JERNINDUSTRIENS FORLAG

Introduktion i struktureret højniveausprog

1985

Instruktioner
Øvelser

Jern- og Metalindustrien

Best. nr. 4061

Forord

Lærebogen anvendes i undervisningen på Metalindustriens Efteruddannelseskursus nr. 5822 Introduktion i struktureret højniveausprog og EFG-uddannelsen, Datamekaniker, 2. del, trin 2.

Lærebogen er udarbejdet på foranledning af Metalindustriens Efteruddannelsesudvalg.

Faglærere fra Sønderborg tekniske skole har udarbejdet og tilrettelagt lærebogen i samarbejde med Jernindustriens Forlag

Lærebogen er delt op i følgende afsnit:

Instruktioner
Øvelser

Instruktionerne omfatter:

Programstruktur
Procedurer
Funktioner
Parametre
Moduler

Øvelserne omfatter:

Programstruktur
Procedurer
Funktioner
Parametre
Moduler

Bladene er forsynet med huller og kan sættes ind i ringbind, efterhånden som de tages i brug.

Til brug ved undervisningen har lærebogen fortløbende sidenumre nederst på siderne.

I forbindelse med brug af lærebogen skal der anvendes en diskette med "Kildetekst-programmel". Den kan bestilles på Jernindustriens Forlag.

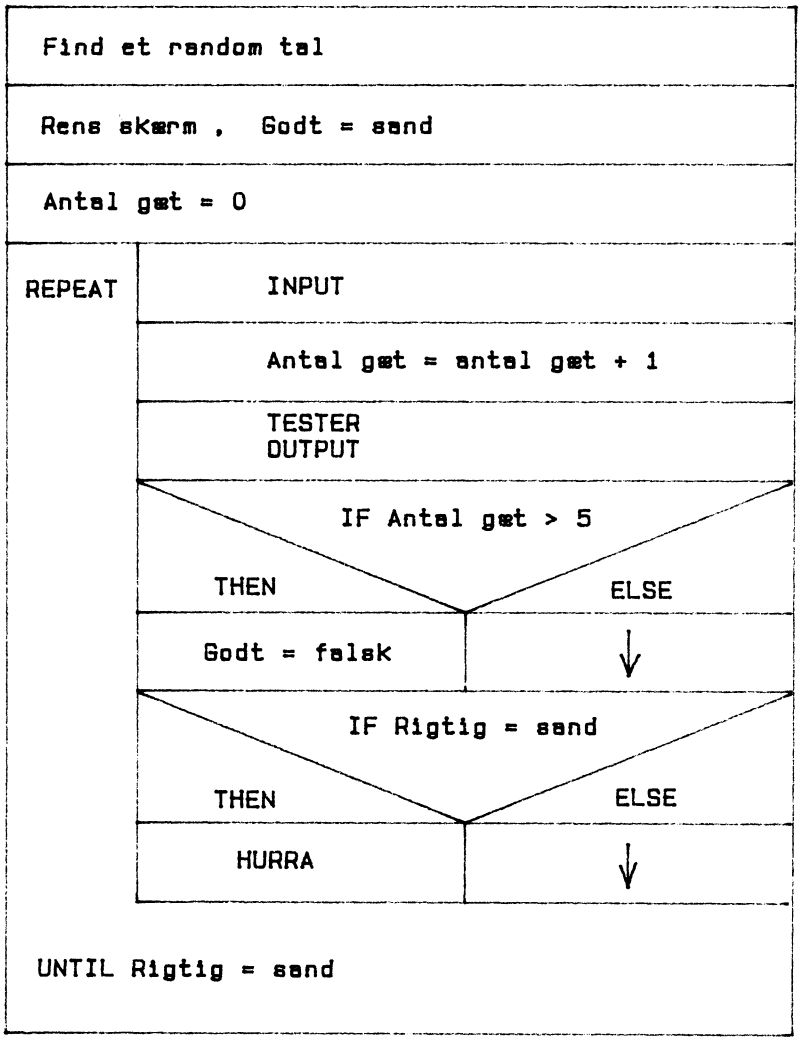
Ud over programmel, der er på "Kildetekst-disketten", skal der anvendes færdigkøbt programmel, som fremgår af udstyrslisten til kurset ME 5822 eller Datamekaniker 2.del, trin 2.

København, september 1985

JERNINDUSTRIENS FORLAG

Indholdsfortegnelse

STRUKTURERET HØJNIVEAUSPROG



Introduktion i struktureret højniveausprog

INDLEDNING

Hvad betyder Struktureret Højniveausprog ?

Først struktureret, ordet struktur betyder en entydig opdeling og en veldefineret indre opbygning .

Derefter Højniveausprog, for at en microcomputer kan udføre et program, skal dette foreligge på maskinkode, dvs. det "sprog" som den pågældende microprocessor direkte forstår.
Programmering i maskinkode er imidlertid meget kompliceret.
Hver eneste instruktion har en talkode, og udfører kun en lille proces.
For at lette denne kodning har man indført "symbolsk maskinsprog" også kaldet "assembler sprog", hver instruktion er nu en bogstavskode, som er lettere at huske end en tal kode, men ellers svarer en bogstavskode til en maskinkode.
Det betyder at programmering stadig er ret kompliceret.
Det har man prøvet at råde bod på ved udvikle forskellige "højere programmeringssprog", der skulle gøre programmering lettere. Hver højniveau sætning indeholder en række maskinkoder, som selve programmeringssprogets program oversætter til maskinkoder.

Fordele ved at bruge højniveausprog frem for maskinsprog:

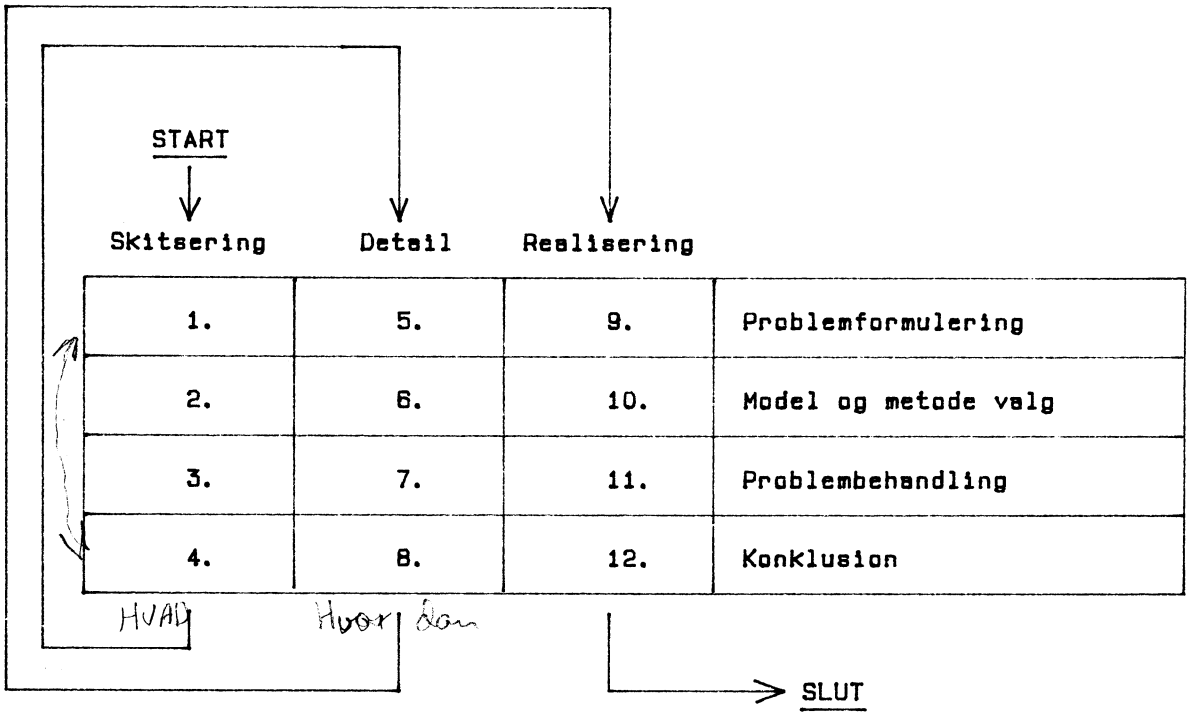
- Hurtigere udviklingstid.
- Lettere at rette og modificere.
- Lettere at forstå for mennesker.
- Mindre dokumentation nødvendig.
- Lettere at overføre fra en computer til en anden.

Ulemper ved at bruge højniveuasprog frem for maskinsprog:

- Langsommere ekseveringstid.
- Større lagerkrav.

PROGRAM UDVIKLING

ARBEJDSSKEMA



PROGRAM UDVIKLING

I forbindelse med programudvikling er det vigtigt at have en entydig arbejdsgang. I det efterfølgende er der vist en måde hvorpå en arbejdsgang kan forløbe gennem 12 punkter som i hovedtræk hedder SKITSERING - DETAILERING - REALISERING og hvor hver af underpunkterne hedder PROBLEMFORMULERING - MODEL OG METODE VALG - PROBLEMBEHANDLING - KONKLUSION.

SKITSERING

Formålet med skitsering er at:
- Skabe afklaring og overblik over opgaven.
- Få udformet en brugsanvisning
- Tidsrammen for projektet fastlægges.
- Økonomien bag projektet fastlægges.

SKITSERING 1 - PROBLEMFORMULERING

Under problemformuleringen er det vigtigt at få udformet en brugsanvisning af det program der skal udvikles udfra kundens ønsker, således vedkommende kan genkende programmets funktion.
Under problemformuleringen er det nødvendig med en afgrænsning af opgaven d.v.s. at konklusionen udfyldes løbende.

SKITSERING 2 - MODEL OG METODE VALG

Det er nødvendig med en systematisk metode til at fastlægge et programs overordnede struktur. Det er ikke en fordel med eet stort program da det er vanskeligt at dokumentere og vedligeholde. Derfor skal der foregå en opdeling af programmet i del-funktioner, en metode der kan anbefales er en funktionsnedbrydning efter det princip der hedder TOP-DOWN.
Se funktionsnedbrydning side 6.

SKITESERING 3 - PROBLEMBEHANDLINGEN

Består i at få en passende overbliksviden. Således at når arbejdet er tilendebragt er der tegnet et Hierarkisk Input Process Output diagram forkortet et HIPO diagram. Hvor input og output beskriver de variable der skal overføres mellem proces modulerne og proces beskrivelsen går ud på at beskrive hvad der skal udføres i modulet men ikke hvordan.

SKITSERING 4 - KONKLUSION

Vil primært bestå i en afgrænsning af opgavens art og omfang. Således at programmøren ikke fristes til at udvikle sit eget program i stedet for kundens. Dvs. brugsanvisningen skal være fyldestgørende.
Under skitseringsfasen er der vigtigt at problemformuleringen og konklusionen hænger logisk sammen

DETAILERING

Detail behandlingen har det formål at gå i dybden med en egentlig data bearbejdning.
Når detail behandlingen er grundigt gennemarbejdet er mere end halvdelen af opgaven tilendebragt.

DETAILERING 5 - PROBLEMFORMULERINGEN

Problemformuleringen skal nu gerne være så præcis at der ikke opstår tvivl og usikkerhed under løsning af opgaven. Det betyder, at der skal beskrives en rækkefølge af programmet, og hvornår de enkelte moduler kaldes således, at hele programmet fremtræder som en helhed.

DETAILERING 6 - MODEL OG METODE VALG

Består her i at fastlægge hvilke kontrolstruktur der skal anvendes, således at hvert modul kan beskrives enten som PSEUDO KODER - STRUKTURERET FLOWCHART eller NASSI-SCHNEIDER-MAN-STRUKTUGRAM.

DETAILERING 7 - PROBELEMBEHANDLING

Problembehandlingen går ud på at beskrive de enkelte moduler enten som SEKVENS - UDVÆLGELSE eller GENTAGELSE, således at modulerne kan overføres til et struktureret programmeringssprog f.eks. Pascal.

DETAIL 8 - KONKLUSION

Der træffes konklusion ud fra de enkelte delproblemer, således at modulerne tilsammen danner en helhed.

REALISERING

Realisering har det formål at udforme selve programmet. Og med den foregående arbejde er det under realiseringsfasen kun nødvendig at koncentrere sig om programmeringssprogets syntaks.

REALISERING 9 - PROBLEMFORMULERING

Under problembehandlingen må det afklares, hvordan de enkelte moduler kan realiseres, der laves oversigt over konstanter og variable til hvert modul, således at overgangen til et struktureret programmeringssprog, som f.eks. Pascal ikke bliver nogen problem.

REALISERING 10 - MODEL OG METODE VALG

Her må man sikre sig at vælge et programmeringssprog som kan overholde kontrolstrukturene, og det er ikke alle programmeringssprog som er bygget struktureret op. Ved at vælge programmeringssproget Pascal er vi sikker på at kunne overholde kontrolstrukturene, da Pascal er et struktureret programmeringssprog.

REALISERING 11 - PROBLEMBEHANDLING

Problembehandlingen er nu mere eller mindre låst fast til programmeringssprogets syntaks som en konsekvens af model og metode valg.
Samtidig er det vigtigt at lave en løbende dokumentation.

REALISERING 12 - KONKLUSION

Desværre hænder det ofte at opgaveløsningen forringes væsentligt af en ufuldstænding og usystematisk dokumentation, hvilket gør at en senere servicering af programmet er umulig -det kan betyde at hele programmet må skives om. Så som den sidste konklusion må det slås fast, at både en overordnet og en løbende dokumentation er vigtig.

FUNKTIONSNEDBRYDNING

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 (brugsanvisning på programmet). Hvorledes det gøres gemmes til en senere detailbearbejdning.

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

De enkelte separate delfunktioner kan derefter hver for sig yderligere neddeles i separate delfunktioner, hvorved der fremkommer en programstruktur med et topmodul samt et hierarki af underliggende moduler, hvor de nederste moduler tager sig af kommunikation med omverdenen (Tastatur,Skærm,Printer osv.). Nedbrydningen skal være således at de enkelte moduler kan realiseres uafhængig af hinanden.

Funktionsnedbrydning er ikke een bestemt teknik og der findes ikke een bestemt løsning. Fremgangsmåden afhænger helt af programmørens erfaring.

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.

Ved at starte med den overordnede problemstillinge, og derefter bevæge sig ned gennem problemet, er der hele tiden en sammenhæng til det overordnede niveau.
Det vil sige man laver et TOP DOWN princip.

Ved trinvis raffinering udskydes behandlingen vedrørende delproblemerne, hvis løsning ikke er umiddelbar synlige på "top-niveauet".

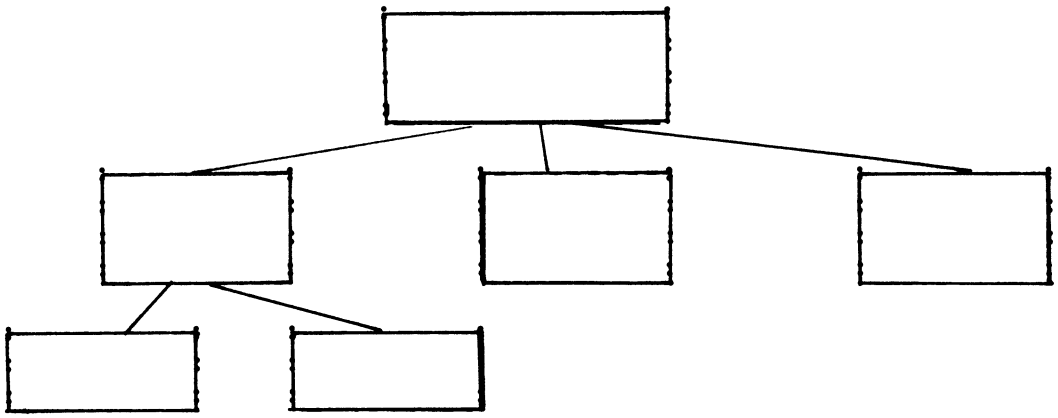
Ved en opdeling af den overordnede problemstilling i et antal separate moduler er det muligt at flere programmører kan løse hver sit modul.

Det gør det muligt at servicere programmet, Hvis programmet skal ændres, vil det være let at finde det/de steder ændringen skal foretages.

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

Funktionsnedbrydning kan foregå på flere måder, f.eks. ved hjælp af hierakisk funktions struktur.

Eks.



De enkelte moduler kendetegnes ved input, process og output kaldet HIPO Hierakisk Input, Process og Output.
Hvor input og output beskriver modulets grænseflade d.v.s. de parametre der skal føres til og fra modulet.
Og proces er en modulbeskrivelse der fortæller hvad modulet udfører, men ikke hvordan.

Formålet med funktionsnedbrydningen er bla. også ud fra modulbeskrivelsen at se om der er proces gentagelser. Således at man kan bruge den samme proces flere gange.

Hvorledes de enkelte moduler realiseres, er beskrevet under KONTROLSTRUKTURER.

KONTROLSTRUKTURER

Hvorledes bør logikken i et modul udformes, for at funktionen bliver udført korrekt ?
Vi vil her se på hvorledes programmets logik kan udformes så det bliver let at fejlrette og dermed også let at læse.
For at lette læseligheden af et program bør der kun anvendes bestemte kontrolstrukturer som:

SEKVENS

UDVÆLGELSE

GENTAGELSE

I de efterfølgende eksempler vises der brug af SEKVENS - UDVÆLGELSE og GENTAGELSE kontrolstrukturer i fobindelse med PSEUDO KODER - STRUKTUREKET FLOWCHART og NASSI-SCHNEIDERMAN-STRUKTUGRAM.

PSEUDOKODER

I pseudokoder bruges en grundlæggende kontrolstruktur og margin-indrykinig på en standardiseret måde, men ellers er udtryksformen overladt til programmøren. Pseudokoder kan minde meget om gængse programmeringssprog, men har fordelen ved at være fritaget for de forskellige programmeringssprogs strikse syntaks.

Hver af disse kontrolstrukturer kan i et struktureret højniveau-sprog direkte omsættes til programlinier.

SEKVENS

Ved en sekvens forstås en række programlinier der udføres i den rækkefølge de forekommer i programmmet. En sekvens kan også bestå af nogle programmoduler der udføres i rækkefølge.

UDVÆLGELSE

IF-THEN

IF-THEN-ELSE

I den grundlæggende kontrolstruktur af udvælgelsen, vil der være en eller to processer at vælge i mellem.

Eks.1

```
IF  (logisk udsagn)
   THEN

     (Process A)

END
```

Eks.2

```
IF  (logisk udsagn)
   THEN

     (Process A)

   ELSE

     (Process B)

END
```

I de her viste eksempler er der to forskellige måder at udføre udvælgelse på
1. eks.
I mange programmer kan der ofte være tale om at en del af et program under bestemte forudsætninger skal udelades, dvs. et valg om en proces skal udføres eller springes over.
2. eks.
I det her viste eksempel er det et spørgsmål om at vælge mellem to forskellige processer, og dersom den ene ikke udføres skal den skal den anden udføres.

CASE-SÆTNING

En anden form for udvælgelse er valg mellem flere alternativer ved hjælp af CASE-SÆTNING, idet der hermed opnås en mere overskuelig programstruktur.

Eks.

```
CASE valgudtryk OF

  valg 1 (process A)

  valg 2 (process B)
     :
     :
     :
  valg n (process X)

END
```

CASE-sætningen udføres ved at først defineres valgudtryk derefter sammenlignes denne værdi med hver af valg muligherderne. Hvis værdien findes i valg listen, så udføres den tilhørende proces.

GENTAGELSE

Den tredie kontrolstruktur er gentagelsen, der med et eller andet logisk udsagn tester påbetingelsen for gentagelsen. Der er to former for gentagelser, den ene skal altid behandles mindst een gang, det er REPEAT-UNTIL. Ved den anden form skal startbetingelsen være opfyldt for at der overhovedet foretages en gentagelse, dette kaldes WHILE-DO.

REPEAT-UNTIL

En gentagelse der altid gennemløbes een gang kan se således ud:

```
REPEAT

  (Process A)

UNTIL (logisk udsagn)
```

WHILE-DO

En anden gentagelse der gentages indtil det logiske udsagn ikke er opfyldt ser således ud:

```
WHILE (logisk udsagn)

  DO (Process A)

END
```

For de to viste gentagelser gælder at det logiske udsagn skal ændres v.h.a. processen da programmet ellers aldrig vil komme videre.

FOR-TO

En gentagelse der gentages et forud bestemt antal gange uanset processen ser således:

```
FOR (startværdi) TO (slutværdi)

  DO (process A)
```

Det er jo meget godt, at der findes 3 forskellige repetionskonstruktioner, men hvornår skal den ene benyttes og hvornår skal den anden? Det er muligt altid at klare sig med WHILE-DO konstruktionen, men det er ikke altid hensigtsmæssig.
Følgende retningslinier kan være en hjælp ved valg af repetionskonstruktion:

Vælg WHILE-DO konstruktion når der kan være tilfælde, hvor man ikke ønsker at programmet gennemløbes i sløjfen.

Vælg REPEAT-UNTIL konstruktionen når stopbetingelsen er logisk betinget, dvs. at den først kan fastlægges ved gennemløb af programmet. Det er altså et krav at programmmet altid gennemløbes mindst een gang i sløjfen.

Vælg FOR-TO konstruktion når program sløjfen kan fastlægges udenfor sløjfen. Dvs. start- og slutværdi for kontrolvariable kan være beregnet, før vi når til programsløjfen.

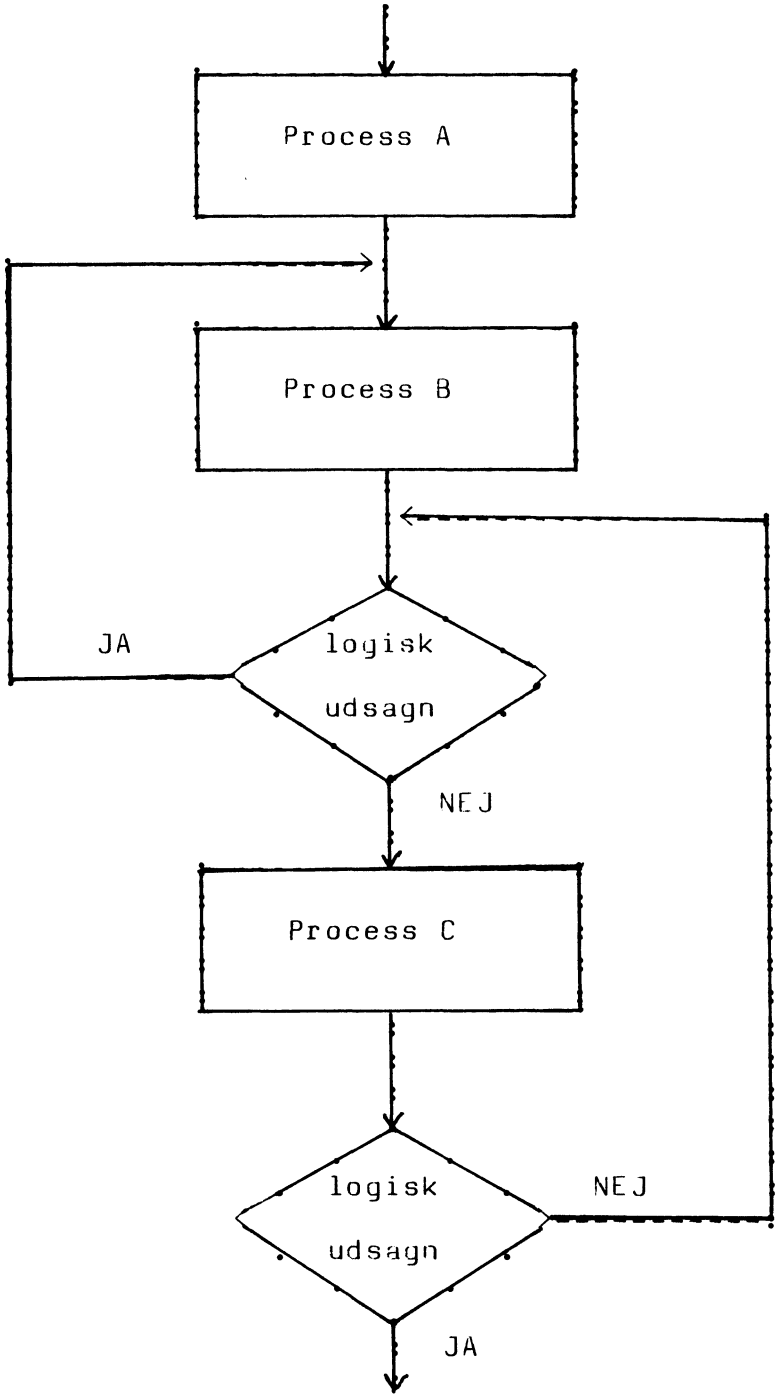
STRUKTURERET FLOWCHART

Sekvens

Anvendelsen af flowcharts som programmeringsgrundlag, giver gode muligheder for et ustruktureret program, idet der er meget få regler for hvodan man tegner sit flowchart.
Flowcharts alene er ikke skyld i at et program er/bliver ustruktureret.

PROGRAMMØREN HAR ANSVARET ! !

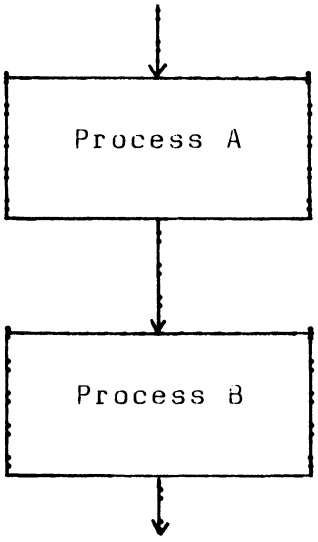
Et eksempel på et ikke struktureret flowchart:



Det der bl.a. kendetegner et ikke struktureret flowchart er at programsløjferne krydser hinanden.

SEKVENS

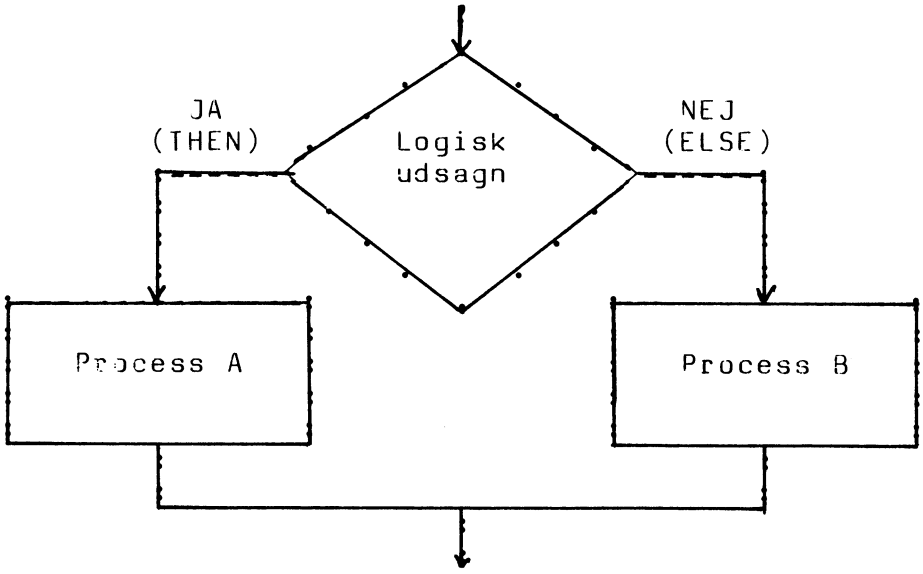
Den første grundlæggende struktur er sekvens.
Et struktureret flowchart af nogle sekvenser ser således ud:



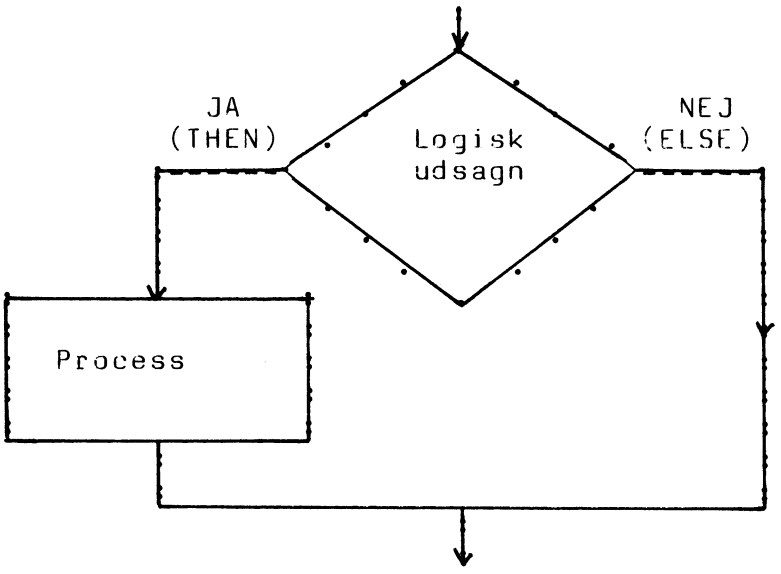
Det der kendetegner en sekvens er en række programmoduler der udføres i den rækkefølge de forekommer i programmmet.
I en sekvens har programmodulerne kun en indgang og en udgang. Dvs. man hopper ikke ind midt i et programmodul og forlader det heller ikke midt i programforløbet.

UDVÆLGELSE

Den anden grundlæggende kontrolstruktur er udvælgelsen. Ved udvælgelsen vil der være en indgang og to udgange.
Et flowchart ser således ud:



I det her viste eksempler er det et spørgsmål om at vælge mellem to forskellige processer og dersom den ene ikke udføres skal den anden udføres.
I mange programmer kan der ofte være tale om, at en del af et program under bestemte forudsætninger skal udelades, så er det ikke længere et valg mellem to processer, men et valg om en proces skal udføres eller springes over. Dette ser således ud i et flowchart:



Betingelsen i spørgekassen skal altid udformes så det kun kan besvares med et JA eller et NEJ.

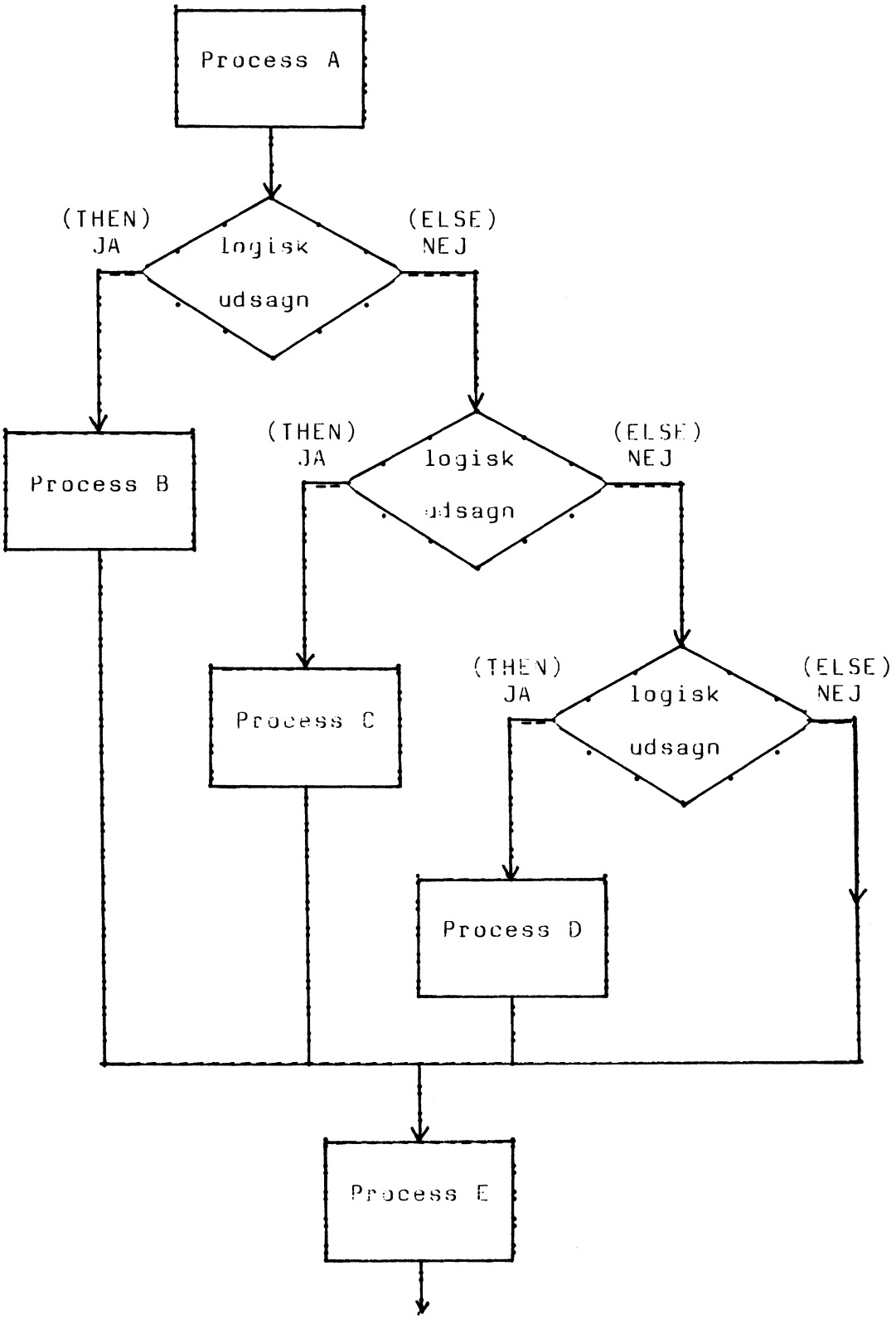
For at lette læseligheden, vælges venstre udgang altid som JA.

Ser man på udvælgelsen som en helhed vil man se, at den kan betragtes som en sekvens.

CASE-SÆTNING

Hvis der er behov for at udtrykke valg mellem flere muligheder, kan dette gøres ved at gentagen anvendelse af IF-THEN-ELSE.

Eks:

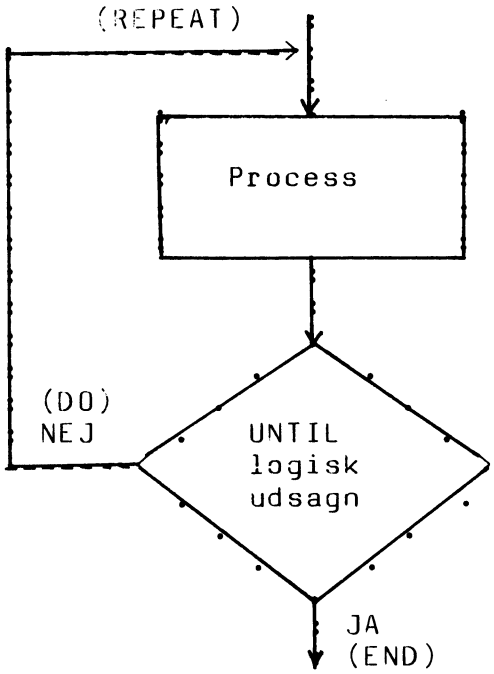


GENTAGELSE

Den tredie kontrolstruktur er gentagelsen der udfra et eller andet logisk udsagn giver betingelsen for gentagelsen. De to former for gentagelse, er for det første den hvor programmet altid skal behandles mindst een gang, det er REPEAT-UNTIL. Den anden form er hvor startbetingelsen skal være opfyldt for at der overhovedet foretages en gentagelse, det er WHILE-DO.

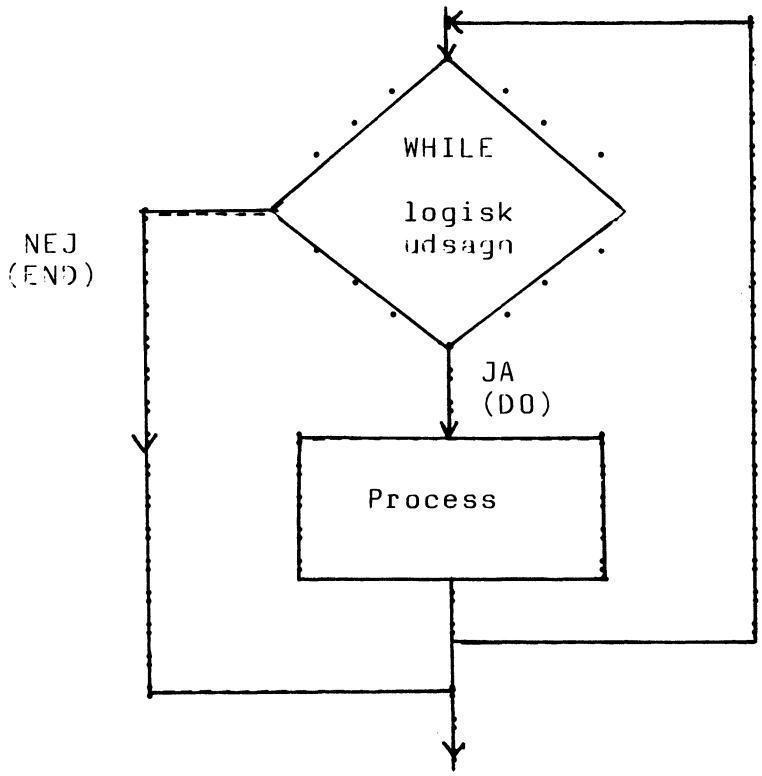
REPEAT-UNTIL

En repeat-until vil som flowchart således ud:



WHILE-DO

While-do, ser således ud i et flowchart:



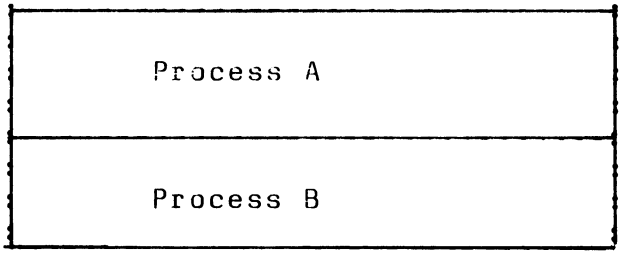
Når logikken i modulerne er opbygget ved hjælp af de her viste kontrolstrukturer bliver det lettere at læse programmet idet hver kontrolstruktur har kun een indgang og een udgang og således kan de allesammen betragtes som sekvenser.

NASSI-SCHNEIDERMAN STRUKTUGRAM

Nassi-Schneiderman struktugrammet, også kaldet "Struktugram" er især velegnet ved indlæring i at skrive sine programmer strukturerede.
Da den grundlæggende kontrolstruktur sikre, at det ikke kan laves ustruktureret.

SEKVENS

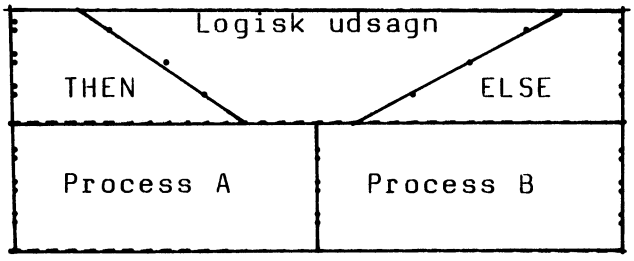
Et Nassi-Schneidermann struktugram med nogle sekvenser ser således ud:



Et program vil altid set udefra bestå af en række sekvenser der påbegyndes når programmet startes og afsluttes når programmmet stoppes.

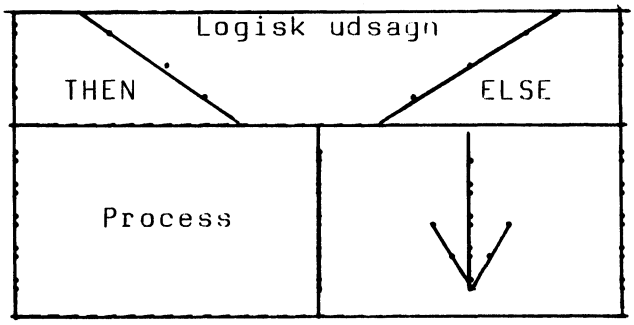
UDVÆLGELSE

I de her viste eksempler er det et spørgsmål om at vælge mellem to forskellige processer og dersom den ene ikke udføres skal den anden udføres. Dette ser således ud i et Nassi-Schneiderman strukturgram:



For at lette læseligheden vælges venstre søjle altid som THEN udgang og venstre søjle som ELSE udgang på det logiske udsagn:

I mange programmer kan der ofte være tale om at en del af et program under bestemte forudsætninger skal udelades. Så er det ikke længere et valg mellem to processer, men et valg om en proces skal udføres eller springes over.

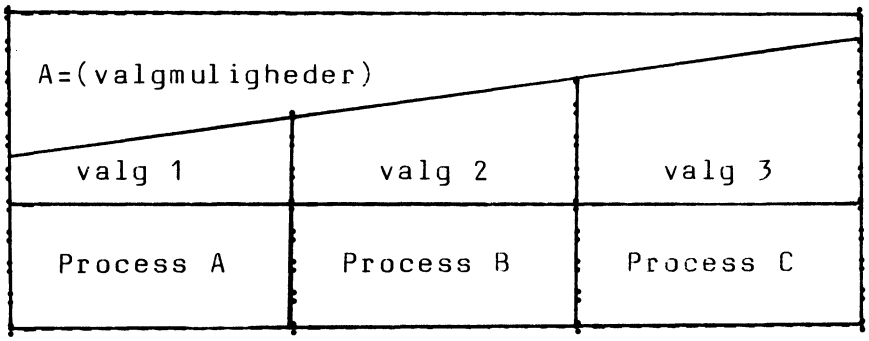


For at lette læseligheden vælges venstre søjle altid som THEN udgang på det logiske spørgsmål i spørgekassen. Og det er også den der indeholder processsen.

CASE-STRUKTUR

Hvis der er behov for flere valgmuligheder kan det også lade sig gøre ved hjælp af Nassi-Schneiderman struktugram.

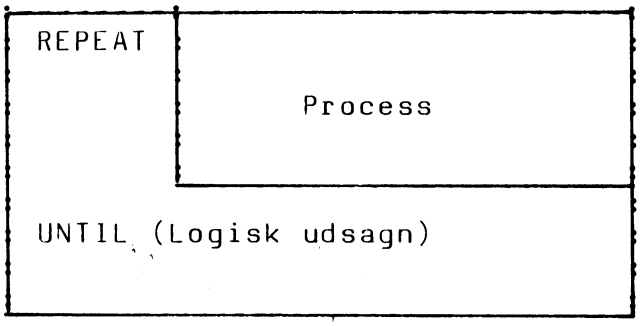
Eks:



GENTAGELSE

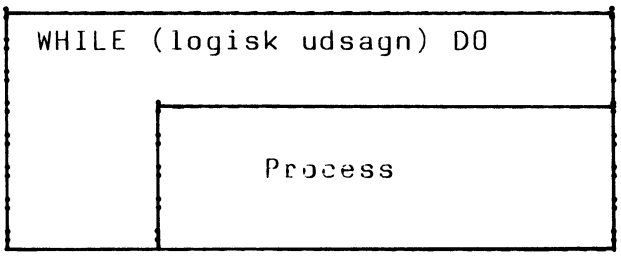
REPEAT-UNTIL

Der er to former for gentagelser, den ene medfører at processen altid skal behandles mindst een gang, det er REPEAT-UNTIL som ser således ud som Nassi Schneidermann struktugram:



WHILE-DO

Ved den anden form skal startbetingelsen være opfyldt for at der overhovedet foretages en gentagelse, dette kaldes WHILE-DO.



Når man bruger Nassi-Schneiderman struktugrammer er det vigtigt at starte ned at tegne bredt da alle kontrolstrukturer kan placeres i alle proceskasser og derved bliver proceskasserne hurtigt for smalle.

Fordele ved at bruge Nassi-Schneiderman struktugram er:

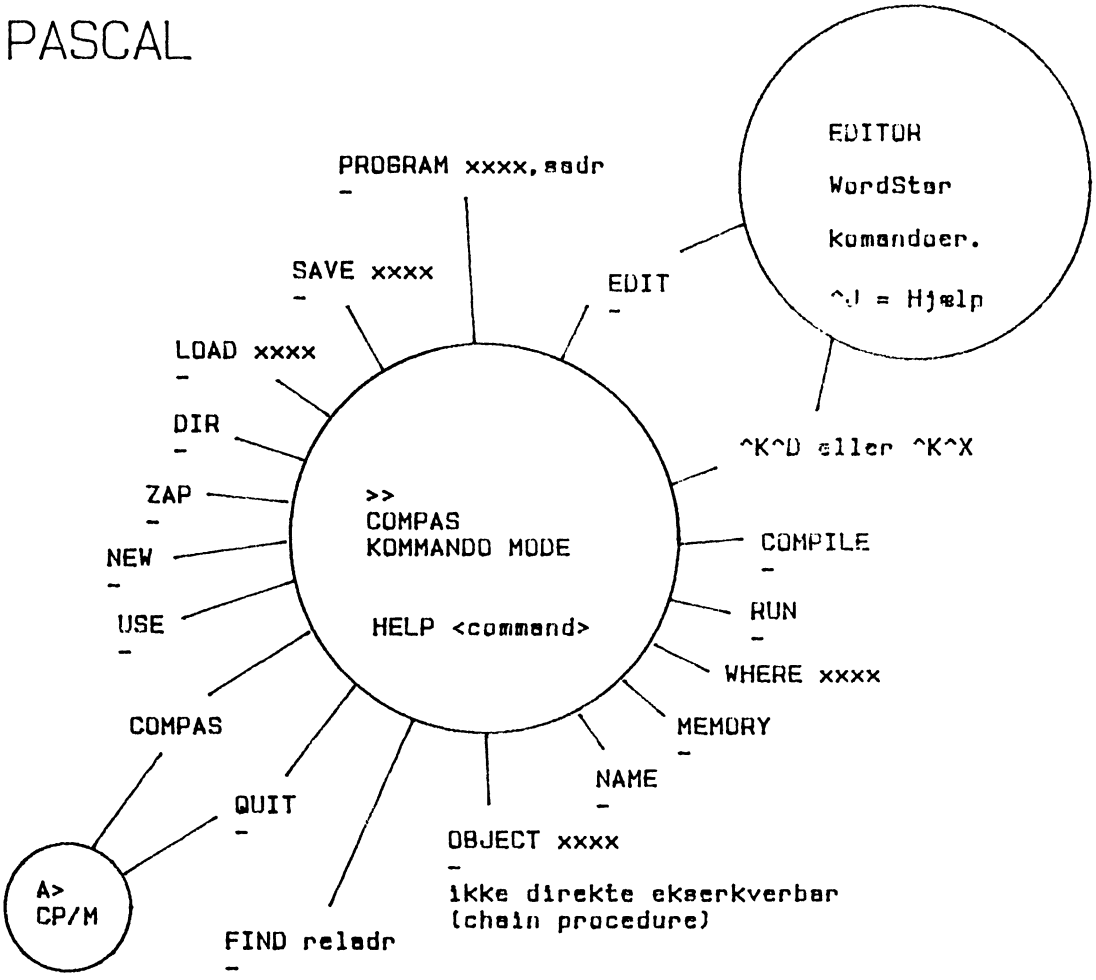
- God indlæring ved struktureret programmering.
- Afspejler programmets blokstruktur.
- Ingen muligheder for at bryde med tegnereglerne.
- Kan let tegnes uden brug af skabelon.

Konklusion

Når logikken i modulerne er opbygget ved hjælp af de her viste kontrolstrukturer uanset om man bruger PSEUDO KODER - STRUKTURERET FLOWCHART eller NASSI-SCHNEIDERMAN-STRUKTUGRAM bliver det lettere at læse programmet idet hver kontrolstruktur kun har een indgang og een udgang. Samtidig bliver programmet lettere at servicere da det er forholdsvis let at indføje ændringer uden at skulle skrive hele programmet om.

Brugsanvisning på COMPAS V. 3.3

COMPAS V. 3.3
POLY PASCAL



Alle kommandoer afsluttes med <return>.
Kommandoer kan forkortes til det understregede.

Program udvikling's eksempel

GÆT ET TAL

SKITSERING - PROBLEMFORMULERING

Programmet skal generere et tilfældigt tal, som spilleren skal gætte.
Spilleren skal indtaste intervallet af det tilfældige tal, samt max antal gæt.
Programmet skal fortælle spilleren om gættet er for højt eller for lavt.
Hvis spilleren gætter rigtig skal programmet gratulere med "HIP - HIP - HURRA" for gæt mindre end halvdelen af max antal gæt, ellers "HURRA - begynder". Hvis spilleren ikke gætter tallet inden max antal gæt vises det tilfældige tal.

SKITSERING - MODEL OG METODE VALG

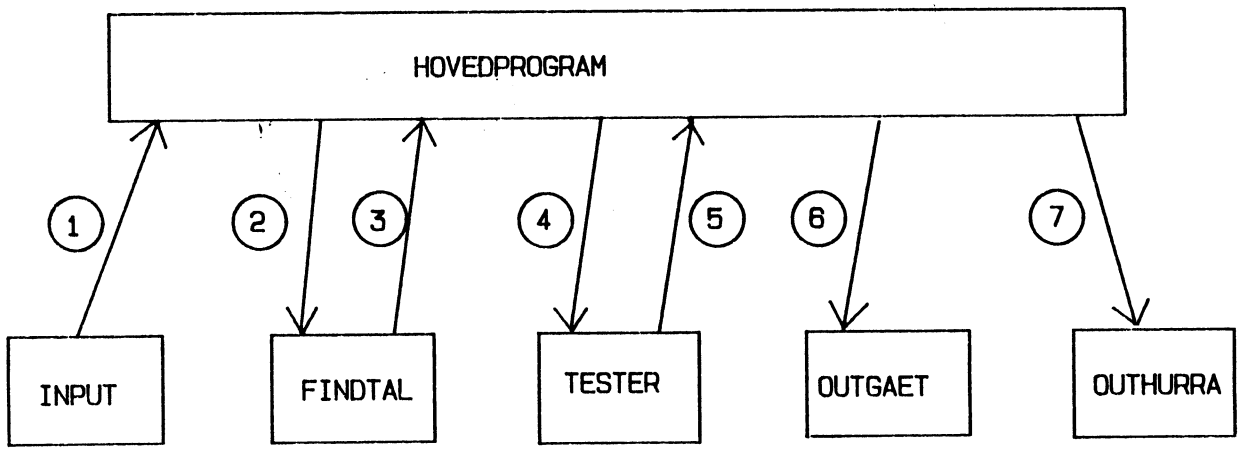
Programmet skal bestå af følgende moduler:

1. INPUT
   Henter nedre grænse tal, øvre grænse tal og max antal gæt samt udskriver ledetekster.
2. FINDTAL
   Modulet finder tilfældig tal mellem nedre og øvre grænse.
3. TESTER
   Modulet henter gæt og tester om gæt er for lavt, rigtig eller for højt.
4. OUTGAET
   Skriver på skærm om gæt er for højt eller for lavt samt antal gæt.
5. OUTHURRA
   Skriver på skærm hvis spiller finder det hemmelige tal "HIP - HIP - HURRA" eller "HURRA - begynder" og hvis spiller ikke gætter det hemmelige tal inden max antal gæt vises det hemmelige tal.

SKITSERING - PROBLEMBEHANDLING

HIPODIAGRAM

SKITSERING PROBLEMBEHANDLING



1. MINTAL
   MAXTAL
   MAXGAET

2. MINTAL
   MAXTAL

3. COMPUTAL

4. COMPUTAL

5. LAVT
   RIGTIG
   HOJT

6. LAVT
   HOJT
   ANTAL

7. RIGTIG
   MAXANTAL
   ANTAL
   COMPUTAL

SKITSERING - PROBLEMBEHANDLING fortsat

Input - Process - Output beskrivelse:

INPUT

INPUT: ingen

PROCESS: Hent interval af tal der skal gættes og max antal gæt, samt udskriv ledetekster.

OUTPUT: MINTAL,MAXTAL,MAXGAET

FINDTAL

INPUT: MINTAL,MAXTAL

PROCESS: Genererer et tilfældig tal mellem MINTAL og MAXTAL v.h.a. random funktion.

OUTPUT: COMPUTAL

TESTER

INPUT: COMPUTAL

PROCESS: Hent gæt og test om gæt er for lavt, rigtigt, eller for højt og aflever et sandt udsagn enten LAVT, RIGTIG eller HOJT.

OUTPUT: LAVT,RIGTIG,HOJT

OUTGAET

INPUT: LAVT,HOJT,ANTAL

PROCESS: Skriv på skærm om gæt er for højt eller for lavt afhængig om LAVT eller HOJT er sand, samt skriv ANTAL gæt.

OUTPUT: Ingen

OUTHURRA

INPUT: RIGTIG,MAXGAET,ANTAL,COMPUTAL

PROCESS: Hvis RIGTIG er sand skrives "HIP - HIP - HURRA" hvis antal gæt er mindre end halvdelen af max antal, ellers skrives "HURRA - begynder" Hvis RIGTIG ikke er sand vises det hemmelige tal.

OUTPUT: Ingen

SKITSERING - KONKLUSION

Det tilfældige tal er et heltal i intervallet 0 - 32767.
Max antal gæt skal være et heltal fra 1 til 32767.
Der testes ikke om intervallet er forkert.

```
DETAILERING - PROBLEMFOMULERING
-------------------------------

Hovedprogram sætter ANTAL = 0
Hovedprogram kalder INPUT
             kalder FINDTAL
Hovedprogram gentager
          kalder TESTER
          kalder OUTGAET
          ANTAL + 1
indtil ANTAL = MAXGAET eller RIGTIG = SAND
Hovedprogram kalder OUTHURRA

DETAILERING - MODEL OG METODE VALG
----------------------------------

Der vælges NASSI-SCHNIEDERMAN-STRUKTUGRAM

DETAILERING - PROBLEMBEHANDLING
-------------------------------
```

STRUKTUGRAMMER

Eks. på PSEUDO-KODER og STRUKTURERET FLOWCHART vises.

HOVEDPROGRAM

Gæt et tal

<table>
<tr><td colspan="2">Rens skærm og skriv init. besked</td></tr>
<tr><td colspan="2">Sæt ANTAL = 0</td></tr>
<tr><td colspan="2">Kald INPUT</td></tr>
<tr><td colspan="2">Kald FINDTAL</td></tr>
<tr><td>REPEAT</td><td>Sæt LAVT, RIGTIG og HOJT = FALSK</td></tr>
<tr><td></td><td>Kald TESTER</td></tr>
<tr><td></td><td>ANTAL + 1</td></tr>
<tr><td></td><td>Kald OUTGAET</td></tr>
<tr><td colspan="2">UNTIL RIGTIG = SAND eller ANTAL = MAXANTAL</td></tr>
<tr><td colspan="2">KALD OUTHURRA</td></tr>
</table>

DETAILERING - PROBLEMBEHANDLINGEN

STRUKTUGRAMMER fortsat

INPUT

SKRIV "Indtast interval af tal der skal gættes f.eks 10 20"
Hent MINTAL, MAXTAL
Skriv "Max antal gæt"
Hent MAXGAET

FINDTAL

REPEAT	COMPUTAL = TILFÆLDIG TAL
UNTIL COMPUTAL >= MINTAL OG COMPUTAL <= MAXTAL	

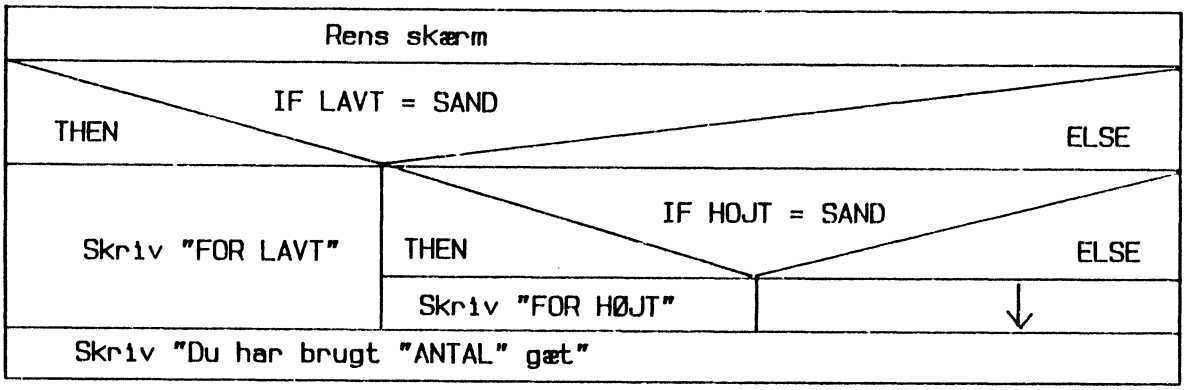
TESTER

Skriv "Indtast gæt"		
Hent GAET		
IF GAET < COMPUTAL		
THEN		ELSE
LAVT = SAND	IF GAET = COMPUTAL	
	THEN	ELSE
	RIGTIG = SAND	HOJT = SAND

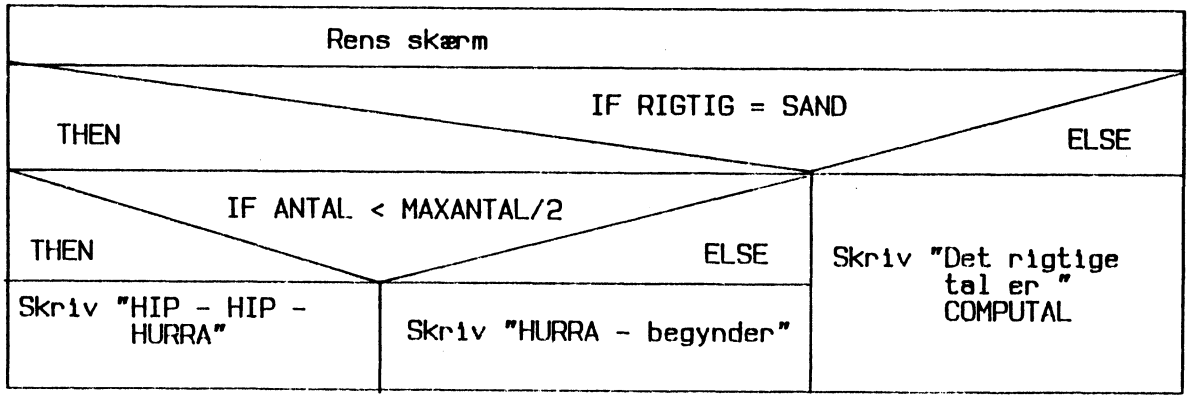
DETAILERING - PROBLEMBEHANDLINGEN

STRUKTUGRAMMER fortsat

OUTGAET



OUTHURRA



DETAILERING - PROBLEMBEHANDLINGEN

PSEUDO KODER

GÆT ET TAL

```
Begynd GÆT ET TAL (Hovedprogram)
  Rens skærm og skriv init. besked
  ANTAL = 0
  kald INPUT
  kald FINDTAL
  REPEAT
    kald TESTER
    kald OUTGAET
    ANTAL + 1
  UNTIL ANTAL = MAXGAET OR RIGTIG = SAND
  kald OUTHURRA
slut (Hovedprogram)

Begynd INPUT (underprogram)
  Skriv "Indtast interval af tal der skal gættes f.eks. 10 20"
  Hent MINTAL, MAXTAL
  Skriv "Max antal gæt"
  Hent MAXANTAL
slut INPUT

Begynd FINDTAL (underprogram)
  REPEAT
    COMPUTAL = TILFÆLDIG TAL
  UNTIL COMPUTAL >= MINTAL eller COMPUTAL <= MAXTAL
Slut FINDTAL

Begynd TESTER (underprogram)
  Skriv "Indtast gæt"
  Hent GAET
  IF GAET < COMPUTAL
  THEN
    LAVT = SAND
  ELSE
    IF GAET = COMPUTAL
    THEN
      RIGTIG = SAND
    ELSE
      HOJT = SAND
    END
  END
Slut TESTER
```

DETAILERING - PROBLEMBEHANDLINGEN

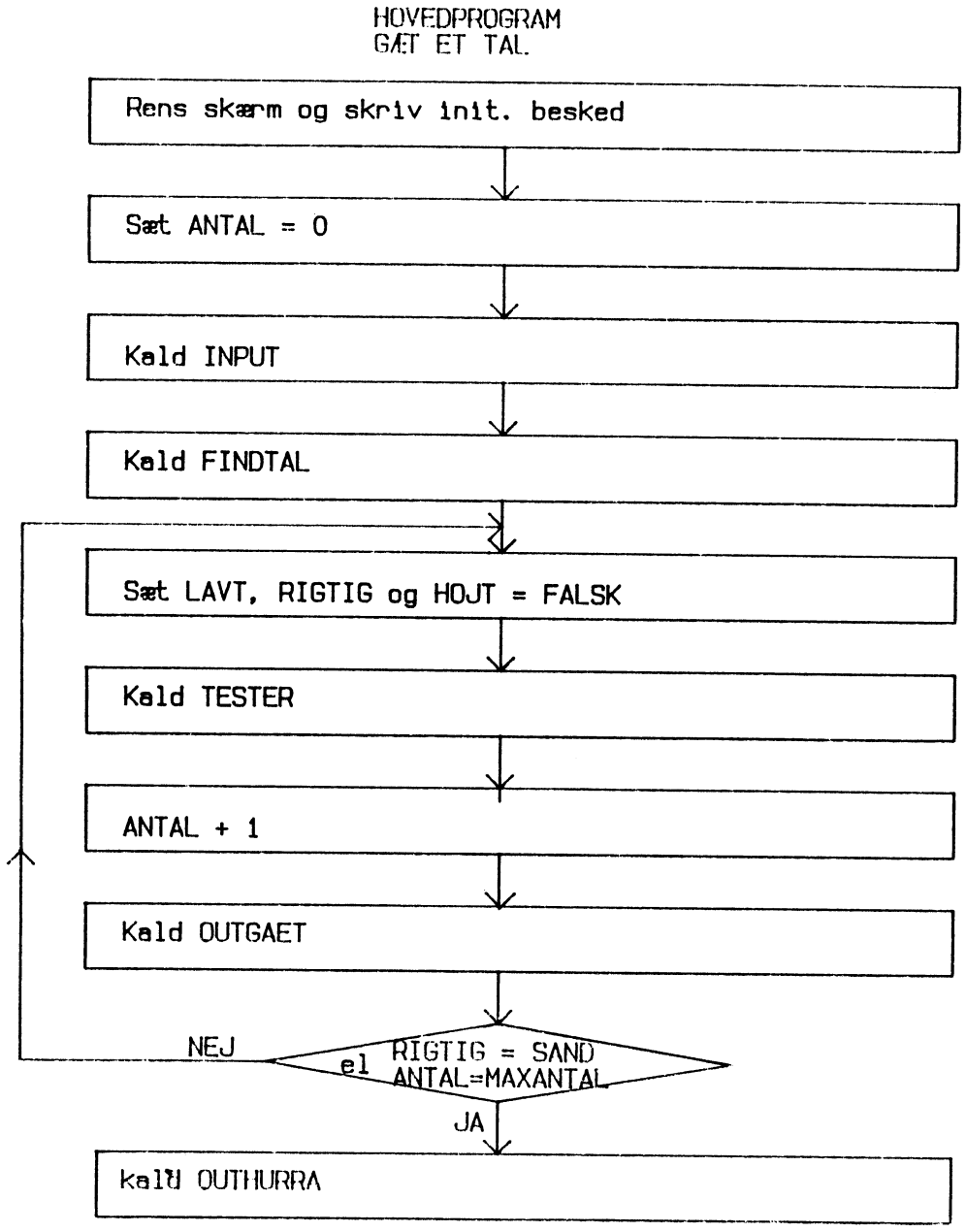
PSEUDO KODER fortsat

```
Begynd OUTGAET (underprogram)
  Rens Skærm
  IF LAVT = SAND
  THEN
    Skriv "FOR LAVT"
  ELSE
    IF HOJT = SAND
    THEN
      Skriv "FOR HØJT"
    END
  END
  Skriv "Du har brugt "ANTAL" gæt"
slut OUTGAET

Begynd OUTHURRA (underprogram)
  Rens skærm
  IF RIGTIG = SAND
  THEN
    IF ANTAL < MAXANTAL/2
    THEN
      Skriv "HIP - HIP - HURRA"
    ELSE
      Skriv "HURRA - begynder"
    END
  ELSE
    Skriv "Det rigtige tal er "COMPUTAL
  END
Slut OUTHURRA
```

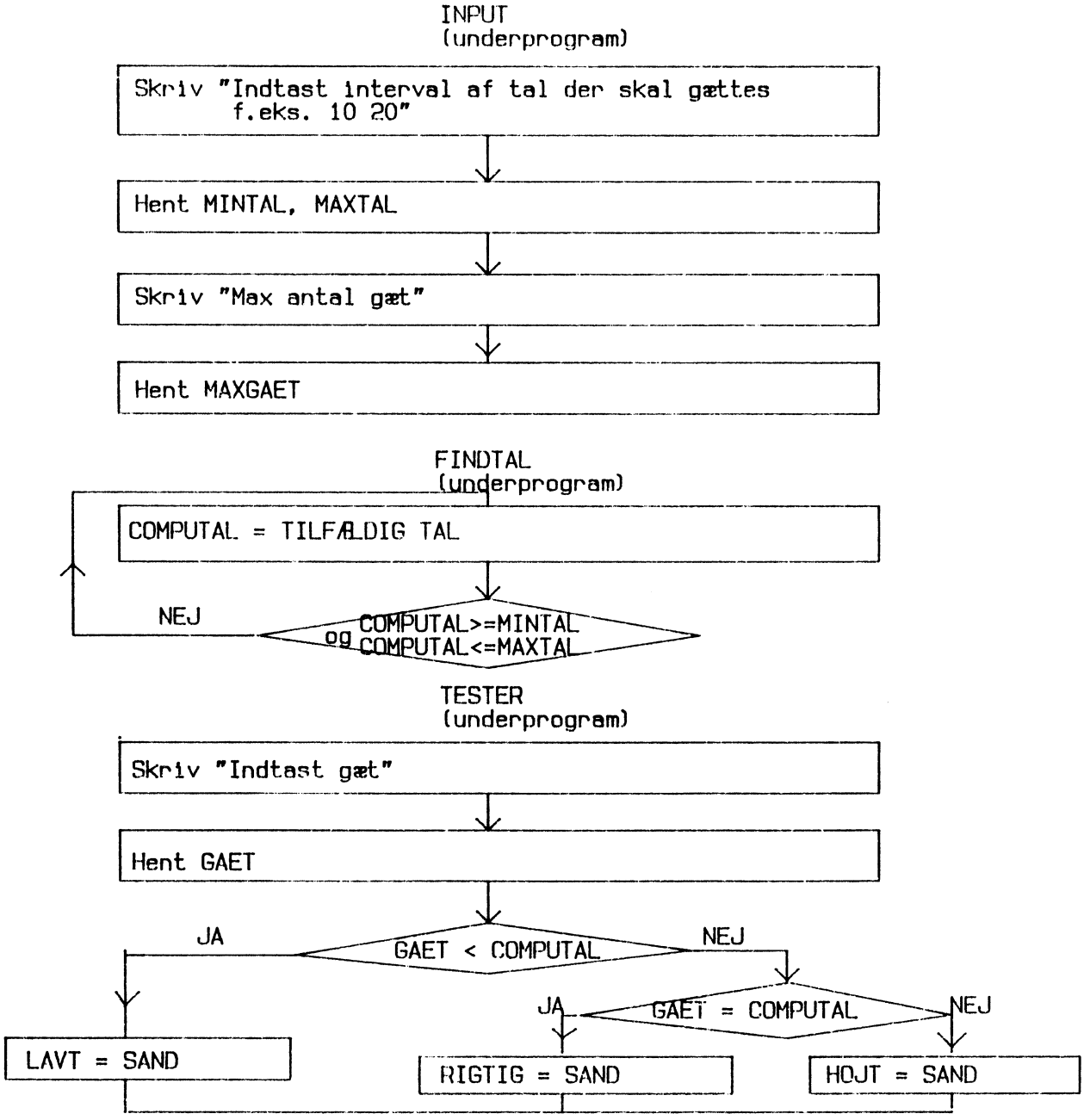
DETAILERING - PROBLEMBEHANDLINGEN

STRUKTURERET FLOWCHART



DETAILERING - PROBLEMBEHANDLINGEN

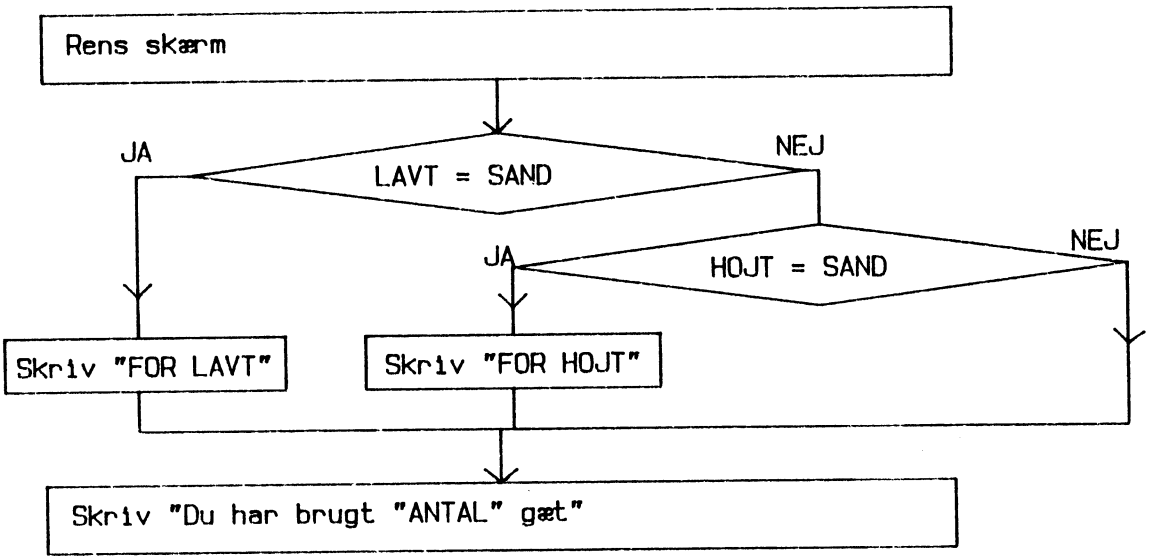
STRUKTUREREET FLOWCHART fortsat



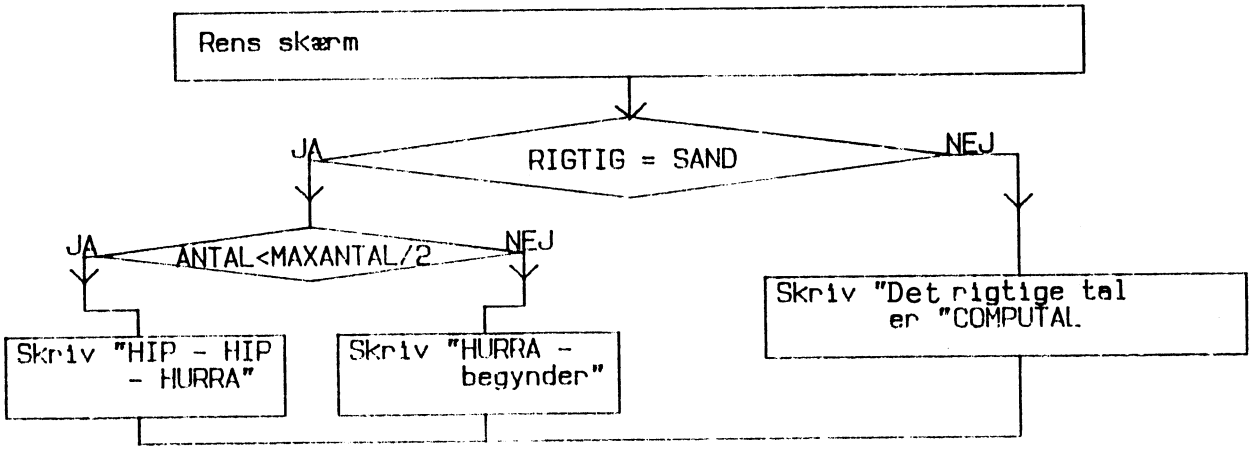
DETAILERING - PROBLEMBEHANDLINGEN

STRUKTURERET FLOWCHART fortsat

OUTGAET (underprogram)



OUTHURRA (underprogram)



DETAILERING KONKLUSION

Det skal kontrolleres at der er sammenhæng mellem Skitsering problemformulering og kontrolstrukturerne.

```
REALISERING PROBLEMFORMULERING
------------------------------

               OVERSIGT OVER KONSTANTER OG VARIABLE

INPUT:          ingen

FINDTAL:        ingen

TESTER:         VARIABEL: GAET

OUTGAET:        ingen

OUTHURRA:       ingen

HOVEDPROGRAM:   VARIABLE:  MINTAL, MAXTAL, MAXGAET, COMPUTAL,
                           LAVT, RIGTIG, HOJT, ANTAL

REALISERING MODEL OG METODE VALG
--------------------------------

Programmeringssproget PASCAL vælges.

REALISERING PROBLEMBEHANDLING
-----------------------------

PROGRAM GAETTAL;

VAR
  MINTAL,MAXTAL,MAXGAET,COMPUTAL:INTEGER;
  LAVT,RIGTIG,HOJT:BOOLEAN;
  ANTAL:INTEGER;


(* Underprogram INPUT henter interval af tal og max antal gæt, *)
(* samt udskriver ledetekster *)

  PROCEDURE INPUT (VAR MIN,MAX,MAXGAET:INTEGER);
  BEGIN
    GOTOXY (10,15);
    WRITE ('Indtast interval af tal der skal gættes f.eks. 10 20  ');
    READLN (MIN,MAX);
    GOTOXY (10,17);
    WRITE ('Max antal gæt ?  ');
    READLN (MAXGAET);
  END;


(* Underprogram FINDTAL genererer et tilfældig tal *)
(* mellem MINTAL og MAXTAL v.h.a. random funktion *)

    PROCEDURE FINDTAL (MINTAL,MAXTAL:INTEGER; VAR RANDOMTAL:INTEGER);
    BEGIN
      REPEAT
        RANDOMTAL:=RANDOM(MAXTAL)+1;
      UNTIL RANDOMTAL >= MINTAL;
    END;
```

```
(* Underprogram TESTER henter gæt og tester om gæt er for lavt *)
(* rigtig eller for højt og afleverer et sandt udsagn enten *)
(* LAVT, RIGTIG eller HOJT *)

    PROCEDURE TESTER (COMPUTAL:INTEGER; VAR LAVT,RIGTIG,HOJT:BOOLEAN);

    VAR
      GAET:INTEGER;

    BEGIN
      GOTOXY (10,20);
      WRITE ('Indtast gæt  ');
      READLN (GAET);
      IF GAET < COMPUTAL
      THEN
        LAVT:=TRUE
      ELSE BEGIN
        IF GAET = COMPUTAL
        THEN
          RIGTIG:=TRUE
        ELSE BEGIN
          IF GAET > COMPUTAL
          THEN
            HOJT:=TRUE;
        END;
      END;
    END;


(* Underprogram OUTGAET skriver på skærm om gæt er for lavt *)
(* eller for højt afhængig om LAVT eller HOJT er sand, *)
(* samt skriver antal gæt *)

    PROCEDURE OUTGAET (LAVT,HOJT:BOOLEAN; ANTAL:INTEGER);
    BEGIN
      WRITE (CLRHOM);
      GOTOXY (10,5);
      IF LAVT
      THEN
        WRITELN('FOR LAVT !')
      ELSE BEGIN
        IF HOJT
        THEN
          WRITELN('FOR HØJT !');
      END;
      GOTOXY (25,7);
      WRITELN('Du har brugt ',ANTAL,' gæt');
    END;
```

```
(* Underprogram OUTHURRA tester om RIGTIG er sand, hvis RIGTIG *)
(* er sand skrives "HIP - HIP - HURRA" hvis antal gæt er mindre *)
(* end halvdelen af max antal, ellers skrives "HURRA - begynder" *)
(* hvis RIGTIG ikke er sand vises det hemmelige tal *)

    PROCEDURE OUTHURRA (RIGTIG:BOOLEAN; MAXGAET,ANTAL,COMPUTAL:INTEGER);
    BEGIN
      GOTOXY (15,20);
      IF RIGTIG
      THEN BEGIN
        IF ANTAL < (MAXGAET/2)
        THEN
          WRITELN('HIP - HIP - HURRA')
        ELSE
          WRITELN('HURRA - begynder');
      END
      ELSE
        WRITELN ('Det rigtige tal er ',COMPUTAL ,' FARVEL !');
    END;

                        (* HOVEDPROGRAM *)
(* Hovedprogrammet skriver init. tekst og kalder INPUT *)
(* Derefter gentages Kald FINDTAL, kald TESTER, ANTAL + 1 *)
(* og kald OUTGAET indtil rigtig gæt eller ANTAL = MAXGAET *)
(* og til sidst kaldes OUTHURRA *)

BEGIN
  WRITE (CLRHOM);
  GOTOXY (10,8);
  WRITELN ('              GÆT ET TAL');
  WRITELN ('Du skal indtaste talinterval og antal gæt ');
  WRITELN ('hvorefter programmet genererer et tilfældig tal ');
  WRITELN ('i intervallet som du skal gætte ');
  GOTOXY (20,12);
  WRITELN ('GOD FORNØJELSE !');

  ANTAL:=0;      (* ANTAL GÆT SÆTTES TIL 0 *)

  INPUT (MINTAL,MAXTAL,MAXGAET);
  FINDTAL (MINTAL,MAXTAL,COMPUTAL);

  REPEAT
    LAVT:=FALSE;RIGTIG:=FALSE;HOJT:=FALSE;
    TESTER (COMPUTAL,LAVT,RIGTIG,HOJT);
    ANTAL:=ANTAL+1;
    OUTGAET (LAVT,HOJT,ANTAL);
  UNTIL RIGTIG OR (ANTAL = MAXGAET);

  OUTHURRA (RIGTIG,MAXGAET,ANTAL,COMPUTAL);
END.
```

REALISERING KONKLUSION

Husk kommentarer !

□