS-OPERATORER

---

### DEFINITION:

= : LIGE MED  
<> : FORSKELLIG FRA  
> : STØRRE END  
< : MINDRE END  
>= : STØRRE END LIGE MED  
<= : MINDRE END LIGE MED

### EKSEMPLER:

'C' < 'D'  
ANTAL > 0  
ANTAL <> 0  
A = B  
A >= B  
FALSE < TRUE

### BEMÆRK:

RELATIONS-OPERATORER VIRKER PÅ SAMTLIGE SIMPLE DATATYPER  
(INTEGER, REAL, CHAR OG BOOLEAN).

RESULTATET ER EN BOOLEAN.

LOGISKE OPERATORER:

---

AND : LOGISK "OG"  
OR : LOGISK "ELLER"  
NOT : LOGISK "IKKE"

PRIORITET:

HØJEST	{	NOT
		AND
LAVEST	}	OR

EKSEMPLER:

P AND Q  
NOT P  
P OR Q AND NOT R

BEMÆRK:

LOGISKE OPERATORER VIRKER KUN PÅ BOOLEAN.

RESULTATET ER EN LOGISK VÆRDI TRUE ELLER FALSE.

# LOGISKE OPERATORER FORTSAT

## AND OR NOT

OPERATORER TIL SAMMANSÆTNING AF LOGISKE UDTRYK.

LOGISKE OPERATORER:

AND "OG"

OR "ELLER"

NOT "IKKE"

## SANDHEDSTABELLER:

A AND B

A \ B	TRUE	FALSE
	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

A OR B

A \ B	TRUE	FALSE
	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

NOT A

A	
TRUE	FALSE
FALSE	TRUE

# OPERATOR PRIORITERING

---

HØJESTE PRIORITET

.  
.  
.

LAVESTE PRIORITET

NOT

|,/,DIV,MOD, AND

+,-,OR

=,&lt;&gt;,&gt;,&lt;,&gt;=,&lt;=, IN

:=

## EKSEMPLER:

(z&lt;50) OR (x&gt;100)

(z&gt;50) AND (z&lt;100) OR NOT (P=Q)

((z+7)&gt;100)

---

## PREDEFINERED TYPEN I PASCAL

DET DER KARAKTERISERER EN VARIABEL AF EN GIVEN TYPE ER VÆRDIOMRÅDE, SAMT DE OPERATORER SOM KAN BENYTTES PÅ VARIABLEN.

### GRUNDTYPER:

- |           |            |
|-----------|------------|
| - INTEGER | HELTAL     |
| - REAL    | REELLE TAL |
| - CHAR    | KARAKTERER |
| - BOOLEAN | BOOLSKE    |

## HELTAL (INTEGER)

---

### DEFINITION:

HELTAL ERKLÆRES SOM INTEGER OG LIGGER I OMRÅDET

- MAXINT..MAXINT

MAXINT ER EN FORUDDEFINERET KONSTANT, DER ER IMPLEMENTATIONS-AFHÆNGIG.

### EKSEMPLER:

134

0

-12

## REELLE TAL (REAL)

---

### DEFINITION:

REELLE TAL ERKLÆRES SOM REAL.

TALOMRÅDE SAMT NØJAGTIGHED ER IMPLEMENTATIONSafhængigt.

### EKSEMPLER:

3.5 - 6.78901 0.1234 4.0

4E3 SVARER TIL  $4 \cdot 10^3$

3.123E-12 SVARER TIL  $3.123 \cdot 10^{-12}$

### BEMÆRK:

TEST ALDRIG REELLE TAL FOR LIGHED (DE ER IKKE EKSakte).

## KARAKTERER (CHAR)

---

### DEFINITION:

KARAKTERER ERKLÆRES SOM CHAR.

NORMALT INDGÅR MINIMUM FØLGENDE TEGN:

1. STORE BOGSTAVER A..Z
2. CIFRE 0..9
3. BLANK-TEGN
4. SPECIALTEGNE + - | / ( ) ' . : ; =

### EKSEMPLER:

EN KARAKTER OMSLUTTES AF APOSTROF-TEGN:

'A'  
'Z'  
'1'  
'|'  
' ' (|BLANK|)

### BEMÆRK:

UNDTAGELSEN ER APOSTROF-TEGNET, DER SKRIVES:

''''

## KARAKTERSÆTTET

## KARAKTERSÆTTET ORDNET IFLG. ASCII

AMERICAN STANDARD CODE FOR INFORMATION INTER-  
CHANGE (ASCII)

Code	Char	Code	Char	Code	Char	Code	Char				
Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex				
0	00	NUL	32	20	SP	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	\$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F		127	7F	DEL

## LOGISKE VÆRDIER (BOOLEAN)

---

### DEFINITION:

LOGISKE VÆRDIER ERKLÆRES SOM BOOLEAN.  
DER FINDES KUN TO LOGISKE VÆRDIER TRUE (SAND) OG FALSE  
(FALSK).

### EKSEMPLER:

TRUE  
FALSE

SIMPEL INPUT  
STANDARDKONTROLLER OG FILEN INPUT

---

DEFINITION: READ(INPUT, VARIABLE..., VARIABLE);  
READLN(INPUT, VARIABLE, ..., VARIABLE);  
INPUT: FIL (SAMLING AF DATA), HVORFRA DATA LÆSES.  
READ: INDLÆSER EN VÆRDI TIL HVER AF DE ANGIVNE  
VARIABLE. NÆSTE READ FORTSÆTTER INDLÆSNING  
FRA STEDET HVOR FOREGÅENDE READ "SLAP".  
READLN: VIRKER SOM READ, MED DEN UNDTAGELSE, AT NÆSTE  
INDLÆSNING SKER FRA POSITION 1 PÅ NÆSTE  
LINIE.

EKSEMPLER: SAMMENLIGNING AF READ OG READLN:

INDDATA:

INDLÆSNING:

RESULTAT:

1 2 3 4 5 6	1 2 3 4 5 6
READ(A,B); READ(C,D);	READLN(A,B); READLN(C,D);
A:= 1, B:= 2 C:= 3, D:= 4	A:= 1, B:= 2 C:= 4, D:= 5

BEMÆRK: STANDARDFILNAVNET INPUT KAN UDELADES.

## INPUT STANDARDFUNKTIONER

---

DEFINITION:    READ(INPUT, VARIABLE, ..., VARIABLE);  
                  READLN(INPUT, VARIABLE, ..., VARIABLE);  
                  READ OG READLN ER DEFINERET UNDER SIMPEL INPUT  
                  EOLN(INPUT): SAND, HVIS SLUTNINGEN AF LINIEN ER  
                                  NÅET, ELLERS FALSK.  
                  EOF(INPUT) : SAND, HVIS SLUTNINGEN AF FILEN  
                                  INPUT ER NÅET, ELLERS FALSK.

EKSEMPLER:       :  
                      (\*TEGNVIS INDLÆSNING OG BEHANDLING AF EN  
                      TEKST. EOLN-MÆRKET BEHANDLES IKKE\*)  
                  WHILE NOT EOF(INPUT) DO  
                      BEGIN  
                          READ(INPUT, CH);  
                          IF EOLN(INPUT) THEN  
                              READLN(INPUT)  
                          ELSE  
                              BEHANDLING  
                      END;  
                      :

BEMÆRK:        INDLÆSNING KAN FORETAGES FOR TYPERNE INTEGER,  
                  REAL OG CHAR.  
                  MAN BØR IKKE BLANDE INDLÆSNING AF TAL OG TEGN.  
                  KAN MAN IKKE UNDGÅ DETTE BØR MAN KUN BENYTT  
                  READLN.  
                  MAN KAN KUN FORETAGE TEGNVIS INDLÆSNING TIL  
                  EN PAKKET DATASTRUKTUR.  
                  STANDARDFILNAVNET INPUT KAN UDELADES.

## SIMPEL OUTPUT

### STANDARDFUNKTIONER OG FILEN OUTPUT

---

DEFINITION: WRITE(OUTPUT, ELEMENT, ...., ELEMENT);  
WRITELN(OUTPUT, ELEMENT, ...., ELEMENT);

OUTPUT: FIL, HVORTIL DATA UDLÆSES

WRITE : UDLÆSER EN VÆRDI FOR HVER AF DE ANGIVNE  
ELEMENTER. NÆSTE WRITE FORTSÆTTER UDLÆSNING  
FRA STEDET, HVOR FOREGÅENDE WRITE "SLAP".

WRITELN: VIRKER SOM WRITE, MED DEN UNDTAGELSE, AT  
NÆSTE UDLÆSNING SKER TIL POSITION 1 PÅ  
NÆSTE LINIE (LINIESKIFT).

EKSEMPLER: T:= 1; B:= 2;

WRITE('TYKKELSEN ER ',T);  
WRITE(' BREDDEN ER ',B);

WRITELN(OUTPUT); (\*LINIESKIFT\*)  
WRITELN('TYKKELSEN ER ',T);  
WRITELN(' BREDDEN ER ',B);

GIVER UDSKRIFTEN:

TYKKELSEN ER 1 BREDDEN ER 2  
TYKKELSEN ER 1  
BREDDEN ER 2

BEMÆRK: STANDARDFILNAVNET OUTPUT KAN UDELADES.

OUTPUT  
STANDARDFUNKTIONER

---

DEFINITION: WRITE (OUTPUT, ELEMENT, ..., ELEMENT);  
WRITELN(OUTPUT, ELEMENT, ..., ELEMENT);  
WRITE OG WRITELN ER DEFINERET UNDER SIMPEL  
OUTPUT.  
PAGE(OUTPUT): SIDESKIFT.  
":": STYRER FORMATET FOR UDLÆSNINGEN.

INTEGER:

(I:N) HELTALLET I UDLÆSES HØJREJUSTERET I ET  
FELT, DER ER N POSITIONER BREDT.

REAL:

(R:N) DET REELLE TAL R UDLÆSES HØJREJUSTERET  
I ET FELT, DER ER N POSITIONER BREDT.  
UDLÆSNINGEN FORETAGES I EKSPONENTIEL  
NOTATION.

(R: N:M) DET REELLE TAL R UDLÆSES I DECIMAL FORM  
MED M DECIMALER I ET FELT, DER ER N  
POSITIONER BREDT.

CHAR:

(C: N) TEGNET C UDLÆSES HØJREJUSTERET I ET FELT,  
DER ER N POSITIONER BREDT.

EKSEMPEL: WRITELN(OUTPUT, I:5, R:10:3, C:13);

BEMÆRK: UDLÆSNING KAN FORETAGES FOR TYPERNE INTEGER, REAL  
CHAR OG BOOLEAN.  
STANDARDFILNAVNET OUTPUT KAN UDELADES.

## EKSEMPLER

Eks 5.A »Udskrivning af lede-tekst ved indlæsning af tal«

Når man indlæser data fra terminal, er det bedst at udskrive en lede-tekst, der angiver, at indtastning ønskes. Følgende eksempel viser dette.

```
program add;
var
  a, b, c, sum: integer;
begin
  writeln(output, 'INDTAST TRE HELTAL');
  read(input, a, b, c);
  sum := a + b + c;
  writeln(output, 'SUMMEN AF TALLENE ER ', sum);
end.
```

Ind- og udlæsning kan se sådan ud:

```
INDTAST TRE HELTAL
12 45 -3
SUMMEN AF TALLENE ER 54
```

Eks 5.B »Indlæsning af tal efter lede-tekst på samme linie«  
Man kan også udskrive en ledetekst og indtaste tal på samme linie.

```
program root;  
  
var  
  tal, rod: real;  
begin  
  write(output, 'INDTAST ET TAL: ');  
  read(input, tal);  
  rod := sqrt(tal);  
  writeln(output, 'KVADRATRODEN ER ', rod);  
end.
```

Konversationen på terminalen kan se sådan ud:

```
INDTAST ET TAL: 12.5  
KVADRATRODEN ER 3.53553
```

Eks 5.C »Tabeludskrift«

Følgende program udskriver  $\sin(x)$ ,  $\cos(x)$  og  $\tan(x)$  for  $x=0.1$  og  $0.2$ .

```
program trigtab;  
  
begin  
  writeln(output, 'FUNKTION': 12, '0.1': 10, '0.2': 10);  
  writeln(output);  
  writeln(output, 'SIN': 12, sin(0.1): 10:4, sin(0.2): 10:4);  
  writeln(output, 'COS': 12, cos(0.1): 10:4, cos(0.2): 10:4);  
  writeln(output, 'TAN': 12, sin(0.1)/cos(0.1): 10:4,  
                                     sin(0.2)/cos(0.2): 10:4);  
end.
```

Følgende tabel fremkommer under udførelsen af programmet:

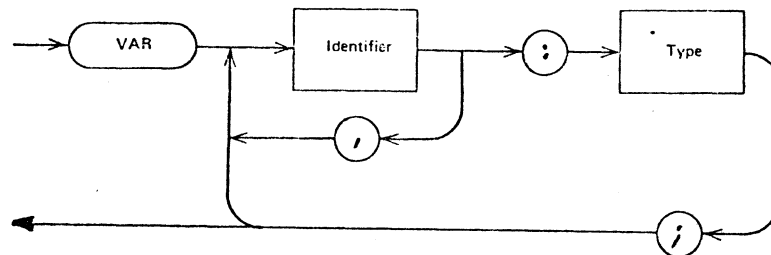
FUNKTION	0.1	0.2
SIN	0.0998	0.1987
COS	0.9950	0.9801
TAN	0.1003	0.2027

## VARIABLE ERKLÆRING

---

DEFINITION: VAR

VARIABELNAVN, ..... ,VARIABELNAVN: TYPE;



EKSEMPLER: VAR

I,J,K: INTEGER;  
R: REAL;  
FLAG: BOOLEAN;  
CH,KAR: CHAR;  
RADIUS,AREAL: REAL;

## PROGRAMSTRUKTUR

---

```
PROGRAM      Kostplan;
  PROCEDURE Lav_kostberegning;
  BEGIN
  END;

  PROCEDURE Vis_kost_data;
  BEGIN
  END;

  PROCEDURE Aendre_i_kost_data;
  BEGIN
  END;

  PROCEDURE Specielle_oversigter;
  BEGIN
  END;

BEGIN      (* kostplan *)

              Lav_kostberegning;
              Vis_kost_data;
              Aendre_i_kost_data;
              Specielle_oversigter;

END.
```

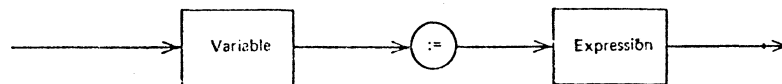
} Hovedprogram

# VÆRDITILSKRIVNING (ASSIGNMENT) OG SIMPEL ARITMETIK

---

DEFINITION: VÆRDITILSKRIVNING (ASSIGNMENT):

VARIABEL := UDTRYK;

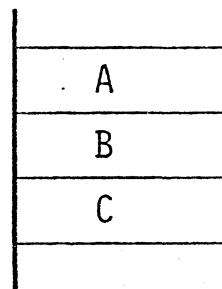
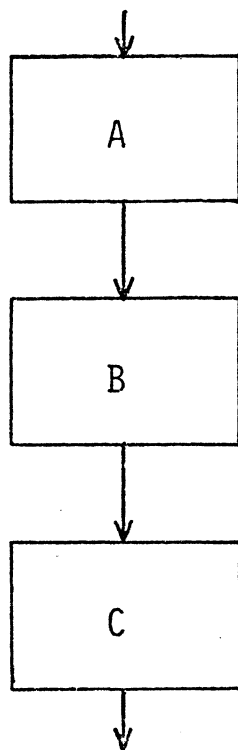


EKSEMPLER: I := 1;  
J := 2;  
K := I

BEMÆRK: VÆRDIEN FOR UDTRYKKET SKAL VÆRE AF SAMME TYPE  
SOM VARIABLEN.  
UNDTAGELSE: EN VARIABEL AF TYPE REAL KAN TILSKRIVES  
EN VÆRDI AF TYPE INTEGER.

## FUNDAMENTALE BYGGEKLODSE (PRIMITIVE)

### SEKVEN S:



UDFØR A

UDFØR B

UDFØR C

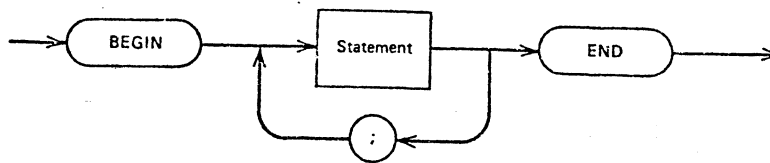
### KARAKTERISTIK:

1. PROGRAMSÆTNINGER UDFØRES I RÆKKEFØLGE.
2. PROGRAM-AFSNIT (SEGMENTER), DER ALLE HAR EEN INDGANG OG EEN UDGANG.

SEKVENSS  
(COMPOUND STATEMENT)

---

DEFINITION: BEGIN  
          SÆTNING;  
          :  
          SÆTNING  
          END

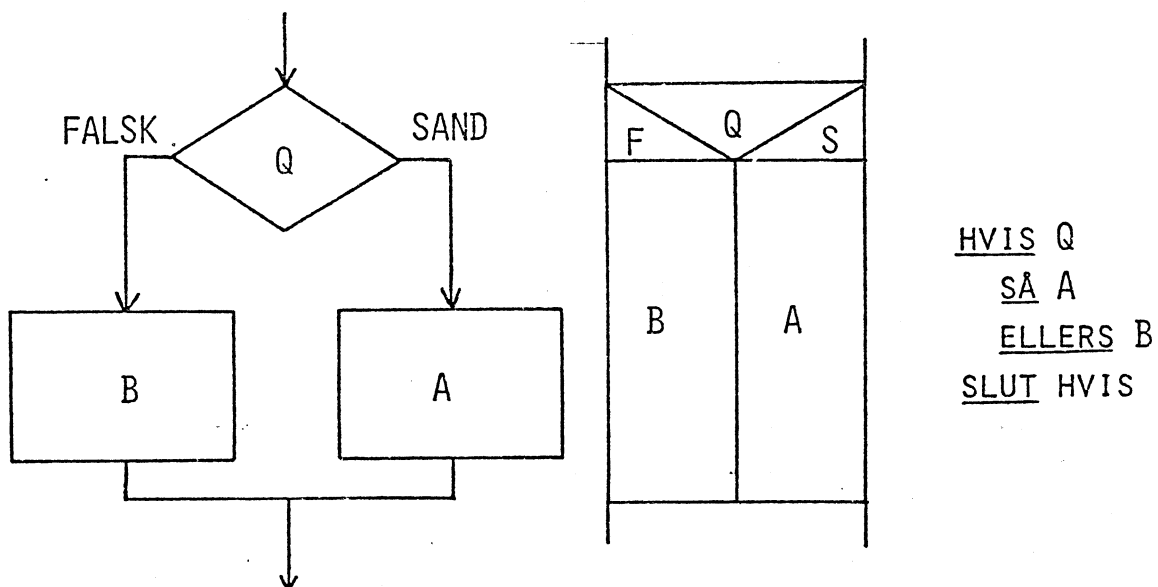


EKSEMPLER: BEGIN  
          WRITELN (OUTPUT, 'INDTAST ET TAL');  
          READLN (INPUT, TAL);  
          SUM:= SUM+TAL  
          END

BEMÆRK:       SEMIKOLON (';') ADSKILLER SÆTNINGER

## FUNDAMENTALE BYGGEKLODSER (FORTSAT)

### UDVÆLGELSE:

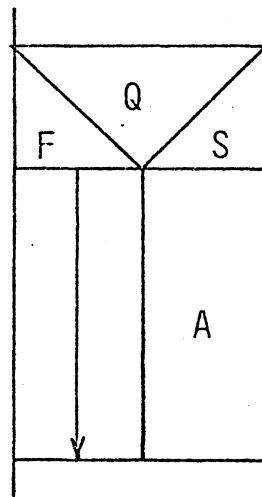
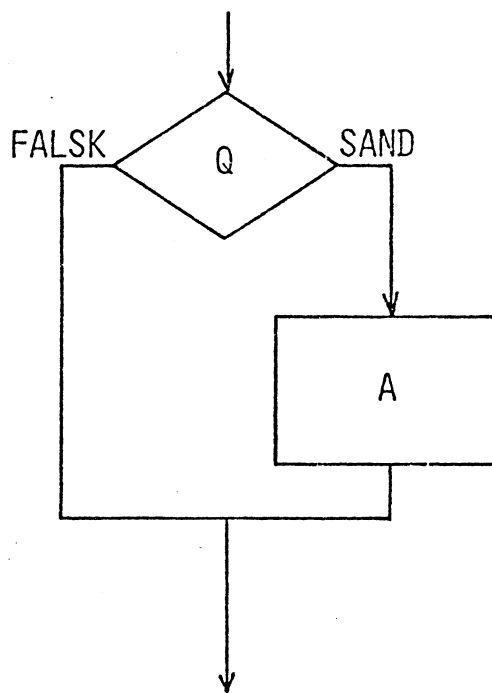


### KARAKTERISTIK:

1. BESTÅR AF EN BESLUTNINGSDEL, SAMT OPERATIONSDELENE A OG B.
2. Q EVALUERES ALTID TIL VÆRDIEN ENTEN SAND ELLER FALSK.
3. A OG B ER ET PROGRAMSEGMENT AF VILKÅRLIG KOMPLEKSITET.
4. VALGET AF OM A ELLER B SKAL UDFØRES, BESTEMMES AF Q'S VÆRDI.

## FUNDAMENTALE BYGGEKLODSE (FORTSAT)

### U D VÆ L G E L S E (FORTSAT):



HVIS Q  
SÅ A  
SLUT HVIS

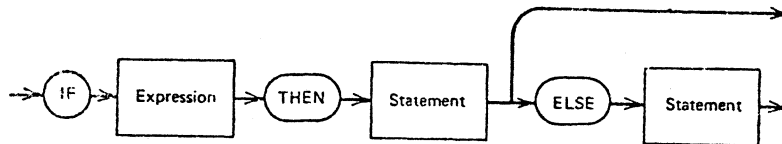
### KARAKTERISTIK:

1. SOM FOR HVIS-SÅ-ELLER, MEN MED EN "TOM" ELLERS-DEL.

## UDVÆLGELSE IF-SÆTNINGEN

---

DEFINITION: IF LOGISK UDTRYK THEN SÆTNING  
ELLER  
IF LOGISK UDTRYK THEN SÆTNING ELSE SÆTNING



EKSEMPLER: IF A>B THEN  
MAX:= A  
ELSE  
MAX:= B;  
⋮

PRAKTISK  
NOTATION: IF LOGISK UDTRYK THEN  
BEGIN  
SÆTNING;  
SÆTNING;  
SÆTNING  
END  
ELSE  
BEGIN  
SÆTNING;  
SÆTNING;  
SÆTNING  
END;  
⋮

## EKSEMPLER

### Eks 6.A »Sammenlign to tal«

Indlæs to reelle tal og udskriv det største.

```
program greatest;  
var  
    x, y, max: real;  
begin  
    writeln(output, 'INDTAST TO REELLE TAL: ');  
    readln(input, x, y);  
    if x > y then max := x else max := y;  
    writeln(output, 'DET STØERSTE TAL ER ', max);  
end.
```

### Eks 6.B »Sammenlign tre tal«

Indlæs tre heltal og udskriv det største.

```
program greatest2;  
var  
    largest, x, y, z: integer;  
begin  
    writeln(output, 'INDTAST TRE HELTAL: ');  
    readln(input, x, y, z);  
    if x > y  
    then if x > z  
         then largest := x  
         else largest := z  
    else if y > z  
         then largest := y  
         else largest := z;  
    writeln(output, 'DET STØERSTE TAL ER ', largest);  
end.
```

Eksemplet viser, hvordan if-sætninger kan indgå i andre if-sætninger.

**Eks 6.C »Sammenlign to tal – alternativ løsning«**

Samme opgave som i Eks 6.B, men if-sætningen erstattes med følgende tre sætninger:

```
largest: = x;
if y > largest then largest: = y;
if z > largest then largest: = z;
```

**Eks 6.D »Simpel test på inddata«**

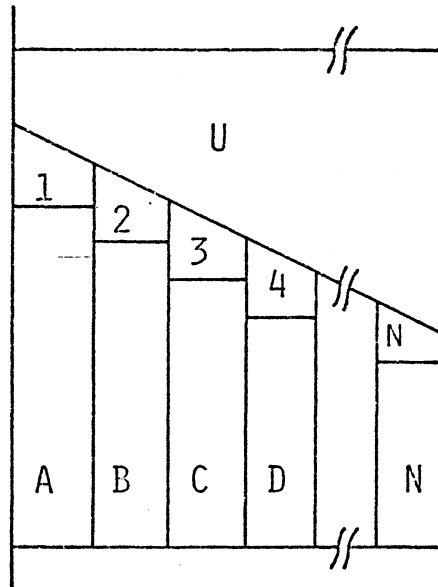
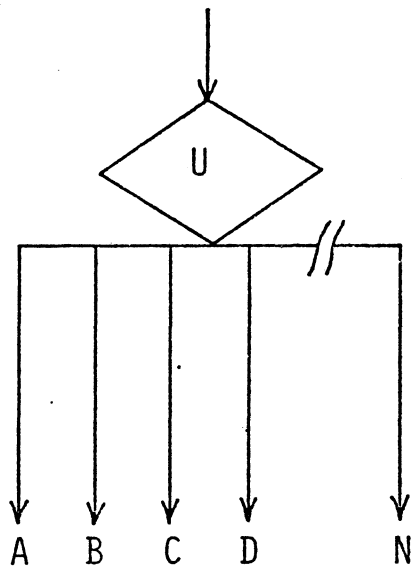
Programmet indlæser et heltal og beregner kvadratet og kvadratroden af tallet. Der udføres test på, om det indlæste tal er negativt, og i givet fald laves en fejludskrift.

```
program calc;

var
  number: integer;
begin
  writeln(output, 'INDTAST ET HELTAL: ');
  readln(input, number);
  if number < 0
  then writeln(output, 'TALLET ER NEGATIVT: ', number)
  else begin
    writeln(output, 'KVADRATRODEN ER ', sqrt(number));
    writeln(output, 'KVADRATET ER ', sqr(number));
  end;
end.
```

## FUNDAMENTALE BYGGEKLODSER (FORTSAT)

### M U L T I P L E   V A L G :



CASE U AF

KONST1: A

KONST2: B

KONST3: C

KONST4: D

⋮

KONSTN: N

### KARAKTERISTIK:

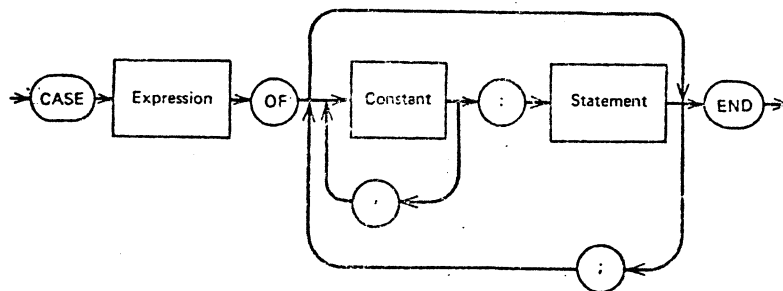
1. U ER ET UDTRYK, DER EVALUERES TIL EN VÆRDI.
2. A-N ER SEPARATE KODESEGMENTER AF VILKÅRLIG KOMPLEKSITET.
3. NÅR DEN EVALUEREREDE VÆRDI SVARER TIL EN AF KONSTANTERNE (KONST 1-N) I KONSTANTLISTEN, UDFØRES DET DERTIL SVARENDE KODESEGMENT.

## UDVÆLGELSE CASE-SÆTNINGEN

DEFINITION: CASE VALGUDTRYK OF

$V_1$ : SÆTNING;  
 $V_2$ : SÆTNING;  
⋮  
 $V_N$ : SÆTNING  
END

HVOR  $V_1..V_N$  ER LISTER AF KONSTANTER AF SAMME TYPE  
SOM VALGUDTRYKKET. (IKKE REAL).



EKSEMPLER: VAR  
MAANED, DAGE: INTEGER;

CASE MAANED OF  
1,3,5,7,8,10,12 : DAGE:= 31;  
2 : DAGE:= 29;  
4,6,9,11 : DAGE:= 30  
END

BEMÆRK: HVIS VÆRDIEN AF VALGUDTRYKKET IKKE SVARER TIL  
EEN AF DE KONSTANTE VÆRDIER I LISTEN, VIL MAN  
FÅ EN FEJLUDSKRIFT OG EFFEKTEN AF CASE-SÆTNINGEN  
VIL VÆRE UDEFINERET.

*Example 3I*

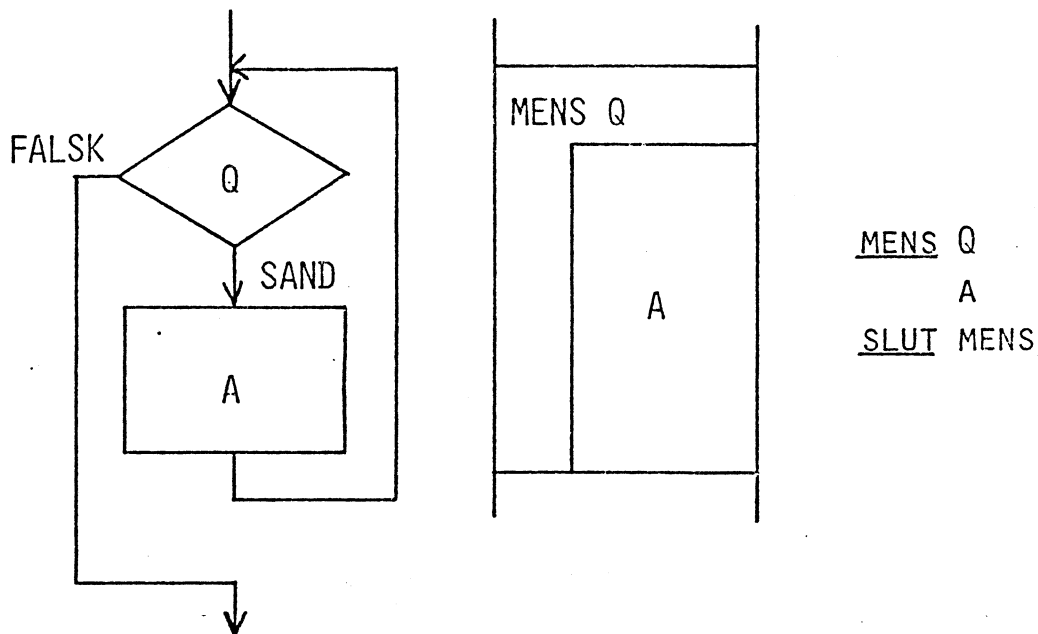
```
(* READ NO, PRINT DAY OF WEEK *)
PROGRAM EX3I(INPUT,OUTPUT);
VAR DAYNO :INTEGER;
BEGIN
  READ(DAYNO);
  CASE DAYNO OF
    1: WRITELN('MONDAY');
    2: WRITELN('TUESDAY');
    3: WRITELN('WEDNESDAY');
    4: WRITELN('THURSDAY');
    5: WRITELN('FRIDAY');
    6: WRITELN('SATURDAY');
    7: WRITELN('SUNDAY')
  END (*OF CASE*)
END.
```

*Example 4B*

```
(* PROGRAM TO ACT AS A HAND CALCULATOR *)
PROGRAM EX4B(INPUT,OUTPUT);
VAR OPERATOR      :CHAR;
    ANSWER,NEWNO  :REAL;
BEGIN
  ANSWER := 0; OPERATOR := '+';
  REPEAT
    READ (NEWNO);
    CASE OPERATOR OF
      '+': ANSWER := ANSWER + NEWNO;
      '-': ANSWER := ANSWER - NEWNO;
      '*': ANSWER := ANSWER * NEWNO;
      '/': ANSWER := ANSWER / NEWNO
    END;
    READ (OPERATOR)
  UNTIL OPERATOR = '=' ;
  WRITELN ('ANSWER IS', ANSWER)
END.
```

## FUNDAMENTALE BYGGEKLODSE (FORTSAT)

### G E N T A G E L S E:



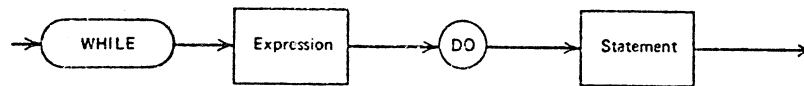
### KARAKTERISTIK:

1. BESTÅR AF EN BESLUTNINGSDEL (Q) OG EN OPERATIONSDEL (A).
2. Q ER ET LOGISK UDTRYK, DER ALTID EVALUERES TIL VÆRDIEN SAND ELLER FALSK.
3. A ER ET PROGRAMSEGMENT AF VILKÅRLIG KOMPLEKSITET.
4. A UDFØRES INDTIL Q EVALUERES TIL VÆRDIEN FALSK. HVIS Q ER FALSK FRA STARTEN UDFØRES A IKKE.

# GENTAGELSE WHILE-SÆTNINGEN

---

DEFINITION: WHILE LOGISK UDTRYK DO SÆTNING



EKSEMPLER:

```
:  
(*SUMMEN AF DE FØRSTE 10 HELTAL*)  
SUM:= 0; ANTAL:= 10;  
WHILE ANTAL > 0 DO  
  BEGIN  
    SUM:= SUM+ANTAL;  
    ANTAL:= ANTAL -1  
  END;  
:
```

PRAKTISK  
NOTATION:

```
WHILE LOGISK UDTRYK DO  
  BEGIN  
    SÆTNING;  
    SÆTNING;  
    SÆTNING  
  END;  
:
```

## EKSEMPLER

### Eks 6.E »Summation«

Heltal indlæses og summeres. Summationen afsluttes, når tallet 0 er indlæst.

```
sum := 0;
readln(input, i);
while i <> 0
do begin
    sum := sum + i;
    readln(input, i);
end;
```

### Eks 6.F »Beregning af produkt«

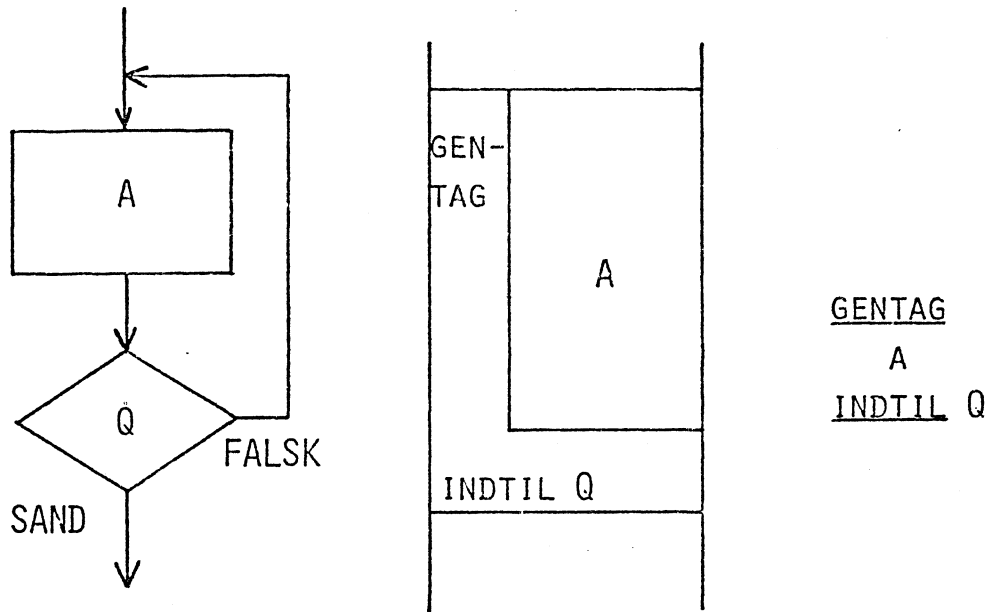
Beregn antallet af led, som er nødvendige for at

$$3^2 \cdot 6^2 \cdot 9^2 \cdot \dots \cdot (3i)^2 > 10000$$

```
prod := 1;
k := 3;
antal := 0;
while prod <= 10000
do begin
    prod := prod * sqr(k);
    k := k + 3;
    antal := antal + 1;
end;
```

## FUNDAMENTALE BYGGEKLODSE (FORTSAT)

### G E N T A G E L S E (FORTSAT):



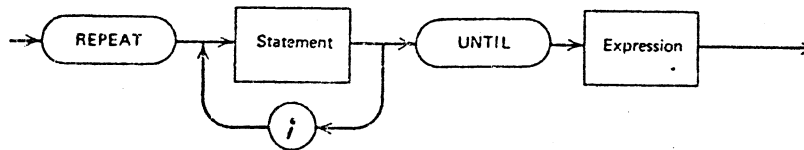
### KARAKTERISTIK:

1. BESTÅR AF EN BESLUTNINGSDEL (Q) OG EN OPERATIONSDEL (A).
2. Q ER ET LOGISK UDTRYK, DER ALTID EVALUERES TIL VÆRDIEN SAND ELLER FALSK.
3. A ER ET PROGRAMSEGMENT AF VILKÅRLIG KOMPLEKSITET.
4. A UDFØRES INDTIL Q EVALUERES TIL VÆRDIEN SAND. HVIS Q ER SAND FRA STARTEN UDFØRES A EEN GANG.

## GENTAGELSE REPEAT-SÆTNINGEN

---

DEFINITION: REPEAT SÆTNING ;....;SÆTNING UNTIL LOGISK UDTRYK



EKSEMPLER:

```
⋮
(*SUMMEN AF DE 10 FØRSTE HELTAL*)
SUM:= 0; ANTAL:= 10;
REPEAT
    SUM:= SUM+ANTAL;
    ANTAL:= ANTAL-1
UNTIL ANTAL<= 0;
⋮
```

PRAKTISK  
NOTATION:

```
REPEAT
    SÆTNING;
    SÆTNING;
    SÆTNING
UNTIL LOGISK UDTRYK;
⋮
```

## EKSEMPLER

### Eks 6.H »Summation igen«

Samme som i Eks 6.E, dvs summation af tal, som indlæses.

Summationen afsluttes, når tallet 0 er indlæst.

```
sum := 0;
repeat
  readln(input, i);
  sum := sum + i;
until i = 0;
```

### Eks 6.I »Indlæsning, bearbejdning og udskrivning af tal igen«

Her løser vi samme opgave som i Eks 6.G med **repeat**-sætningen.

```
writeln(output, 'INDTAST DATA: ');
readln(input, tal);
repeat
  bearbejd tal

  writeln(output, resultat);
  readln(input, tal);
until tal = slutvaerdi;
```

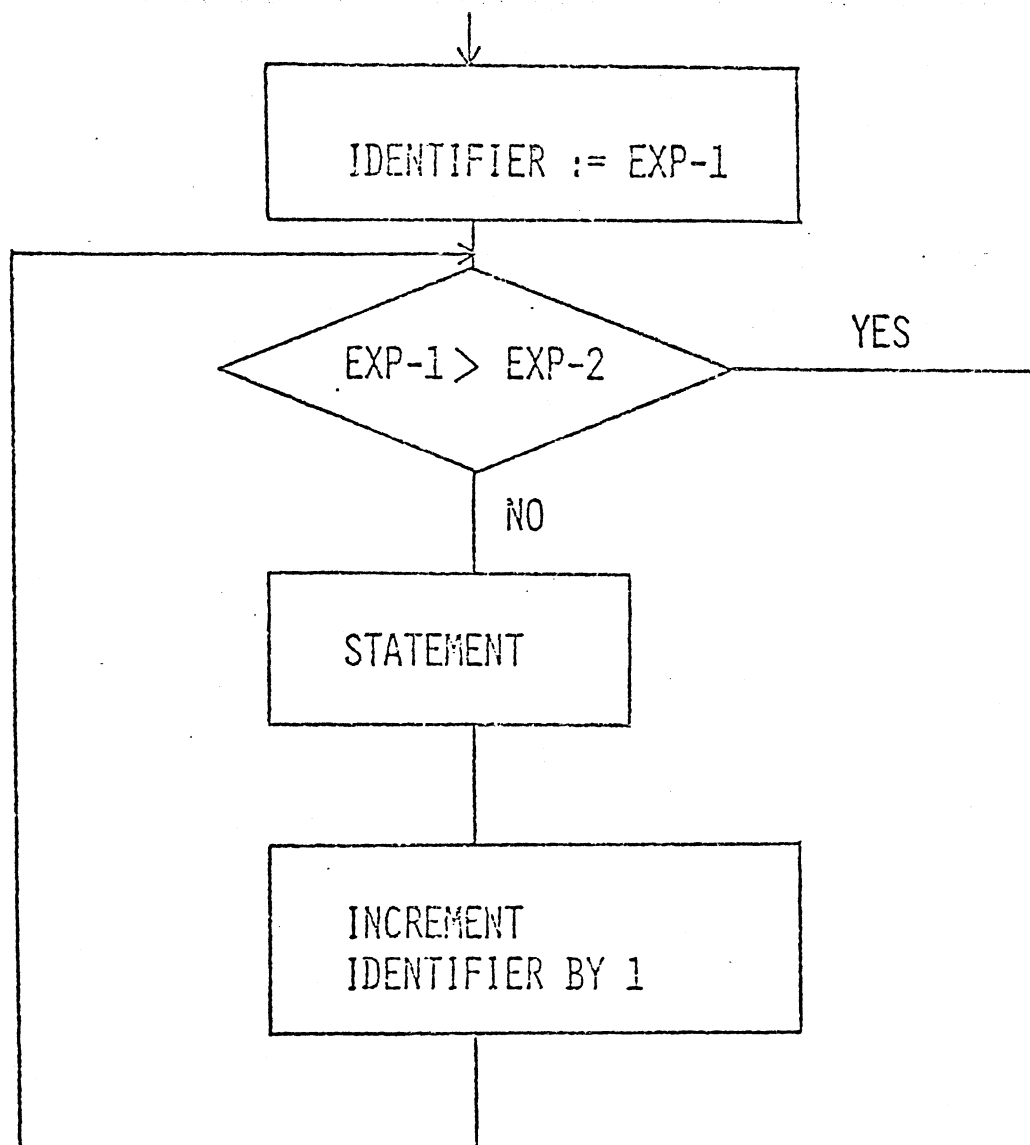
---

```
var
  slutflag: boolean;
begin
  .
  .
  slutflag := false;
  repeat
    indlæs et tal
    if tal = slutvaerdi
    then slutflag := true
    else begin
      bearbejdning af tallet
      udskrift af resultatet
    end;
  until slutflag;
  .
  .
end.
```

GENTAGELSE  
FOR-SÆTNINGEN

---

FOR IDENTIFIER := EXPRESSION-1 TO EXPRESSION -2 DO STATEMENT

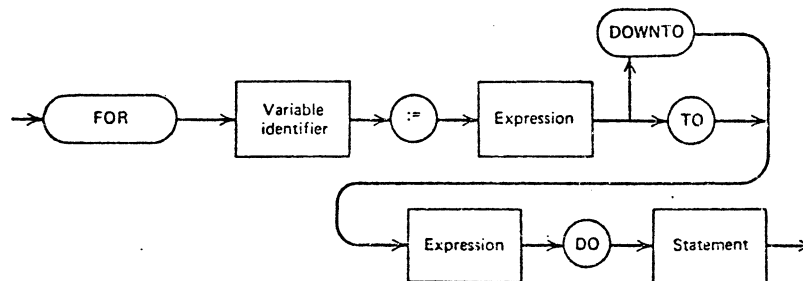


"DOWNTO" IS ALSO AVAILABLE

## GENTAGELSE FOR-SÆTNINGEN

---

DEFINITION: FOR STYREVARIABLE:= STARTVÆRDI TO SLUTVÆRDI  
DO SÆTNING  
  
ELLER  
FOR STYREVARIABLE:= STARTVÆRDI DOWNTO SLUTVÆRDI  
DO SÆTNING



EKSEMPLER:

```
⋮
(*SUMMEN AF DE 10 FØRSTE HELTAL*)
SUM:= 0;
FOR ANTAL:= 1 TO 10 DO
    SUM:= SUM+ANTAL;
⋮
```

PRAKTISK  
NOTATION:

```
FOR STYREVARIABLE:= START TO SLUT DO
    BEGIN
        SÆTNING;
        SÆTNING;
        SÆTNING
    END;
⋮
```

GENTAGELSE  
FOR-SÆTNINGEN

---

REGLER I FORBINDELSE MED FOR-SÆTNINGER:

1. STYREVARIABLEN I EN FOR-SÆTNING HAR EN UDEFINERET VÆRDI, NÅR FOR-SÆTNINGEN AFSLUTTES.
2. STYREVARIABLEN OG SLUTVÆRDIEN MÅ IKKE ÆNDRES INDE I FOR-SÆTNINGEN.
3. SÅFREMT MAN HAR EN FOR-SÆTNING INDE I EN ANDEN FOR-SÆTNING, KAN DE TO SÆTNINGER IKKE BENYTTE SAMME STYREVARIABLE.
4. SÅFREMT STARTVÆRDI ER STØRRE END SLUTVÆRDI I FOR-TO-DO-SÆTNINGEN (MINDRE END SLUTVÆRDI FOR-DOWNT-DO-SÆTNINGEN) UDFØRES SÆTNING IKKE.

**Eks 6.M »Tabeludskrift«**

Lav en tabel over  $n!$ , dvs  $1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$

```
program faktab;
var
  n, fak: integer;
  i: integer;
begin
  writeln(output, 'INDTAST ET HELTAL < 8:');
  readln(input, n);
  writeln(output);
  writeln(output, 'N' : 3, 'N!' : 10);
  writeln(output);
  fak := 1;
  for i := 1 to n
  do begin
    fak := fak * i;
    writeln(output, i: 3, fak: 10);
  end;
end.
```

Såfremt man indtaster værdien 7 fås følgende tabel:

N	N!
1	1
2	2
3	6
4	24
5	120
6	720
7	5040

**Eks 6.N »Oversættelsestabel«**

Oversættelsestabel mellem Celsius og Farenheit grader. Formel:

$$\text{Farenheit} = 9/5 \cdot \text{Celsius} + 32$$

Tabellen omfatter intervallet 0–99 Celsiusgrader.

```
program tempconv;
var
  c, i, j: integer;
begin
  for i:= 1 to 4 do write(output, 'C': 5, 'F': 5, ' ');
  writeln(output);
  for i:= 99 downto 75
  do begin
    c:= i;
    for j:= 1 to 4
    do begin
      write(output, c: 5, round(1.8*c+32): 5, ' ');
      c:= c - 25;
    end;
    writeln(output);
  end;
end.
```

Begyndelsen af tabellen bliver:

C	F	C	F	C	F	C	F
99	210	74	165	49	120	24	75
98	208	73	163	48	118	23	73
97	207	72	162	47	117	22	72
96	205	71	160	46	115	21	70
95	203	70	158	45	113	20	68
94	201	69	156	44	111	19	66
93	199	68	154	43	109	18	64

1985.08.30

NH/kc/6/2

I N T R O D U K T I O N   T I L

P R O C E D U R E R   O G

F U N K T I O N E R

ERKLÆRING:

PROCEDURE PROCEDURENAVN (INDP : TYPE;...; INDP :TYPE;

VAR UDP : TYPE;...;VAR UDP : TYPE);

DEFINITIONER OG ERKLÆRINGER

BEGIN

SÆTNINGER

END;

EN PROCEDURE BESTÅR AF ET PROCEDUREHOVED OG EN PROCEDUREKROP.

PROCEDUREHOVEDET DEFINERER PROCEDURENS GRÆNSEFLADE TIL OMGIVELSERNE.

PROCEDUREKROPPEN ER EN SELVSTÆNDIG BLOK, BESTÅENDE AF DEFINITIONER, ERKLÆRINGER OG SÆTNINGER PRÆCIS SOM I ET HOVEDPROGRAM.

EN PROCEDURE INDLEDES DOG ALTID MED ORDET PROCEDURE.

KALD AF:

EN PROCEDURE AKTIVERES VED AT SKRIVE PROCEDURENAVNET.

PROGRAM ABC;

  (\*DEMO AF SIMPEL PROCEDURE\*)

VAR X,Y,Z: INTEGER;

  PROCEDURE SKRIVEUD;

  BEGIN

    WRITELN(X\*Y\*Z);

  END; (\*SKRIVEUD\*)

  BEGIN (\*ABC\*)

    READLN (X,Y,Z);

    SKRIVEUD

  END  (\*ABC\*)

PROGRAM VARIABLE

VAR

A: INTEGER

B: INTEGER

PROCEDURE NO\_1:

VAR

C: INTEGER

D: INTEGER

PROCEDURE NO\_2:

VAR

A: INTEGER

E: INTEGER

LOKALE: A OG E

GLOBALE: B, C OG D

LOKALE: C OG D

GLOBALE: A OG B

LOKALE: A OG B

LOKALE PROCEDURER, VARIABLE, KONSTANTER OG TYPER

EN PROCEDURE VARIABLE, DER ER ERKLÆRET I EN PROCEDURE (BLOK), SIGES AT VÆRE LOKAL TIL DENNE BLOK.

KONSTANTER OG TYPER ER LIGELEDES LOKALE, NÅR DE ER DEFINERET INDE I EN BLOK.

GLOBALE PROCEDURER, VARIABLE, KONSTANTER OG TYPER

I EN BLOK KALDES DE VARIABLE OG PROCEDURER, DER ER ERKLÆRET I OVERORDNEDE BLOKKE FOR GLOBALE.

KONSTANTER OG TYPER ER LIGELEDES GLOBALE, NÅR DE ER DEFINERET I OVERORDNEDE BLOKKE.

---

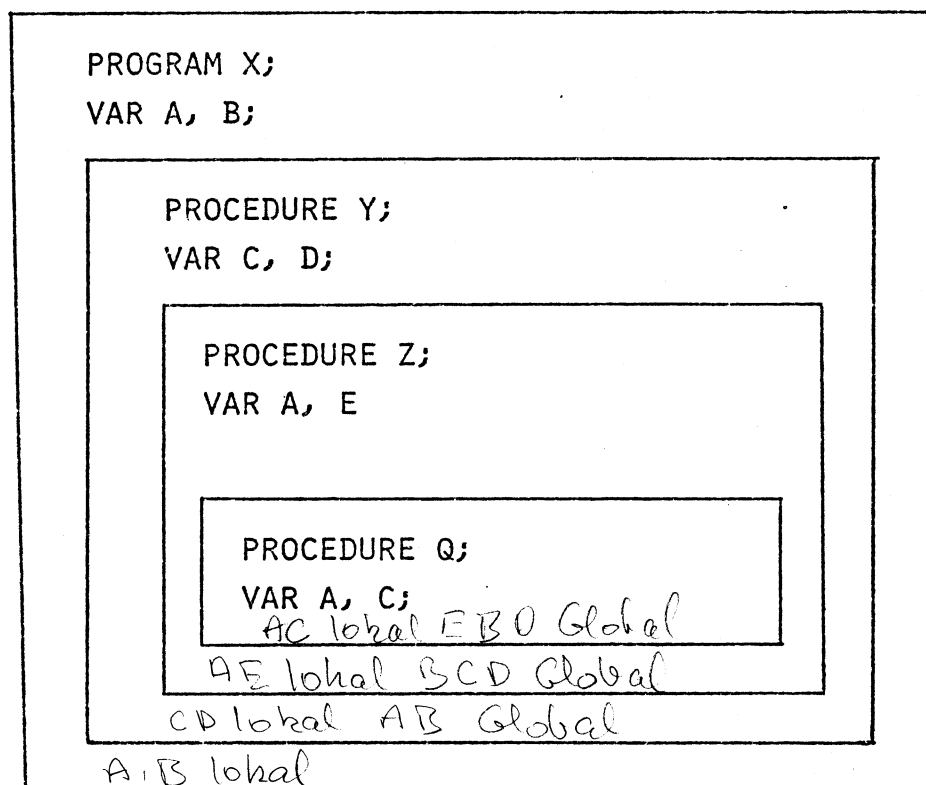
PROCEDURER KAN ERKLÆRES INDEN I EN ANDEN PROCEDURE.  
MAN SIGER DA, AT DE ER FLETTEDE (NESTEDE), UDTRYKKET  
- EN LOKAL PROCEDURE - KAN OGSÅ BRUGES.

STRUKTUREN AF ET PASCAL-PROGRAM KAN SÅLEDES BESTÅ AF  
ET ANTAL BLOKKE FLETTET INDEN I HINANDEN.

ET NAVNS RÆKKEVIDDE (SCOPE) ER DET OMRÅDE, HVORI  
NAVNET MÅ BRUGES (REFERERES).

ET NAVNS SCOPE DEFINERES SOM:

- I. DEN BLOK HVORI NAVNETS ERKLÆRING FOREKOMMER, SAMT  
I ALLE DE BLOKKE, SOM ER INDESLUTTET I DENNE BLOK,  
SE DOG REGEL II.
- II. HVIS NAVNET GENERKLÆRES I EN FLETTET BLOK ER  
NAVNET IKKE TILGÆNGELIGT I DEN NYE OG UNDER-  
LIGGENDE BLOKKE.



VURDER SCOPE (RÆKKEVIDDEN) AF DE TRE VARIABLE MED NAVNET A.

I PASCAL ER ET NAVN TILGÆNGELIGT I HELE DEN BLOK, I HVILKEN DET ER DEFINERET, OG I DE UNDERLIGGENDE BLOKKE I HVILKEN NAVNET IKKE ER BLEVET REDEFINERET.

FOR PROGRAM X ANGIV:

LOKALE NAVNE I X:

AB

kan bruge Y

FOR RUTINE Y ANGIV:

LOKALE NAVNE I Y:

CD

GLOBALE NAVNE I X:

AB

ZY

FOR RUTINE Z ANGIV:

LOKALE NAVNE I Z:

AE

GLOBALE NAVNE I Y:

CD

GLOBALE NAVNE I X:

B

QYZ

FOR RUTINE Q ANGIV:

LOKALE NAVNE I Q:

AC

GLOBALE NAVNE I Z:

E

GLOBALE NAVNE I Y:

D

GLOBALE NAVNE I X:

B

YZQ

DET RIGTIGE STED AT DEFINERE/ERKLÆRE SINE NAVNE GIVES OFTE HELT NATURLIGT VED DESIGN AF PROGRAMMET.

FØLGENDE REGLER KAN VÆRE EN RETTESNOR:

DEFINER/ERKLÆR ET NAVN I DEN BLOK, HVORI DEN BRUGES, DETTE GIVER LÆSBARHED OG MINIMERER RISIKOEN FOR EN UTILSIGTET REFERENCE TIL NAVNET.

NÅR ET NAVN SKAL BRUGES I FLERE BLOKKE DEFINERES/ERKLÆRES NAVNET I DEN BLOK, SOM UMIDDELBART OMSLUTTER "BRUGEN" AF NAVNET.

*simple tællere og hjælpe variable er altid lokale*

BEMÆRK:

NÅR EN PROCEDURE SKAL BEVARE EN VARIABEL FRA GANG TIL GANG, SKAL VARIABLEN ERKLÆRES I DEN BLOK, DER SIKRER, AT VARIABLEN "OVERLEVER" PROCEDUREKALDET.

GLOBALE VARIABLE BØR IKKE BENYTTES TIL DATAOVERFØRSEL  
MELLEM PROGRAMMODULER.

HER NÆVNES HVORFOR:

- HVIS FLERE PROGRAMMØRER ARBEJDER SAMMEN OM ET PROJEKT  
SKAL ALLE KENDE VARIABELNAVNE PÅ GLOBALNIVEAU.

- VANSKELIGGØR PROGRAMAFPRØVNINGEN, IDET EN PROCEDURE  
IKKE KOMMER TIL AT VIRKE SOM ET SELVSTændIGT PROGRAM  
MED VELDEFINERET GRÆNSEFLADE.

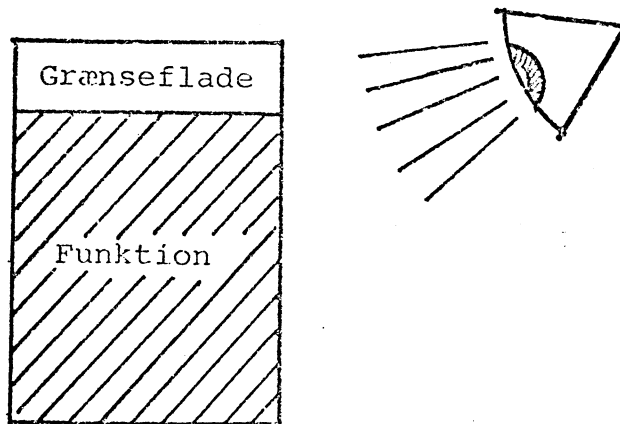
- VANSKELIGGØR GENBRUG AF PROCEDURER FORDI:

VARIABELNAVNE IKKE I OVERENSSTEMMELSE MED NYT  
PROGRAM.

KAN KUN ARBEJDE PÅ ET SÆT VARIABELNAVNE I SAMME  
PROGRAM.

- GIVER PROBLEMER VED PROGRAMVEDLIGEHOLEDSE DA PROCE-  
DURERNE IKKE HAR VELDEFINERET GRÆNSEFLADE.

SOM BRUGER AF ET MODUL ØNSKER VI KUN AT SE GRÆNSEFLADE OG FUNKTION.



GRÆNSEFLADEN ER EN BESKRIVELSE AF HVILKE DATATYPER SOM MODULET SKAL HAVE UD/IND FOR AT UDFØRE DEN ØNSKEDE FUNKTION.

PARAMETRE BRUGES TIL AT GIVE MODULER DENNE VELDEFINEREDE GRÆNSEFLADE.

```
PROCEDURE PROCEDURENAVN (INDP: TYPE,,,,; INDP: TYPE;  
    VAR UDP: TYPE,,,,; VAR UDP: TYPE);
```

DEFINITIONER OG ERKLÆRINGER

BEGIN

SÆTNINGER

END;

PARAMETRE, DER INDGÅR I PROCEDUREERKLÆRINGEN KALDES  
<sup>Formal</sup>  
FORMELLE PARAMETRE.

DE FORMELLE PARAMETRES TYPE ANGIVES I PROCEDUREHOVEDET.

EN PROCEDURE KALDES (AKTIVERES) VED AT ANGIVE PROCEDURE-  
NAVNET SAMT DE AKTUELLE PARAMETRE (VÆRDIER), DER SKAL  
OVERFØRES TIL PROCEDUREN.

VED AKTIVERING AF EN PROCEDURE, ERSTATTES DE FORMELLE  
PARAMETRE MED AKTUELLE PARAMETRE.

VED AKTIVERING AF EN PROCEDURE, SKAL DER VÆREN EN EEN TIL  
EEN OVERENSSTEMMELSE MELLEM FORMELLE OG AKTUELLE PARAMET-  
RE. D.V.S. AT PARAMETRENE ANTAL, RÆKKEFØLGE OG TYPE SKAL  
VÆRE DEN SAMME I PROCEDURENS PARAMETERLISTE OG I DEN LISTE  
DER BENYTTES VED AKTIVERING AF PROCEDUREN.

DE FORMELLE PARAMETRE VIRKER SCOPE-MÆSSIGT SOM OM DE ER  
LOKALE VARIABLE I PROCEDUREN.

```
PROGRAM XYZ;  
  
  VAR X, Y, Z : REAL;  
  
  PROCEDURE SKRIV (P:REAL);  
  BEGIN  
    P := P * P;  
    WRITELN (P);  
  END;  
  BEGIN  
    SKRIV (X);  
    SKRIV (Y);  
    SKRIV (Z)  
  END.
```

PROGRAM XYZ

X

Y

Z

PROCEDURE SKRIV (P: REAL)

P  LOKAL VARIABEL

P := P \* P  
WRITELN (P)

SKRIV (X)  
SKRIV (Y)  
SKRIV (Z)

DE FORMELLE PARAMETRE ER OPDELT I TO TYPER:

VALUE-PARAMETRE:

BRUGES SOM INDPARAMETER TIL PROCEDUREN.

VAR-PARAMETRE:

BRUGES SOM UDPARAMETER FRA PROCEDUREN.  
KAN OGSÅ BRUGES SOM INDPARAMETER.

PROGRAM XYZ

X

Y

Z

PROCEDURE OPLOEFT (VAR P:INTEGER);

P:=P\*P;

END;

OPLOEFT (X);

OPLOEFT (Y);

OPLOEFT (Z);

```
PROGRAM STOERSTE;
```

```
VAR TAL_1, TAL_2: INTEGER;  
    RESULTAT: INTEGER;
```

```
PROCEDURE MAX (A, B: INTEGER; VAR RES: INTEGER);  
    BEGIN  
        IF A > B THEN RES := A ELSE RES := B;  
    END;
```

```
BEGIN (*STOERSTE*)  
    READLN (A, B);  
    MAX (A, B, RESULTAT);  
    WRITELN (RESULTAT);  
END. (*STOERSTE*)
```

DEFINITION: VALUE-PARAMETER: VED KALD KOPIERES DEN AKTUELLE  
PARAMETER OVER I DEN FORMELLE.

VAR-PARAMETER : AKTUEL OG FORMEL PARAMETER ER LIG  
MED SAMME LAGERPLADS.

EKSEMPLER: PROGRAM DEMO;

VAR

A,X: INTEGER;

PROCEDURE ADD5 (X:INTEGER;VAR Y: INTEGER);

BEGIN (\*2\*)

X:= X+5;

Y:= Y+5;

END; (\*3\*)

BEGIN

A:= 5;

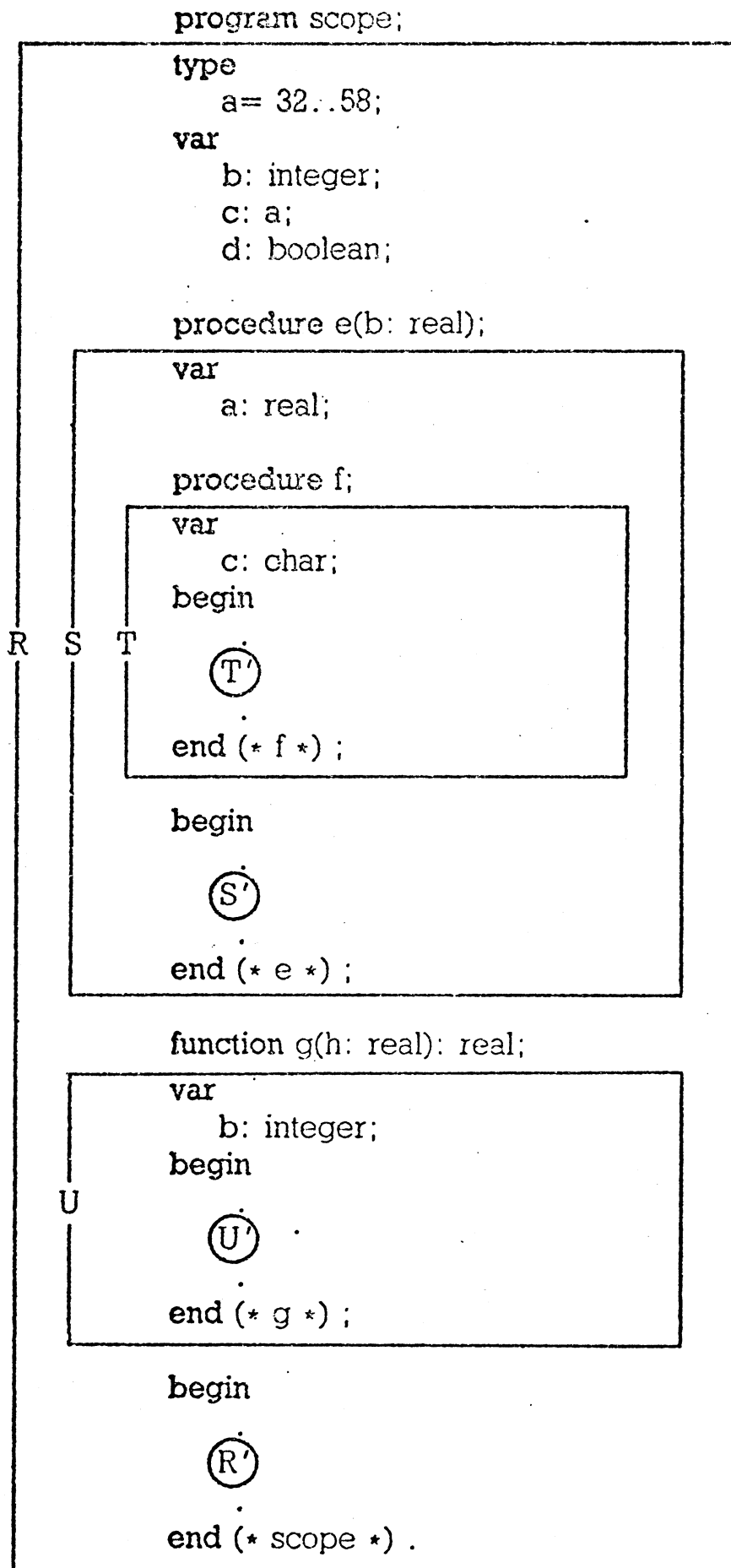
X:= 10; (\*1\*)

ADD5(A,X)

END. (\*4\*)

VED DE ANGIVNE PUNKTER INDEHOLDER LAGERET  
FØLGENDE:

(*1*)	(*2*)	(*3*)	(*4*)
A: 5	A: 5	A: 5	A: 5
X: 10	X,Y: 10	X,Y: 15	X: 15
	X': 5	X': 10	



## FUNKTIONER

DEFINITION: FUNCTION FUNKTIONSNAVN (FORMELLE PARAMETRE: TYPE; ...;  
FORMELLE PARAMETRE: TYPE): RESULTATTYPE;

DEFINITIONER OG ERKLÆRINGER

BEGIN

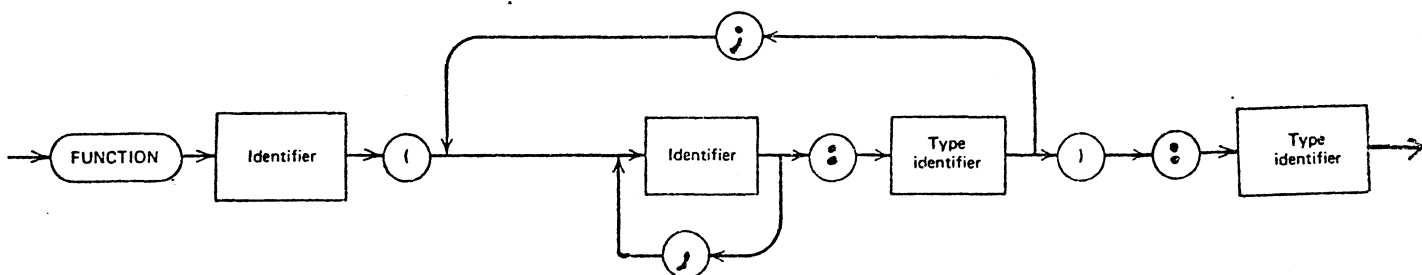
SÆTNINGER;

FUNKTIONSNAVN := VÆRDI

END;

HVOR

RESULTATTYPE = SKALAR DATATYPE.



EKSEMPLER: FUNCTION MAX (X,Y: REAL):REAL;

BEGIN

IF X>Y THEN

MAX := X

ELSE

MAX := Y

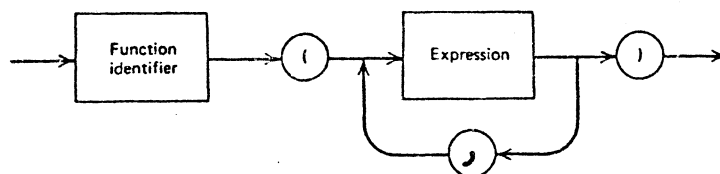
END;

## FUNKTIONER

### KALD AF FUNKTIONER

---

DEFINITION: FUNKTIONSNAMN (AKTUEL PARAMETER, ..., AKTUEL  
PARAMETER);



EKSEMPLER:

```
...  
VAR  
  STOERST, TAL1, TAL2 : REAL;  
...  
BEGIN  
  ...  
  STOERST := MAX (TAL1, TAL2);  
  ...  
END.
```

BEMÆRK:

MAN MÅ IKKE BENYTTE ET ELEMENT FRA EN PAKKET STRUKTUR  
SOM AKTUEL PARAMETER. ELEMENTET MÅ KOPIERES OVER I  
EN VARIABEL AF SAMME TYPE SOM ELEMENTET SELV.

1985.08.22

NH/kc/7/5

## STANDARDFUNKTIONER I PASCAL

---

MATEMATISKE FUNKTIONER

---

PASCAL	FUNKTION	ARGUMENT-TYPE	RESULTAT-TYPE
ABS	ABSOLUTTE VÆRDI	REAL/INTEGER	REAL/INTEGER
ARCTAN	ARCUS-TANGENS	REAL/INTEGER	REAL
COS	COSINUS	REAL/INTEGER	REAL
EXP	EXPONENTIALFUNKT.	REAL/INTEGER	REAL
LN	NATURLIGE LOGARITME	REAL/INTEGER	REAL
SIN	SINUS	REAL/INTEGER	REAL
SQR	KVADRATET	REAL/INTEGER	REAL/INTEGER
SQRT	KVADRATROD	REAL/INTEGER	REAL

EKSEMPLER:

ABS(2.1) = 2.1

ABS(-2) = 2

SQR(3.3) = 9.9

SQR(-4) = 16

BEMÆRK:

ARGUMENTET TIL FUNKTIONERNE KAN VÆRE BÅDE REAL OG INTEGER.

## TYPEKONVERTERINGSFUNKTIONER

### DEFINITION:

CHR(I) : GIVER KARAKTEREN, DER HAR POSITION I DEN ORDNEDE MÆNGDE AF KARAKTERER.

ORD(C) : KARAKTEREN C'S POSITION (ORDINALNUMMER) I DEN ORDNEDE MÆNGDE AF KARAKTERER.

PASCAL	ARGUMENT-TYPE	RESULTAT-TYPE
CHR	INTEGER	CHAR
ORD	CHAR	INTEGER
ROUND	REAL	INTEGER
TRUNC	REAL	INTEGER

### EKSEMPLER:

TRUNC(2.5) = 2  
 TRUNC(-3.8) = -3  
 ROUND(2.5) = 3  
 ROUND(-3.8) = -4  
 ORD('A') = 65  
 CHR(65) = 'A'

DECIMALDEL SKÆRES BORT

AFRUNDING TIL NÆRMESTE HELTAL

SE ASCII-TABEL SIDE

---

## ANDRE FUNKTIONER

### DEFINITION:

PRED('c') : FORGÆNGEREN TIL KARAKTEREN C.  
SUCC('c') : EFTERFØLGEREN TIL KARAKTEREN C.  
ODD(I) : FUNKTIONEN HAR VÆRDIEN TRUE, HVIS I ER  
ULIGE - ELLERS FALSE

### EKSEMPEL:

PRED('D') = 'C'  
SUCC('C') = 'D'  
ODD(17) = TRUE

### BEMÆRK:

SUCC OG PRED KAN BRUGES PÅ ALLE SIMPLE TYPER, UNDTAGEN  
REAL.

## ARRAYSTRUKTURER

### Indledning

For at kunne håndtere store datamængder, er det ofte hensigtsmæssigt at lagre disse data i en arraystruktur. En arraystruktur er et eksempel på en datastruktur, d.v.s. man har grupperet en mængde af data og ordnet dem efter et bestemt mønster. En datastruktur, som er opbygget af simple datatyper, kaldes for en struktureret datatype.

En arraystruktur består af et antal dataelementer, der alle er af samme type. Man refererer et element ved at opgive et index, der angiver, hvor i arraystrukturen elementet findes.

Alternative betegnelser for en arraystruktur er vektor (endimensional arraystruktur), matrix (todimensional arraystruktur), tabel, række eller simpelt hen array.

En endimensional arraystruktur kan grafisk illustreres som i fig. 1. Denne arraystruktur, som vi kan kalde PRIME indeholder de første 10 primtal. Hvert primtal er et heltal, d.v.s. af typen integer.

dataelement	2	3	5	7	11	13	17	19	23	29
index	1	2	3	4	5	6	7	8	9	10

Fig. 1. Grafisk illustration af en arraystruktur

---

ENDIMENSIONALE ARRAYSTRUKTURER

---

Erklæring:

Erklæringen består af en typebeskrivelse, der fortæller hvordan datastrukturen ser ud, samt en erklæring af en variabel af denne type.

TYPE

```
TYPENAVN = ARRAY [STARTINDEX..SLUTINDEX]
                      OF DATATYPE
```

VAR

```
VARIABELNAVN : TYPENAVN
```

EKSEMPEL:

CONST

```
MAX = 10;
```

TYPE

```
PRIMETYPE = ARRAY [1..MAX] OF INTEGER;
```

VAR

```
PRIME, PRIME1:PRIMETYPE;
```

```
I:INTEGER
```

---

ENDIMENSIONALE ARRAYSTRUKTURER

---

KOPIERING AF ARRAY:

Værdierne af samtlige elementer i en arraystruktur kan kopieres over i en arraystruktur af samme type ved blot én værditilskrivning.

## EKSEMPLER

PRIME1 :=PRIME; kopiering af en hel arraystruktur.

ELEMENTREFERENCE:

Man refererer et element i en arraystruktur med konstruktionen:

arraynavn    index

hvor index kan være et vilkårligt aritmetisk udtryk, hvis værdi er et heltal eller en anden skala værdi, der ligger indenfor de specificerede indexgrænser startindex og slutindex.

EKSEMPLER jfr. fig. 1.

prime [8]            indeholder 19  
prime [20-i]        indeholder 11, såfremt i har værdien 15

I:=PRIME [8]; I tildeles værdien 19  
PRIME [10]:=PRIME [7]; PRIME 10 tildeles værdien 17  
PRIME1 [1]:=PRIME [10];

---

## ENDIMENSIONALE ARRAYSTRUKTURER

---

### IND- OG UDLÆSNING TIL ARRAYS:

Man kan ikke ind/udlæse en hel arraystruktur på én gang, men må ind/udlæse et element ad gangen.

### EKSEMPEL:

BEGIN

```
(* indlæsning af en arraystruktur *)  
for i:=1 to 5 do read(input, PRIME [i] );
```

```
(* kopiering af en hel arraystruktur *)  
PRIME1:=PRIME;
```

```
(* udskrivning af en hel arraystruktur *)  
for i:=1 to 5 do write(output, PRIME1 [i] );
```

FLERDIMENSIONALE ARAYSTRUKTURER

---

Fig. 2 viser en todimensional arraystruktur.

$a_{11}$	$a_{12}$	$a_{13}$	$a_{14}$
$a_{21}$	$a_{22}$	$a_{23}$	$a_{24}$
$a_{31}$	$a_{32}$	$a_{33}$	$a_{34}$
$a_{41}$	$a_{42}$	$a_{43}$	$a_{44}$
$a_{51}$	$a_{52}$	$a_{53}$	$a_{54}$

Fig. 2. Todimensional arraystruktur med navnet a.

EKSEMPEL:

TYPE

TODIM = array [ 1..5, 1..4 ] OF integer;

var

a, b: TODIM

I:integer

---

## KOPIERING AF ARRAY - TO DIMENSIONALE

Kopiering af hele arrays kræver at arraystrukturerne er af samme type

`a:=b;` kopiering af en hel arraystruktur

### ELEMENTREFERENCE:

Man refererer et element i en todimensional arraystruktur ved

`arraynavn [ index1, index2 ]`

### EKSEMPEL:

reference til element `a43` i fig. 2.

```
I:=a [4,3];  
b [1,2] :=a [5,1];
```

En todimensional arraystruktur består af rækker og søjler. I foranstående eksempel har vi 5 rækker og 4 søjler.

Vi kan i Pascal have et vilkårligt antal dimensioner på en arraystruktur.

---

ARRAYS - RESUME

---

DEFINITION:

TYPE

TYPENAVN = ARRAY [ INDEX TYPE, ..., INDEX TYPE ]  
OF TYPE;

ELEMENTREFERENCE:

ARRAYNAVN [ INDEX, ..., INDEX ]

EKSEMPLER:

TYPE

TAB = ARRAY [ 1..5, 1..10 ] OF INTEGER;

VAR

TABEL1, TABEL2: TAB;

BEGIN

TABEL1 [ 1, 7 ] := 17; tildeling til element

TABEL1:=TABEL2; kopi af hele array

END.

BEMÆRK:

IKKE DIREKTE IND/UDLÆSNING AF ET HELT ARRAY - SKAL SKE  
ELEMENTVIS.

ARRAYS PAKKEDE (STRENG)

---

DEFINITION:

TYPE

TYPENAVN = PACKED ARRAY [1..INDEX] OF CHAR;

HVOR

INDEX = POSITIVT HELTAL

EKSEMPLER:

TYPE

STRENG = PACKED ARRAY [1..80] OF CHAR;

NAVN = PACKED ARRAY [1..15] OF CHAR;

VAR

S : STRENG;

N : NAVN;

BEGIN

N := 'HANS HANSEN';

END.

---

INDLÆSNING TIL PAKKEDE ARRAYS

---

I standard Pascal er indlæsning af strenge ikke defineret, men strenge kan naturligvis indlæses tegnvis.

Givet erklæringerne

```
var
  fornavn: packed array [1..6] of char;
  efternavn: packed array [1..10] of char;
  ch:char;
```

Så kan man indlæse tegn til strengene med

```
for i: = 1 to 6
do begin read (input, ch); fornavn[i] := ch end;
for i: = 1 to 10
do begin read (input, ch); efternavn[i] := ch end;
```

---

STRING

Findes ikke i Standardpascal, men findes i de fleste Pascalversioner.

EKSEMPEL;

TYPE

STR20 = STRING [20] ;

VAR

FORNAVN, EFTERNAVN: STR20;

BEGIN

FORNAVN: = 'HENRIK';

WRITELN (FORNAVN);

WRITELN ('INDTAST EFTERNAVN');

READ (EFTERNAVN);

END.

BEMÆRK:

Det er ikke nødvendigt at udfylde hele stingen;

DATASTRUKTURER  
RECORD (POST)

---

- DEFINITION:
- SAMLING AF ELEMENTER (FELTER) SOM IKKE BEHØVER AT HAVE SAMME TYPE
  - ENDELIGT ANTAL ELEMENTER (FELTER)
  - ELEMENTER (FELTER) REFERERES VED ET FELTNAVN.

EKSEMPEL:

KURSIST:

FORNAVN	EFTERNAVN	ADRESSE		
ANNETTE	NIELSEN	VEJ	NR.	BY
		SKRIVEVEJ	17	NYBORG

KURSIST, EFTERNAVN=NIELSEN

↑                      ↑

RECORD NAVN    FELT NAVN

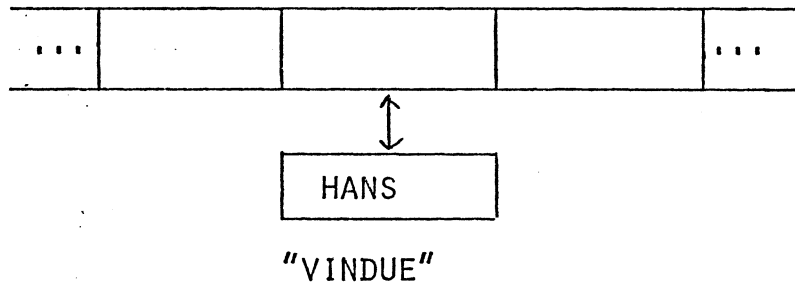
## DATASTRUKTURER

### FILE

---

- DEFINITION:
- SAMLING AF ELEMENTER AF SAMME TYPE.
  - VILKÅRLIGT ANTAL ELEMENTER.
  - ELEMENTER KAN KUN UNDERSØGES ET AF GANGEN GENNEM ET "VINDUE"
  - ADGANG TIL ET BESTEMT ELEMENT KRÆVER SEKVENTIEL UNDERSØGELSE AF ELEMENTER FRA BEGYNDELSEN AF FILEN.
  - KUN SEKVENTIEL TILFØJELSE AF ELEMENTER I SLUTNINGEN.
  - LIGGER OFTE PÅ BAGGRUNDSLAGER.

EKSEMPEL: KURSISTER:



## EKSTRA ØVELSE 1.

### ERKLÆRINGER, VÆRDITILSKRIVNING OG SIMPEL ARITMETIK

---

SKRIV ET PROGRAM DER INDLÆSER TO HELLAL A OG B FRA TASTATURET OG BEREGNER VÆRDIEN AF UDTRYKKENE:

$$A + B$$

$$A - B$$

$$A * B$$

$$A / B$$

UDSKRIFTEN FRA PROGRAMMET KUNNE SE UD SOM NEDENSTÅENDE (INDDATA ER UNDERSTREGET).

INDTAST 2 TAL:  $A = \underline{2}$   
 $B = \underline{5}$

RESULTATER:  $A + B = 7$   
 $A - B = -3$   
 $A * B = 10$   
 $A / B = 0,40000$

## EKSTRA ØVELSE 2.

### ERKLÆRINGER, VÆRDITILSKRIVNING OG SIMPEL ARITMETIK

---

- A) FØLGENDE PROGRAM SKAL UDSKRIVE EN SIMPLIFICERET LØN-  
SEDDER FOR EN MEDARBEJDER I ET FIRMA. PROGRAMMET IND-  
LÆSER LØN, FRADrag SAMT TRÆKPROCENT FRA TASTATURET  
OG BEREgNER

KILDESKAT:= (LØN - FRADrag - ATP)\* TRÆKPROCENT/100;  
DISPONIBEL LØN:=LØN - KILDESKAT - ATP;

DER REGNES MED ET KONSTANT ATP-BIDrag PÅ KR. 14,00 PR.  
MÅNED.

UDSKRIFTEN FRA PROGRAMMET KUNNE SE UD SOM NEDESTÅENDE  
(INDDATA ER UNDERSTREGET)

INDTAST LØN : 9514.50  
FRADrag : 3500.50  
TRÆKPROCENT : 50

LØNSEDDER FOR HANS HANSEN

LØN : 9514.50  
ATP : 14.00  
KILDESKAT : 3.000000E3  
DISPONIBEL LØN: 6.50050E3

- B) ATP-BIDragET ER VOKSET TIL KR. 16,00 PR. MÅNED.  
KORRIGER PROGRAMMET FRA SPØRGSMÅL A I OVERENSSTEMMELSE  
MED DETTE.

### EKSTRA ØVELSE 3.

#### KONTROLSTRUKTUREN GENTAGELSE

---

LAV ET PROGRAM, DER LÆGGER TAL SAMMEN EFTER FØLGENDE FORMEL:

$$\text{SUM} = 0+1+2+3+4+5+6+7+\dots+N$$

DETTE KAN GØRES VED INDE I EN SLØJFE AT HAVE FØLGENDE OPERATIONER:

```
NR:= NR+1;  
SUM:= SUM+NR;
```

OPGAVEFORMULERING:

- A: LAV ET PROGRAM (INDEHOLDENDE) EN SLØJFE, DER LÆGGER TAL SAMMEN EFTER OVENSTÅENDE FORSKRIFT, PROGRAMMET SKAL UDSKRIVE DET TAL (NR), DER FÅR SUMMEN TIL AT VÆRE LIG MED ELLER OVERSTIGE 1000.
- B: LAV EN "LILLE" ÆNDRING I PROGRAMMET, SÅLEDES AT DEN VÆRDI, DER SKAL TERMINERE PROGRAMMET NU INDLÆSES FRA TERMINALENS TASTATUR, PROGRAMMET SKAL TERMINERE, NÅR SUM ER LIG MED ELLER OVERSTIGER DET INDLÆSTE TAL.

## EKSTRA ØVELSE 4.

### ARRAYS OG UNDERPROGRAMMER

---

- I. LAV ET PROGRAM, DER KAN INDLÆSE 10 TAL I ET  
ARRAY.  
FIND HEREFTER DET STØRSTE TAL I ARRAYET OG UDLÆS  
DET PÅ SKÆRMEN.
  
- II. LAV "FIND STØRSTE TAL"-DELEN SOM ET UNDERPROGRAM.

Eva Hansen  
VME/10 5.PASCAL  
KOGEBOG.SA

```
*****  
*                                     *  
*           K O G E B O G           *  
*               I                   *  
*   S Y S T E M A R B E J D E       *  
*               O G                 *  
*   P R O G R A M M E R I N G       *  
*                                     *  
*****
```

# I N D H O L D

---

	side
1. SYSTEMARBEJDET .....	3
1.1. Systemarbejdets faser .....	3
1.2. Systemarbejdets forløb .....	4
1.2.1. Traditionelt systemarbejde .....	5
1.2.2. Prototype udvikling .....	5
2. SYSTEMBESKRIVELSE .....	6
2.1. Systembeskrivelsens formål og anvendelse .....	6
2.2. Systembeskrivelses værktøjer .....	6
2.2.1. Blanketoversigt .....	7
2.2.2. Blanketindhold .....	8
2.2.3. Datasammenhænge .....	9
2.2.4. Dataliste .....	10
2.2.5. Maskinkonfiguration .....	11
2.2.6. Skærbillede layout .....	12
2.2.7. Layout for udskrifter .....	15
2.2.8. Moduldiagram, pseudokode og rutediagram .....	16
3. PROGRAMMERING .....	22
3.1. Hvornår kan programmeringen starte .....	22
3.2. Programmeringsgrundlaget .....	23
4. PROGRAM LAYOUT .....	24
4.1. Hvordan er et godt program opbygget .....	24
4.2. Typiske delprogrammer/moduler .....	25
4.3. Typiske procedurer/funktioner .....	25
4.4. Programmets læsbarhed .....	26
5. FEJLBEHANDLING .....	27
6. AFPRØVNING AF PROGRAMMET .....	28
7. LITTERATUR .....	29

## 1. SYSTEMARBEJDET

Systemarbejdet omfatter

- indførelse af nye systemer (f.eks. et bogholderisystem).
- ændringer i eksisterende systemer.

### 1.1. Systemarbejdets faser.

Problemformulering (konkretisering af opgaven), herunder beskrivelse af det eksisterende system:

1. Foranalyse og udarbejdelse af tilbud.
2. Problemstudier.
3. Kravspecifikation.

Beskrivelse af det nye/ændrede system:

4. Detailstudie og analyse.
5. Design og programmering.

Indføring af det nye system:

6. Implementering.
7. Test og afprøvning.

Vurdering, revision og opfølgning af systemet:

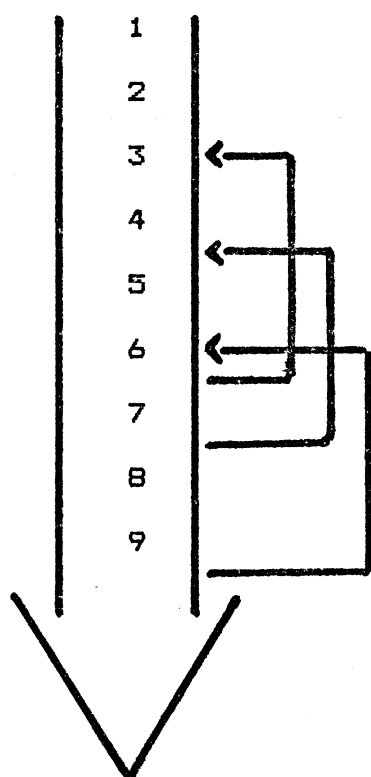
8. Brugeraccept.
9. Drift og vedligeholdelse.

## 1.2. Systemarbejdets forløb.

Systemarbejdets faser udføres i en vedtaget rækkefølge. Her nævnes to mulige måder.

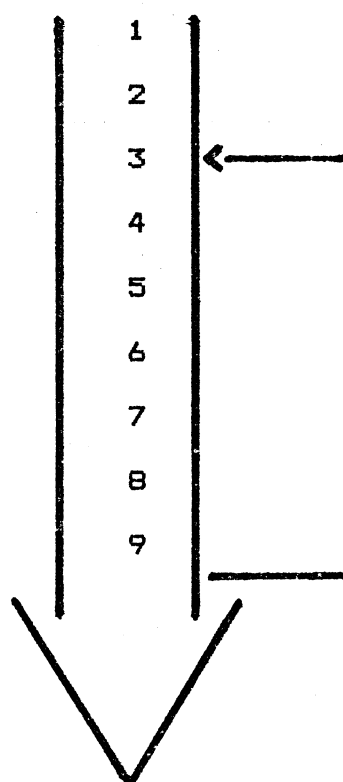
- traditionelt lineært forløb
- prototype udvikling

Den traditionelle lineære, iterative strategi.



Mange iterationer giver et uoverskueligt projekt.

Prototype strategien.



Et nyt forsøg for hver prototype.

Den lineære strategi tillader ikke flere gennemløb af processen. Hvis man ikke får alle detaljer med i første forsøg, bliver man nødt til at foretage mange, mindre iterationer - og det giver et uoverskueligt projekt/system.

Prototype strategien giver systemarbejderen flere chancer for at forbedre det første forsøg.

### 1.2.1. Traditionelt systemarbejde.

Anvendes når

1. Brugeren præcis ved, hvad han ønsker.
2. Designet har afgørende indflydelse på systemets kvalitet.
3. Der er ikke krav om få en tidlig demonstration af systemet.
4. Tidsplanerne er løse, der er god tid.
5. Der skal skaffes et hurtigt overblik over det endelige system.  
(F.eks. af økonomiske grunde).

De enkelte faser udføres en ad gangen i den viste rækkefølge.

Arbejdets enkelte faser er adskilt fra hinanden.

Det er vanskeligt at hoppe tilbage i processen og foretage ændringer.

### 1.2.2. Prototype udvikling.

Anvendes når

1. Brugeren ikke er fuldstændig klar over, hvad han vil have.
2. Hovedtrækkene i systemet fremgår tydeligt, men detalillerne er uklare.
3. Brugeren vil se resultater tidligt (demonstration af systemet).
4. Tidsplanen er stram.
5. Økonomien tillader at det endelige system ikke er fastlagt.  
på forhånd.

Prototypeudvikling er en måde at udvikle systemer på i samarbejde med brugeren.

Metoden går ud på

- at beskrive systemet i hovedtræk, en grovskitse.
- at programmere grovskitsen
- at demonstrere grovskitse-systemet
- at ændre grovskitsen og programmet indtil brugeren er tilfreds
  
- at beskrive systemet lidt mere i detaljer
- at programmere detalillerne
- at demonstrere programmet og ændre indtil brugeren er tilfreds.
  
- at beskrive flere detaljer
- osv.

Der laves altså fra starten af et system-skelet, der kan demonstreres. Man kan herefter lave forskellige varianter af systemet, indtil man finder en passende løsning.

Derefter udbygges systemet, så flere og flere detaljer kommer med.

Fordelen ved prototype-udviklingen er, at systemet fra starten af laves, så det er let at foretage konstruktions-ændringer i systemet/programmet.

Normalt har man på forhånd lagt sig fast på et design, og det er derfor meget svært at lave større ændringer i systemet/programmet.

## 2. SYSTEMBESKRIVELSEN

### 2.1. Systembeskrivelsens formål og anvendelse.

1. Skabe klart og entydigt overblik over den problemstilling, der indgår i systemet.
2. Anvende beskrivelsen til kommunikation mellem de personer, der konstruerer, ændrer og bruger systemet.
3. Hjælpemiddel ved konstruktion og vedligeholdelse af systemet.
4. Grundlag for algoritmebeskrivelse til brug ved programmeringen.
5. Beslutningsgrundlag ved indførelse/ændring af systemet.

### 2.2. Beskrivelses værktøjer.

Systembeskrivelsen skal skrives i et letforståeligt sprog. Den skal være præcis og overskuelig. Beskrivelsen deles op i fornuftige kapitler, der igen er delt op i mindre afsnit med gode overskrifter, så den kan anvendes som et opslagsværk.

Systembeskrivelsen kan suppleres med forskellige former for illustrationer. Figurer og oversigter letter forståelsen.

Der kan f.eks. nævnes følgende systembeskrivelses-værktøjer

- |                              |        |
|------------------------------|--------|
| - blanketoversigt            | se (1) |
| - blanketindhold             | se (1) |
| - datasammenhæng             | se (1) |
| - dataliste                  | se (1) |
| <br>                         |        |
| - maskinkonfiguration        | se (1) |
| - moduldiagram (JSP-diagram) | se (3) |
| - rutediagram                | se (1) |
| - pseudokode                 | se (4) |
| <br>                         |        |
| - layout for skærbilleder    | se (1) |
| - layout for udskrifter      | se (1) |

NR.	INDDATA	REGISTERDATA	UDDATA

[illegible]



## DATALISTE

Udfyldt af		Udfyldt den	Opgavenr	Bilagnr	Sidenr
------------	--	-------------	----------	---------	--------

Nr.	Datanavn				Datatype
	Forkortelse				
	Definition				
					Værdisæt

Nr.	Datanavn				Datatype
	Forkortelse				
	Definition				
					Værdisæt

Nr.	Datanavn				Datatype
	Forkortelse				
	Definition				
					Værdisæt

Nr.	Datanavn				Datatype
	Forkortelse				
	Definition				
					Værdisæt

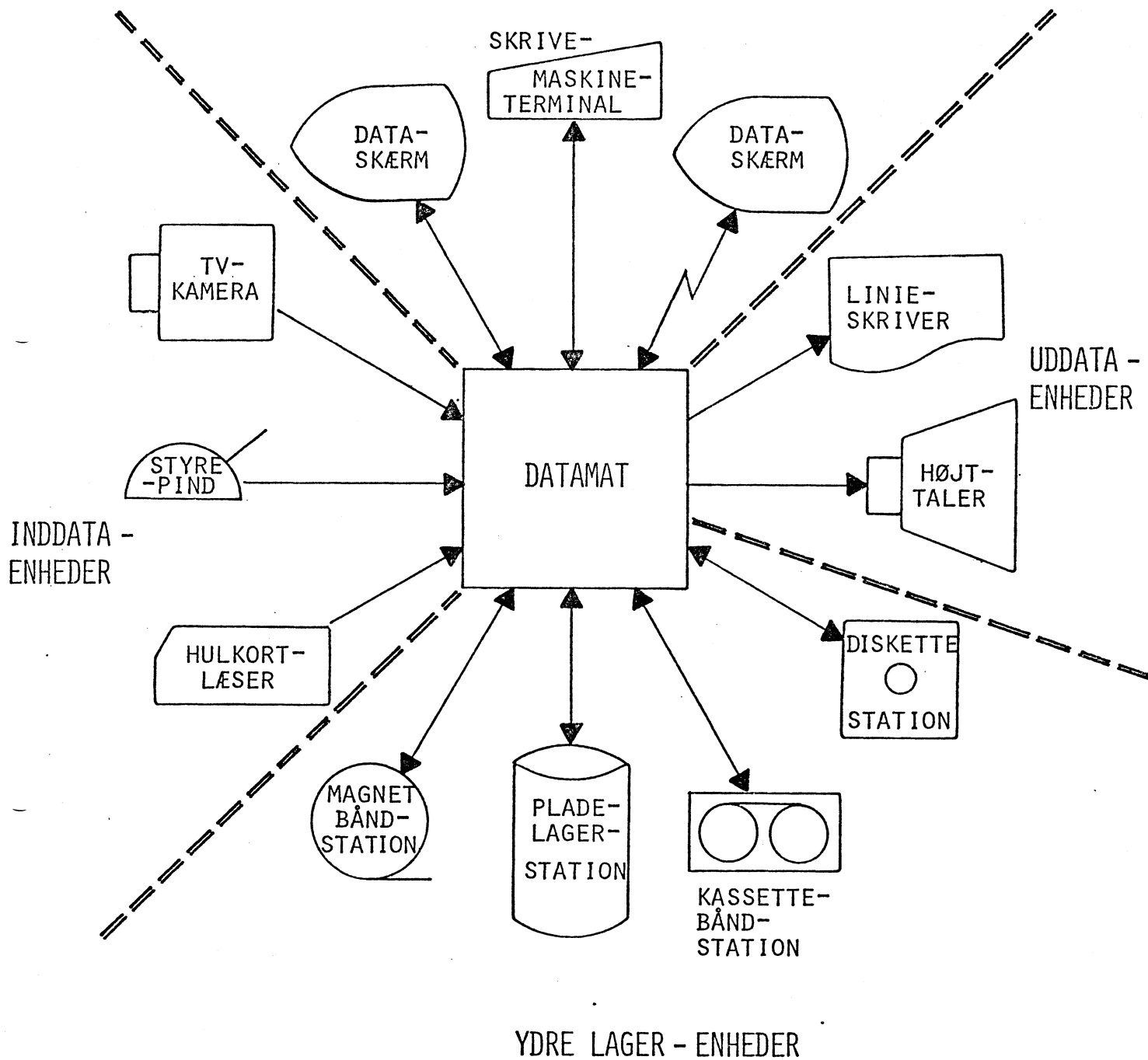
Nr.	Datanavn				Datatype
	Forkortelse				
	Definition				
					Værdisæt

Nr.	Datanavn				Datatype
	Forkortelse				
	Definition				
					Værdisæt

MASKINKONFIGURATION

KOMBINEREDE IND / UD - DATA - ENHEDER



Øvelse.

— Skitser en maskinkonfiguration, der består af en minidatamat, 1 operator-konsol, 1 skærmterminal, 1 grafisk farveskærm, 1 matrix-skriver og 1 skrivemaskineterminal, der er tilsluttet over telefonnettet via modem.

Øvelse.

Der skal tilsluttes nogle ydre lagerenheder til systemet ovenfor. Vælg ydre lagerenhed (-er) og begrund valget.

### 2.2.6. Skærbillede layout.

#### Eksempel.

Opdeling af skærbilledet i faste vinduer.

	0	10	20	30	40	50	60
0-	<.. firmanavn .....>			<. overskrift .....>		<. dato .>	
	<.. systemnavn .....>			<. deloverskrift .....>		<. kl .>	
5-							
10-	D A T A						
15-							
20-							
	<.. linje til programmets ledetekster og anvisninger .....>						
	<.. linje til kommandoer og inddata .....>						
	<.. linje til fejlmeddelelse .....>						

Eksempel.  
Menu-billede.

```

      0      10      20      30      40      50      60
      |      |      |      |      |      |      |
+-----+-----+-----+-----+-----+-----+
0-| <.. firmanavn .....>  <. overskrift .....>  <. dato .>|
  | <.. systemnavn .....>  <. deloverskrift .....>  <. kl .>|
  |-----|-----|-----|-----|-----|-----|
5-|                                     M E N U
  |                                     -----
  |
  |      1.aaaaaaaaaaaaaa  (... forklaring .....)|
  |      2.aaaaaaaaaaaaaa  (... forklaring .....)|
10-|      3.aaaaaaaaaaaaaa  (... forklaring .....)|
  |      4.aaaaaaaaaaaaaa  (... forklaring .....)|
  |
  |      5.aaaaaaaaaaaaaa  (... forklaring .....)|
  |      6.aaaaaaaaaaaaaa  (... forklaring .....)|
15-|      7.aaaaaaaaaaaaaa  (... forklaring .....)|
  |      8.aaaaaaaaaaaaaa  (... forklaring .....)|
  |
  |      99. Stop.
20-|
  | <.. linje til programmets ledetekster og anvisninger .....> |
  | <.. linje til kommandoer og inddata .....> |
  | <.. linje til fejlmeddelelse .....> |
+-----+-----+-----+-----+-----+-----+

```

### 2.2.7. Layout for udskrifter.

#### Øvelse.

Lav layout for udskrift af en lønseddel.

### 2.2.8. Moduldiagram, pseudokode og rutediagram.

Et system kan beskrives ved hjælp af ord og figurer.

Det er tit nemmere at forstå en figur – end en kompliceret tekst. Man skal dog være sikker på, at alle opfatter figuren på den samme måde, ellers er den ikke noget værd.

Her nævnes 3 forskellige grafiske måder, at beskrive et system/program på.

1. Moduldiagram (= JSP diagram).
2. Pseudokode.
3. Rutediagram (= blokdiagram).

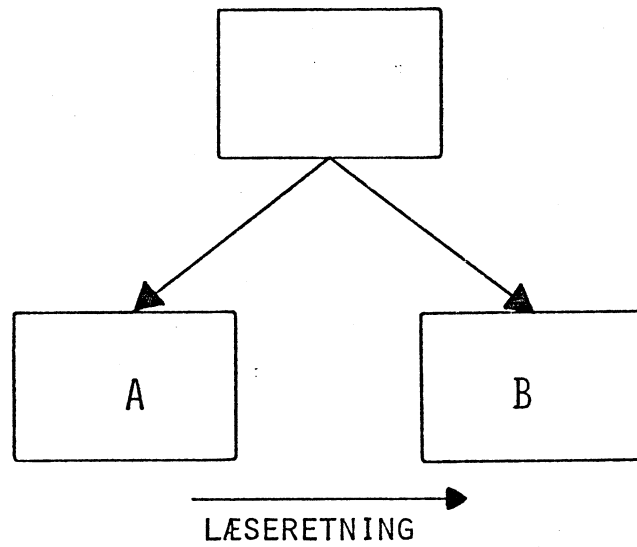
Alle programmer vil kunne opbygges af 4 basale kontrolstrukturer.

1. Sekvens.
2. Valg.
3. Repetition.
4. Hop.

De 4 kontrolstrukturer beskrives som moduldiagram, pseudokode og rutediagram i det følgende.

SEKVEN

M. A. JACKSON  
MODULDIAGRAM:

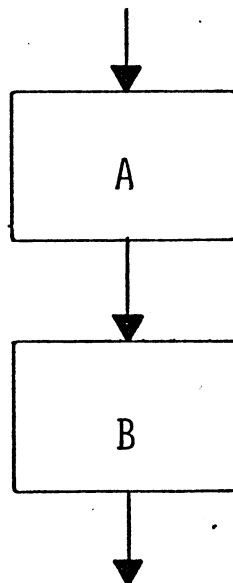


PSEUDOKODE:

UDFØR ( A )

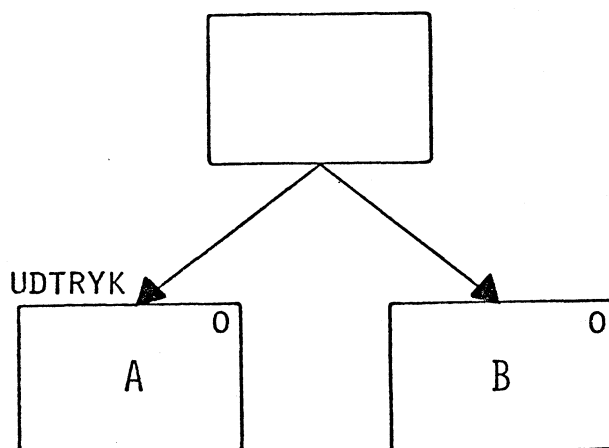
UDFØR ( B )

RUTEDIAGRAM:



VALG

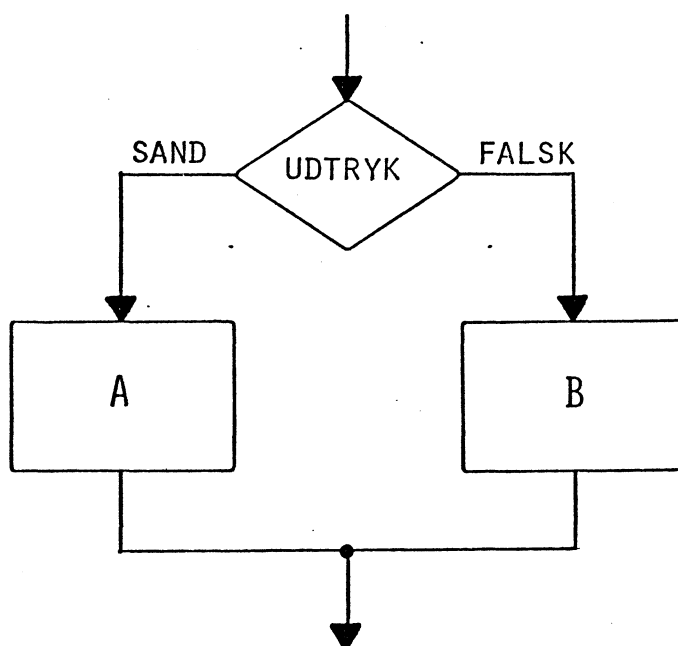
M. A. JACKSON  
MODULDIAGRAM:



PSEUDOKODE:

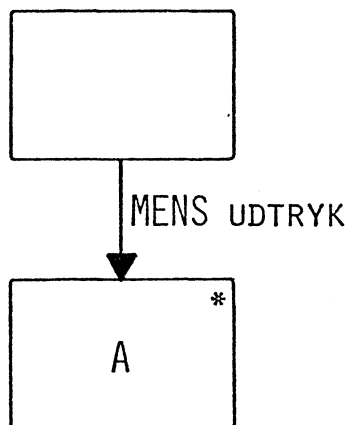
```
HVIS UDTRYK SÅ
    UDFØR ( A )
ELLERS
    UDFØR ( B )
SLUT HVIS
```

RUTEDIAGRAM:



REPETITION MENS - UDFØR

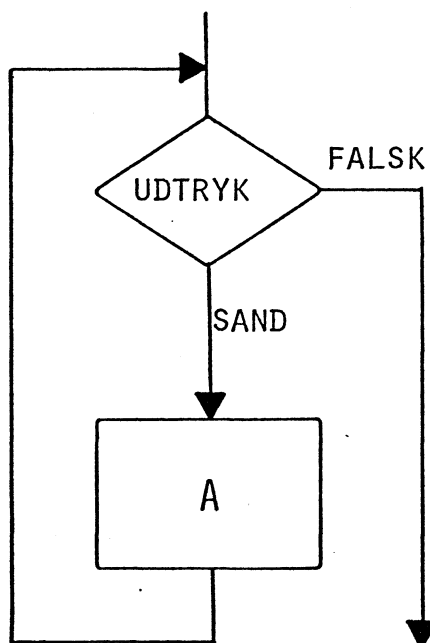
M. A. JACKSON  
MODULDIAGRAM:



PSEUDOKODE:

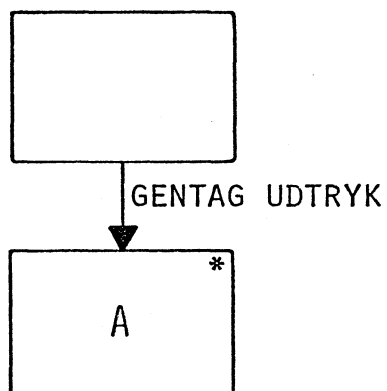
MENS UDTRYK UDFØR  
UDFØR ( A )  
SLUT MENS

RUTEDIAGRAM:



REPETITION GENTAG - INDTIL

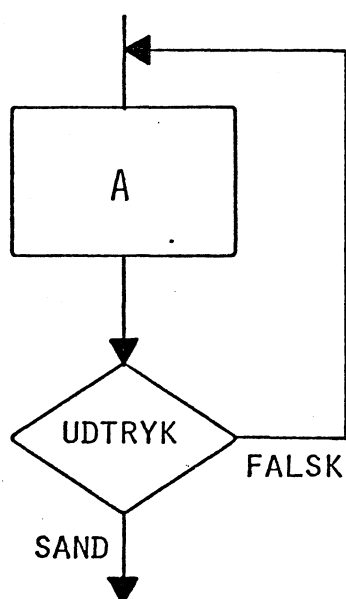
M. A. JACKSON  
MODULDIAGRAM:



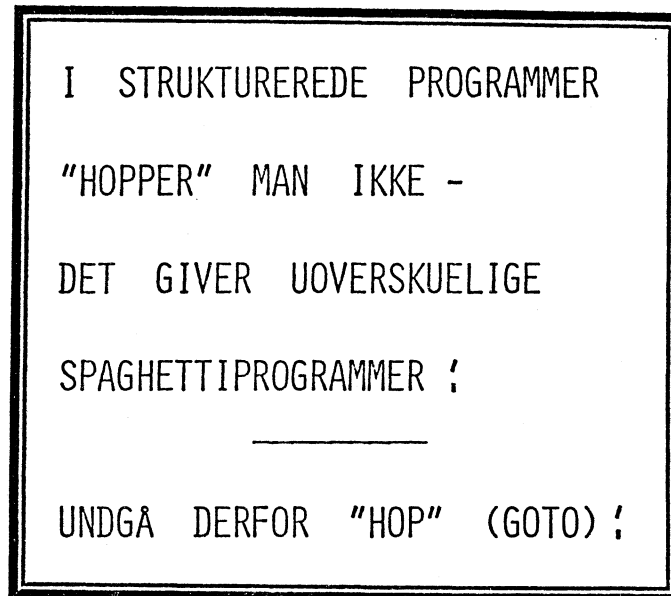
PSEUDOKODE:

GENTAG  
    UDFØR ( A )  
INDTIL UDTRYK

RUTEDIAGRAM:



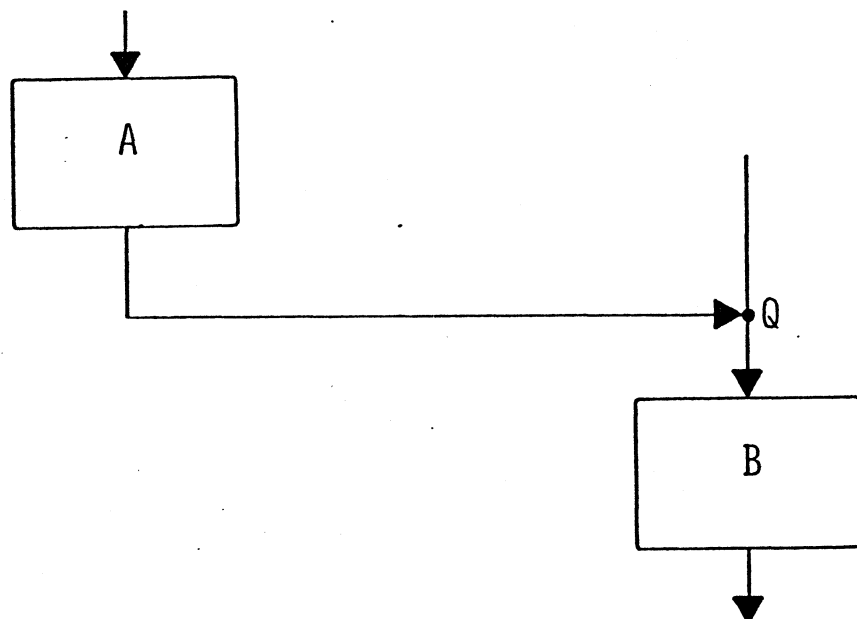
M. A. JACKSON  
MODULDIAGRAM:



PSEUDOKODE:

```
UDFØR ( A )  
HOP TIL Q  
:  
:  
:  
Q: UDFØR ( B )
```

RUTEDIAGRAM:



### 3. PROGRAMMERING

#### 3.1. Hvornår kan programmeringen starte.

Programmeringen kan starte, når arbejdsgrundlaget er i orden.

Der skal altså foreligge en godkendt beskrivelse af det ønskede system og en arbejdsplan, der angiver tidsfrister og tidsforbrug.

Programmeringsgrundlaget består af

- databeskrivelse
- beskrivelse af skærbilleder
- beskrivelse af udskrifter
- grovskitse af systemet
- detailskitse af systemet

Arbejdsplanen består af

- plan over programmerings arbejdets forløb
- rækkefølgen systemets moduler skal programmeres i
- tidsfrister og tidsforbrug for de enkelte programmeringsopgaver
- beskrivelse af afprøvningsstrategi

## 3.2. Programmeringsgrundlaget.

### 1. Databeskrivelse.

Der skal forligge en beskrivelse af

- inddata
- registerdata
- uddata

på en standardiseret form, f.eks. i form af datalister og datasammenhænge (Dansk Standard).

### 2. Skærbilleder.

Systemets skærbilleder skal være beskrevet.

Der skal være angivet et layout for skærbillederne, og systemets skærbilleder skal følge denne standard.

Ud over det generelle skærbillede layout, skal hvert enkelt skærbilledes ledetekster og ind- og uddata felter være beskrevet.

Skærbillederne er menuer, rettebilleder, oversigter og lignende.

### 3. Udskrifter.

Systemets udskrifter skal være beskrevet.

Der skal være angivet et generelt layout for udskrifterne.

Desuden skal hver enkelt udskrift være beskrevet. Dette gælder udskriftens overskrifter, ledetekster og samtlige datafelter.

Udskrifter, der kan vises på både skærm og på papir, skal helst følge samme standard.

### 4. Grovskitse af systemet, modulbeskrivelse.

Systemets opbygning afspejles i modulbeskrivelsen (moduldiagrammet).

Modulbeskrivelsen angiver direkte hvilke programmer eller moduler, der indgår i systemet, og hvorledes systemet fungerer som helhed.

### 5. Detailskitse af systemet, pseudokode.

Systemets enkelte moduler skal være beskrevet i detaille. F.eks. ved hjælp af pseudokode (eller rutediagram). Pseudokode er at foretrække.

Pseudokoden laves ud fra modulbeskrivelsen. Fordelen ved at benytte pseudokode er at

- beskrivelsen er uafhængig af programmeringssproget.
- beskrivelsen er på et højt niveau, man abstraherer fra uinteressante detaljer.
- modulstrukturen afspejles direkte i pseudokoden.
- det er let at lave velstrukturerede programmer ud fra pseudokode.

#### 4. PROGRAM LAYOUT

Et program er ikke en stor rodet bunke kodelinjer, der ved et mirakel er kommet til at virke.

Et program er resultatet af kreativ tænkning og systematisk arbejde. Kvaliteten af dette arbejde fremgår direkte af programmets udseende.

Programmer skal være velstrukturerede. Ellers får man problemer.

Rutediagram-teknik har tendens til at give spaghetti-programmer, pseudokode-teknik giver velstrukturerede programmer.

##### 4.1. Hvordan er et godt program opbygget?

Programmet (eller programkomplekset) består af et hovedprogram (hovedmodul) og evt. nogle delprogrammer (delmoduler).

Et hoved- eller delprogram er delt op i procedurer og funktioner.

Hele systemet består altså af mindre dele, der er stykket sammet til det samlede system. Herved bliver systemet mere overskueligt, og lettere at arbejde med.

##### Hovedprogram

! Programnavn.	!	Programmet skal have et passende navn.
! Beskrivelse af program.	!	En kommentar skal beskrive, hvad programmet laver, hvornår det er lavet, og hvem der har lavet det.
! Liste over konstanter og variable.	!	Konstanter og variable skal defineres, og deres brug beskrives i en kommentar. Konstanter og variable skal have "sigende" navne.
! Liste over funktioner og procedurer.	!	Procedurer og funktioner skal erklæres. De skal være overskuelige (højst 40-60 linjer). Navne på funktioner og procedurer skal være "sigende".
! Kode for hovedprogrammet.	!	Hovedprogrammet er den overordnede kode, som har ansvaret for den overordnede styring. Det underordnede sker i procedurerne og funktionerne. Hovedprogrammet har samme længde som en procedure.

Delprogrammer opbygges på samme måde som hovedprogrammet.

#### 4.2. Typiske delprogrammer/moduler.

Nogle typiske moduler kan f.eks. være:

- hovedmodul
- initialisering af systemets filer
- initialisering af systemet (ved opstart)
- opret registerpost
- ret/slet registerpost
- beregningsmodul
- udskrivningsmodul
- sikkerhedsmodul

#### 4.3. Typiske procedurer/funktioner.

Procedurer kan opdeles i forskellige typer, afhængig af deres opgave. Der kan f.eks. nævnes følgende opgave områder:

- fejlbehandling
- inddatakontrol (validering)
- særlige kontroller
- initialisering af datastrukturer (konstanter, variable, tabeller ...)
- læsning fra skærm
- skrivning til skærm
- læsning fra filer
- skrivning på filer
- læs data
- skriv data
- beregninger

Funktionerne og procedurerne defineres i det modul (delprogram), hvor de hører hjemme. De grupperes efter deres type eller opgave, så systemet bliver så overskueligt som muligt. Procedurer som er generelt anvendelige kan samles i et program-bibliotek, som programmerne kan trække på.

#### 4.4. Programmets læsbarhed.

Et program kan blive lettere at læse og forstå, hvis man tager hensyn til følgende tommelfinger-regler.

1. Indryk programteksten i nivåer.
2. Fremhæv nøgleord.
3. Brug kun 1 operation per linje.
4. Brug blanktegn som adskillelse, så teksten ikke bliver gnidret.
5. Brug blanke linjer som adskillelse, for at opdele teksten i logiske afsnit.
6. Brug forklarende og relevante kommentarer.
7. Undgå intetsigende og kryptiske kommentarer.
8. Skriv kommentarer på dansk (eller relevant menneskesprog).  
Dvs: Stort begyndelsesbogstav, almindelig tegnsætning og hele sætninger.
9. Brug sigende navne på programmer, moduler, funktioner, procedurer, variable, konstanter og labels.  
Tællevariable gives simple navne som f.eks. "i", "j" og "k".
10. Erklær alle variable og beskriv deres funktion i en kommentar.  
Undgå desuden at bruge samme variabel til flere forskellige ting.
11. Undgå "mystiske" talkonstanter, brug konstanter i stedet.
12. Struktur programmet.  
Dvs: Opdel programmet i moduler og anvend små og fornuftige procedurer.  
En procedure må fylde fra 1 - 50 linjer kode. Saml endelige de logisk sammenhængende operationer i procedurer - også selv om de kun bliver kaldt en enkelt gang.
13. Kommunikationen mellem programmets enkelte dele foregår ved udveksling af data gennem parameteroverførsel.  
Brug parameteroverførsel, hvor det er "fornuftigt".
14. Fejlmeddelelser og andre udskrifter skal være brugervenlige.
15. Forbered programmet på at det skal afprøves. Indføj en test-variabel, så test kan slås til og fra.
16. Lav fornuftige testudskrifter. Og lad være med at fjerne dem fra programmet - bare fordi du TROR at programmet virker (for det gør det ikke).

## 5. FEJLBEHANDLING.

En af programmets vigtigste funktioner er fejlbehandlingen. Inddata skal kontrolleres og fejlbehandling af data skal undgås. Endelig skal programmet tage højde for særlige systemfejl.

### Validering af inddata.

- Kontroller datatypen.
- Kontroller værdigrænser.
- Kontroller om kombinationen af inddata er konsistent (dvs. passer sammen).

### Kontrol af beregningsprocessen.

- Husk at initialisere data.
- Test for division med 0.
- Test for over- og underløb.
- Giv advarsel, når brugeren vil slette større datamængder, tillad evt brugeren at fortryde.

### Uddata kontrol.

- Kontroller at beregningsresultaterne er rimelige.

### Kontrol af adressering.

- Test på indeks-grænser ved tabeller og filer.
- Test på om ønsket fil eksisterer.
- Kontroller at ønsket post i fil findes.
- Kontroller at ønsket ydre enhed (f.eks. printer) er parat.
- Kontroller at filoperation gik godt.

## 6. AFPRØVNING AF PROGRAMMET.

Man kan ikke BEVISE, at et program er fejlfrit, -  
man kan kun påvise, at visse fejl IKKE er tilstede.

Derfor skal man afprøve programmet så omhyggeligt som muligt, for  
at sikre sig at så mange fejl som muligt er fraværende!

Sædvanligvis laves 2 forskellige afprøvninger af programmet:

### 1. Programmørens afprøvning.

Programmøren skal lave en systematisk afprøvning, for at  
sikre at programmet virker korrekt i alle tænkelige situ-  
ationer - både generelle tilfælde og specialtilfælde.

### 2. Brugerens afprøvning.

Brugeren skal afprøve programmet, for at sikre sig, at  
det virker på den ønskede måde. Denne test skal også være  
meget omhyggelig og systematisk.

Det er en KUNST at afprøve programmer, og det kræver stor disciplin  
at opstille testdata og udføre en "komplet" afprøvning.  
Ligeledes er det et problem at dokumentere afprøvningen.

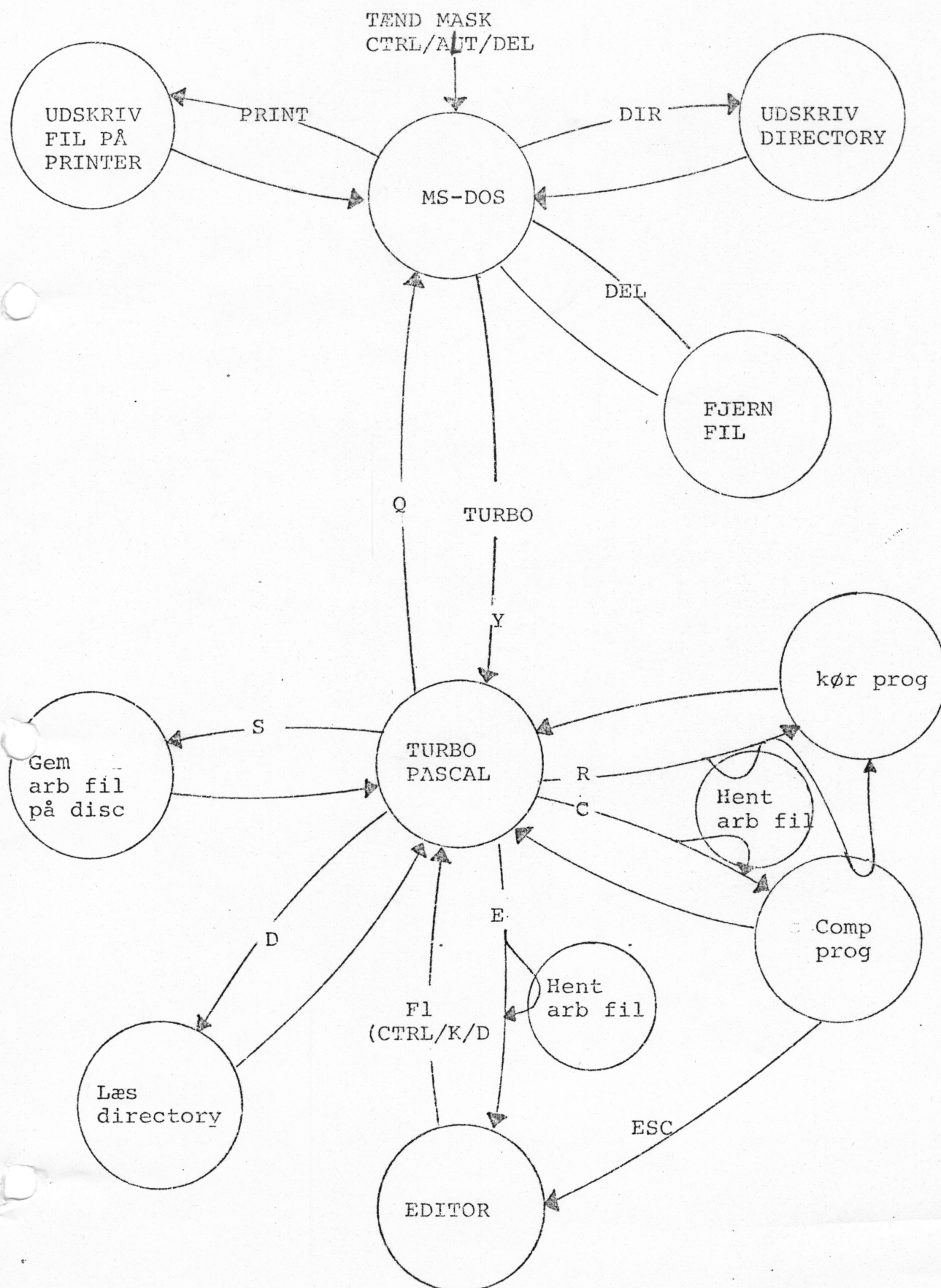
Til indføring i emnet anbefales bogen "Kunsten at teste edb-programmer"  
(6).

## 7. LITTERATUR.

1. A. Egebjerg Johansen, J. P. Jensen, H. Fabricius Olesen, F. Dybkjær: "Databehandling for EDB brugeren". Systime, 1981.
2. Tom Eriksen, Freddy Hafskjold: "Kan vi frigjøre oss fra EDB-eksperten?". Nordisk Datanytt, nr. 2, 14. febr. 1983.
3. Bo Sandén: "Systemprogrammering med JSP". Studentlitteratur, Lund 1983.
4. Aimo Törn: "Programmering - från problem till dokumentation". Studentlitteratur, Lund 1981.
5. Poul Østergaard: "Programmering i COMAL-80". Teknisk Forlag a/s, 1983.
6. Glenford J. Meyers: "Kunsten at teste edb-programmer". Borgens Forlag 1984.

# TURBO-PASCAL SYNTAKS DIAGRAM

=====



```

(**)
PROGRAM Kost_beregning (INPUT, OUTPUT);

(*****)
(*-----*)
(* Filnavn:          \PASCAL.GRLVKOST1.PAS          *)
(* Programør:        Eva Hansen.                    *)
(* Programnavn:       Kost_beregning.                *)
(* Sidste ændring:    14. august 1985, kl. 14:30. (BGP) *)
(*                                                         *)
(* Program beskrivelse:                               *)
(*   Programmet kan lave en kostberegning paa grundlag af de *)
(*   produkter, der er defineret i programmet.            *)
(*   Programmet er menu-styret.                          *)
(*                                                         *)
(*   Programmet viser en hovedmenu, ud fra hvilken man kan vælge *)
(*   at udføre en kostberegning, se en oversigt over programmets *)
(*   produkter, ændre i produkt-data eller udskrive en nærmere *)
(*   specificeret oversigt over produkterne.              *)
(*                                                         *)
(*   Produkterne er beskrevet i faste tabeller i programmet. *)
(*   Antallet af produkter er givet ved konstanten "Max_antal". *)
(*                                                         *)
(*   Data for produkterne er fiber-indhold pr 100 g *)
(*               og pris          pr 100 g. *)
(*                                                         *)
(*   Kostberegningen foregaar saaledes: *)
(*   Der afgives en bestilling, hvor brugeren indtaster antallet *)
(*   af gram, som de enkelte produkter skal indgaa i beregningen *)
(*   med. *)
(*   Under kostberegningen, beregnes bestillingens samlede pris, *)
(*   fiberindhold mm. samt prisen pr 1000 fibre. *)
(*                                                         *)
(* Bemærkning: *)
(*   Programmet tager ikke højde for fejlagtig indtastning fra *)
(*   brugeren, idet programmet bruger standardfunktionen READ. *)
(*   Ved saadanne fejl, "gaar programmet ned" med Run-time fejl. *)
(*-----*)
(*****)

```

```

CONST
  Max_antal = 15;          (* Antal produkter. *)
  FF = 12;

TYPE
  Navn_type= STRING (20);
  Tekst= STRING(80);
  Navne_tabel= ARRAY [1..Max_antal] OF Navn_type;
  Tabel_type = ARRAY [1..Max_antal] OF REAL;

VAR
  Nr:      INTEGER;        (* Læst nr i hovedmenu. *)
  Produkt_navn: Navne_tabel; (* Tabel over produkternes navne. *)
  Fiber_indhold: Tabel_type; (* Tabel over fiberindhold pr 100 g. *)
  Pris      : Tabel_type; (* Tabel over priser i kr. pr 100 g. *)
  Bestilling : Tabel_type; (* Tabel over bestilte produkter i g. *)

```

```

(*

```

\*)

PROCEDURE PAGE ;

2.

BEGIN

CLRSCL;

(\* Clear Screen \*)

END;

(\*

```

*)
(*****)
(*-----*)
(*                                           *)
(* Initialisering af tabeller.                *)
(*                                           *)
(*-----*)
(*****)

```

```

PROCEDURE Init_produkt_navn;
(*****)
(* Initialiser produktnavnene.                *)
(*****)

```

```

BEGIN (* Init_produkt_navn *)
  Produkt_navn [ 1 ] := 'Rugbrød';
  Produkt_navn [ 2 ] := 'Franskbrød';
  Produkt_navn [ 3 ] := 'Kartofler';
  Produkt_navn [ 4 ] := 'Ymer';
  Produkt_navn [ 5 ] := 'Jordbær';
  Produkt_navn [ 6 ] := 'Svinekød';
  Produkt_navn [ 7 ] := 'Oksekød';
  Produkt_navn [ 8 ] := 'Gulerødder';
  Produkt_navn [ 9 ] := 'Mel';
  Produkt_navn [10] := 'Havregryn';
  Produkt_navn [11] := 'Salt';
  Produkt_navn [12] := 'Wienerbrød';
  Produkt_navn [13] := 'Lagkage';
  Produkt_navn [14] := 'Chokolade';
  Produkt_navn [15] := 'Kaffe';
END; (* Init_produkt_navn *)

```

```

PROCEDURE Init_fiber_indhold;
(*****)
(* Initialiser produkternes fiberindhold pr. 100 g. *)
(*****)

```

```

BEGIN (* Init_fiber_indhold *)
  Fiber_indhold [ 1 ] := 100;
  Fiber_indhold [ 2 ] := 50;
  Fiber_indhold [ 3 ] := 300;
  Fiber_indhold [ 4 ] := 5;
  Fiber_indhold [ 5 ] := 20;
  Fiber_indhold [ 6 ] := 0;
  Fiber_indhold [ 7 ] := 0;
  Fiber_indhold [ 8 ] := 500;
  Fiber_indhold [ 9 ] := 15;
  Fiber_indhold [10] := 365;
  Fiber_indhold [11] := 0;
  Fiber_indhold [12] := 0;
  Fiber_indhold [13] := 0;
  Fiber_indhold [14] := 0;
  Fiber_indhold [15] := 0;
END; (* Init_fiber_indhold *)

```

```

(*)

```

```
PROCEDURE Init_pris;
(*****)
(* Initialiser produkternes pris i kr pr 100 g. *)
(*****)
BEGIN (* Init_pris *)
  Pris [ 1 ] := 0.10;
  Pris [ 2 ] := 0.70;
  Pris [ 3 ] := 0.05;
  Pris [ 4 ] := 0.26;
  Pris [ 5 ] := 1.00;
  Pris [ 6 ] := 0.89;
  Pris [ 7 ] := 5.00;
  Pris [ 8 ] := 0.25;
  Pris [ 9 ] := 8.50;
  Pris [10] := 1.20;
  Pris [11] := 7.00;
  Pris [12] := 6.90;
  Pris [13] := 12.00;
  Pris [14] := 9.50;
  Pris [15] := 36.00;
END; (* Init_pris *)

PROCEDURE Init_bestilling;
(*****)
(* Nulstil bestillingstabellen til 0 g pr. produkt. *)
(*****)
VAR
  i: INTEGER;
BEGIN (* Init_bestilling *)
  FOR i := 1 TO Max_antal DO
    Bestilling [i] := 0;
  END; (* Init_bestilling *)
```

```

(*)
(*****
(*-----*)
(*)
(* Lav kostberegning. *)
(*)
(*-----*)
(*****)

PROCEDURE Lav_kostberegning;
(*****)
(* Lav kostberegning. *)
(* 1. Modtag bestilling af produkter. *)
(* 2. Beregn fiberindhold, pris mm for bestilling. *)
(* 3. Vis bestilling og kostberegnings-resultater. *)
(*****)
VAR
  Nr: INTEGER;
  Antal_fibre,
  Pris_f_bestilling,
  Pris_f_1000_fibre: REAL;
  i: INTEGER;

PROCEDURE Vis_spiseseddel;
(*****)
(* Vis spiseseddel og de bestilte mængder. *)
(*****)
VAR
  i: INTEGER;
BEGIN (* Vis_spiseseddel *)
  PAGE;
  WRITELN (OUTPUT, 'SPISE-SEDEL. ');
  WRITELN (OUTPUT, '-----');
  WRITELN (OUTPUT, 'Nr. Produkt                      Bestilt mængde. ');
  WRITELN (OUTPUT, '-----');
  FOR i := 1 TO Max_antal DO
    BEGIN
      WRITE (OUTPUT, i: 2, '. ');
      WRITE (OUTPUT, Produkt_navn [i]);
      WRITELN (OUTPUT, Bestilling [i]: 16: 1, ' gram. ');
    END;
  WRITELN (OUTPUT, '-----');
  WRITELN (OUTPUT, '99. Start beregning. ');
  WRITELN (OUTPUT); WRITELN (OUTPUT);
  WRITE (OUTPUT, 'Indtast et nr: ');
END; (* Vis_spiseseddel *)

PROCEDURE Modtag_bestilling;
(*****)
(* Modtag bestilling for givet produkt. *)
(* Bestillingen afgives i antal g. *)
(*****)
VAR Antal_gram: REAL;
BEGIN (* Modtag_bestilling *)
  WRITE (OUTPUT, 'Indtast mængde i gram: ');
  READLN (Antal_gram);
  Bestilling [Nr] := Antal_gram;
END; (* Modtag_bestilling *)

```

```

(*)

```

```

*)
PROCEDURE Vis_bestilling;
(*****)
(* Vis skærbillede over de bestilte produkter, *)
(* der indgår i kostberegningen. *)
(*****)
VAR i: INTEGER;
BEGIN (* Vis_bestilling *)
    PAGE;
    WRITELN (OUTPUT, 'BESTILLING. ');
    WRITE (OUTPUT, '-----');
    WRITELN (OUTPUT, '-----');
    WRITELN (OUTPUT, 'Mængde ': 12, ' Produkt ',
               'Antal fibre': 12, 'Pris': 12);
    WRITE (OUTPUT, '-----');
    WRITELN (OUTPUT, '-----');

    FOR i := 1 TO Max_antal DO
    BEGIN
        IF Bestilling [i] > 0 THEN
        BEGIN
            WRITE (OUTPUT, Bestilling [i]: 11: 1);
            WRITE (OUTPUT, Produkt_navn [i]: 22);
            WRITE (OUTPUT, Bestilling [i] / 100 * Fiber_indhold [i]: 12: 2);
            WRITE (OUTPUT, Bestilling [i] / 100 * Pris [i]: 12: 2);
            WRITELN(OUTPUT);
        END;
    END;

    WRITE (OUTPUT, '-----');
    WRITELN (OUTPUT, '-----');
    WRITELN(OUTPUT); WRITELN(OUTPUT);
END; (* Vis_bestilling *)

PROCEDURE Vis_kost_beregnings_resultat;
(*****)
(* Vis resultatet af kostberegningen. *)
(* - antal fibre i alt for bestillingen. *)
(* - pris i alt for bestillingen. *)
(* - pris pr 1000 fibre for bestillingen. *)
(*****)
BEGIN (* Vis_kost_beregnings_resultat *)
    WRITELN (OUTPUT, 'Fibre i alt: ', Antal_fibre: 10: 0);
    WRITELN(OUTPUT);
    WRITELN (OUTPUT, 'Pris i alt: ', Pris_f_bestilling: 10: 2, ' Kr. ');
    WRITELN(OUTPUT);
    WRITELN (OUTPUT, 'Pris for 1000 fibre:', Pris_f_1000_fibre: 10: 2, ' Kr. ');
    WRITELN(OUTPUT);
    WRITELN (OUTPUT, 'Tryk på RETUR for at komme til hovedmenu. ');
    READLN;
END; (* Vis_kost_beregnings_resultat *)

```

```

(

```

```

*)
BEGIN (* Lav_kostberegning *)
  (* Initialiser. *)
  Antal_fibre := 0;
  Pris_f_bestilling := 0;
  Pris_f_1000_fibre := 0;
  Nr := 0;
  Init_bestilling;
  (*****)
  (* 1. Modtag bestilling af produkter. *)
  (*****)
  WHILE Nr <> 99 DO
    BEGIN
      Vis_spiseseddel;
      READLN (Nr);
      IF (1 <= Nr) AND (Nr <= Max_antal)
      THEN Modtag_bestilling;
    END;

    (*****)
    (* 2. Beregn fiberindhold, pris mm for bestilling. *)
    (*****)
    FOR i := 1 TO Max_antal DO
      BEGIN
        IF Bestilling [i] > 0 THEN
          BEGIN
            Antal_fibre := Antal_fibre +
              Bestilling [i] * Fiber_indhold [i] / 100;
            Pris_f_bestilling := Pris_f_bestilling +
              Bestilling [i] * Pris [i] / 100;
          END;
        IF Antal_fibre <= 0 THEN
          BEGIN
            Antal_fibre := 0;
            Pris_f_1000_fibre := 0;
          END
        ELSE
          Pris_f_1000_fibre := Pris_f_bestilling * 1000
            / Antal_fibre;
        END;
      END;

    (*****)
    (* 3. Vis bestilling og kostberegning-resultater. *)
    (*****)
    Vis_bestilling;
    Vis_kost_beregnings_resultat;

  END; (* Lav_kostberegning *)

```

```

(*)

```

```

*)
(*****
(*-----*)
(*                                           *)
(* Vis produkt-oversigt.                    *)
(*                                           *)
(*-----*)
(*****

PROCEDURE Vis_produkt_oversigt;
(*****
(* Vis oversigt over produkter og tilhørende data *)
(* (fiberindhold, pris, mm) paa skærmen.          *)
(*****
VAR i: INTEGER;
BEGIN (* Vis_produkt_oversigt *)
  PAGE;
  WRITELN (OUTPUT, 'KOST-DATA. ');
  WRITE  (OUTPUT, '-----');
  WRITELN (OUTPUT, '-----');
  WRITELN (OUTPUT, 'Nr. ': 4,
    'Produkt navn': 12, 'Fibre pr. 100 g':18, 'Pris pr. 100 g':15);
  WRITE  (OUTPUT, '-----');
  WRITELN (OUTPUT, '-----');

  FOR i := 1 TO Max_antal DO
  BEGIN
    WRITE (OUTPUT, i: 2, '. ');
    WRITE (OUTPUT, Produkt_navn [ i]: 20);
    WRITE (OUTPUT, Fiber_indhold [ i]: 10: 2);
    WRITE (OUTPUT, Pris [i]: 15: 2);
    WRITELN(OUTPUT);
  END;
END; (* Vis_produkt_oversigt *)

PROCEDURE Vis_kost_data;
(*****
(* Vis oversigt over produkter og tilhørende data *)
(* (fiberindhold, pris, mm) paa skærmen.          *)
(*****
BEGIN (* Vis_kost_data *)
  Vis_produkt_oversigt;

  WRITELN(OUTPUT);
  WRITELN (OUTPUT, 'Tryk på RETUR for at komme tilbage til hoved-menuen. ');
  READLN;
END; (* Vis_kost_data *)

(*)

```

```

*)
(*****
(*-----*)
(*                                           *)
(* Ændre i kost-data.                      *)
(*                                           *)
(*-----*)
(*****

PROCEDURE Aendre_i_kost_data;
VAR Nr: INTEGER;

PROCEDURE Aendre_data_for_produkt (Nr: INTEGER);
BEGIN (* Endre_data_for_produkt *)
  WRITE (OUTPUT, 'Indtast fiber-indhold pr 100 g.:');
  READLN (Fiber_indhold [Nr]);
  WRITE (OUTPUT, 'Indtast pris i kr pr 100 g.:');
  READLN (Pris [Nr]);
END; (* Endre_data_for_produkt *)

BEGIN (* Endre_i_kost_data *)
  Nr := 0;
  WHILE Nr <> 99 DO
  BEGIN
    Vis_produkt_oversigt;

    WRITELN(OUTPUT);
    WRITELN (OUTPUT, 'Ændring af kost-data. ');
    WRITELN (OUTPUT, 'Indtast Nr paa produkt, der skal ændres. ');
    WRITELN (OUTPUT, '                (99 = Tilbage til hoved-menuen). ');
    READLN (Nr);

    IF (1 <= Nr) AND (Nr <= Max_antal)
    THEN Aendre_data_for_produkt (Nr);
  END; (* WHILE *)
END; (* Endre_i_kost_data *)

(*)

```

```

*)
(*****)
(*-----*)
(*                                           *)
(* Specielle oversigter.                      *)
(*                                           *)
(*-----*)
(*****)

PROCEDURE Specielle_oversigter;

VAR
  Nr : INTEGER;

PROCEDURE Msg ( Col, Lin: INTEGER; Str: Tekst);
(*****)
(* Skriver tekststrengen Str ved punktet angivet *)
(* af x,y-koordinaterne Col og Lin.              *)
(*****)
BEGIN
  GOTOXY(Col,Lin);
  WRITE(OUTPUT,Str);
END;

PROCEDURE Vis_menu;
(*****)
(* Skriver oversigten over de specielle udskrif- *)
(* ter, der kan laves.                             *)
(*****)
BEGIN
  Page;
  Msg( 10,10,' Specielle oversigter: ');
  Msg( 10,11,'-----');
  Msg( 10,13,' 1: Råvarer med fiberindhold større end et angivet antal ');
  Msg( 10,14,' 2: Råvarer der koster mindre end en angivet pris pr 100 g. ');
  Msg( 10,15,' 3: 1 og 2 kombineret. ');
  Msg( 10,17,'99: Tilbage til hovedmenu. ');
  Msg( 10,20,' Indtast et nr: ');
END;

(*)

```

```

PROCEDURE Lav_st_skema;
(*****)
(* Proceduren tegner et standard skema, der kan anven-*)
(* des ved udskrift af de specielle oversigter.      *)
(*****)
BEGIN
    Msg(1,5,'');      (* Msg kan med en tom streng bruges som GOTOXY*)

    WRITELN(OUTPUT,'Nr. ':4,'Produkt navn':12,'Fibre pr. 100 g':22,
              'Pris pr. 100 g':20);
    WRITELN(OUTPUT,'-----');
    Msg(1,23,'');
    WRITELN(OUTPUT,'-----');
    Msg(1,8,'');      (* Klar til udskrift i skema *)
END;

```

```

PROCEDURE Udskriv_st_format( Produktnr:INTEGER);
(*****)
(* Proceduren udskriver produkter i standard format så*)
(* passer i standard skemaet                          *)
(*****)
BEGIN
    WRITE (OUTPUT,Produktnr: 2, '. ');
    WRITE (OUTPUT,Produkt_navn [Produktnr]: 20);
    WRITE (OUTPUT,Fiber_indhold [Produktnr]: 10: 2);
    WRITE (OUTPUT,Pris [Produktnr]: 18: 2);
    WRITELN(OUTPUT);
END;

```

(\*)

```

*)
PROCEDURE Fiber_antal;
(*****)
(* Skriver en oversigt over de råvarer der indeholder *)
(* flere fibre end den indlæste grænse pr. 100 g. *)
(*****)
VAR
  I: INTEGER;
  Fiber_graense: REAL;

BEGIN
  PAGE;
  Msg(5,10,' Indlæs grænseværdien for det mindste fiberindhold pr 100 g.: ');
  READLN(Fiber_graense);

  PAGE;
  Msg(1,1,'Oversigt over råvarer hvis fiberindhold pr. 100 g. er større end: ');
  WRITELN(Fiber_graense:4:0);
  WRITELN('=====');

  Lav_st_skema;
  FOR I:=1 to Max_antal DO
    IF Fiber_indhold[I] >= Fiber_graense THEN
      Udskriv_st_format(I);
  Msg(1,24,'Aktiver RETURN For at komme tilbage til udskriftsoversigt. ');
  READLN;

END;

PROCEDURE Pris_graense;
(*****)
(* Skriver en oversigt over de råvarer der koster *)
(* mindre end den indlæste pris pr. 100 g. *)
(*****)
VAR
  I: INTEGER;
  Prisen: REAL;

BEGIN
  PAGE;
  Msg(15,10,' Indlæs den maksimale pris i kroner pr 100 g.: ');
  READLN(Prisen);

  PAGE;
  Msg(1,1,'Oversigt over råvarer hvis pris pr. 100 g. er mindre end kr: ');
  WRITELN(Prisen:4:2);
  WRITELN('=====');

  Lav_st_skema;
  FOR I:=1 to Max_antal DO
    IF Pris[I] <= Prisen THEN
      Udskriv_st_format(I);
  Msg(1,24,'Aktiver RETURN For at komme tilbage til udskriftsoversigt. ');
  READLN;

END;

```

(\*)

\*)

```

PROCEDURE Pris_og_fiber_graense;
(*****)
(* Skriver en oversigt over de råvarer der koster *)
(* mindre end den indlæste pris pr. 100 g. samt *)
(* indeholder flere fibre end det indlæste antal. *)
(*****)
VAR
  I: INTEGER;
  Prisen, Fiber_antal: REAL;

BEGIN
  PAGE;
  Msg(5,10,' Indlæs den maksimale pris i kroner pr 100 g.: ');
  READLN(Prisen);
  Msg(5,12,' Indlæs grænseværdien for det mindste fiberindhold pr 100 g.: ');
  READLN(Fiber_antal);

  PAGE;
  Msg(1,1,'Oversigt over råvarer hvis pris pr. 100 g. er mindre end kr: ');
  WRITELN(Prisen:4:2);
  Msg(1,2,'                og indeholder flere fibre pr. 100 g end: ');
  WRITELN(Fiber_antal:5:0);
  WRITELN('=====');

  Lav_st_skena;
  FOR I:=1 to Max_antal DO
    IF (Pris[I] <= Prisen) AND (Fiber_indhold[I] >= Fiber_antal) THEN
      Udskriv_st_format(I);
  Msg(1,24,'Aktiver RETURN For at komme tilbage til udskriftsoversigt. ');
  READLN;
END;

BEGIN
  REPEAT
    Nr := 0;
    Vis_menu;
    READLN(Nr);
    CASE Nr OF
      1: Fiber_antal;
      2: Pris_graense;
      3: Pris_og_fiber_graense;
    END; (* CASE *)
  UNTIL Nr = 99;
END;

```

(\*)

```

*)
(*****)
(*-----*)
(*                                     *)
(* Hovedmenu.                         *)
(*                                     *)
(*-----*)
(*****)

```

```

PROCEDURE Vis_hovedmenu;
(*****)
(* Vis systemets hovedmenu paa skermen. *)
(*****)
BEGIN (* Vis_hovedmenu *)
  PAGE;
  WRITELN (OUTPUT, ' HOVEDMENU. ');
  WRITELN (OUTPUT, '-----');
  WRITELN (OUTPUT, ' 1. Lav kostberegning. ');
  WRITELN (OUTPUT, ' 2. Vis kost-data. ');
  WRITELN (OUTPUT, ' 3. Endre i kost-data. ');
  WRITELN (OUTPUT, ' 4. Specielle oversigter. ');
  WRITELN (OUTPUT, '-----');
  WRITELN (OUTPUT, '99. Forlad programmet. ');
  WRITELN (OUTPUT); WRITELN (OUTPUT); WRITELN (OUTPUT);
  WRITE (OUTPUT, 'Indtast et nr: ');
END; (* Vis_hovedmenu *)

```

```

(*****)
(*-----*)
(*                                     *)
(* Kost-beregning - Hovedprogram.    *)
(*                                     *)
(*-----*)
(*****)

```

```

BEGIN (* Kost-beregning *)
  (*****)
  (* Initialiser tabeller. *)
  (*****)
  Init_produkt_navn;
  Init_fiber_indhold;
  Init_pris;

  (*****)
  (* Initialisering. *)
  (*****)
  Init_bestilling;

  (*****)
  (* Styring af hovedmenu. *)
  (*****)
  REPEAT
    Nr := 0;
    Vis_hovedmenu;
    READLN (Nr);
    CASE Nr OF
      1: Lav_kostberegning;
      2: Vis_kost_data;
      3: Aendre_i_kost_data;
      4: Specielle_oversigter;
    END; (* CASE *)
  UNTIL Nr = 99;
END. (* Kost_beregning *)

```

Type: Person oplysninger = Record

Record

Nr ; tal integer

Navn : streng [max]

Født : integer

Score : integer

Aktio : Boolean

End;

Var person 1 ; person 2 = person oplysning

Person 2. Navn := Jonny

Whit person 1 Do

Navn := lotte;

Nr := 100;

Person talel

Array af Person oplysninger

VAR

Gemmer Data

Person\_Fil : File OF personoplysninger

Begin

ASSIGN (personfil, 'B:Person.DAT');

REWRITE (personfil);

For i = 1 to N DO

WRITE (personfil, Person tabel [i],)

CLOSE : (personfil, ?);

End;

Hente Data

Begin

ASSIGN (personfil, 'B:Person.DAT');

Reset : personfil;

While Not EOF Personfil and i < max  
DO

Begin

Read (Personfil, Person tabel [i]);

close (Personfil)

End;

End;

Linære Lister

Konstanter der er nyttige i program  
telst streng [11-11-85] version 1.2

VAR

TEST : Boolean

if Test then writel;

Be

$$P_{vis} 100 = \text{belijb} / \text{Antal fibre} \cdot 1000$$

5 fibre

0,26

0,05 · 1000

IF

$P_{vis} 100$

if

belieb ~~de~~  
then

0 or Antal fibre  $\angle 0$

$a \leq b;$

Type = char

Type noun = Packed Array [1 .. Index] of char;

string = Packed Array [1 .. 80] of char;

Noun = " [1 .. 15] of char;

Var

S : string

N : Noun

Begin

N := 'Hans Hansen';

Forman ; packed array [1 .. 6] of char  
Afterman ; " "

ch; char

for i = 1 to 6

do begin read (input ch; for k from

Packed array begins to char

Product Moon of  $[i]$

Write

Data type

Real  
integer

Erkl ring

Program test.pas

const

max = 15;

Type = Produkt type

Produkt type = string [20]

Flushing of data <sup>Array</sup> ~~for~~ 1 Tabel ~~tbl~~ et  
andet array of en anden dimension  
med for next loop

For i: 1 to 10 do

Tabel 1 [i] := Tabel 2 [i];

For i: 1 to 15 do read Prince[i];

Type

Todim = array [1..5, 1..4] of integer

var

a, b = Todim

integer

MS DOS med Turbo Pascal 3.01A  
SPERRY med 8086 processor

test .pas

turbo Kostplan - PAS

Kostplan - PAS

hoved program sidst i program  
hvor kald til under programmer står

write udskrift

writeln udskrift + return + linefeed

input og output behøves ikke at specificeres  
hvis det er keyboard og skærm

under program

Procedure Procedure navn :

Definere og erklære

If hvis så.

then

else

If Nr = 1 Then lav kostberegning

Else IF Nr = 2 THEN Vis kostdata

Else IF Nr = 3 Then ABNDRE Kost Data

Else IF Nr = 4 special oversigten

case Helvæl

VI : Sætning

Else : writeln ('Fejl i case')

otherwise

case skal være konstanter

If = then kan være variable

skal afsluttes END; (\* case \*)

Else ('Fejl i case');  
(Prøv igen)

Type

Typename = ARRAY [1 .. 10] OF Integer

Var

Prime = Typename

Begin

Prime [1] :=

Prime [1] < Prime ;

End

Const Max = 100 (feels man antel  
Produktar

Var: Konst integer

for I = 1 to 5 do

Begin

Read (Konst)

Prime [I] := Konst

End;

Program

Konst

Type

String

Produkttype = String [20];

Var Produktname : Produkttype

Writeln : Vi du <sup>fortsett</sup>  
Ja/nej )

Funktion Ja : Boolean;  
VAR svar : string [3]

Begin

Readln : (Svar);

Svar := CAPITAL (Svar)

IF SVAR = 'JA' Then

Then Ja := true

Else IF SVAR = 'NEI'

Then Ja := False

Until svar = 'ja' or (SVAR) Nej )

End;

uge 4

uge Mart

uge May

control

Break

3

bestilling > 0

- mangle udskives for 1 til max<sup>tel</sup>

total fibre

total pris /  
fibre \* 1000

Kaffe 100

Rog 100

100

100

36,10

36,10

144

EVA Hansen

Procedure DEL (var  $X$  : Real);

Begin ~~DEL (X)~~

$X := X/2$   
DEL(X);

END;

$X := X/2$

IF  $X > \text{Tolerance}$

Then Del(X);

opt/ft

~~X: Real~~; var i integer

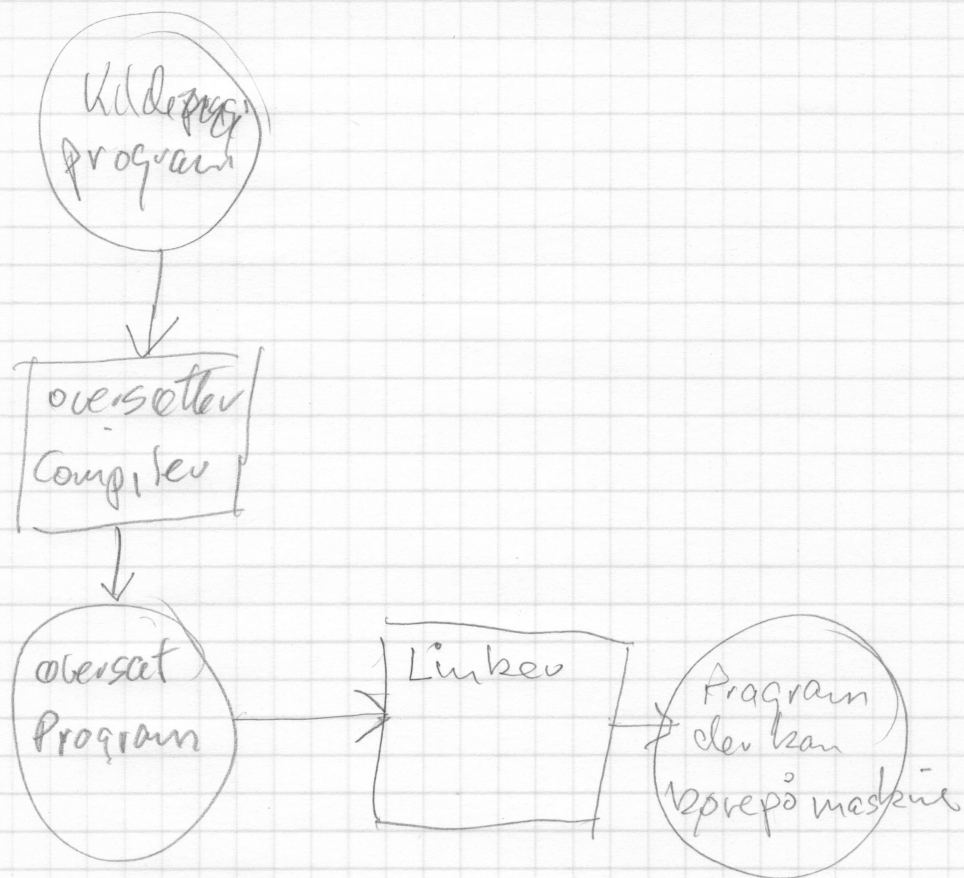
CVAR X : Real;

VAR I; integer)

Begin

End

sumpte tellere



Procedurer

initialisering  
skrive

lese

Tæse

Skrive

andre

belegne

Tabel  
Filer

Fejl behandling control på inddata

Error meddelelser

Menuer - tilbeder

Beregnings

Hjælpes funktioner (specielle som ikke er i systemet)

Fælles procedurer skrive sammen

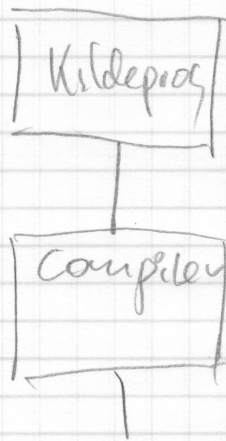
~~Fæl~~ procedures skal defineres for de kaldes

global variabel

Type  
Bestilling  
var real

Sound, Delay : integer

Type converting  
Kildeprogram



Procedurer skal defineres før de kaldes ellers Fejl  
kan det ikke lade sig gøre  
kan man skrive Procedure Fejl: Forward

Niveau

Globale

Lokale

[ Hællevariable  
Hjælpevariable ] lokal

Betydningsfulde variable er globale Max\_tal

Parametre

Funktion Sin ( ARG: Real ) : Real  
Begin  
~~~~~  
End;

Procedurer      contra      Funktioner

for I to max - tal  
 Hirs <sup>menge</sup> fiber indhold  $> 0$  then  
 bestilling  
 fiber indhold i fiber

$$\text{Fiber indhold} / 100g \times \text{Pris} / 100$$

ind i Pris

hvis fiber indhold i

bestilling

|   |  |
|---|--|
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |

kontagren

fiber indhold

fiber indhold

Pris

- 1 Find fornuftige procedurer / funktioner
- 2° Definere P/F inden den skal bruges  
Placere rigtigt i ram
3. Overvej om P/F skal være Global eller lokal
4. Globale eller lokale variable
- 5 Globale variable kan bruges uafhængt  
P/F (husk scoping) ikke lokale med samme navn
- 6 Bestemme P/F grænseflade dvs Parametre.
- 7 Værdi (variabel) eller Variable overført

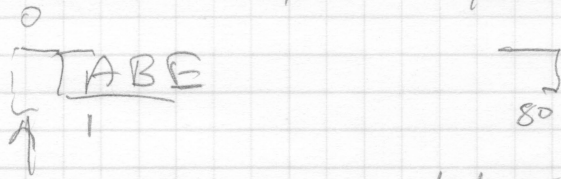
Start med hovedmenu

Procedurer for hvert element i hovedmenu

1 Funktioner

ch : Boolean

2 Funktion length (textstreng : string[80]) : integer



tegn der svarer til sidste udfyldte  
plads i streng

Begin

length := OrdStreng [0]

End

3) Funktion Capital (whereas string) : string[80]

! : Var diff := Ord('A') - Ord  
Begin

For i := 1 to len(string) do

IF (string[i] >= 'a') and (string[i] <= 'z')

Then (string[i] := Chr(Ord(string[i]) - diff))

Capital := string

End

IF nu IN [1..5, 99] Then