Interne notater

STATISTISK SENTRALBYRÅ

86/42

26. september 1986

HÅNDBOK I BRUK AV SAS

AV

LIV DAASVATN, SYSTEMKONTORET, OSLO

FORORD

Dette heftet er skrevet for å dekke et informasjonsbehov nye SAS-brukere i Byrået ofte har. De stiller gjerne spørsmålene:

- Hva er egentlig SAS?
- Hva kan SAS brukes til?
- Hvordan går en frem når en skal bruke SAS?

Det er en forutsetning at leseren er fortrolig med IBM-maskinen, dvs program-editoren og andre viktige punkter under ISPF. Det er også en forutsetning at man vet hva JCL er, og at man vet hvordan man skal få kjørt en jobb, få en utskrift styrt til printer osv.

Bemerk at denne håndboken er ufullstendig på svært mange områder. Den kan derfor på ingen måte erstatte de originale SAS-håndbøkene (SAS-manualene). Bruken av utenlandsk 'dataspråk' er forsøkt begrenset. Likevel er enkelte engelske ord og uttrykk som er vesentlige begrep i SAS beholdt (eksempler: statement, option). Dette for at notasjonen i de forskjelige brukerveiledningene i SAS skal være noenlunde ensartet.

Når dette skrives, kan SAS kjøres på begge Byråets IBM-maskiner, altså både i Oslo og på Kongsvinger. Dette er imidlertid kun kortvarig. For fremtiden vil SAS bare foreligge på Oslo-maskinen. (Dette vil ikke bety at brukere som sitter på Kongsvinger ikke kan bruke SAS!)

Annikken Seip og Sissel Solvin skrev i mai 1985 et internt hefte kalt 'Kurs i bruk av SAS'. Imidlertid er mye endret siden da. Det var derfor mer hensiktsmessig å skrive noe helt nytt, enn å oppdatere heftet til Seip og Solvin.

Takk til Kjell Stræte og Hanne Modahl som har kommet med verdifulle råd under skrivingen!
Ytterligere kommentarer/forslag fra lesere vil bli mottatt med takk.

INNHOLDSFORTEGNELSE

1. INNLEDNING

En interessert, som spør hva SAS er, kan få mange svar: 'SAS er en statistikk-pakke, en standard program pakke, et 4.generasjonsspråk, et programmeringsverktøy' og liknende. Hvis man i tillegg hører utsagn som 'Ring SAS, da vel!' kan forvirringen virke total.

Hva er så egentlig SAS?

* SAS er navnet på et meget omfattende programprodukt.
* SAS er også navnet på programmeringsspråket som dette produktet inneholder.
* Dertil er SAS navnet på firmaet som produserer det hele.

SAS er en forkortelse for Statistical Analysis System, som er et programprodukt laget for å tilrettelegge og analysere data. SAS produseres av firmaet SAS Institute Inc i USA. Produktet SAS inneholder et enkelt programmeringsspråk. Men produktet SAS inneholder også ferdigskrevne prosedyrer som utfører statistisk analyse (alt fra enkle gjennomsnittsberegninger og aggregater, til tidsserier, regresjonsanalyse og kompliserte modellsimuleringer). Når man skal kalle opp en slik ferdigskrevet prosedyre, bruker man SAS-språket. Dette språket er enkelt i bruk, så det er også lett å lage skreddersydde programmer for å tilfredsstille ens egne behov. På bakgrunn av dette, kan man forstå at SAS både kalles en statistikk-pakke, et 4.generasjonsspråk, et programmeringsverktøy osv.

Programproduktet SAS foreligger som 'pakker': Det er en basispakke med flere tilleggspakker som eventuelt kan kjøpes separat.

Av hensyn til effektivitet, må data som skal behandles i SAS foreligge på en spesiell form, som et såkalt SAS-datasett. Basispakken i SAS inneholder muligheter til å tilrettelegge data av ymse slag til slike SAS-datasett. Basispakken inneholder dessuten muligheter for å analysere dataene statistisk, med tabeller, plott osv.

Tilleggspakkene er tilpasset spesielle behov, som f.eks analyse av tidsserier, grafisk fremstilling i farger, skjermorientert håndtering

av data, regneark, beslutningsstøtteverktøy, kvalitetskontroll, interaktiv matrisebehandling med mer.
Basispakken benevnes i litteraturen 'baseSAS'. Under følger en oversikt over noen av de spesialpakkene som dessuten finnes. Byrået har de som er merket * .

*	SAS/ETS	Tidsserier, økonomi, modell-simulering
*	SAS/GRAPH	Farge-grafikk
*	SAS/FSP	Regneark, brevark, interaktiv datahåndtering
	SAS/IML	Interaktiv matrisehåndtering
	SAS/AF	Verktøy for å lage menystyrte applikasjoner
	SAS/OR	Beslutningsstøtteverktøy
	SAS/QC	Kvalitetskontroll
	SAS/REPLY-CICS	SAS-grafikk for CICS-brukere

Hver delpakke har sin egen brukerveiledning (manual, User's Guide). Brukerveiledningen har samme navn som delpakken, f.eks SAS/ETS User's Guide. Til basisdelen hører to manualer: SAS User's Guide: Basics og SAS User's Guide: Statistics.

2. VIKTIGE BEGREP I SAS

Det er 4 sentrale begrep i SAS:

* SAS-datasett
* SAS-statements
* DATA-steg
* PROC-steg

Alle data som skal behandles må foreligge som SAS-datasett. Et SAS-program kan deles i to slags deler, nemlig DATA-steget og PROC-steget. I DATA-steget bearbeides data og det lages SAS-datasett. I PROC-steget analyseres SAS-datasett. Et SAS-program kan bestå av ett eller flere DATA-steg og ett eller flere PROC-steg. DATA- og PROC-steg kan forekomme i en hvilken som helst rekkefølge. Selve programmeringen i SAS foregår med såkalte SAS-statements.

2.1 SAS-datasett

En samling data lagret på en datamaskin, kan kalles et datasett (begrepene datasett, datafil og fil blir brukt om hverandre). Det finnes flere typer datasett, f.eks brukerdatasett, rådatasett og SAS-datasett. Det som skiller disse tre datasett-typene er måten de er organisert på (lagret) i maskinen.
Brukerdatasett er organisert som PDS, partisjonerte datasett. Et PDS består av membere som man kan skrive på i editoren (pkt 2 i ISPF). Et SAS-program lagres på et slikt brukerdatasett-member. Data som SAS skal behandle, må imidlertid være lagret som SAS-datasett. Med hensyn til Byråets rådatamateriale, foreligger dette oftest organisert som sekvensielle datasett.

For at SAS skal kunne behandle data, må altså dataene først leses fra den type datasett de er lagret på (oftest en sekvensiell fil), inn i et SAS-datasett. Et SAS-datasett kan lagres permanent, eller det kan vare bare under en enkelt kjøring.
Det som på rådatafilen er records (poster), blir hetende observasjoner i SAS-datasettet. Til SAS-datasettet hører et sett variable. Til hver observasjon hører et sett variabelverdier.

Et eksempel:
Et SAS-datasett inneholder inntekt- og formues-opplysninger for et utvalg personer. Personene er entydig definert med et identnummer. Foruten inntekt og formue, er også kommunenummeret til personenes skattekommuner oppgitt. Hver av personene er representert med en observasjon i SAS-datasettet, og hver person (observasjon) har sitt sett av verdier for variablene identnr, inntekt, formue og kom_nr.

SAS-datasettet kan representeres slik:

	identnr	inntekt	formue	kom_nr
observasjon nr 1	1359	140000	86000	0103
observasjon nr 2	2433	78000	0	0102
observasjon nr 3	2610	210000	0	0103
osv	3002	115000	74000	0101

2.2 SAS-statements

I programmeringsspråket SAS kalles setningene SAS-statements. Et SAS-program består altså av SAS-statements. SAS-statementene begynner nesten alltid med et såkalt nøkkelord og avsluttes alltid med et semikolon. Noen SAS-statements kan kun brukes i DATA-steget, noen kan kun brukes i PROC-steget, og noen kan brukes begge steder. Eksempler på nøkkelord er DATA, PROC, INFILE, INPUT, DO, IF, KEEP, DROP og SELECT. Statementene har navn etter det nøkkelordet de starter med, f.eks INFILE-statementet, IF-statementet osv.

SAS-statementene har sin egen skrivemåte som man må følge. Men ellers er det ingen strenge regler for hvordan et SAS-program skal skrives. Følgende gjelder:

- SMÅ OG STORE BOKSTAVER KAN BRUKES OM HVERANDRE.
- STATEMENTENE KAN SKRIVES HVOR SOM HELST PÅ LINJEN.
- DET KAN SKRIVES FLERE STATEMENTS PÅ SAMME LINJE.
- DET TRENGS IKKE NOE FORTSETTELSESTEGN HVIS ET STATEMENT GÅR OVER FLERE LINJER.
- BLANKE KAN SKRIVES HVOR SOM HELST, UNNTATT INNI ORD.

Det er likevel på sin plass å oppfordre til god programmeringsskikk ved å følge disse råd:

- BRUK BLANKE LINJER FOR Å DELE INN PROGRAMMET !
- BRUK INNRYKK FOR Å GJØRE PROGRAMMET MER LESBART !
- SKRIV MANGE OG INFORMATIVE KOMMENTARER !

2.3 DATA-steg

I DATA-steget lages det SAS-datasett, og data bearbeides. DATA-steget starter alltid med et DATA-statement. Vær obs på forskjellen mellom et DATA-statement og et DATA-steg: Et DATA-steg består av en rekke state-

ments, hvorav det første alltid er et DATA-statement. Et DATA-steg består av alle statementene fra og med et DATA-statement inntil et nytt DATA- eller PROC-statement.

2.4 PROC-steg

SAS har diverse ferdige prosedyrer for å analysere data. Disse kalles opp i PROC-steget. Input til PROC-steget må være et SAS-datasett. Resultatet av analysen er output fra et PROC-steg. (Resultatet av analysen i PROC-steget kan også legges ut i et SAS-datasett.)
Et PROC-steg starter alltid med et PROC-statement. PROC-steget varer fra og med et PROC-statement inntil et nytt PROC- eller DATA-statement opptrer. Vær igjen klar over forskjellen mellom et PROC-steg og et PROC-statement.

2.5 SAS-navn

Navn på SAS-datasett og variable må følge disse reglene:

- MAKSIMAL LENGDE PÅ ET NAVN ER 8 TEGN.
- ET NAVN KAN KUN BESTÅ AV BOKSTAVER, TALL OG '_' (UNDERSTREK).
- DISSE BOKSTAVER ER IKKE LOV: Æ Ø Å æ ø å .
- NAVNET KAN IKKE BEGYNNE MED ET TALL.
- ET SAS-NØKKELORD KAN IKKE BRUKES SOM ET NAVN.

En må særlig merke seg at ingen andre spesialtegn enn '_' er lov. Et navn som inneholder en bindestrek (-) vil altså være ulovlig.

2. SAS-MANUALENE

Til basisdelen av SAS hører to brukerveiledninger (manualer). Det er SAS User's Guide: Basics og SAS User's Guide: Statistics. Disse manualene kan virke svært store og uhåndterlige, og det krever en viss

innsats å bli fortrolig med dem. Det er likevel vel verdt innsatsen! Manualene til delpakkene SAS/ETS, SAS/GRAPH osv. er av et langt mindre avskrekkende format. På innsiden av permen foran i manualene er det en nyttig oversikt over statements og PROC's. Bakerst i manualene er det et 'Reference Card' til å rive ut. Kortet er et sammendrag av alt en strengt tatt trenger å vite om den aktuelle delen av SAS.
Foruten disse User's Guides, finnes det en manual med eksempler: SAS Applications Guide, det finnes en manual for bruk av SAS under det operativsystem vi bruker: SAS Companion for OS and TSO, det finnes en SAS Programmers Guide, en Supplemental Library User's Guide og en mengde små Technical Reports om spesielle emner.

4. SASLOG og SASLIST

Hver gang en kjører en SAS-jobb vil det produseres en SASLOG. SASLOGen er en meget nyttig dokumentasjon av jobben. Den inneholder alle slags meldinger fra SAS, både eventuelle feilmeldinger med hensyn på syntaks og meldinger om antall observasjoner/variable på datasett som lages. Dersom en f.eks lager et program der det for en observasjon blir dividert med 0, vil det komme en advarsel om dette på SASLOGen.
SASLOGen kan dirigeres til en egen fil. I så fall oppgir en dette i et JCL-kort. Dersom JCLen ikke inneholder noen opplysning om hvor SASLOGen skal sendes, blir den automatisk lagt i kjørerapporten for jobben. Denne kan en se på i ISPF pkt 8 H .

De aller fleste SAS-PROC's gir resultater i form av output. Output fra et PROC-steg vil komme på SASLISTen. Som for SASLOG, gjelder også for SASLISTen at den styres til jobbens kjørerapport dersom ikke annet blir oppgitt i et JCL-kort. Siden en ofte vil ha SASLOGen og SASLISTen ut på papir, er det mest hensiktsmessig å styre disse til egne filer. Hvordan dette skal gjøres, står i dette heftes kapittel om JCL.

5. DATA-STEGET

I DATA-steget legges dataene tilrette for analyse. Det innebærer at en her kan beregne avledete variable, velge ut hvilke observasjoner og/eller variable som skal analyseres osv. DATA-steget er meget fleksibelt: med få og enkle statements kan en forme datamaterialet helt som en ønsker.
Et DATA-steg begynner alltid med et DATA-statement, mens et PROC-steg begynner med et PROC-statement. DATA-steget vil derfor være den delen av programmet som ligger mellom to DATA-statements eller ett DATA-statement og et PROC-statement.

5.1 Statements som kan brukes i et DATA-steg

Følgende statements kan brukes i DATA-steget:

ABORT	DROP	KEEP	OUTPUT
ARRAY	END	LABEL	PUT
assignment	ERROR	labels	RENAME
ATTRIB	FILE	LENGTH	RETAIN
BY	FORMAT	LINK	RETURN
CALL	GO TO	LIST	SELECT
CARDS	IF	LOSTCARD	SET
DATA	INFILE	MERGE	STOP
DELETE	INFORMAT	MISSING	sum
DO	INPUT	null	UPDATE

Av disse er det bare assignment-, sum-, labels- og null-statementene som ikke starter med et nøkkelord.

Under følger en nærmere beskrivelse av noen av de mest brukte statementene i DATA-steget. Det er viktig å være klar over at de statementene som ikke gjennomgås her også er nyttige å kunne! Les om dem i Basics-manualen.

Klammeparenteser betyr at det som står inni er valgfritt å ha med. Vil man ha med noe som er valgfritt, skal klammeparentesene selv ikke være med. Dersom noe står i vanlige parenteser, skal derimot parentesene selv også skrives.

5.2 Statements brukt ved innlesing/utskriving

Hvordan får en hentet dataene sine inn til et DATA-steg?

Dersom det er snakk om rådata lagret som en sekvensiell fil på disk eller tape, brukes INFILE-statementet for først å få lest dataene inn i et SAS-datasett. Foreligger dataene allerede som SAS-datasett, brukes som oftest SET-eller MERGE-statementet. Man kan også taste inn rådata i selve programmet, da aller sist i DATA-steget. I det tilfellet brukes CARDS-statementet.

5.2.1 DATA-statementet

DATA-statementets oppgave er å produsere SAS-datasett, samt å si 'Her starter et DATA-steg'. Den generelle formen på DATA-statementet er:

```
DATA  [ SAS-datasettnavn [ (dsoptions) ] ] ;
```

I sin enkleste form kan DATA-statementet kun bestå av nøkkelordet DATA etterfulgt av et semikolon. Dataset-options (dsoptions) er omtalt under avsnitt 8.1.1. Dersom man ikke oppgir noe datasett-navn, vil SAS produsere datasett med navn data1, data2, data3 osv. Det er lov å lage mange datasett i ett DATA-statement. Skriver man f.eks

```
DATA nordland troms finnmark ;
    SET n_norge ;
    IF fylke = 18 THEN OUTPUT nordland ;
    IF fylke = 19 THEN OUTPUT troms ;
    IF fylke = 20 THEN OUTPUT finnmark ;
```

vil det produseres 3 SAS-datasett med navn hhv 'nordland', 'troms' og 'finnmark'.

Navn på datasett kan bestå av ett eller to ledd. I eksempelet over er navnene ett-leddet (nordland troms finnmark). Det betyr at datasettene som produseres er midlertidige, dvs de varer bare så lenge SAS-kjøringen varer. Dersom man vil lage et SAS-datasett som er permanent,

må datasettets navn være to-leddet, f.eks 'lager.romsdal'. De to leddene adskilles med et punktum. Et datasett med to-leddet navn vil bli lagret på en fil, og man vil kunne lese det i en senere kjøring. Filen som SAS-datasettet skal lagres på, må være opprettet på forhånd (se oppskrift i kap.7). Det første leddet i navnet til et permanent SAS-datasett henviser til den filen datasettet skal lagres eller er lagret på.

Eksempel:

```
//
//LAGER   DD  DSN=TS61.S6911.LDA.SASFIL,DISP=OLD
//
//SYSIN   DD  *

DATA  lager.nordland  lager.troms  lager.finnmark ;
    SET lager.n_norge ;
    IF  .....    osv.
```

I eksempelet over er første ledd i datasett-navnene 'lager'. Vi ser at dette er det samme som DD-navnet til filen 'TS61.S6911.LDA.SASFIL'. Det betyr at denne filen er lagringsfil for SAS-datasettene nordland, troms og finnmark. Første ledd i et to-leddet datasett-navn knytter altså forbindelsen mellom SAS-programmet og filen som datasettene skal lagres på. Navnet på dette leddet (DD-navnet) kan velges fritt, men det må stemme overens i JCL'en og i SAS-programmet. (DD-navnet kan bestå av maksimalt 8 bokstaver/tall, første tegn må være bokstav.)

5.2.2 INFILE-statementet

Dersom man skal lese data fra en rådatafil inn i et SAS-datasett, må man bruke INFILE-statementet. Den generelle formen på dette statementet er:

```
INFILE  navn ;
```

'navn' er DD-navnet til filen med rådata. DD-navnet er det som står

rett ut for '//' i DD-kortet som gjelder rådata-filen. Dette navnet er det en selv som velger.

Eksempel:

```
//
//INN    DD  DSN=P681.S6613.I453A1A1.G8500.V00,DISP=SHR
//
//SYSIN  DD  *

DATA nytt ;
    INFILE inn ;
    INPUT fylke 1-4 naering 9-12 tilskudd 24-30 ;
```

Når en første gang leser rådata inn til et SAS-datasett, må en foruten INFILE-statementet også ha med et INPUT-statement som sier hvilke felt (=variable) som skal leses fra recorden (posten), og hva disse variablene skal hete.

5.2.3 CARDS-statementet

CARDS-statementet brukes når input-dataene skal tastes direkte inn i DATA-steget, altså ikke leses fra en rådatafil eller et eksisterende SAS-datasett.
CARDS-statementet skrives bare slik:

```
CARDS ;
```

Statementet må stå på slutten av programmet, kun etterfulgt av selve datalinjene og et semikolon på aller siste linje. I siste versjon av SAS er det kommet to nye statements som kan brukes på nøyaktig samme måte som CARDS-statementet. Disse heter LINES hhv DATALINES. Dette for at statementene skal være mer selvforklarende. CARDS-/LINES-/DATA-LINES-statementet brukes i praksis (i Byrået) kun når en skal øve seg i SAS, evt. teste hvordan et program virker på noen data man finner på selv.

5.2.4 INPUT-statementet

Når man leser rådata inn til et SAS-datasett, må man foruten et INFILE- eller CARDS-statement også ha med et INPUT-statement som definerer og lokaliserer variablene som skal leses inn. Den generelle formen på INPUT-statementet er:

```
INPUT variabelnavnliste ;
```

Variablene kan av type være numeriske eller tegnvariable. Når tegnvariable skal leses inn, må det stå en 'Å' etter variabelnavnet. I SAS-manualen står dette angitt med en '$'. Overalt der det står en '$' i manualen, må dette erstattes av en 'Å'. (Dette er spesielt for oss.)

Variablene som skal leses inn (input'en) kan beskrives på flere måter:

Hvis man i INPUT-statementet bare lister opp variabelnavnene fortløpende, kalles det 'listeinput'. (Tegnvariable må også ha en 'Å' etter seg.) Ved 'listeinput' er det en forutsetning at det er minst 1 blank som adskiller variabel-verdiene. Dersom en variabel ikke har noen verdi (er missing), må dette være angitt på recorden med f.eks en '.' (ellers vil neste variabels verdi bli lagt i den variabelen som egentlig er missing).

En annen form for input er 'kolonneinput'. Da må man foruten å angi om variabelen er en tegnvariabel, også oppgi dens posisjon (feltet) på recorden.

Den tredje og mest detaljerte form for å beskrive input, er 'formattert input'. Da oppgir man variabelens navn, etterfulgt av et innlesningsformat. Formater er omtalt i et eget kapittel i Basics-manualen (Kap.15). NB! Alle innlesningsformat inneholder et punktum, som regel som siste tegn.

De forskjellige former for input kan blandes i ett og samme INPUT-statement. En kan også lese samme felt eller deler av et felt, flere ganger. For å navigere lesehodet frem og tilbake, evt over flere linjer, brukes spesielle tegn. OBS! Vi bruker andre tegn enn de som

står i manualen! Under følger en liste over de tegnene vi bruker for å navigere. Tegnene som er oppgitt i manualen står i parentes.

Øn	(@n)	flytter lesehodet til posisjon n på linjen
+n		flytter lesehodet n plasser mot høyre
/		flytter lesehodet til posisjon 1 på neste linje
Æn	(#n)	flytter lesehodet til posisjon 1 på linje n

Gitt følgende datarecord, med posisjon angitt øverst:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
P	E	D	E	R	S	E	N		3	4		1	7	9			6	1	.	5		

Denne recorden kan leses med listeinput, kolonneinput eller formattert input:

liste: INPUT navn Å alder lengde vekt ;

kolonne: INPUT navn Å 1-8 alder 10-11 lengde 13-15 vekt 18-21 ;

formattert: INPUT navn Å8. Ø10 alder 2. +1 lengde 3.2 Ø18 vekt 4. ;

Vær klar over at listeinput sjelden kan brukes, da data oftest ligger fortløpende på recorden!
I eksempelet over er lengden oppgitt i cm på recorden. Med rent kolonneinput leses lengden inn som 179, altså i cm. Med formattert input kan man få lengden i m direkte, med formatet 3.2 . Det første tallet angir antall posisjoner som skal leses, mens det siste tallet angir antall desimaler. Siden input-typene kan blandes, kan en lese inn lengden med både kolonneinput og format, og på den måten også få lengden i meter:

```
INPUT ....     lengde 13-15 .2      ;
```

Dersom mange variable med samme format ligger etter hverandre på recorden, kan de leses inn på følgende måte:

```
INPUT ....  (pris1 pris2 pris3 pris4) (6.)  ;
```

Det finnes altså mange måter å skrive et INPUT-statement på, som vil gi samme resultat. Les i Basics-manualen om INPUT-statementet og kapitlet om formater, det er viktig!

5.2.5 SET-statementet

SET-statementet brukes for å lese allerede eksisterende SAS-datasett inn i et DATA-steg. Statementet kan brukes til å kopiere, sette etter hverandre, eller flette eksisterende SAS-datasett. Dette statementet brukes svært ofte. Statementet skrives slik:

```
SET [ SAS-datasettnavn [ (dsoptions) ] ] ;
```

Vil man kopiere et datasett, kan SET-statementet brukes slik:

```
DATA kopi ;
    SET gammelt ;
```

Det eksisterende datasettet 'gammelt' blir her kopiert, og kopien lagt i et datasett med navn 'kopi'. Antall observasjoner og antall variable vil da i utgangspunktet være nøyaktig det samme i de to datasettene. Dersom man ikke oppgir noe datasettnavn i SET-statementet, vil det forrige SAS-datasett produsert i samme jobb bli kopiert.
I praksis er det vanlig at man har et permanent datasett som man vil arbeide videre med. Statementene vil da kunne se slik ut:

```
DATA smaavilt;
    SET lager.smaavilt;
```

Vil man legge flere eksisterende SAS-datasett etter hverandre i ett nytt datasett, brukes SET-statementet slik:

```
DATA storvilt;
    SET elg hjort rein raadyr;
```

Antall observasjoner i 'storvilt' vil være summen av antall observasjoner i 'elg', 'hjort', 'rein' og 'raadyr'.

Dersom datasettene som listes opp i SET-statementet inneholder forskjellige variable (mer korrekt: forskjellig navn på variablene), vil det nye datasettet inneholde samtlige variable:

```
DATA begge ;
    SET dta1 dta2;
```

DTA1:

navn	alder
kari	39
nils	37

DTA2:

navn	adresse
per	torvgaten
guri	storgaten

BEGGE:

navn	alder	adresse
kari	39	
nils	37	
per	.	torvgaten
guri	.	storgaten

Datasettet 'dta1' inneholder ikke variabelen 'adresse'. De observasjonene som stammer fra datasettet 'dta1' vil derfor ha missing verdier for 'adresse' i datasettet 'begge'. Tilsvarende for observasjonene fra 'dta2' med hensyn på variabelen 'alder'.

Hvis en vil flette flere datasett, må en foruten SET-statementet også ha med et BY-statement. I BY-statementet oppgis den variabel datasettene skal flettes sammen etter. Det er en forutsetning at hvert datasett på forhånd er sortert på denne variabelen (f.eks ved hjelp av PROC SORT).

Eksempel:

```
DATA ansatte;
    SET avd1 avd2 avd3 avd4;
    BY navn;
```

5.2.6 MERGE-statementet

Dersom en vil lage et nytt datasett ved å koble observasjoner fra flere datasett sammen til en observasjon i det nye datsettet, brukes MERGE-statementet. Dette statementet brukes også svært ofte. MERGE-statementet kan brukes sammen med et BY-statement, da får en matching på BY-variabelen.

Eksempel på MERGE uten BY:

```
DATA spleis;
    MERGE bil eier;
```

BIL:

merke	aar
honda	86
opel	83

EIER:

navn	adresse
ellen	bomveien 3
bente	snarveien 5

SPLEIS:

merke	aar	navn	adresse
honda	86	ellen	bomveien 3
opel	83	bente	snarveien 5

Hvis man bruker 'MERGE uten BY' som vist ovenfor, må man være helt sikker på at første observasjon på datasettet 'bil' virkelig hører sammen med første observasjon på datasettet 'eier' osv. Eksempelet på neste side viser hva som skjer hvis dette ikke er tilfellet. Man må også være obs på variabelnavnene:

```
DATA spleis;
    MERGE bil eier;
```

BIL:

navn	merke	aar
cato	datsun	79
anne	honda	86
ellen	rover	81
frank	opel	83

EIER:

navn	adresse
cato	mogaten 9
bente	snarveien 5
ellen	bomveien 3

SPLEIS:

	navn	merke	aar	adresse
1	cato	datsun	79	mogaten 9
2	bente	honda	86	snarveien 5
3	ellen	rover	81	bomveien 3
4	frank	opel	83	

Vi ser at i observasjon nr 2 er navnet 'anne' endret til 'bente'. Dette skjer fordi variabelen 'navn' finnes i begge datasettene og det 'merges uten by'. Variabelen i det datasettet som nevnes sist i MERGE-statementet vil da bli den gjeldende. Siden antall observasjoner i de to datasettene er forskjellig, får observasjon nr 4 missing verdi for adresse.

Dersom en skal koble observasjoner som hører sammen, bør dette derfor gjøres ved å koble observasjonene etter en match-variabel. En bruker da 'MERGE med BY'. I dette tilfellet må da selvsagt match-variabelen finnes på begge datasett.

Eksempel på MERGE med BY:

```
DATA spleis;
    MERGE bil eier;
    BY navn;
```

BIL:

navn	merke	aar
anne	honda	86
cato	datsun	79
ellen	rover	81
frank	opel	83

EIER:

navn	adresse
bente	snarveien 5
cato	mogaten 9
ellen	bomveien 3

SPLEIS:

	navn	merke	aar	adresse
1	anne	honda	86	
2	bente			snarveien 5
3	cato	datsun	79	mogaten 9
4	ellen	rover	81	bomveien 3
5	frank	opel	83	

En må her merke seg at begge datasettene må være sortert på BY-variabelen før de kan merges sammen.

5.2.7 UPDATE-statementet

UPDATE-statementet er en spesialvariant av MERGE-statementet. Det brukes dersom en har et datasett med opplysninger som skal oppdateres. Rettelsene må da også foreligge på et SAS-datasett, et transaksjonsdatasett. Statementene skrives slik:

```
DATA  masterdatasettnavn ;
    UPDATE  masterdatasettnavn  transaksjonsdatasettnavn ;
    BY variabelnavn ;
```

Ved hjelp av UPDATE-statementet kan eksisterende variable oppdateres, nye variable kan legges til observasjonene og nye observasjoner kan legges til datasettet.

Eksempel:

```
DATA  master ;
    UPDATE  master  trans ;
    BY  persnr ;
```

MASTER:

	persnr	navn	alder	bosted
1	10001	mette	43	bergen
2	10002	kari	39	oslo
3	10003	liv	32	finse
4	10006	knut	23	drammen

TRANS:

	persnr	navn	alder	bosted
1	10002		33	
2	10002			narvik
3	10005	marit	54	skien

MASTER:

	persnr	navn	alder	bosted
1	10001	mette	43	bergen
2	10002	kari	33	narvik
3	10003	liv	32	finse
4	10005	marit	54	skien
5	10006	knut	23	drammen

Både masterdatasett og transaksjonsdatasett må være sortert på identifikasjonsvariabelen.

5.2.8 OUTPUT-statementet

OUTPUT-statementet har følgende form:

```
OUTPUT  [ SAS-datasettnavn ] ;
```

I et DATA-steg er det vanligvis automatisk output til det datasett man oppgir navnet på i DATA-statementet. I noen tilfelle er det imidlertid nødvendig med et eget OUTPUT-statement:

* hvis det skal lages 2 eller flere datasett
* hvis det skal lages flere observasjoner av hver observasjon som leses inn
* hvis det skal lages 1 observasjon av flere observasjoner som leses inn fra ett datasett

Eksempel på å lage flere datasett:

```
DATA nordland troms finnmark;
    SET n_norge;
    IF fylke = 18 THEN OUTPUT nordland ;
    IF fylke = 19 THEN OUTPUT troms ;
    IF fylke = 20 THEN OUTPUT finnmark ;
```

Eksempel på å lage flere observasjoner av hver observasjon som leses:

```
DATA students;
    INPUT fakultet $ oslo bergen tromso ;
    DROP oslo - - tromso ;
        antall = oslo ;
    OUTPUT;
        antall = bergen ;
    OUTPUT;
        antall = tromso ;
    OUTPUT;
    CARDS;
       matnat   2359  1280   973
       histfil  5721  3814  1092
       medisin   625     .   306
            .......
            .......        osv.
        ;
```

I dette eksempelet består hver observasjon som leses av 4 variable: 'fakultet', 'oslo', 'bergen' og 'tromso'. Variablene inneholder antall studenter ved de forskjellige fakultet i de tre byene.

Ønsket er å lage 3 observasjoner ut fra hver observasjon som leses: en som inneholder verdien i 'oslo', en som inneholder verdien i 'bergen' og en som inneholder verdien i 'tromso'. Det nye datasettet som lages skal bare inneholde to variable: 'fakultet' og 'antall'. Variablene 'oslo', 'bergen' og 'tromso' skal altså sløyfes i ut-datasettet. Dette gjøres med DROP-statementet. De tre variablene ligger etter hverandre på SAS-datasettet (rekkefølgen bestemmes i INPUT-statementet). For å utelate alle tre variablene er det da tilstrekkelig bare å skrive navnene til første og siste variabel, med to bindestreker mellom.

Den nye variabelen 'antall' tilordnes så i tur og orden verdiene i 'oslo', 'bergen' og 'tromso', men legges ut på datasettet med et OUTPUT-statement mellom hver tilordning. (Tilordningen skjer ved assignment-statementet.) Datasettet 'students' vil inneholde 3 ganger så mange observasjoner som det antall som leses inn.

Eksempel på å lage 1 ut-observasjon av flere inn-observasjoner:

```
PROC SORT DATA=sjekker;
    BY kontonr;

DATA sumsjekk ;
    SET sjekker;
    BY kontonr;
    IF FIRST.kontonr THEN total = 0;
        total = total + kroner ;
    DROP kroner;
    IF LAST.kontonr THEN OUTPUT;
```

SJEKKER:

kontonr	kroner
29542	3000
32607	500
16233	2000
32607	1500
16233	500
32607	4000
16233	1500
21570	5000
16233	2500
29542	400

SJEKKER etter sortering:

kontonr	FIRST.kontonr	LAST.kontonr	kroner
16233	sann	ikke sann	2000
16233	ikke sann	ikke sann	500
16233	ikke sann	ikke sann	1500
16233	ikke sann	sann	2500
21570	sann	sann	5000
29542	sann	ikke sann	3000
29542	ikke sann	sann	400
32607	sann	ikke sann	500
32607	ikke sann	ikke sann	1500
32607	ikke sann	sann	4000

SUMSJEKK:

kontonr	total
16233	6500
21570	5000
29542	3400
32607	6000

I dette eksempelet brukes mange nyttige statements. Utgangspunktet er et SAS-datasett 'sjekker', som inneholder alle sjekker registrert i en bank over en periode. 'Sjekker' inneholder variabelen 'kontonr', og pålydende beløp i variabelen 'kroner'. Ønsket er å summere alle sjekkene som har samme kontonr, og lage et datasett 'sumsjekk' som har 1 observasjon for hvert kontonr. Denne observasjonen skal ha en variabel 'total' som inneholder det summerte beløp.

Først må datasettet 'sjekker' sorteres på kontonr. Alle sjekker med samme kontonr vil da bli lagt etter hverandre. Når et datasett er sor-

tert på en variabel, lager SAS automatisk to temporære, Boolske størrelser i tilknytning til den sorterte variabelen. Disse størrelsene sier noe om når sorteringsvariabelen skifter verdi. Størrelsene betegnes 'FIRST.variabelnavn' og 'LAST.variabelnavn'. At størrelsene er Boolske, vil si at de har verdien 'sann' eller 'ikke sann'. 'FIRST.kontonr' er 'sann' for den første observasjonen som har et bestemt kontonr. (Husk at observasjonene nå ligger gruppevis sortert på kontonr.) I den siste observasjonen som har dette kontonummeret er 'LAST.kontonr' 'sann'. Alle andre tilfelle av hhv 'FIRST. ...' og 'LAST. ...' vil være 'ikke sann'.

I en IF-THEN-konstruksjon vurderes uttrykket som står mellom 'IF' og 'THEN'. Hvis dette uttrykket er sant, så utføres det som står etter 'THEN'. Hvis det ikke er sant, går SAS videre til neste statement. I eksempelet over vil betingelsen 'FIRST.kontonr' være sann for den observasjonen som ligger først av flere observasjoner med samme kontonr. Variabelen 'total' settes da til 0 i et assignment-statement. Deretter legges variabelen 'kroner' til 'total' i et sum-statement. I det aller siste statementet sjekkes om observasjonen er den siste med samme verdi for kontonr. Dersom dette ikke er tilfellet, behandles neste observasjon og variabelen 'total' vokser med verdien i 'kroner'. Dette gjentas helt til 'LAST.kontonr' er sann, dvs når siste observasjon med et bestemt kontonr behandles. Da først legges observasjonen ut på datasettet 'sumsjekk'.

5.2.9 PUT-statementet

PUT-statementet er motstykket til INPUT-statementet. I INPUT-statementet defineres hva som skal leses inn fra en datafil, mens i PUT-statementet beskrives det som skal skrives ut på en datafil. Med datafil menes her et rådatasett, hvilket altså ikke er det samme som et SAS-datasett. Formen på PUT-statementet er slik:

```
PUT  [ spesifikasjoner ] ;
```

Dersom det ikke oppgis noen spesifikasjoner i PUT-statementet, skrives hele observasjonen ut. Spesifikasjoner kan være variabel-navn eventu-

elt etterfulgt av format, men det kan også spesifiseres tekststrenger. En tekststreng må skrives med en ' først og sist.
Eksempel:

```
PUT navn Å15. 'Vekten er:' +2 vekt 5.1 'kg' ;
```

Filen som PUT-statementet skriver på, er definert i et FILE-statement, tilsvarende som INFILE-statementet. Dersom programmet ikke inneholder noe FILE-statement, skriver PUT ut på SASLOG'en (les om SASLOG i dette heftet, kap.4).
Bemerk forskjellen mellom OUTPUT- og PUT-statementene: OUTPUT skriver data ut på et SAS-datasett, mens PUT skriver data ut på en rådatafil eller SASLOG'en.

5.2.10 FILE-statementet

Dersom en vil skrive et SAS-datasett ut på en fil (dette kan benevnes som en rådatafil, en ekstern fil, et OS-datasett, et PDS-member osv.), brukes i tillegg til PUT-statementet et FILE-statement. Dette skrives slik:

```
FILE navn ;
```

Navnet henviser til DD-navnet på den filen det skal skrives ut på. Se forøvrig eksempelet som står under INFILE-statementet.

5.3 Statements brukt for å bearbeide data

Et SAS-datasett vil i utgangspunktet inneholde de variable som er definert i INPUT-statementet. Dersom man i senere DATA-steg ønsker å utelate noen av variablene, kan dette gjøres med KEEP- eller DROP-statementet. Ønsker man å endre navn på variable, brukes RENAME-statementet.
SAS-språket inneholder også tradisjonelle programmerings-muligheter som DO-løkker, IF-THEN/ELSE-konstruksjoner og tilordningssetninger. Noen av disse statementene vil bli kort beskrevet under.

5.3.1 KEEP- og DROP-statementene

Statementene har formen:

```
KEEP variabelliste ;
```

eller

```
DROP variabelliste ;
```

Er det bare noen få variable som skal sløyfes, brukes DROP-statementet, mens hvis de variable som skal beholdes er i mindretall, brukes KEEP-statementet.

Dersom variablene som inngår i variabellisten har nummererte navn (f.eks gruppe1 gruppe2 gruppe3 gruppe4 osv), kan man istedenfor å skrive alle variabelnavnene bare skrive:

```
DROP gruppe1-gruppe4;
```

Dersom variabelnavnene ikke er nummererte, men variablene ligger etter hverandre på datasettet (sjekkes med PROC CONTENTS), kan man istedenfor å skrive alle navnene i listen bare skrive det første og det siste, med to bindestreker (--) mellom. F.eks:

```
KEEP matutg--boutg;
```

Vær oppmerksom på at det i noen PROC's blir laget spesielle variable som kan bli lagt til et eventuelt ut-datasett. Dette gjelder f.eks PROC CORR og PROC SUMMARY. Variablene heter i så fall _TYPE_, _NAME_, _FREQ_ eller liknende. Slike variable kan også sløyfes eller beholdes med med et KEEP- eller DROP-statement.

5.3.2 RENAME-statementet

Statementet skrives slik:

```
RENAME gammelt variabelnavn = nytt variabelnavn ;
```

En kan endre navn på mange variable i ett RENAME-statement, altså:

```
RENAME inntekt1=br_innt inntekt2=nt_innt inntekt3=renter;
```

Vær klar over at de nye variabelnavnene først trer ikraft når observasjonen legges ut på det nye SAS-datasettet. Bruk derfor de gamle variabelnavnene i inneværende DATA-steg, uansett hvor RENAME-statementet er plassert i dette DATA-steget.

5.3.3 assignment-statementet

Dette er et viktig statement. Det starter ikke med noe nøkkelord, men har følgende form:

```
variabelnavn = uttrykk ;
```

Variabelnavnet på venstre side av likhetstegnet kan enten betegne en variabel som allerede eksisterer på datasettet eller en ny variabel. Hvis variabelen ikke finnes på datasettet, blir den opprettet der og da. Assignment-statementet brukes altså når en skal opprette nye variable. Ytterligere deklarasjon av en ny variabel trengs ikke.
Uttrykket på høyresiden av likhetstegnet kan bestå av konstanter, eksisterende variable og/eller funksjoner, knyttet sammen av passende operatorer eller parenteser. I Basics-manualen er det eget kapittel om SAS-uttrykk (Kap.5).

ARITMETISKE OPERATORER:

**	potensering
*	multiplikasjon
/	divisjon
+	addisjon
-	subtraksjon

LOGISKE OPERATORER:

AND
OR
NOT

SAMMENLIKNENDE OPERATORER:

=	lik	kan også skrives	EQ
^=	ikke lik	-	NE
<	mindre enn	-	LT
<=	mindre eller lik	-	LE
>	større enn	-	GT
>=	større eller lik	-	GE

Dersom et uttrykk inneholder parenteser, vil det som står inni parentesene bli beregnet først.

Det er viktig å merke seg følgende: Dersom en i et uttrykk skal summere variable og bruker operatoren '+', kan dette skje: Hvis et eneste av leddene som skal summeres er missing, vil hele uttrykket bli missing.

Eksempel:

```
total = a + b ;
```

a	b	total
3	5	8
6	.	.

I tilfeller der det kan forekomme missing-verdier, må altså SUM-funksjonen brukes istedenfor operatoren '+' i assignment-statementet. SUM-funksjonen returnerer summen av argumentene som listes opp. SUM-funksjonen behandler missing som '0'.

Eksempel:

```
total = SUM (a,b) ;
```

a	b	total
3	5	8
6	.	6

5.3.4 RETAIN-statementet

SAS behandler et datasett sekvensielt, det vil si at første observasjon i datasettet behandles helt ferdig før neste observasjon leses. Før en ny observasjon leses inn, settes alle dens variable til missing. Dersom en ønsker at en variabel ikke skal 'nullstilles' før neste observasjon leses, må en bruke RETAIN-statementet på variabelen. Statementet skrives:

```
RETAIN  [ variabelnavn  [ initialverdi ] ] ;
```

Dersom en ikke skriver noe variabelnavn i statementet, blir alle variabelverdiene beholdt.

I eksempelet under er utgangspunktet et datasett 'stevne' med resultater fra en konkurranse. Hver observasjon representerer en deltager. Det konkurreres i flere klasser, og hver observasjon inneholder blant annet variablene 'klasse' og 'poeng'. Ønsket er å lage et datasett som inneholder vinnerne i hver klasse, altså den observasjon som har flest poeng i hver klasse.

```
PROC SORT DATA=stevne;
    BY klasse;

DATA vinnere;
    SET stevne;
    BY klasse;
    IF FIRST.klasse THEN topp =. ;
    RETAIN topp ;
        topp = MAX ( topp, poeng ) ;
    DROP poeng;
    IF LAST.klasse THEN OUTPUT;
```

Først må datasettet 'stevne' sorteres på 'klasse'. 'Stevne' leses så inn i et DATA-steg der det skal lages et datasett 'vinnere'. En ny variabel 'topp' settes til missing dersom observasjonen som behandles er den første med en bestemt verdi av 'klasse'. (Se OUTPUT-statementet for mer beskrivelse av FIRST. og LAST.!) RETAIN-statementet sier så at

'topp' skal beholde verdien sin når neste observasjon leses. Ved hjelp av MAX-funksjonen som returnerer det høyeste av to tall, vurderes så 'topp' (som er hittil høyeste poeng i denne klassen) mot 'poeng' (som er denne observasjonens poeng). Er 'poeng' høyere enn 'topp', tilordnes 'topp' ny verdi, ellers ikke. Når siste observasjon med en gitt 'klasse' behandles, legges observasjonen som da inneholder klassens høyeste poeng-verdi ut på datasettet 'vinnere'.

5.3.5 sum-statementet

Dette statementet er blant de få SAS-statements som ikke starter med et nøkkelord. Det brukes til å lage akkumuleringsvariable, og det skrives slik:

```
variabelnavn + uttrykk ;
```

Variabelnavnet angir en akkumuleringsvariabel som vil beholde sin verdi når neste observasjon leses inn, akkurat som en variabel som opptrer i et RETAIN-statement.
Et sum-statement har samme virkning som å bruke SUM-funksjonen sammen med et RETAIN-statement.

Det viktige med sum-statementet er at uttrykket som skal legges til akkumuleringsvariabelen settes til 0 dersom uttrykket er missing. Dette i motsetning til hva som skjer i et assignment-statement, der hele uttrykket blir missing hvis et av leddene som skal summeres er missing.

5.3.6 IF-setninger

Det finnes to slags IF-setninger:

* subsetting IF
* IF-THEN/ELSE

5.3.6.1 subsetting IF

Subsetting IF brukes hvis bare noen av de observsjonene som leses inn i DATA-steget skal være med i datasettet som lages. Statementet skrives slik:

```
IF betingelse ;
```

Betingelsen må være et SAS-uttrykk. Kun de observasjoner der betingelsen er tilfredsstilt blir behandlet videre. Hvis betingelsen ikke er oppfylt, går SAS til neste observasjon.

Eksempel:

```
DATA rev ;
    SET rovdyr;
    IF art = 'fjellrev' OR art = 'rødrev' ;
```

I eksempelet over hentes datasettet 'rovdyr' inn med SET-statementet, men bare de observasjonene som har verdi 'fjellrev' eller 'rødrev' i variabelen 'art' legges ut på datasettet 'rev'.

5.3.6.2 IF-THEN/ELSE-statementet

En IF-THEN-konstruksjon brukes dersom man vil utføre et SAS-statement på noen observasjoner, men ikke alle. Dette skrives slik:

```
IF betingelse THEN statement ;
```

Betingelsen må være et SAS-uttrykk, se omtalen av assignment-statementet eller kap.5 i Basics-manualen.

Uttrykket som er betingelsen kan ha verdi 'sann', 'ikke sann' eller være missing. Er betingelsen 'sann' blir statementet etter 'THEN' utført, ellers ikke. Statementet som skal utføres, kan være et hvilket som helst eksekverbart SAS-statement eller en DO-gruppe.

Eksempel:

```
IF ktr_nr = 61  THEN kontor = 'system' ;
```

Hvis variabelen 'ktr_nr' har verdi '61' skal variabelen 'kontor' tilordnes verdien 'system'. Hvis mer enn ett statement skal utføres når betingelsen er oppfylt, må statementene stå inni en DO-END-gruppe:

```
IF postnr = 3590 THEN
    DO;
        sted = 'finse';
        folketal = 164 ;
        OUTPUT ulvik ;
    END;
```

Rett etter IF-THEN-statementet kan det følge et ELSE-statement som spesifiserer hva som skal utføres dersom IF-betingelsen ikke er sann. ELSE-statement skrives:

```
ELSE statement ;
```

Det som skal utføres kan være ett enkelt statement, eller flere statements omgitt av en DO-END-gruppe.

Eksempel:

```
IF kjonn = 1  THEN OUTPUT  menn ;
ELSE OUTPUT  kvinner ;
```

IF-THEN-setninger kan også settes inni hverandre:

```
IF kjonn = 1 THEN
    IF estatus = 1  THEN PUT 'Mann, gift' ;
    ELSE PUT 'Mann, ugift' ;
ELSE PUT 'Kvinne, gift eller ugift' ;
```

I ovenstående tilfelle kunne en SELECT-WHEN/OTHERWISE-END-konstruksjon også vært brukt, se neste avsnitt.

5.3.7 SELECT-WHWN/OTHERWISE-END-statementet

SELECT-WHEN/OTHERWISE-END-setningene vises her bare med et eksempel. Det er imidlertid en meget nyttig konstruksjon som det kan leses mer om i Basics-manualen. Eksempelet er det samme som ble brukt sist i forrige avsnitt:

```
SELECT (kjonn);
    WHEN (1) DO;
                SELECT (estat);
                    WHEN (1) PUT 'Mann, gift' ;
                    OTHERWISE PUT 'Mann, ugift' ;
                END;
            END;
    OTHERWISE PUT 'Kvinne, gift eller ugift' ;
END;
```

5.3.8 DO-setninger

Det er mange former for DO-statements i SAS:

* enkel DO-END
* iterativ DO-END
* DO WHILE-END
* DO UNTIL-END
* DO OVER-END

5.3.8.1 enkel DO-END

En enkel DO-END-gruppe brukes i forbindelse med IF-THEN/ELSE-statements når flere statements skal utføres ved en gitt betingelse.

```
DO;
    statements
END;
```

5.3.8.2 iterativ DO-END

En DO-END-gruppe kan utføres gjentatte ganger ved hjelp av en indeksvariabel som spesifiseres i DO-statementet:

```
DO indexvar = start [ TO stopp [ BY skritt ] [ WHILE uttrykk ] ];
    statements
END;
```

eller

```
DO indexvar = start [ TO stopp [ BY skritt ] [ UNTIL uttrykk ] ];
    statements
END;
```

Dersom en ikke oppgir skrittlengde, er default verdi 1.

I iterativ DO er det mange spesifikasjoner man kan velge å ha med eller ikke. Det medfører at iterativ DO kan anta svært mange former.

Eksempler på lovlige former for iterativ DO:

Går gjennom DO-gruppen en gang:

```
DO tall = 5 ;
    .....
END;
```

Går gjennom DO-gruppen 10 ganger:

```
DO tall = 1 TO 10 ;
    .....
END;
```

Går gjennom DO-gruppen 5 ganger:

```
DO tall = 1 TO 10  BY 2 ;
    .....
END;
```

Går gjennom DO-gruppen 4 ganger:

```
DO tall = 1, 3, 7, 16 ;
    .....
END;
```

Går gjennom DO-gruppen maksimum 5 ganger, mens betingelsen i WHILE er oppfylt:

```
DO tall = 1 TO 5 WHILE ( a < b ) ;
    .....
END;
```

Går gjennom DO-gruppen maksimum 5 ganger, inntil betingelsen i UNTIL er oppfylt:

```
DO tall = 1 TO 5 UNTIL ( a > b ) ;
    .....
END;
```

5.3.8.3 DO WHILE-END

DO WHILE-END benyttes for å utføre en DO-gruppe gjentatte ganger, mens en betingelse er oppfylt. Statementet skrives:

```
DO WHILE (uttrykk) ;
    statements
END;
```

Uttrykket som er betingelsen, testes på toppen av DO-løkken.

5.3.8.4 DO UNTIL-END

DO UNTIL-END benyttes for å utføre en DO-gruppe gjentatte ganger, inntil en betingelse er oppfylt. Statementet skrives:

```
DO UNTIL (uttrykk) ;
    statements
END;
```

I DO UNTIL testes betingelsen på slutten av DO-løkken, i motsetning til i DO WHILE. I DO UNTIL vil derfor løkken alltid bli utført minst en gang.

5.3.8.5 DO OVER-END

DO OVER-END brukes dersom man vil ha utført samme statements for alle elementene i en 'implisitt subscripted' array. Dette er en array som er deklarert uten å oppgi antall elementer den inneholder. Statementet har formen:

```
DO OVER arraynavn;
    statements
END;
```

Et eksempel:

```
DATA justerte;
    SET faktiske;
    ARRAY priser  pris1 pris2 pris3 pris4 pris5 ;
    DO OVER priser;
        priser = priser * 1.035 ;
    END;
```

6. PROC-STEGENE

I PROC-steg analyseres SAS-datasett. Et PROC-steg innledes ved at man kaller opp en ferdiglaget SAS-prosedyre (en PROC). De delpakkene av SAS som Byrået har, inneholder over 100 prosedyrer for ymse formål. Det er blant annet PROC's innenfor emnene

* deskriptiv statistikk
* variansanalyse
* regresjonsanalyse
* multivariat analyse
* kategorisk analyse
* diskriminantanalyse
* scoring
* clustering
* tidsserier
* lineære systemer
* ikke-lineære systemer

Det er dessuten PROC's for grafikk, regneark og brevark. Beskrivelsen av de enkelte prosedyrer innenfor følgende emner finnes i manualene:

- deskriptiv statistikk: SAS User's Guide: BASICS
- mer avansert statistikk: - : STATISTICS
- økonomi, tidsserier: SAS/ETS User's Guide
- regneark, brevark: SAS/FSP User's Guide
- grafikk: SAS/GRAPH User's Guide

På innsiden av permen til manualene står det en oversikt over hvilke PROC's som beskrives i nettopp den manualen.

6.1 Statements som kan brukes i et PROC-steg

Sammenliknet med DATA-steget er det få statements tilknyttet et PROC-steg. Dette er fordi den vesentligste tilretteleggingen av data (og dermed behovet for statements) skjer i DATA-steget.

Følgende statements kan brukes i PROC-steget:

ATTRIB	FREQ	OUTPUT	TITLE
BY	ID	PARMCARDS	VAR
CLASS	LABEL	PROC	WEIGHT
FORMAT	MODEL		

Disse statementene brukes blant annet for å spesifisere hvilke variable som skal være med i analysen, for å identifisere klassifikasjonsvariable, for å spesifisere at det skal analyseres gruppevis på en variabels verdi, for å spesifisere avhengige - og forklaringsvariable i en modell, for å knytte nærmere beskrivelser (labels) til variabelnavn med henblikk på en forståelig utskrift, osv.

6.2 Litt om noen utvalgte PROC's

Siden det er så mange SAS-prosedyrer å velge mellom, er det vanskelig å ha oversikten over alle. For å finne ut hvilken eller hvilke PROC's en skal bruke i det enkelte tilfelle, er derfor grundige studier i manualen(e) ikke til å unngå. Men for å gjøre manual-lesingen mest mulig effektiv, bør følgende strategi følges:
Først må man finne ut hvilket område ens analysebehov ligger i. Ønsker man deskriptiv statistikk, mer avansert statistikk, tidsserier, eller kanskje litt av hvert? Etter en slik vurdering, velger man hvilken manual som skal brukes. Se oversikten på forrige side! Man studerer så innsiden av permen til den aktuelle manual (evt flere manualer) og ser der hvilke PROC's som finnes innen de forskjellige emnene.

I de fleste prosjekt vil det på et innledende stadium være behov for beskrivende statistikk på datamaterialet. Ved f.eks å sjekke maksimum- og minimum-verdier for de enkelte variablene, kan man blant annet se om dataene er korrekt lest inn. I det følgende vil noen av de mest anvendelige prosedyrene beskrives svært kort. Når det gjelder prosedyrer til spesielle formål som tidsserier, modellsimulering, regresjons- og varians-analyser, osv. osv. henvises det til manualene.

6.2.1 PROC PRINT

Dette er den prosedyre som brukes når en vil ta utskrift av et SAS-datasett. Dersom man etter å ha laget et SAS-datasett i et DATA-steg skriver

```
PROC PRINT;
```

vil man få skrevet ut dette datasettet med alle dets observasjoner og alle variable.

Dersom man i selve PROC PRINT-statementet ikke har med noen spesifikasjon av hvilket datasett som skal skrives ut, vil prosedyren skrive ut det datasett som ble laget sist før PROC-steget. Det er imidlertid en god regel alltid å spesifisere hvilket datasett som skal skrives ut. Dette gjør man ved å ha med en DATA=-option i PROC-statementet (se eksempelet nedenfor).
Sammen med en DATA=-option kan man også angi hvor mange av datasettets observasjoner som skal skrives ut dersom man ikke vil ta ut alle.
Hvis man ønsker at ikke alle variablene skal være med i utskriften, kan man i tillegg til PROC PRINT-statementet skrive et VAR-statement. I dette statementet lister man opp de variable som skal være med i utskriften.
Siden variabelnavn kan bestå av max 8 tegn, er det begrenset hvor informativt et slikt navn kan være. For å avhjelpe dette, kan man knytte beskrivelser (labels) til den enkelte variabel. Dette gjøres med et LABEL-statement. Ved å føye en LABEL-option til PROC PRINT-statementet, er det labelen som blir skrevet ut i resultatet istedenfor variabelnavnet.

Eksempel:
SAS-datasettet 'familie' består av 2000 observasjoner, hver med 100 variable som inneholder familiens husholdningsutgifter og diverse andre opplysninger. Variablene som gjelder husholdningsutgiftene har de lite opplysende navnene 'var1', 'var2', 'var3' osv. Vi ønsker å ta en utskrift av de første 50 observasjonene på datasettet. Vi vil kun ha med noen utvalgte variable, og i utskriften vil vi ikke ha med variabelnavnene, men derimot beskrivelser av variablene (labels). Ut-

skriften skal dessuten ha en passende overskrift.

```
PROC PRINT DATA=familie(OBS=50) LABEL ;
    VAR personer var3 var6 var7 var24 var30;
    LABEL var3  = 'matvarer'
          var6  = 'klær, sko'
          var7  = 'brennevin'
          var24 = 'bil, båt'
          var30 = 'bolig' ;
    TITLE  'UTGIFTER PR. ÅR TIL NOEN UTVALGTE VARER' ;
    TITLE3 'KUN DE FØRSTE 50 OBSERVASJONER' ;
```

6.2.2 PROC MEANS

PROC MEANS er en meget anvendelig prosedyre. Den produserer deskriptiv statistikk for numeriske variable.
For hver numerisk variabel på datasettet beregnes gjennomsnittsverdi, standard avvik, minimumsverdi og maksimumsverdi. Det oppgis også hvor mange observasjoner beregningen av disse størrelsene er basert på.

Når rådata er lest inn til et SAS-datasett er det en god regel alltid å kjøre PROC MEANS på datasettet. Ved å studere utskriften fra denne prosedyren kan en få en god oversikt over datamaterialet. Ved f.eks å se på gjennomsnitt, maksimum- og minimumsverdier, kan en lett se om dataene er lest korrekt inn.

Hvis man istedenfor størrelsene snitt, standard avvik, maksimum og minimum ønsker f.eks standard feil, varians, variasjonskoeffisient, skewness eller kurtosis, kan dette spesifiseres som options i selve PROC MEANS-statementet.

Dersom intet annet oppgis, vil størrelsene i utskriften fra PROC MEANS inneholde et nokså høyt antall desimaler. Ved å ha med en MAXDEC-option i PROC MEANS-statementet, kan en selv angi det maksimale antall desimaler utskriften skal inneholde.

6.2.3 PROC FORMAT

Ofte vil variablene vi arbeider med være kodet på en spesiell måte. F.eks vil en variabel for kjønn med verdi '1' bety 'mann', mens verdien '0' vil bety 'kvinne'. I en tabellutskrift kan det i slike tilfelle være ønskelig å få skrevet ut noe annet enn variablenes formelle verdi. Dette er mulig ved hjelp av en PROC FORMAT.

I PROC FORMAT inngår ett eller flere VALUE-statements. I et VALUE-statement oppgis først et formatnavn, deretter knyttes beskrivelser (formater) til enkeltverdier eller verdiområder. I eksempelet som følger, gjelder det første VALUE-statementet kjønn. Formatnavnet er 'kj'. Verdiene 0 og 1 blir gitt 'formatene' kvinne og mann. I det andre VALUE-statementet er formatnavnet 'alder'. Her blir formatene ikke knyttet til enkeltverdier, men til verdiområder. Legg merke til at man slipper å oppgi nedre og øvre grense, det holder å skrive low hhv high. Formatene må ha en ' først og sist. Vær klar over forskjellen mellom et formatnavn og selve formatene! Formatnavnet skal være bindeleddet mellom en variabel og formatene variabelen skal skrives ut med. Når en vil bruke slike utskriftsformater, må derfor formatnavnet knyttes sammen med det aktuelle variabelnavnet i et FORMAT-statement.

Et eksempel:

```
PROC FORMAT;
    VALUE kj
              0 = 'kvinne'
              1 = 'mann' ;
    VALUE alder
              low-29   = 'under 30 år'
               30-34   = '30 - 34 år'
               35-high = '35 år og over'  ;

PROC PRINT DATA=utvalg;
    VAR kjonn alder;
    FORMAT kjonn kj. alder alder. ;
    TITLE 'Eksempel på bruk av utskriftsformater.' ;
```

I eksempelet over er det et FORMAT-statement i PROC-steget der utskriften skal lages. I FORMAT-statementet knytter en sammen variabelnavnet 'kjonn' med formatnavnet 'kj'. Husk at det alltid må stå et

punktum etter formatnavnet! Formatnavnet lager man selv. Det lønner seg å lage et formatnavn som likner på variabelnavnet. Formatnavnet kan forsåvidt godt være lik variabelnavnet, slik det er sist i eksempelet over, der alder både er variabelnavn og formatnavn. Husk bare at det alltid skal stå et punktum etter formatnavnet! (Les evt. mer om regler for formatnavn i Basics-manualen, side 917.)

> Vær klar over forskjellen/likheten mellom FORMAT og LABEL! LABEL går på en variabels navn, mens FORMAT går på en variabels verdi.

6.2.4 PROC FREQ

Dette er en prosedyre som lager frekvenstabeller. PROC FREQ gir brukeren mange muligheter: Det kan lages enkle frekvenstabeller der bare en variabel inngår, eller det kan lages krysstabeller i flere dimensjoner.

I PROC FREQ kreves det foruten PROC-statementet også et TABLES-statement. I dette oppgir man variabelen eller variablene som skal inngå i tabellen. Pass på at variablene som oppgis i TABLES-statementet er kategoriske! En variabel som kjønn, fylke eller ekteskaplig status, vil uten videre være velegnet til å inngå i et TABLES-statement. Variable som inntekt eller alder derimot, vil måtte deles inn i passende størrelsesgrupper før de eventuelt kan være med i et TABLES-statement.

En slik inndeling i grupper kan gjøres på to forskjellige måter: Enten kan man lage en ny variabel, en grupperingsvariabel, eller man kan bruke PROC FORMAT til å lage størrelsesgrupper. Eksempelet på neste side viser begge deler:

```
DATA ;
    SET ifu.utvalg;
*-------------------------------------------------------*;
*   Deler inntekten inn i 7 størrelsesgrupper:          *;
*-------------------------------------------------------*;
    IF           0 <= inntekt <   10000 THEN igruppe = 1;
    ELSE IF  10000 <= inntekt <=  25000 THEN igruppe = 2;
    ELSE IF  25000 <  inntekt <=  50000 THEN igruppe = 3;
    ELSE IF  50000 <  inntekt <= 100000 THEN igruppe = 4;
    ELSE IF 100000 <  inntekt <= 150000 THEN igruppe = 5;
    ELSE IF 150000 <  inntekt <= 200000 THEN igruppe = 6;
    ELSE                                     igruppe = 7;

    KEEP kjonn alder igruppe;

PROC FORMAT;
    VALUE kj  0 = 'KVINNER'
              1 = 'MENN' ;
    VALUE innt  1 = ' under 10 000    '
                2 = ' 10 000 - 25 000'
                3 = ' 25 001 - 50 000'
                4 = ' 50 001 -100 000'
                5 = '100 001 -150 000'
                6 = '150 001 -200 000'
                7 = ' over 200 000    '  ;

    VALUE ald  low-29  = 'under 30 år'
                30-60  = '30 - 60 år'
                61-high= 'over 60 år' ;

PROC FREQ ;
    TABLES kjonn*igruppe*alder;
    FORMAT kjonn kj.
           igruppe innt.
           alder ald.  ;
    LABEL kjonn='kjønn'
          igruppe='inntekt' ;
    TITLE 'Inntekt fordelt på aldersgrupper og kjønn' ;
```

2

Inntekt fordelt på aldersgruppe og kjønn

TABLES OF INNTEKT BY ALDER
CONTROLLING FOR KJØNN=MENN

1

Inntekt fordelt på aldersgruppe og kjønn

TABLES OF INNTEKT BY ALDER
CONTROLLING FOR KJØNN=KVINNER

INNTEKT	ALDER		
	under 30 år	30-59 år	over 60 år
under 10 000			
10 000 - 25 000			
25 001 - 50 000			
50 001 - 100 000			
100 001 - 150 000			
150 001 - 200 000			
over 200 000			

6.2.5 PROC SUMMARY

PROC SUMMARY er en prosedyre som lager deskriptiv, gruppevis statistikk. Foruten selve PROC SUMMARY-statementet har man som regel med et CLASS-statement som identifiserer grupperingsvariable, og et VAR-statement som sier hvilke variable det skal beregnes statistikk på. PROC SUMMARY er en av de få SAS-prosedyrer som ikke produserer resultatet på en utskrift. PROC SUMMARY lager derimot automatisk et nytt SAS-datasett som inneholder de statistiske størrelsene som er blitt beregnet. Hver observasjon i dette nye datasettet inneholder beregninger for grupper av observasjoner i input-datasettet. For å se på resultatet fra PROC SUMMARY, må vi ta en PROC PRINT på resultat-datasettet.

PROC MEANS kan også produsere gruppevis statistikk. Forskjellen mellom PROC MEANS og PROC SUMMARY er at PROC MEANS bare lager gruppevise beregninger når en har med et BY-statement. Datasettet må da først være sortert på BY-variabelen. For mer enn en slik grupperingsvariabel må dertil PROC MEANS kjøres flere ganger, mens det er tilstrekkelig med en PROC SUMMARY-kjøring for å få samme resultat.

Et eksempel:

Vi har et datasett 'kommuner' der hver observasjon består av data for en kommune. En observasjon inneholder blant annet kommunens inntekter og utgifter, samt hvilket fylke den tilhører. I eksempelet ønsker vi å summere inntekter og utgifter til alle kommunene innen et fylke. Vi vil også finne ut hva som er laveste og høyeste kommunale inntekt/utgift innen fylket. Til en slik problemstilling kan man bruke PROC SUMMARY. Det datasettet som PROC SUMMARY lager, vil inneholde en observasjon for hvert fylke, inneholdende de størrelsene vi ønsket å beregne. I tillegg vil datasettet inneholde en observasjon der beregningsgrunnlaget er hele 'kommune-datasettet'.
Det datasettet som produseres av PROC SUMMARY, kan senere brukes til videre analyser på fylkes-nivå, evt landsnivå.

```
PROC SUMMARY DATA=kommuner;
    CLASS fylke;
    VAR inntekt utgift;
    OUTPUT OUT=resultat
        SUM  = suminn sumut
        MAX  = maksinn maksut
        MIN  = minstinn minstut ;
```

Kommentar til eksempelet:

Det første statementet er selve PROC SUMMARY-statementet, der prosedyren kalles. I dette statementet har vi med en DATA=-option, der vi spesifiserer hvilket datasett prosedyren skal behandle.
I CLASS-statementet angir vi så grupperingsvariabelen. I vårt tilfelle skal altså alle observasjoner med samme verdi for variabelen 'fylke' behandles under ett.
I neste statement, VAR-statementet, oppgis hvilke variable det skal beregnes statistikk på, nemlig variablene 'inntekt' og 'utgift'.
Deretter følger OUTPUT-statementet. Her oppgis først navnet på det datasettet PROC SUMMARY lager, det skal hete 'resultat'. Deretter

oppgis hva slags statistikk som skal beregnes, og hvilke navn disse størrelsene skal ha. (Dette blir altså variabelnavnene på det nye datasettet.)

KOMMUNE:

komnr	fylke	inntekt	utgift
1020	18	3405	2307
3123	18	6380	6804
4225	18	7102	6746
6312	14	2340	2287
5420	14	6529	6471
6425	14	5091	6709
7500	14	4402	1985

RESULTAT:

fylke	suminn	sumut	maksinn	maksut	minstinn	minstut	_TYPE_	_FREQ_
.	35249	33309	7102	6804	2340	1985	0	7
18	16887	15857	7102	6804	3405	2307	1	3
14	18362	17452	6529	6709	2340	1985	1	4

Vi ser at datasettet 'resultat' i tillegg inneholder to variable: _FREQ_ og _TYPE_ . Dette er variable som SAS automatisk lager. Variabelen _FREQ_ inneholder det antall observasjoner i inn-datasettet ('kommuner') som ligger bak en observasjon i ut-datasettet ('resultat').

Variabelen _TYPE_ angir aggregeringsnivå for observasjonen. Vi ser i eksempelet at den første observasjonen på 'resultat' har _TYPE_ lik 0. Det betyr at observasjonen representerer hele datasettet 'kommuner'. De neste observasjonene har _TYPE_ lik 1, hvilket betyr at de representerer første undergruppe i CLASS-statementet.

I vårt eksempel er det bare en undergruppe i CLASS-statementet, nemlig fylke. Men la oss tenke oss at det var oppgitt 2 grupperingsvariable i CLASS-statementet, fylke og kommunestyrets politiske 'farge' (A eller H):

```
CLASS farge fylke;
```

La oss også tenke oss at inn-datasettet vårt inneholdt en observasjon for hver av landets 454 kommuner. Da ville vi i ut-datasettet fått 1

observasjon med _TYPE_=0, basert på hele landet. Dertil ville vi fått 20 observasjoner, en for hvert fylke, med _TYPE_=1. Vi ville dessuten fått 2 observasjoner med _TYPE_=2, der den ene ville være basert på alle landets kommuner med A-styre, og den andre basert på de med H-styre. I tillegg til dette ville vi fått 40 (=2*20) observasjoner av _TYPE_=3, der grunnlaget ville være 'fylke'*'farge'. Altså alle A-styrte kommuner i ett fylke, alle H-styrte kommuner i dette fylke, så dette gjentatt for alle de 20 fylkene.

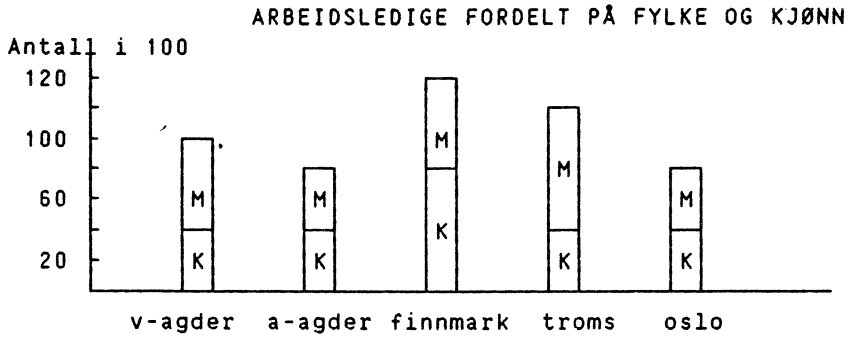
Når vi senere vil bearbeide de aggregerte dataene, kan vi ved hjelp av _TYPE_-variabelen velge ut de observasjonene som gjelder nettopp dette aggregeringsnivå.

6.2.6 PROC CHART

Dersom man ikke har tilgang til en grafisk terminal, men likevel ønsker en grafisk fremstilling av dataene, kan man f.eks bruke PROC CHART. Den lager blant annet søylediagrammer (både horisontale og vertikale), og paidiagrammer.

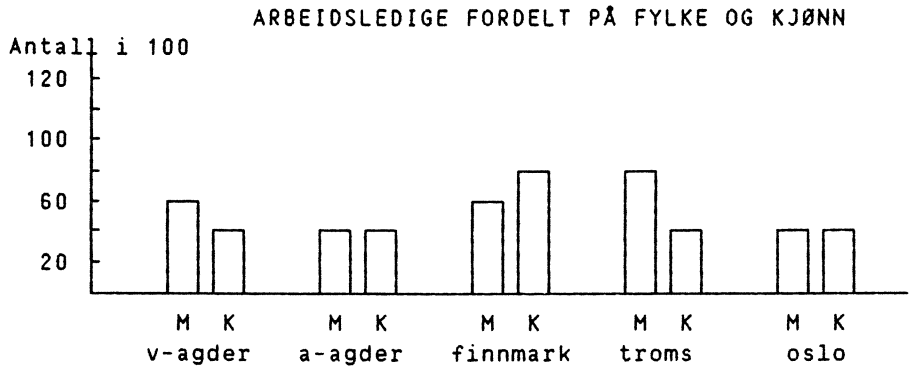
Eksempel:
Et permanent SAS-datasett 'aku.ledige' inneholder informasjon om personer uten arbeid. Datasettet inneholder en observasjon for hver arbeidsledig person. Blant variablene er personens kjønn, alder og hjemstedsfylke samt antall måneder i ledighet. Vi ønsker et diagram der hver søyle representerer ledige i ett fylke. Søylen skal også vise kjønnsfordelingen av de arbeidsledige i fylket.

```
PROC CHART DATA=aku.ledige;
    VBAR fylke / SUBGROUP=kjonn;
    TITLE 'Arbeidsledige fordelt på fylke og kjønn';
```



Dersom vi ønsker at menn og kvinner skal være representert med hver sin søyle, kan programmet se slik ut:

```
PROC CHART DATA=aku.ledige;
    VBAR kjonn / GROUP=fylke;
    TITLE 'Arbeidsledige fordelt på fylke og kjønn';
```



6.2.7 PROC PLOT

Med PROC PLOT kan man lage diagram der flere variable er plottet mot hverandre. Dette kan gi en god visuell oppfatning av dataene. Med utgangspunkt i datasettet i eksempelet ovenfor, kan vi f.eks plotte de arbeidslediges tid i ledighet som en funksjon av alder:

```
PROC PLOT DATA=aku.ledige;
    PLOT mnd*alder;
    TITLE 'Antall mnd i ledighet som funksjon av alder';
```

7. STYREKORT (JCL) BRUKT I FORBINDELSE MED SAS

I dette kapitlet beskrives de JCL-oppsett som trengs når en skal bruke SAS. Følgende er viktig å merke seg:

- Det er tre typer JCL-kort: JOB-kort, EXEC-kort og DD-kort
- Alle linjer i JCL-oppsettet må begynne med //
- Det må kun brukes store bokstaver
- Parametre må adskilles av komma, ikke mellomrom
- Hvis et kort skal fortsette på neste linje, må det stå et komma etter siste tegn på linjen, og minst ett tegn etter // på fortsettelseslinjen må være blank

Alle JCL-eksemplene er laget for bruker LO61LDA. Vær derfor nøye med at alle forekomster av LO61LDA erstattes med ens egen brukerident.

7.1 Opprette fil for lagring av permanente SAS-datasett

Når et SAS-datasett skal lagres permanent må datasett-navnet være to-leddet, og første ledd må være DD-navnet til filen der datasettet skal lagres. Etter gjeldende regler må en fil som skal lagre permanente SAS-datasett være organisert på en bestemt måte. For å opprette en slik fil MÅ en derfor bruke et JCL-oppsett tilsvarende det som er vist i eksemplet på side 52.

> En fil som skal lagre permanente SAS-datasett kan IKKE opprettes i pkt 3.2 i ISPF !!!

SAS-datasett kan lagres på testfiler eller på produksjonsfiler, alt etter hva dataene skal brukes til.

En testfil kan ha følgende form på navnet:

TSnn.Snnnn.AAA.filnavn

```
TSnn.                    <---  T=test S=SAS nn=brukers ktr.nr
     Snnnn.              <---  nnnn=stat.nr
          AAA.           <---  brukers initialer
              filnavn    <---  hensiktsmessig navn velges
```

> En testfil varer bare 30 dager
> etter siste gang den ble lest
> fra eller skrevet til !!!

En produksjonsfil skal ha følgende form på navnet:

Sonn.Snnnn.ident.Gnn.Vnn

```
Sonn.                    <---  S=SAS o=filens levetid nn=ktr.nr
     Snnnn.              <---  nnnn=stat.nr
          ident.         <---  identen må fåes av System/Drift
               Gnnnn.    <---  nnnn=generasjonsnr
                    Vnn  <---  nn=versjonsnr
```

Filens levetid oppgis i antall 100 dager. Les mer om navnestandarden for Byråets filer i 'EDB brukerveiledning IBM' kapittel 2. Der står det også mer om filenes oppbevaringstid, identer, generasjonsnr og versjonsnr.

JCL for å opprette en lagringsfil til permanente SAS-datasett:

```
//LO61LDA  JOB nnnn,'brukers navn',NOTIFY=LO61LDA,
//             CLASS=A,MSGLEVEL=(1,1),MSGCLASS=X
//         EXEC PGM=IEFBR14
//LAGER    DD  DSN=TS61.Snnnn.LDA.SASFIL,
//             DISP=(NEW,CATLG),
//             UNIT=SSB,
//             DCB=(RECFM=U,DSORG=DA),
//             SPACE=(TRK,(2,1),RLSE
```

Kommentarer til JOB-kortet:

- Erstatt alle forekomster av LO61LDA med brukers egen ident.
- Erstatt nnnn med det aktuelle stat.nr.
- Skriv brukers navn med bokstaver og en ' først og sist.

Kommentar til EXEC-kortet:

- PGM=IEFBR14 brukes når en kun skal opprette evt slette filer.

Kommentarer til DD-kortet:

- LAGER er DD-navnet til filen som SAS-datasettene skal lagres på. Dette navnet bestemmer en selv.
- I DISP-parameteren angis at filen er ny og at den skal katalogiseres.
- I UNIT-parameteren kan man skrive SSB eller BATCH, hvilket angir navnet på disken som filen skal legges på. SSB-diskene er av type 3375, mens BATCH-diskene er av type 3350.
- I SPACE-parameteren oppgis hvor mye plass som skal avsettes til filen. Først oppgis enheten som plass skal avsettes i. I dette eksempelet avsettes plassen i tracks (TRK). Et track tilsvarer 35616 tegn på en disk av type 3375. (SSB-diskene er av type 3375).
- Etter at plass-enheten er oppgitt, må det angis hvor mange enheter som skal avsettes. I eksempelet står det (2,1). Det første tallet angir at det skal avsettes 2 tracks i første omgang. Det siste tallet angir at det deretter om nødvendig skal avsettes 1 track inntil 15 ganger til.

Første gang man trenger en lagringsfil for permanente SAS-datasett, opprettes den med et JCL-oppsett som det ovenfor. Når filen alt er opprettet, kan den skrives til og leses fra i SAS. JCL-oppsettet for å bruke filen vil da se slik ut:

```
//LO61LDA  JOB   ....
//
//         EXEC SAS,OPTIONS='GEN=1 PS=46 NOCENTER'     <--- NB
//LAGER    DD  DSN=TS61.Snnnn.LDA.SASFIL,DISP=OLD      <--- NB
//SYSIN    DD  *                                       <--- NB

DATA lager.kvinner;
       .....
              osv.
```

I EXEC-kortet knyttes nå forbindelse med SAS. I DD-kortet til lagringsfilen er DISP-parameteren endret til OLD (filen eksisterer jo allerede). Det er dessuten med et SYSIN DD-kort, og SAS-programmet kan skrives rett etter JCL-linjene.

7.2 Opprette filer for SASLOG og SASLIST

Dersom man i JCL-oppsettet ikke angir hvor SASLOG og SASLIST skal styres, blir de automatisk lagt bakerst i kjørerapporten fra IBM. Når jobben er ferdig, vil kjørerapporten ligge i 'hold-køen' i pkt 8 i ISPF og kan der sees på. Dersom man imidlertid ønsker å styre SASLOG og SASLIST til egne filer, må disse først opprettes.

JCL for å opprette filer til SASLOG og SASLIST:

```
//L061LDA   JOB nnnn,'brukers navn',NOTIFY=L061LDA,CLASS=A,
//             MSGLEVEL=(1,1),MSGCLASS=X
//          EXEC PGM=IEFBR14
//FT11F001  DD  DSN=TK61.Snnnn.LDA.SASLOG,
//             DISP=(NEW,CATLG),
//             UNIT=BATCH,
//             DCB=(RECFM=VBA,LRECL=137,BLKSIZE=141),
//             SPACE=(141,(100,100))
//FT12F001  DD  DSN=TK61.Snnnn.LDA.SASLIST,
//             DISP=(NEW,CATLG),
//             UNIT=BATCH,
//             DCB=(RECFM=VBA,LRECL=137,BLKSIZE=141),
//             SPACE=(141,(300,100))
```

Kommentar til DD-kortene:

- SASLOG og SASLIST må alltid ha DD-navn FT11F001 hhv FT12F001.
- SASLOG og SASLIST skal være TK-filer (ikke TS !).
- Navnene til SASLOG og SASLIST følger forøvrig standarden for testfiler, se ovenfor.
- Det eneste en selv kan bestemme, er antall enheter som skal opprettes (i SPACE-parameteren).
- Antallene som er oppgitt i eksempelet, (100,100) for SASLOG og (300,100) for SASLIST, vil være tilstrekkelig for de fleste formål.

SASLOG og SASLIST opprettes første gang man har bruk for dem. Når de først er opprettet, kan man i SAS-jobben knytte forbindelse med dem på følgende måte:

```
//LO61LDA   JOB   ....
//
//          EXEC SAS,  ....
//FT11F001  DD  DSN=TK61.Snnnn.LDA.SASLOG,DISP=OLD    <--- NB
//FT12F001  DD  DSN=TK61.Snnnn.LDA.SASLIST,DISP=OLD   <--- NB
//SYSIN     DD  *
```

Med DISP-parameteren OLD vil SASLOG og SASLIST inneholde kun inneværende kjørings meldinger og resultater. Dette er oftest det mest hensiktsmessige. Ønsker man derimot også å beholde SASLOG og SASLIST fra forrige kjøring (uten å ta disse ut på papir), kan DISP-parameteren settes lik MOD. Denne kjørings SASLOG og SASLIST blir da lagt etter LOG og LIST produsert i forrige kjøring. Ønsker man dette, må en være klar over at SASLOG- og SASLIST-filene ganske raskt kan overskride den plassen man har avsatt for dem.

7.3 Noen praktiske JCL-eksempler

Når en har opprettet lagringsfil for permanente SAS-datasett og evt opprettet filer for SASLOG og SASLIST, kan en starte med selve SAS-jobbene.

7.3.1 Lese rådata fra en tape

Først kommer et eksempel på hvordan en kan lese rådata fra en tape og lage et permanent SAS-datasett. Det forutsettes at filer til SASLOG og SASLIST, samt lagringsfil til SAS-datasett allerede er opprettet.

```
//LO61LDA   JOB 6613,'DAASVATN',NOTIFY=LO61LDA,CLASS=T,
//             MSGLEVEL=(1,1),MSGCLASS=X
//          EXEC SAS,OPTIONS='GEN=1 NOCENTER PAGESIZE=46'
//FT11F001  DD  DSN=TK61.S6613.LDA.SASLOG,DISP=OLD
//FT12F001  DD  DSN=TK61.S6613.LDA.SASLIST,DISP=OLD
//INNTAPE   DD  DSN=P413.S6613.I324A1A1.G8400.V00,DISP=SHR
//UTFIL     DD  DSN=TS61.S6613.LDA.SASFIL,DISP=OLD
//SYSIN     DD  *

DATA utfil.alledata;
    INFILE inntape;
    INPUT hushnr 1-4  komnr 5-8  Ø20 alder 20-21
          skattekl 25  Ø43 (inntekt formue skatt) (7.) ;
```

Legg merke til at CLASS-parameteren i JOB-kortet er satt til T, siden jobben innebærer at det skal leses fra en tape. Legg også merke til at tapens DISP-parameter er SHR. Det betyr at tapen bare skal leses. Når rådata lagret på tape eller disk skal leses inn til et SAS-datasett, må alltid DISP-parameteren være SHR.

```
DD-kortene til SASLOG og SASLIST
må alltid komme rett etter kortet
med EXEC SAS !!!
```

Hvis jobben bruker mer enn 30 CPU-sek, skal CLASS-parameteren være D. En må da også hekte en TIME-parameter på JOB-kortet, f.eks slik:

```
//LO61LDA  JOB  .....      ,CLASS=D,
//              ..... MSGCLASS=X,TIME=2
```

I TIME-parameteren oppgir man maksimal tid jobben vil ta, i CPU-minutter. Dersom en får ABEND på en jobb på grunn av manglende eller for liten TIME-parameter, vil feilkoden som står i IBM-kjørerapporten være 322.

7.3.2 Hvis lagringsfilen blir for liten

Dersom det viser seg at den filen man har opprettet for lagring av SAS-datasett ikke har nok plass til å lagre det man ønsker, kommer det melding om dette i SASLOG'en.

Hvis dette skjer første gang man prøver å lagre noe, har man opprettet filen med en altfor beskjeden SPACE-parameter. Det beste er da først å slette filen, og deretter opprette den på nytt med en større SPACE-parameter. Filen må først slettes sånn:

```
//LO61LDA   JOB 6613,'DAASVATN',NOTIFY=LO61LDA,CLASS=A,
//             MSGLEVEL=(1,1),MSGCLASS=X
//          EXEC PGM=IEFBR14
//SLETT     DD  DSN=TS61.S6613.LDA.SASFIL,DISP=(OLD,DELETE)
```

I eksempelet som viser hvordan man oppretter en lagringsfil, er plassen allokert i tracks: Først to, så ett inntil 15 ganger til (ialt 17 tracks):

```
//    ....  SPACE=(TRK,(2,1),RLSE)
```

Skriver man isteden:

```
//    ....  SPACE=(TRK,(3,2),RLSE)
```

vil man få tildelt først 3 tracks, så inntil 15 ganger 2 tracks til, ialt 33 tracks.

På SASLOG'en står det hvor langt SAS var kommet da plassen på lagringsfilen ble for liten. En får da en pekepinn om hvor mye større filen bør være når man allokerer plass til den på ny.

Dersom en lagringsfil blir for liten idet man skal legge ut på den SAS-datasett nr n (!), bør en følge en annen strategi enn den ovenfor nevnte.
En bør da nøye vurdere om det virkelig er nødvendig å ha alle datasettene som er på filen permanent lagret. Det er viktig å unngå dobbeltlagring. Dersom et datasett er en delmengde av et annet, slik at det bare skal noen få statements til for å lage det ene utfra det andre, er det ingen grunn til å ha begge lagret permanent.
Dersom man etter en slik vurdering kommer til at det trengs en opprydding på lagringsfilen, MÅ dette gjøres ved hjelp av PROC DATASETS, se kap 8.3.

```
Et SAS-datasett kan kun slettes
ved hjelp av PROC DATASETS !!!
```

7.3.3 Lese et permanent lagret SAS-datasett

Vi går nå ut fra at rådata er lest inn og ligger permanent lagret som et SAS-datasett. I neste omgang skal noen av observasjonene fra dette SAS-datasettet leses inn og arbeides videre med:

```
//L061LDA   JOB 6613,'DAASVATN',NOTIFY=L061LDA,CLASS=A,
//               MSGLEVEL=(1,1),MSGCLASS=X
//          EXEC SAS,OPTIONS='GEN=1 NOCENTER PAGESIZE=46'
//FT11F001  DD  DSN=TK61.S6613.LDA.SASLOG,DISP=OLD
//FT12F001  DD  DSN=TK61.S6613.LDA.SASLIST,DISP=OLD
//LAGRET    DD  DSN=TS61.S6613.LDA.SASFIL,DISP=OLD
//SYSIN     DD  *

DATA bearbeid;
    SET lagret.alledata;
    IF 30 <= alder <= 35;
           .....
           .....
                 osv.
```

Legg merke til at CLASS-parameteren i JOB-kortet nå er satt til A igjen.

7.3.4 JCL og SAS-program skrevet på hver sin fil

Dersom man velger å ha JCL'en og selve SAS-programmet på hvert sitt PDS-member må JCL-oppsettet endres litt:

```
//LO61LDA   JOB 6613,'DAASVATN',NOTIFY=LO61LDA,CLASS=A,
//             MSGLEVEL=(1,1),MSGCLASS=X
//          EXEC SAS,OPTIONS='GEN=1 NOCENTER PAGESIZE=46'
//FT11F001  DD  DSN=TK61.S6613.LDA.SASLOG,DISP=OLD
//FT12F001  DD  DSN=TK61.S6613.LDA.SASLIST,DISP=OLD
//LAGRET    DD  DSN=TS61.S6613.LDA.SASFIL1,DISP=OLD
//SYSIN     DD  DSN=LO61LDA.SAS.ANALYSE(PROGRAM),DISP=SHR
```

SAS-programmet er i dette eksempelet skrevet inn på et eget PDS-member. Memberet heter PROGRAM og ligger på et brukerdatasett som heter SAS.ANALYSE, tilhørende bruker LO61LDA. Filen med SAS-programmet må ha DISP-parameter SHR.

8. NYTTIGE HINT

I dette kapittel står litt av hvert som er nyttig å vite for en SAS-bruker.

8.1 Options

En skal ikke lese mange avsnittene i en SAS-manual uten å støte på en 'option'. Det finnes datasett-options (dsoptions) og options tilknyttet forskjellige statements. I tillegg har man et eget OPTIONS-statement, samt PROC OPTIONS .....

8.1.1 Datasett-options

Overalt der en oppgir navnet på et SAS-datasett (unntatt i et OUTPUT-statement!!), kan en føye til visse datasett-options (dsoptions):

```
DATA nytt;
    SET gammelt (KEEP= inntekt formue  FIRSTOBS=250 OBS=400);
```

I dette eksempelet skal datasettet 'gammelt' kopieres. Men av alle variablene på 'gammelt' skal kun 'inntekt' og 'formue' være med på det nye datasettet. Det skal også kun leses inn 150 observasjoner: Den første som leses skal være obs nr 250 på 'gammelt', mens den siste som leses skal være obs nr 400.

Legg merke til at OBS=-optionen angir nummeret til den siste observasjonen som skal leses! Dersom man ikke har med noen FIRSTOBS=-option leses obs nr 1 først, og OBS= vil da angi totalt antall observasjoner som blir lest inn. Har man derimot med en FIRSTOBS=-option, må en regne litt.

Viktig: dsoptions må alltid stå i parenteser!

Blant de vanligste dsoptions er KEEP=, DROP=, OBS=, FIRSTOBS=, RENAME= og IN= . Dersom en skal bruke RENAME=, må det være med et ekstra sett parenteser:

```
DATA kvart_1 ;
    MERGE jan (RENAME=(antall=ant_jan total=tot_jan) IN=januar )
          feb (RENAME=(antall=ant_feb total=tot_feb) IN=februar )
          mar (RENAME=(antall=ant_mar total=tot_mar) IN=mars ) ;
    BY nr;
```

I eksempelet over er brukt både RENAME=- og IN=-options. Disse er mye brukt i forbindelse med SET-, UPDATE og MERGE-statementene.

I eksempelet ønsker vi å lage ett nytt datasett ved å spleise sammenhørende observasjoner fra tre datasett til en observasjon i det nye. På ut-datasettet vil vi beholde variablene fra alle de tre inn-datasettene. Observasjonene spleises sammen ved hjelp av en identifikasjonsvariabel 'nr'. Siden variablene har samme navn på de tre inn-datasettene, må navnene endres for at alle variablene skal være med på det nye datasettet.

IN= er en nyttig option som brukes for å holde rede på hvilket eller hvilke datasett en observasjon på det nye datasettet stammer fra. Når en f.eks skriver IN=mars blir det laget en ny variabel med navn 'mars'. Variabelen 'mars' kan ha verdiene 0 (ikke sann) eller 1 (sann). Dersom en observasjon i det nye datasettet har mars=1, vil det si at datasettet 'mar' har bidratt med verdier til denne observasjonen.
I eksempelet over skal det merges på variabelen 'nr'. Helst skal tre observasjoner med samme 'nr' settes sammen til en observasjon. Men det kan jo tenkes at det forekommer observasjoner som ikke har 'makker' på de andre datasettene. For å finne ut slikt, brukes variablene som er laget med IN=options. Hvis vi i det nye datasettet f.eks kun vil ha med observasjoner som har bidrag fra alle de tre inn-datasettene, kan dette skrives slik:

```
DATA kvart_1 ;
    MERGE jan (RENAME=(antall=ant_jan total=tot_jan) IN=januar )
          feb (RENAME=(antall=ant_feb total=tot_feb) IN=februar )
          mar (RENAME=(antall=ant_mar total=tot_mar) IN=mars ) ;
    BY nr;
    IF januar AND februar AND mars ;
```

De fleste dsoptions foreligger også som egne statements i DATA-steget. Legg her merke til at som dsoptions får de i tillegg et = etter seg!

8.1.2 Options tilknyttet enkelte statements

Mange statements, særlig i PROC-steget, har knyttet til seg en mengde valgmuligheter, options. Disse står alltid beskrevet i manualen sammen med statementet. Ved å spesifisere options kan en f.eks få utført spesielle beregninger, eller man kan spesifisere en bestemt metode som skal brukes ved beregningene.

> Les om options tilknyttet de PROC-statements som skal brukes! Kanskje vil det finnes en option for nettopp ditt spesielle behov!

8.1.3 PROC OPTIONS

Når SAS installeres på en datamaskin, er det visse system-options som fastsettes. Disse kan f.eks være hvorvidt output fra PROC-steg skal sentreres på siden, antall linjer som skal skrives ut pr side i en utskrift, om det skal være lov med både store og små bokstaver osv. Disse system-opsjonene kan en finne verdiene til, ved å kjøre prosedyren PROC OPTIONS.

Hvis det er en eller flere av disse verdiene en vil endre i sine egne SAS-kjøringer, kan dette gjøres ved hjelp av OPTIONS-statementet.

8.1.4 OPTIONS-statementet

Dette er et statement som kan brukes til å endre verdiene av system-opsjonene. Endringene gjelder kun ens egen kjøring. OPTIONS-statementet er et meget omfattende statement, som kan brukes både innenfor DATA- og/eller PROC-steg, men også utenfor disse. En lister da opp de system-options en vil endre. Noen er på form f.eks CAPS/NOCAPS, mens andre krever en verdi oppgitt, f.eks PAGESIZE=antall .

Eksempel:

```
OPTIONS nocenter skip=3;
DATA ....;
PROC ....;
```

I dette eksempelet står OPTIONS-statementet aller først i SAS-programmet, før både DATA- og PROC-steget.

Hvis en ønsker spesielle verdier av noen system options hver gang en kjører SAS, kan en sette disse ved initieringen av SAS i EXEC-kortet i JCL-oppsettet:

```
//RK61LDA  JOB ......
//         EXEC SAS,OPTIONS='GEN=2 NOCENTER PS=46'
//   -
```

Dette eksempelet viser system-options som en bør merke seg: GEN=2 angir at det kun skal samles på 2 historiske generasjoner av dataset. For å unngå å sløse med plass, henstilles det til SAS-brukere å ha med denne opsjonen når SAS kalles!
Neste option angir at prosedyre-resultatene ikke skal sentreres på siden. Dette er en smaksak.
Den siste opsjonen angir antall linjer pr side i utskriften. Dette vil variere alt etter hva slags papir det er i den skriveren en bruker for å ta utskrift av resultatene.

8.2 PROC CONTENTS

Når man over tid har laget diverse SAS-datasett og lagret disse permanent på en fil, dukker før eller senere følgende spørsmål opp:

* Hvilke datasett finnes egentlig på lagringsfilen?
* Hva heter de forskjellige datasettene?
* Hvilke variable er med på datasettet 'nnn'?
* Hvor mye plass er igjen på lagringsfilen?

For å få svar på slike og liknende spørsmål, må en bruke prosedyren PROC CONTENTS.

Et eksempel:

Vi har lagret SAS-datasett på en fil med navn TS61.S6613.LDA.SASFIL. Men hvilke datasett ligger nå egentlig lagret her?

```
//
//LEVE   DD  DSN=TS61.S6613.LDA.SASFIL,DISP=OLD
//SYSIN  DD  *

PROC CONTENTS DATA=leve._ALL_ NODS ;
```

Ved å lage et oppsett som i dette eksempelet, får vi en liste over de datasettene som er lagret på filen. Siden det står _ALL_istedenfor et datasettnavn, får vi en liste over alle datasettene, med antall observasjoner, hvor stor plass de tar osv.

Legg merke til at det er med en option til PROC CONTENTS-statementet, nemlig NODS. Dette betyr at vi bare vil ha en oversikt over hvilke datasett som er lagret på filen, vi vil i dette tilfelle ikke ha med lange beskrivelser av de enkelte dataset.

Dersom en derimot ønsker å finne ut hvilke variable som egentlig er med på et spesielt dataset, f.eks et som heter 'region', kan en skrive følgende:

```
//
//LEVE   DD  DSN=TS61.S6613.LDA.SASFIL,DISP=OLD
//SYSIN  DD  *

PROC CONTENTS DATA=leve.region ;
```

8.3 PROC DATASETS

Hvis man f.eks etter å ha kjørt PROC CONTENTS vil slette datasett som er lagret permanent, MÅ dette gjøres i SAS-prosedyren PROC DATASETS.

> Når SAS-datasett skal slettes
> MÅ dette gjøres ved hjelp av
> PROC DATASETS !

Eksempel:

På filen med DD-navn 'gato' har man lagret mange SAS-datasett. Man ønsker så å slette datasettene 'menn', 'kvinner' og 'enslige':

```
//
//GATO   DD  DSN=TS61.S6911.LDA.SASFIL,DISP=OLD
//SYSIN  DD  *

PROC DATASETS DDNAME=gato ;
    DELETE menn kvinner enslige ;
```

I PROC DATASETS kan man blant mye annet også forandre navn på datasett.

8.4 Automatiske funksjoner i SAS

En automatisk funksjon er en rutine som returnerer en verdi utfra en eller flere oppgitte argumenter. SAS inneholder automatiske funksjoner av forskjellig slag: Noen har med dato eller tid å gjøre, noen er matematiske, noen statistiske og andre gjelder tekst.

Eksempel:

MAX-funksjonen returnerer den største verdien blant de oppgitte argumentene. I dette eksempelet blir en variabel 'dyrest' tilordnet den største verdien blant variablene 'pris1', 'pris2' og 'pris3':

```
dyrest = MAX(pris1, pris2, pris3);
```

En slik SAS-funksjon kan brukes overalt der en kan bruke et SAS-uttrykk, ikke bare i assignment-statements som i eksempelet over.

Et lite utvalg funksjoner:

ARITMETISKE:

ABS	MIN
DIM	MOD
MAX	SQRT

TRUNKERINGS:

CEIL	INT
FLOOR	ROUND

MATEMATISKE:

EXP	GAMMA
ERF	LOG

STATISTISKE:

CSS	USS	KURTOSIS
MEAN	N	NMISS
RANGE	STD	SKEWNESS
STDERR	SUM	VAR

Kapittel 6 i BASICS-manualen omhandler SAS-funksjonene.

8.5 Ta random-utvalg av store datasett

I SAS finnes en automatisk funksjon som trekker et tilfeldig utvalg av observasjonene. Størrelsen på utvalget bestemmer man selv. Se eksempel på neste side!

Eksempel:

Vi har et datasett 'stort' og vil lage et datasett 'lite' som skal bestå av ca 20% av observasjonene i 'stort'.

```
DATA lite;
    SET stort;
    IF UNIFORM(0) <= .20 ;
```

8.6 TITLEn-statementet

TITLEn-statementet brukes for å få overskrifter på utskriftene. Statementet kan skrives både i og utenfor PROC- eller DATA-steg. Dette er et nyttig statement som bør brukes flittig! Man kan ha opptil 10 linjer med overskrift. Hver linje krever et TITLE-statement. Man angir hvilken linje overskriften skal skrives på i fortløpende etter TITLE . Skriver man ikke noe tall, kommer overskriften på første linje. Overskriftene må stå i anførselstegn (') og inneholde maksimalt 132 tegn.

Eksempel:

```
PROC PRINT DATA= ... ;
       ...
TITLE  'INNTEKT OG FORMUESUNDERSØKELSEN 1986';
TITLE3 'Utskrift av deler av datamaterialet';
TITLE5 'Enslige kvinner over 40 år';
```

I dette tilfellet blir det tre linjer med overskrift, men med en blank linje mellom hver.

8.7 Informasjonsfil om SAS

PDS-memberet 'SSB2.SAS.DIV(info)' inneholder diverse informasjoner om SAS. Foruten rettelser og oppdateringer av dette heftet, vil man her finne tips om fremgangsmåter osv. Alle SAS-brukere oppfordres til å skrive inn SAS-erfaringer på dette memberet!

9. MACRO-SPRÅKET I SAS

Macro-språket er det som gjør SAS til et virkelig anvendelig verktøy innen databehandling og -analyse. Macro-språket kan beskrives som et språk som ligger utenpå eller rundt det vanlige SAS-språket.

```
Ved bruk av macroer må OPTIONS='MACRO'
være med ved initieringen av SAS !!
```

SAS' macro-språk brukes blant annet for å

* repetere grupper av vanlige SAS-statements
* håndtere DATA- og/eller PROC-steg
* gjøre program modulære

Macro-språket består av noen ganske få statements, som kjennes ved at de alltid har en % som første tegn.

Noen viktige macro-statements:

```
%MACRO              starter deklarasjonen av en macro
%MEND               avslutter deklarasjonen av en macro
%IF-%THEN/%ELSE     hvis betingelse, så ..., ellers ...
%DO-%END            utfør blokk
%DO %UNTIL-%END     utfør inntil betingelse ...
%DO %WHILE-%END     utfør mens betingelse ...
%DO mvar=  %TO      utfør skrittvis ...
%GLOBAL             definere global macrovariabel
%LOCAL              definere lokal macrovariabel
%LET                gi verdi til en macrovariabel
```

Innenfor SAS' macro-språk er det tre viktige begrep å holde styr på:

* macro
* macrostatement
* macrovariabel

En SAS-macro er en samling vanlige SAS-statements som er gitt et navn.

En macro er altså en del av et program, en programmodul. For oversiktens skyld skrives macroene som regel i begynnelsen programmet. En macro starter alltid med macrostatementet %MACRO og avsluttes alltid av macrostatementet %MEND. I %MACRO-statementet oppgis hva macroen skal hete. Når macroen senere skal kalles, skriver en bare macronavnet med en % foran. Hver gang macroen kalles, genereres de statementene macroen består av.

Dersom en del av et program skal gjentas flere ganger, kan det være lurt å legge denne delen i en macro. De stedene i programmet der denne delen skulle forekommet, kaller man så bare opp macroen.

Et enkelt eksempel:
I dette programmet lages det mange datasett, som alle skal skrives ut med PROC PRINT. Det skal også kjøres PROC MEANS på datasettene. Ved å legge disse statementene inn i en macro, slipper vi å skrive 'PROC PRINT' og 'PROC MEANS' mange ganger i programmet, vi kaller bare på macroen.

```
%MACRO skriv;       * ------------------------------------;
    PROC PRINT;     *          Her deklareres macroen,     ;
    PROC MEANS;     *          den skal ha navnet 'skriv'. ;
%MEND skriv;        * ------------------------------------;

DATA ... ;          *   <---  Her begynner hovedprogrammet. ;
     ...
%skriv;             *   <---  Her kalles macroen.          ;
DATA ... ;
     ...
%skriv;             *   <---  Her kalles macroen en gang til;
```

Dette var et eksempel på bruk av macro for å spare skrivearbeid. I praksis vil en macro sjelden bestå av så få som to statements, men for forståelsens skyld var denne macroen svært enkel.

En macrovariabel er av en annen type enn variablene som tilhører et SAS-datasett. En macrovariabel har alltid bare en verdi, i motsetning til en datasett-variabel som har en verdi for hver observasjon i datasettet. En macrovariabel kan beholde sin verdi gjennom flere DATA- og/eller PROC-steg. En macrovariabel kan være global eller lokal. En global macrovariabel kan refereres hvor som helst i SAS-programmet,

det være seg innenfor eller utenfor en macro. Er macrovariabelen derimot lokal, kan den bare refereres innenfor den macroen den er laget i. En macrovariabel tilordnes verdi i %LET-statementet. Når det skal refereres til verdien av en macrovariabel, må man skrive en & foran navnet til macrovariabelen.

Et eksempel på bruk av en macrovariabel:

```
      %***-------------------------------------------------*;
      %*  Først deklareres macroen, den heter 'delvis'. I    *;
      %*  macroen refereres det til en macrovariabel: 'fylke'*;
      %***-------------------------------------------------*;
%MACRO delvis;
      DATA;
          SET &fylke;
              ...
              ...  diverse statements

      PROC MEANS;
          TITLE "GJENNOMSNITTSTALL FOR &FYLKE ";
%MEND delvis;
      %***-------------------------------------------------*;
      %*  Så gis macrovariabelen fylke verdi:               *;
      %***-------------------------------------------------*;
%LET fylke=troms;
      %***-------------------------------------------------*;
      %*  Så kalles macroen. Siden macrovariabelen fylke nå  *;
      %*  har blitt tilordnet en verdi, er det datasettet med*;
      %*  navn 'troms' som kopieres. Verdien til macrovaria- *;
      %*  belen skrives også ut i tittelen på utskriften.    *;
      %***-------------------------------------------------*;
%delvis
      %***-------------------------------------------------*;
      %*  Macrovariabelen gis så ny verdi, og macroen kalles *;
      %*  en gang til:                                      *;
      %***-------------------------------------------------*;
%LET fylke=nordland;
%delvis
      %***-------------------------------------------------*;
      %*  Dette gjentas igjen og igjen:                     *;
      %***-------------------------------------------------*;
%LET fylke=finnmark;
%delvis
%LET fylke=buskerud;
%delvis
%LET fylke=akershus;
%delvis
%LET fylke=hedmark;
%delvis
%LET fylke=oppland;
%delvis
```

I et tilfelle som det ovenfor, kunne en ha benyttet en macro med parametre. I %MACRO-statementet oppgir en da foruten navnet på macroen også parametre som refereres som macrovariable senere i macroen. Når så macroen kalles, erstattes parametrene av de aktuelle verdiene:

```
%MACRO delvis(fylke) ;
    DATA;
        SET &fylke;
          ...
          ...  diverse statements

    PROC MEANS;
        TITLE "GJENNOMSNITTSTALL FOR &FYLKE ";
%MEND delvis;

%delvis(troms)
%delvis(nordland)
%delvis(finnmark)
%delvis(buskerud)
%delvis(akershus)
%delvis(hedmark)
%delvis(oppland)
```

Macrostatementene %MACRO og %MEND brukes for å deklarere en macro, mens %LET, %LOCAL og %GLOBAL brukes i forbindelse med macrovariable. Mange av de andre macrostatementene kan brukes for å håndtere DATA- og/eller PROC-steg, i eller utenfor en macro:

Eksempel på at en macro lager 11 datasett:

```
%MACRO dekade;
    %DO i=70 %TO 80;
        DATA aar_&i ;
            INFILE inn&i ;
            INPUT .....
                  ....
                  ....
        PROC SUMMARY;
             ...
            OUTPUT OUT=aggr_&i ;
    %END;
%MEND dekade;

%dekade
DATA tre;
    SET aar_70 aar_71 aar_72 aar_73 aar_74 aar_75
        aar_76 aar_77 aar_78 aar_79 aar_80 ;
```

Et annet eksempel:

```
%MACRO klasse1;
       ...   *-------------------------------------------*;
       ...   * Program som beregner skatt i skatteklasse 1 *;
       ...   *-------------------------------------------*;
%MEND klasse1;

%MACRO klasse2;
       ...   ***-----------------------------------------*;
       ...   * Program som beregner skatt i skatteklasse 2 *;
       ...   ***-----------------------------------------*;
%MEND klasse2;

DATA skatt;
    SET likning;
    IF estatus=1 THEN
        DO;
            %klasse1
        END;
    ELSE
        DO;
            %klasse2
        END;
```

Macroer kan gjøre et stort program mer oversiktlig. Ved å legge passende programbiter i macroer, reduseres hovedprogrammet til bare å bestå av kall på macroene:

Eksempel:

```
    %*-------------------------------------------------------*;
    %*  I dette programmet skal det beregnes skatt for en  *;
    %*  personlig skattyter.                                *;
    %*  Deklarerer først noen hensiktsmessig macroer:       *;
    %*-------------------------------------------------------*;
%MACRO fpensj;
    ...   *-----------------------------------------*;
    ...   * beregner folketrygdens pensjonsdel      *;
    ...   *-----------------------------------------*;
%MEND fpensj;
%MACRO fradra;
    ...   *-----------------------------------------*;
    ...   * beregner fradrag i inntekt              *;
    ...   *-----------------------------------------*;
%MEND fradra;
%MACRO komska;
   ...   *-----------------------------------------*;
   ...   * beregner kommuneskatt                   *;
   ...   *-----------------------------------------*;
%MEND komska;
%MACRO statska;
   ...   *-----------------------------------------*;
   ...   * beregner statsskatt                     *;
   ...   *-----------------------------------------*;
%MEND statska;
%MACRO fordel;
   ...   *-----------------------------------------*;
   ...   * beregner skatt til skattefordelingsfondet *;
   ...   *-----------------------------------------*;
%MEND fordel;
    %*-------------------------------------------------------*;
    %*  Her starter hovedprogrammet.                       *;
    %*  Beregner først folketrygdens pensjonsdel av brutto-*;
    %*  inntekten. Beregner så fradrag i inntekt, og får   *;
    %*  dermed nettoinntekt. Beregner så kommuneskatt,     *;
    %*  statsskatt og skatt til skattefordelingsfondet av  *;
    %*  nettoinntekten.                                    *;
    %*-------------------------------------------------------*;
DATA likning;
    SET ...
    %fpensj
    %fradra
    %komska
    %statsta
    %fordel
    ;
```

10. INTERAKTIV SAS

En SAS-jobb kan utføres på to måter: i batch eller interaktivt. Det vanligste i Byrået er å kjøre SAS i batch. Da må man foruten å skrive selve SAS-programmet også skrive et JCL-oppsett. Når dette er gjort, sender man jobben til eksekvering ved å skrive submit. Maskinen legger jobben i en kø, og når tiden er inne blir jobben utført. Når jobben er ferdig, kommer det en melding om dette på brukerens terminal.

Dersom man kjører SAS interaktivt, slipper man å skrive JCL-kort. Man slipper også å vente på at jobben skal bli gjort, for når man submitter en job interaktivt, blir den utført omgående. Både SASLOG og SASLIST kommer straks opp på skjermen.

Interaktiv SAS kan være en stor fordel f.eks når man skal teste et program. Det kan da spares mye tid ved at man slipper å vente hver gang en jobb skal bli kjørt. Men interaktiv SAS krever store maskinressurser. Det anbefales derfor å bruke små datamengder ved denne type kjøring og forøvrig bruke omhu.

Når man vil kjøre SAS interaktivt, velger man pkt S i hovedmenyen i ISPF. Man kan alternativt velge X i hovedmenyen og så skrive %sas når man får READY på skjermen. Etter en liten stund vil da SAS Display Manager System (heretter kalt DMS) komme frem på skjermen. Skjermen er delt i to, en del for SASLOG'en og en program-editor. I program-editoren gjelder stort sett de samme linjekommandoer som i editoren i ISPF pkt 2. De samme PF-tastene gjelder også, PF3 betyr END, PF2 betyr delt skjermbilde, PF5 betyr REFIND osv. Hvis man er misfornøyd med delingen av skjermbildet, man vil kanskje ha mindre plass til SASLOG'en og mer plass til programmet, er det bare å taste PF2 der man vil at skjermen skal deles.

Et SAS-program som skal kjøres interaktivt bør avsluttes med et RUN-statement (kun RUN;). Forøvrig er det ingen forskjell på et program som skal kjøres interaktivt og ett som skal kjøres i batch. Når man har skrevet SAS-programmet ferdig i editor-delen, skriver man sub i kommandofeltet. (Kommandofeltene i editoren og LOG'delen går like bra.) Jobbens SASLOG vil da komme rett opp på skjermen. Dersom pro-

grammet gir noen resultater, vil disse også komme direkte ut på skjermen. Både for program-editor, SASLOG og SASLIST gjelder PF7 og PF8 for å bla hhv bakover og forover. Når man er ferdig med å se på eventuelle resultater, kommer man tilbake til editoren med PF3.

Dersom man har gjort en feil i programmet, kan dette rettes opp på følgende måte: Ved å skrive RECALL i editorens kommandofelt, kommer siste program tilbake i editoren! Man kan så rette feilen og submitte programmet på ny.

For å få forrige program tilbake i editoren: Tast RECALL i kommandofeltet!

Dersom man lurer på hva de forskjellige PF-tastene betyr, kan man taste KEYS i kommandofeltet. En liste over hva PF-tastenes betyr vil da komme frem på skjermen. SAS DMS er beskrevet i kap. 14 i Basics-manualen. På slutten av kapitlet (fra side 485 og utover) er beskrevet de forskjellige kommandoene som kan brukes, både linje-kommandoene og kommandoene som må skrives i kommandofeltet.
Det er fullt mulig både å ta med seg ferdige programmer, permanente SAS-datasett og rådata inn i interaktiv SAS. Har man for eksempel et ferdig program liggende på et PDS-member, kan man kjøre dette interaktivt på to måter, vist med eksempler under. I eksemplene er programmet skrevet på PDS-memberet 'prog1' på brukerdatasettet 'sas.test' til brukeren 'lo61lda'.

Alternativ 1:

Man tar programmet med inn som en parameter ved initieringen av SAS. Programmet blir utført direkte. Man skriver:

```
%sas input(lo61lda.sas.test(prog1)) ↵
```

Ved denne fremgangsmåten går man automatisk ut av SAS når programmet er utført. Man havner altså ute i READY.

Alternativ 2:

Man går først inn i interaktiv SAS ved å skrive %sas. Programmet hentes så inn med et %INCLUDE-statement i selve SAS-programmet. En må da først ha knyttet forbindelse mellom SAS og brukerdatasettet der programmet ligger med et TSO-statement:

```
===> sub ↵

''''   tso alloc f(dd) da(sas.test) shr;
''''   %include dd(prog1); run;
```

I TSO-statementet allokeres brukerdatasettet lo61lda.sas.test. I f-parentesen oppgis hva som skal være datasettets dd-navn (dette er som vanlig med dd-navn valgfritt). I %INCLUDE-statementet hentes programmet inn. Datasettets dd-navn settes utenfor parentesen, memberets navn inni. Det hele avsluttes med et RUN-statement og submittes. Med denne fremgangsmåten vil man forbli inne i SAS.

Bemerk at når programmet tas med inn ved initieringen av SAS, må brukerens ident være med når datasett-navnet oppgis, mens når datasettet allokeres inne i SAS, kan identen utelates.

Foruten 'input', er det flere parametre en kan ha med ved initieringen av SAS, blant annet system options. Vil man f.eks bruke macroer og spare plass, kan man skrive:

```
%sas options('macro gen=1') ↵
```

Bemerk at det som står inni options-parentesen her må stå i fnutter!

Ønsker man å arbeide med et SAS-datasett som er lagret permanent, må man knytte forbindelse mellom filen SAS-datasettet er lagret på og SAS. Dette kan gjøres med et par setninger inne i editoren i inter-

SAS-datasettet 'menn' er lagret på filen TS61.S6221.LDA.SASFIL . Vi gir den dd-navnet 'person'. For å arbeide videre med SAS-datasettet 'menn', må en gå frem slik:

Først:

```
===> sub ↵

''''   TSO alloc f(person) da('TS61.S6221.LDA.SASFIL'); RUN;
```

Så:

```
===> sub ↵

''''   DATA kopi;
''''        SET person.menn;
''''        RUN;
```

Dersom man har allokert et datasett til bruk i interaktiv SAS (det være seg en lagringsfil for permanente SAS-datasett eller et bruker-datasett), og man senere ønsker å arbeide med datasettet utenfor interaktiv SAS uten å logge seg av, må man først frigjøre datasettet. Dette gjøres inne i SAS slik:

```
===> sub ↵

''''  TSO free f(ddnavn); RUN;
```

Når man ønsker å gå ut av interaktiv SAS skriver man BYE i kommando-feltet.

For å gå ut av interaktiv SAS: Tast BYE i kommandofeltet!

11. STIKKORDREGISTER