Interne notater

STATISTISK SENTRALBYRÅ

92/1

14. januar 1992

EASYTRIEVE PLUS

Av
Kristian Lønø

FORORD Oslo, 14/1-92

Dette heftet tar for seg de mest brukte sidene ved Easytrieve Plus. Det er forsøkt tilpasset SSB's bruk av programmet. Jeg har tatt for meg de sidene ved Easytrieve Plus som jeg anser for å være viktige for oss. Et slikt hefte vil uansett ikke kunne erstatte manualene, men jeg håper det kan være et godt grunnlag for forståelse av Easytrieve Plus. Jeg har lagt vekt på å bruke mange eksempler på program. Mange av disse kan du forhåpentligvis bruke som utgangspunkt for egne programmer.

For å få utbytte av notatet er det en forutsetning at du er fortrolig med SSB's stormaskin, spesielt editoren og SDSF (der du ser på kjørerapportene, valg 8 fra hovedmenyen). Videre må du vite hva JCL er, og du må vite hvordan du kjører en jobb.

Kristian Lønø
Gruppe for EDB,
Personstatistikk (GEP),
Oslo

INNHOLDSFORTEGNELSE

1. INNLEDNING

Easytrieve Plus er et filbehandlingsprogram. Det er det amerikanske firmaet Pansophic som står bak programmet, som finnes både på stormaskin og PC. Vi bruker nå versjon 6.0. Dette notatet dreier som om stormaskinutgaven, men kan tildels brukes også for PC-versjonen.

Programmet er meget bra på de fleste filbehandlingsoperasjoner. Rapportgeneratoren er OK på enkle tabeller, men kan vanskelig brukes til trykkeklare tabeller. Easytrieve Plus er maskineffektivt, og det har automatisk innlesing av data som standard, hvilket gjør programmeringen enklere.

Feilmeldinger ved avbrutte kjøringer er den største svakheten med programmet. Det kan ofte være meget vanskelig å finne ut hvorfor Easytrieve Plus avbryter et program. Meldingene programmet gir er som oftest meget kryptiske og det gir også samme melding på forskjellige feil. Det kan se ut som feilmeldingene er ment for profesjonelle Debuggere, og det har vi lite av i SSB. Programmet er dessuten følsomt for ikke-numeriske tegn i numeriske felt.

Det finnes 2 manualer til Easytrieve Plus:

- Reference Manual
- Application Guide

Videre finnes det en Student Guide, som er en opplæringsbok, og en Installation Guide for installering av programmet.

Easytrieve Plus kan startes med følgende JCL:

```
//O414KRLÆ JOB 8019,'KRL',MSGCLASS=X,CLASS=A,NOTIFY=O414KRL,REGION=6144K
//ISITRIV  EXEC PGM=EZTPA00,REGION=6144K
//STEPLIB  DD   DSN=SSB2.EASYPL60.LOAD,DISP=SHR
//SORTWK01 DD  UNIT=WORK,SPACE=(4096,500,,,ROUND)
//EZTVFM   DD  UNIT=WORK,SPACE=(4096,(100,200),,,ROUND)
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
```

Programmet skal skrives inn på linjene etter //SYSIN DD *
Alle filer du bruker i programmet må dessuten defineres i JCL.

Dette JCL-forslaget finnes på fila EZT.STD.MODULER(JCL). Under datasettene EZT.STD.MODULER og EZT.DIV finnes det forslag til diverse program som er gode å bruke som utgangspunkt for egen programmering.

2. EASYTRIEVE PLUS

2.1. Aktiviteter i Easytrieve Plus

- JOB
- SORT

JOB brukes for å starte en programmeringsaktivitet, mens SORT brukes for å sortere en fil. Vi kan ha så mange JOB- og SORT-aktiviteter vi vil i et program, men programmet blir fort uoversiktlig og komplisert hvis det blir mange av dem.

2.2. Instruksjoner i Easytrieve Plus

- FILE
- DEFINE
- IF, ELSE, END-IF
- DISPLAY
- PUT
- STOP
- MOVE
- MOVE LIKE
- SELECT
- SEARCH
- PRINT
- REPORT
- COPY
- PARM
- DO WHILE, END-DO
- GOTO
- PERFORM

*- GET
*- CALL

- MSTART, MACRO, MEND

Instruksjoner merket med * er ikke beskrevet i notatet.

3. REKKEFØLGEN AV INSTRUKSJONENE

I Easytrieve Plus er rekkefølgen av instruksjonene slik: Først eventuelle makroer. Deretter en eventuell PARM-instruksjon. Så kommer alle definisjonene. I definisjonsdelen inngår definisjoner av:

A) Alle FILER som det skal leses fra og skrives til.
B) Alle felt som skal brukes i programmet, både felt i tilhørende filer og arbeidsfelt.

Etter at alle definisjoner er gjort, er det klart for aktivitetene. Hver aktivitet starter enten med en JOB eller en en SORT-instruksjon. Vanligvis består et Easytrieve Plus program av en aktivitet.

Under JOB følger selve programmet, dvs. de instruksjonene som Easytrieve Plus skal utføre for hver lest record fra innfila (IF, MOVE, PUT, DISPLAY, osv.).

Bruker vi prosedyrer skal disse stå nederst i programmet vårt (de kalles opp med instruksjonen PERFORM).

Skal vi lage rapporter i programmet vårt, skal disse skrives etter selve programmet (rapporter kalles opp med instruksjonen PRINT) og etter eventuelle prosedyrer. Prosedyrer som kalles opp fra rapporter (BEFORE-BREAK, ENDPAGE osv.) skal stå rett etter den rapporten de hører til.

Eksempel på et program:

```
 * Programmet starter med en PARM-instruksjon!!
 PARM DEBUG (NOCLIST NODMAP FLOW NOXREF) LIST (NOPARM) SORT (MSG NO)
 * Her defineres filer og felt
 FILE INN
      INR              1       7   A
      KJØNN           27       1   A
      LIGGEDAGER      56       4   N 0
      TELLER-L         W       2   P 0 VALUE 0
      LIGG-PR-UTS      W       3   P 2 VALUE 0
 FILE DISP PRINTER ASA
 FILE RAPP PRINTER ASA
 * Her kommer selve programmet!!
 JOB INPUT INN START OVERSKRIFT
  IF LIGGEDAGER <= 0
   PERFORM FEIL-LIGGEDAGER
  ELSE
   PRINT RAPP
  END-IF
 * Her følger prosedyrene!!
 FEIL-LIGGEDAGER. PROC
    TELLER-L = TELLER-L + 1
    IF TELLER-L = 43
     PERFORM OVERSKRIFT
     TELLER-L = 0
    END-IF
    DISPLAY DISP 'RECORD: ' RECORD-COUNT ' MED INR: ' INR +
                 ' HAR UGYLDIGE LIGGEDAGER: ' LIGGEDAGER
 END-PROC
 OVERSKRIFT. PROC
  DISPLAY DISP NEWPAGE 'LISTE OVER OBSERVASJONER MED FEIL LIGGEDAGER'
  DISPLAY DISP SKIP 1
 END-PROC
 * Her kommer rapporten!!
 REPORT RAPP PRINTER RAPP NOADJUST NODATE SUMMARY
  SEQUENCE KJØNN
  CONTROL KJØNN
  TITLE 'KJØNNSFORDELING AV LIGGEDAGER'
  LINE KJØNN TALLY LIGGEDAGER LIGG-PR-UTS
 BEFORE-BREAK. PROC
  LIGG-PR-UTS ROUNDED = LIGGEDAGER / TALLY
 END-PROC
```

4. SYNTAKSREGLER

Når vi skal lage program i Easytrieve Plus, må vi følge visse retningslinjer. Easytrieve Plus har sine spesielle syntaksregler. Hvis vi ikke følger disse, vil Easytrieve Plus gi oss feilmeldinger når vi kjører programmet vårt. Vi vil ikke få utført programmet vårt før syntaksen er riktig.

Disse reglene gjelder:

- Vi må holde oss innenfor posisjon 1-72. Tekst utenfor posisjon 72 blir ignorert.
- Punktum (.) eller ny linje skiller instruksjonene
- Det skal være minst ett blankt tegn mellom alle ord, parametere og operander.
- Felt-navn kan være opptil 40 posisjoner. De kan inneholde alle tegn unntatt ' . , ( ) og :, og de må begynne med en bokstav fra A til Z eller et tall.
- En instruksjon som ikke får plass på en linje, kan fortsettes på neste. Da må vi ha fortsettelsestegn, enten + eller -
- Parenteser kan brukes til å gjøre aritmetiske uttrykk lettere å lese og til å endre rekkefølgen regneoperasjonene blir utført på.
- Aritmetiske operatorer (+ - * /) MÅ stå mellom to blanke tegn.
- Logiske relasjoner (= ^= < <= > >=) MÅ stå mellom to blanke tegn.
- Kommentarlinjer starter med en stjerne (*) i posisjon 1.

For oversiktens skyld bør vi aldri ha mer enn EN instruksjon pr. linje.

Bruker vi IF eller DO WHILE tester, bør vi rykke inn de instruksjonene som utføres inne i testen. Dette også for oversiktens skyld. Eksempel på et program som har flere instruksjoner pr. linje:

```
IF MATCHED. MOVE LIKE HOVED TO UTFIL. MOVE LIKE ALDER TO UTFIL
PUT UTFIL. ELSE. IF HOVED
DISPLAY DISPA 'KOMMUNE UTEN ALDERSFORDELING: ' KOMMNR. ELSE
DISPLAY DISPH 'KOMMUNE UTEN HELSEDATA  (F):  ' KOMMNR-A. END-IF. END-IF
```

Samme program, men med en instruksjon på hver linje.

```
IF MATCHED
MOVE LIKE HOVED TO UTFIL
MOVE LIKE ALDER TO UTFIL
PUT UTFIL
ELSE
IF HOVED
DISPLAY DISPA 'KOMMUNE UTEN ALDERSFORDELING: ' KOMMNR
ELSE
DISPLAY DISPH 'KOMMUNE UTEN HELSEDATA  (F):  ' KOMMNR-A
END-IF
END-IF
```

Samme program, nå også med innrykk:

```
IF MATCHED
    MOVE LIKE HOVED TO UTFIL
    MOVE LIKE ALDER TO UTFIL
    PUT UTFIL
ELSE
   IF HOVED
       DISPLAY DISPA 'KOMMUNE UTEN ALDERSFORDELING: ' KOMMNR
   ELSE
       DISPLAY DISPH 'KOMMUNE UTEN HELSEDATA  (F):  ' KOMMNR-A
   END-IF
END-IF
```

Easytrieve Plus har to fortsettelsestegn (tegn som sier at en instruksjon skal fortsette på neste linje), nemlig + og -. Hvis vi bruker + vil instruksjonen fortsette i første ikke-blanke posisjon på neste linje. Med - som fortsettelsestegn vil instruksjonen fortsette i posisjon 1 på den neste linjen.

Vi bør alltid ta oss tid til å skrive noen kommentarlinjer som forteller litt om hva programmet vårt gjør. Skriv gjerne også hvorfor vi programmerer slik vi gjør. Det er sikkert innlysende for oss når vi skriver programmet, men den dagen vi må rette eller oppdatere det er det slett ikke sikkert at det er like innlysende hvorfor vi gjorde som vi gjorde.

5. BETINGELSER

Betingelser brukes for å kunne foreta valg i et program. Det vil ofte være nødvendig å knytte betingelser til utførelsen av instruksjoner. Det er bare i instruksjonene IF og DO WHILE vi kan bruke betingelser.

En betingelse består av et subjekt, en sammenlignende operator og et objekt. Subjektet er enten et felt fra en fil eller et arbeidsfelt, objektet er et felt fra en fil, et arbeidsfelt, et tall, en karakterstreng (bokstaver) eller et aritmetisk uttrykk. Følgende sammenlignende operatorer finnes:

=	==> lik	Alternativ skrivemåte:	EQ
^=	==> ulik	"	NE el. NQ
<	==> mindre enn	"	LT el. LS
<=	==> mindre enn eller lik	"	LE el. LQ
>	==> større enn	"	GT el. GR
>=	==> større enn eller lik	"	GE el. GQ

Betingelser bindes sammen med AND eller OR. Vær forsiktig når vi bruker OR, bruk gjerne parentes for å vise hvilke betingelser som hører sammen. Tabellen under viser når betingelser med AND og OR blir oppfylt. A, B og C er betingelser og JA betyr at betingelsen er oppfylt, mens NEI betyr at den ikke er oppfylt. Hver linje viser forskjellige varianter av A, B og C:

A	B	C	A OR B OR C	A AND B AND C	A OR B AND C	(A OR B) AND C
JA	JA	JA	JA	JA	JA	JA
JA	JA	NEI	JA	NEI	JA	NEI
JA	NEI	JA	JA	NEI	JA	JA
JA	NEI	NEI	JA	NEI	JA	NEI
NEI	JA	JA	JA	NEI	JA	JA
NEI	JA	NEI	JA	NEI	NEI	NEI
NEI	NEI	JA	JA	NEI	NEI	NEI
NEI	NEI	NEI	NEI	NEI	NEI	NEI

Easytrieve Plus skiller mellom numeriske og alfanumeriske felt når det gjelder betingelser. Når alfanumeriske felt brukes som subjekt, vil objektet betraktes som alfanumerisk og sammenligningen skjer karakter for karakter. Objektet skal stå i fnutter enten det inneholder tall eller bokstaver. For numeriske felt som subjekt gjelder at objektet må inneholde tall, ellers vil programmet kunne skjære seg.

Eksempler:

```
 A-FELT1     W  3  A    VALUE 'SSB'
 A-FELT2     W  3  A    VALUE '306'
 N-FELT1     W  6  N 0  VALUE 306211
 N-FELT2     W  6  N 0  VALUE 211306

IF A-FELT1 = 'SSB'                 <== Oppfylt
IF A-FELT1 = 'ssb'                 <== Ikke oppfylt
IF A-FELT1 = ' SSB'                <== Ikke oppfylt
IF A-FELT1 >= ' SSB'               <== Oppfylt
IF A-FELT1 = 'SSB '                <== Oppfylt
IF A-FELT1 = 'SS'                  <== Ikke oppfylt
IF A-FELT1 > 'SS'                  <== Oppfylt
IF A-FELT1 <= 'SS'                 <== Ikke oppfylt
IF A-FELT1 < NFELT1                <== Oppfylt
IF A-FELT2 = 306                   <== Syntaksfeil
IF A-FELT2 = '00306'               <== Ikke oppfylt
IF A-FELT2 = N-FELT1               <== Ikke oppfylt
IF A-FELT2 = N-FELT2               <== Oppfylt
IF A-FELT2 > N-FELT1               <== Oppfylt
IF A-FELT2 > N-FELT2               <== Ikke oppfylt
IF N-FELT1 = A-FELT1               <== Syntaksfeil
IF N-FELT1 = A-FELT2               <== Syntaksfeil
IF N-FELT2 > N-FELT1 / 2           <== Oppfylt
```

Betingelser med intervaller kan skrives slik:

```
IF  FELT = 'A' THRU 'P'    <== oppfylt på alle verdier fra A til P
IF  FELTN = 1 THRU 9       <== oppfylt på alle verdier fra 1 til 9
```

Hvis verdiene ikke er i rekkefølge kan vi skrive slik:

```
IF KJØNNKODE = 1 3 5 7 9   <== oppfylt for verdiene 1, 3, 5, 7 og 9
```

Dette er mye bedre enn å skrive slik (som gir samme resultat):

```
IF KJØNNKODE = 1 OR KJØNNKODE = 3 OR KJØNNKODE = 5 OR KJØNNKODE = 7 +
   OR KJØNNKODE = 9
```

6. SYSTEMDEFINERTE FELT.

Et systemdefinert felt er et arbeidsfelt Easytrieve Plus definerer seg under utføringen av et program. Feltet brukes til å holde rede på forskjellige ting, f.eks. dagens dato, hvor mange records som er lest osv. Vi har 3 typer systemdefinerte felt:

- Generelle
- Filrelaterte
- Rapportrelaterte

Generelle felt:

SYSDATE ==> inneholder dagens dato (format: MM/DD/ÅÅ)
SYSTIME ==> inneholder tidspunktet Easytrieve Plus startet eksekveringen (format: TT/MM/SS)

Filrelaterte felt:

RECORD-LENGTH ==> inneholder lengden til den record som behandles.
RECORD-COUNT ==> inneholder recordnr til den record som behandles.

Rapportrelaterte felt:

LEVEL ==> inneholder nivået linja som skal skrives ut i rapporten har. For hvert felt som er med i CONTROL-instruksjonen dannes det et nivå i rapporten. På det laveste nivået er LEVEL = 0. Nivået over har LEVEL = 1. Slik øker LEVEL sin verdi med 1 for hvert nivå helt til totalsummen som har den høyeste verdien for LEVEL (1 mer enn det øverste nivået). Dette feltet bruker vi når vi skal gjøre forskjellige ting alt etter hvilket nivå i rapporten vi befinner oss.

TALLY ==> inneholder antall records fra innfila som blir aggregert til et forspaltekriterium.

I tillegg til at Easytrieve Plus bruker disse feltene for å holde rede på disse tingene, kan vi bruke disse feltene. Vi behøver ikke å definere dem i DEFINE-instruksjonen, i motsetning til andre arbeidsfelt. Vi gir dem heller ikke verdier, men bruker de verdiene programmet har gitt dem.

Eksempel:

```
 IF INN:RECORD-COUNT > 300
  STOP EXECUTE
 ELSE
  PUT UT FROM INN
 END-IF
```

Som vi ser bruker vi filnavnet og : foran feltet for å skille RECORD-COUNT til fila INN fra RECORD-COUNT til andre filer. Hver fil vi bruker i programmet har et felt med RECORD-COUNT og et med RECORD-LENGTH.

7. AKTIVITETENE

7.1. JOB

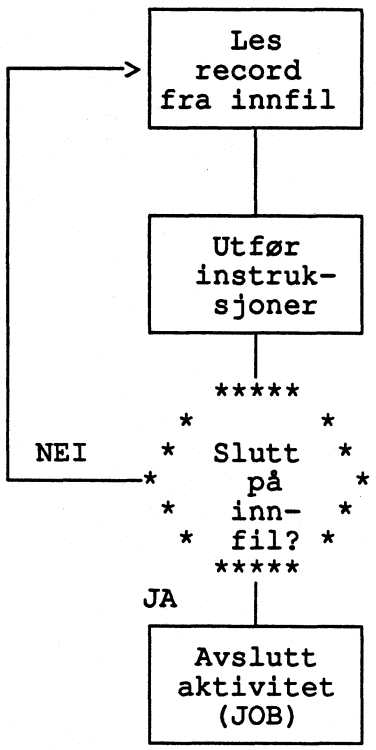
JOB er en instruksjon som starter en aktivitet som utfører filbehandling, lager rapporter, skriver feillister osv. Vanligvis forteller vi programmet hvilken innfil som skal brukes i aktiviteten (JOB). Dette gjøres slik:

```
JOB INPUT (INN)
```

Vi forteller med dette at programmet skal lese fila INN.

Easytrieve Plus benytter automatisk innlesing av innfila, det betyr at vi ikke trenger å bekymre oss for å åpne eller lukke innfila. Dessuten vil programmet avslutte aktiviteten når siste record på innfila er ferdig behandlet, slik at vi slipper å tenke på End-Of-File problematikk.

Den automatiske innlesingen Easytrieve Plus bruker, gjør programmeringen enklere. Programmet fungerer slik at det først leser en record fra innfila. Deretter utføres alle instruksjonene i programmet for denne recorden. Videre leses så neste record og instruksjonene utføres for denne. Slik fortsetter programmet til alle records i innfila er ferdig behandlet.



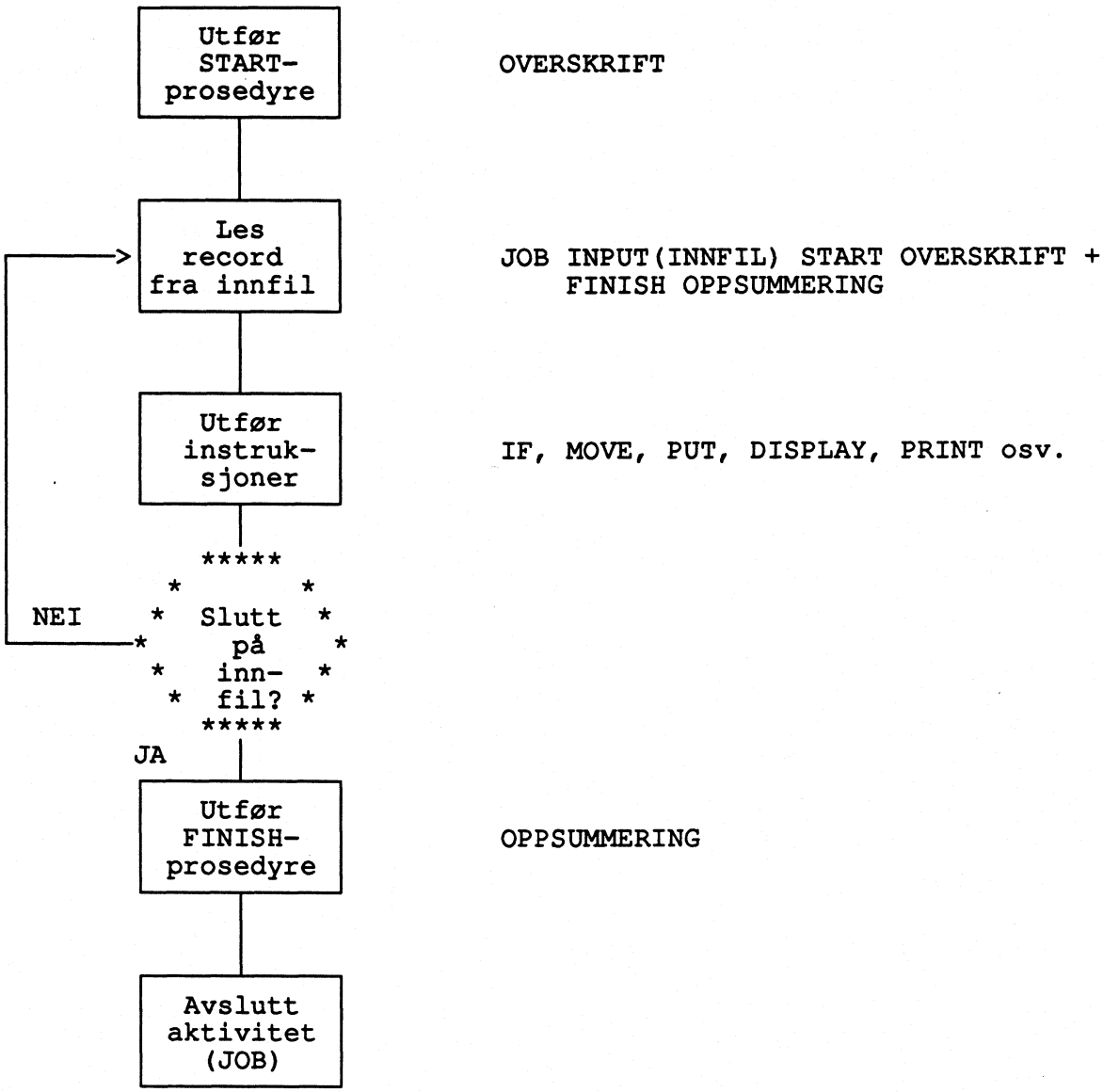

```
JOB INPUT(INNFIL)

IF, MOVE, PUT, DISPLAY, PRINT osv.
```

Det kan ofte være en fordel å få utført noen instruksjoner før programmet starter med lesing av innfila og etter at innfila er ferdig lest. Da må vi oppgi en Start og/eller en Slutt-prosedyre i JOB-instruksjonen.

JOB INPUT (INN) START OVERSKRIFT FINISH OPPSUMMERING

Prosedyrene (her OVERSKRIFT og OPPSUMMERING) må skrives nederst i programmet vårt. JOB-instruksjone sørger for at OVERSKRIFT blir utført før innfila leses, og OPPSUMMERING utført etter at innfila er ferdig behandlet.



Hvis vi ikke vil at programmet skal lese noen innfil, må vi skrive slik:

JOB INPUT (NULL)

Vi kan lese flere innfiler samtidig. Dette kalles synkronisert innlesing eller matching av filer (se dette).

7.2. SORT (med SELECT)

SORT er en egen aktivitet som brukes for å sortere filer. I forbindelse med SORT kan vi bruke instruksjonen SELECT til å selektere ut records under sorteringen. SELECT må være inne i en IF-instruksjon. Programmet selekterer ut de records som oppfyller betingelsen i IF-instruksjonen. SELECT kan bare brukes i SORT-aktiviteten.

Eksempel:

```
 SORT INN TO UT USING KOMMUNENR
```

Fila INN sorteres til fila UT etter KOMMUNENR.

Eksempel:

```
 SORT INN TO UT USING (FNR) BEFORE SELEKSJON
 SELEKSJON. PROC
  IF KOMMUNE = '0301'
   SELECT
  END-IF
 END-PROC
```

Fila INN sorteres til fila UT etter FNR. Bare Oslo (0301) tas med på fila UT.

Synkende sortering med seleksjon:

```
 SORT INN TO UT USING (ALDER D KJØNN) BEFORE SELEKSJON
 SELEKSJON. PROC
  IF KOMMUNE >= '0401' AND ALDER < 67
   SELECT
  END-IF
 END-PROC
```

Kan stoppe en sortering etter et visst antall records:

```
 SORT INN TO UT USING (ALDER D KJØNN) BEFORE SELEKSJON
 SELEKSJON. PROC
  IF RECORD-COUNT < 351
   SELECT
  ELSE
   STOP EXECUTE
  END-IF
 END-PROC
```

SORT bør ikke brukes på større filer fordi sortering internt i Easytrieve Plus krever mere ressurser enn å bruke SYNCSORT direkte (se vedlegg I). Når Easytrieve Plus sorterer en fil, gjøres det ved at Easytrieve Plus kaller opp SYNCSORT som foretar sorteringen.

Ved sorteringer bør vi ha med REGION i JCL, slik:

```
 //O414KRLÆ JOB 8019,'KRL',MSGCLASS=X,CLASS=A,NOTIFY=O414KRL,REGION=6144K
 //ISITRIV  EXEC PGM=EZTPA00,REGION=6144K
 //STEPLIB  DD   DSN=SSB2.EASYPL60.LOAD,DISP=SHR
```

Det kan også tenkes at vi trenger litt ekstra midlertidig plass under sorteringen, og det får vi med en slik linje i JCL:

```
//SORTWK01 DD  UNIT=WORK,SPACE=(4096,500,,,ROUND)
```

8. INSTRUKSJONENE

8.1. FILE

FILE brukes til å definere filer for Easytrieve Plus. Alle filer som skal leses eller skrives i et program, må være definert med hver sin FILE-instruksjon. Easytrieve Plus håndterer både sekvensielle filer og VSAM-filer.

Alle filene må ha unike navn. Disse navnene MÅ være de samme som DD-navnene filene har i JCL-en til programmet vårt. De må derfor følge JCL-syntaks (maks. 8 posisjoner, begynne med bokstav fra A-Z).

Programmet under viser kopiering fra VSAM-fil til sekvensiell fil

```
//0414KRLÅ JOB 8019,'KRL',MSGCLASS=X,CLASS=A,NOTIFY=0414KRL
//         EXEC PGM=EZTPA00
//STEPLIB  DD  DSN=SSB2.EASYPL60.LOAD,DISP=SHR
//INN1     DD  DSN=TK414.S8019.KRL.VSAMFIL,DISP=OLD
//SYSPRINT DD  SYSOUT=*
//UTFIL    DD  SYSOUT=TK414.S8019.KRL.SEKFIL,DISP=OLD
//SYSIN    DD  *
FILE INN1  VS
FILE UTFIL
JOB
 PUT UTFIL FROM INN1
```

For å fortelle Easytrieve Plus at fila vi skal lese er en VSAM-fil, skriver vi VS etter filnavnet i FILE-instruksjonen. Easytrieve Plus vil nå lese VSAM-fila sekvensielt.

De sekvensielle filene kan deles inn i:

- Datafiler
- Utskriftsfiler
- Oppslagsfiler

Datafiler er vanligvis produksjons- eller test-datasett som inneholder statistiske data. Disse har et antall records som alle inneholder de samme felt, og feltene er plassert i de samme posisjonene i alle recordene.

Utskriftsfiler er filer som lages for eventuelt å bli skrevet ut på papir senere. Disse skal ha en styrekarakter til skriveren i første posisjon. De skal ha en recordlengde på maks. 133 posisjoner (inkludert styrekarakteren). Til utskriftsfiler sender vi tabeller, feillister, feilmeldinger osv. Utskriftsfiler definerer vi ved å skrive PRINTER ASA etter filnavnet (se neste eksempel).

Oppslagsfiler er som navnet tilsier filer som vi i løpet av programmet skal slå opp i og hente opplysninger fra. Når vi skal bruke filer som oppslagsfiler må vi fortelle Easytrieve Plus det. Det gjør vi ved å skrive TABLE etter filnavnet i FILE-instruksjonen (se neste eksempel). Omkodingskataloger er typiske oppslagsfiler.

Neste eksempel:

```
//0414KRL  JOB 8019,'LE NEUX',MSGLEVEL=(0,0),
//           MSGCLASS=X,CLASS=A,NOTIFY=0414KRL
//STEP1    EXEC PGM=EZTPA00
//STEPLIB  DD   DSN=SSB2.EASYPL60.LOAD,DISP=SHR
//FOLK     DD DSN=PP414.S8019.TABKURS.DATAFIL2,DISP=SHR
//KOMMTXT  DD DSN=TAB.DIV(KOMMTXT),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//LISTE    DD SYSOUT=*
//SYSIN    DD *
***-------------------------------------------
* DATAFILA
***-------------------------------------------
FILE FOLK
     KOMMUNENR          5      4    N
     BEFOLKNING         9      6    N 0 MASK 'ZZZ ZZ9'
***-------------------------------------------
* UTSKRIFTSFILA
***-------------------------------------------
FILE LISTE PRINTER ASA
***-------------------------------------------
* OPPSLAGSFILA
***-------------------------------------------
FILE KOMMTXT TABLE 466
     ARG                1      4    N
     DESC               6     19    A
*
     KOMMUNETEKST       W     19    A   HEADING 'NAVN'
*
JOB INPUT FOLK
       SEARCH KOMMTXT WITH KOMMUNENR GIVING KOMMUNETEKST
       PRINT INNBYGGERE
*
REPORT INNBYGGERE NOADJUST NODATE SUMMARY SUMCTL DTLCOPY PRINTER LISTE
    SEQUENCE BEFOLKNING D
    CONTROL KOMMUNENR
    TITLE 1 'KOMMUNER I NORGE ETTER ANTALL INNBYGGERE. 1985'
    LINE 1  KOMMUNENR KOMMUNETEKST BEFOLKNING
```

Oppslagsfiler kan enten ligge på en egen fil utenfor programmet (slik som i eksemplet over) eller som en del av programmet. Når den ligger på en egen fil må vi angi maks. antall records (466 i eksemplet over).

Hvis vi vil ha oppslagsfila inne i programmet gjøres det slik:

```
FILE KOMMTXT TABLE INSTREAM
     ARG                1      4    N
     DESC               6     19    A
0101 Halden ...........
0102 Sarpsborg ........
  :
  osv
  :
2028 Båtsfjord ........
2030 Sør-Varanger .....
ENDTABLE
```

Oppslagsfila må ha definert to felt. Det første er ARG. Dette feltet skal inneholde søkeargumentet for oppslaget. Det andre skal hete DESC. Herfra hentes dataene vi trenger fra oppslagsfila. I eksemplet over er kommunenummer ARG mens kommunetekst, som er opplysningene vi skal bruke, er DESC.

Oppslagsfila MÅ være sortert stigende på ARG-feltet!

Oppslagsfila MÅ IKKE inneholde dubletter (flere records med samme verdi for ARG-feltet)!

Av og til hender det at vi har bruk for filer som kun skal brukes i programmet vårt. Det vil si at disse filene lages i løpet av programmet, de brukes av det samme programmet og de blir slettet etter at programmet er kjørt. Disse kalles midlertidige filer.

Midlertidige filer må også defineres for programmet. Vi må også fortelle Easytrieve Plus om en fil er midlertidig. Det gjør vi ved å skrive VIRTUAL etter filnavnet. I tillegg må vi oppgi recordformat, lengde og eventuel blokkstørrelse fila skal ha.

Aktuelle recordformater:

- F (fast recordlenge)
- V (variabel recordlenge)
- FB (fast recordlengde, blokket)
- VB (variabel recordlengde, blokket)

De midlertidige filene trenger vi ikke å definere i JCL. Det er fordi disse filene bare brukes internt av Easytrieve Plus. Derfor må vi oppgi recordformat, lengde (og blokkstørrelse) til midlertidige filer i programmet.

Når vi skal lage lage nye, ikke midlertidige filer i Easytrieve Plus, må vi også oppgi recordformat, lengde og eventuell blokkstørrelse. Dette kan vi gjøre enten i JCL eller i programmet, helst i JCL. Oppgis de både i programmet og i JCL vil det som står i programmet overstyre det som står i JCL'en.

Samme eksempel som tidligere, men med innlagt sortering av datafila. Den sorterte fila legges ut på en midlertidig fil som brukes senere i programmet.

```
//0414KRL  JOB 8019,'LE NEUX',MSGLEVEL=(0,0),
//           MSGCLASS=X,CLASS=A,NOTIFY=0414KRL
//STEP1    EXEC PGM=EZTPA00
//STEPLIB  DD   DSN=SSB2.EASYPL60.LOAD,DISP=SHR
//FOLK     DD DSN=PP414.S8019.TABKURS.DATAFIL2,DISP=SHR
//KOMMTXT  DD DSN=TAB.DIV(KOMMTXT),DISP=SHR
//EZTVFM   DD  UNIT=WORK,SPACE=(4096,(100,200),,,ROUND)
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//LISTE    DD SYSOUT=*
//SYSIN    DD *
***-------------------------------------------
* DATAFILA
***-------------------------------------------
FILE FOLK
     KOMMUNENR          5       4    N
     BEFOLKNING         9       6    N 0 MASK 'ZZZ ZZ9'
***-------------------------------------------
* UTSKRIFTSFILA
***-------------------------------------------
FILE LISTE PRINTER ASA
***-------------------------------------------
* OPPSLAGSFILA
***-------------------------------------------
FILE KOMMTXT TABLE 466
     ARG                1       4    N
     DESC               6      19    A
***-------------------------------------------
* MIDLERTIDIG FIL (SORTERT DATAFIL)
***-------------------------------------------
FILE MIDLFOLK VIRTUAL FB (15 3000)
     COPY FOLK
*
     KOMMUNETEKST       W      19    A   HEADING 'NAVN'
*
SORT FOLK TO MIDLFOLK USING KOMMUNENR
JOB INPUT FOLK
       SEARCH KOMMTXT WITH KOMMUNENR GIVING KOMMUNETEKST
       PRINT INNBYGGERE
*
REPORT INNBYGGERE NOADJUST NODATE SUMMARY SUMCTL DTLCOPY PRINTER LISTE
    SEQUENCE BEFOLKNING D
    CONTROL KOMMUNENR
    TITLE 1 'KOMMUNER I NORGE ETTER ANTALL INNBYGGERE. 1985'
    LINE 1  KOMMUNENR KOMMUNETEKST BEFOLKNING
```

En fil som er definert som midlertidig i Easytrieve Plus vil vanligvis bli slettet etter at den er lest. Skal den midlertidige fila leses flere ganger, må vi ha med en ekstra parameter i fildefinisjonen. Den heter RETAIN (behold) og sørger for at den midlertidige fila ikke blir slettet før hele programmet er utført. Det betyr at midlertidige filer med RETAIN kan leses flere ganger i det samme programmet.

Eksempel på fileinstruksjon for midlertidig fil som skal leses flere ganger i løpet av programmet:

```
FILE MIDLFOLK VIRTUAL RETAIN FB (15 3000)
```

8.2. DEFINE

DEFINE brukes til å definere datafelt. Disse datafeltene kan enten høre til en record eller de kan være arbeidsfelt (felt som ikke hører til en record). Alle felt som defineres må ha angitt lengde og type. Lengden er antall posisjoner og type kan velges blant disse:

- A (Alfanumerisk, max lengde 32767)
- N (Numerisk, max lengde 18)
- P (Pakket desimal, max lengde 10)
- B (Binært, max lengde 4)
- U (Pakket desimal uten fortegn (unsigned), max lengde 9)

De numeriske feltene (N, P, B, U) kan ha desimaler. Binære felt (B) kan ha 0-10 desimaler, de andre kan ha 0-18 desimaler. Skriv antall desimaler etter felttypen (se eksempel under).

For de feltene som tilhører en record, må vi oppgi startposisjonen feltet skal ha. Arbeidsfelt har ingen startposisjon (fordi de ikke er relatert til noen record), isteden må vi fortelle programmet at feltet er et arbeidsfelt. Det gjør vi ved å skrive W istedenfor startposisjon.

Felt som tilhører samme record, må ha forskjellige navn. Arbeidsfelt må ha unike navn.

Eksempel under:

```
FILE FOLK
 DEFINE   FNR          1    11  N  0
 DEFINE   KOMMUNE     12     4  A
 DEFINE   HØYDE       16     3  N  2
 DEFINE   ETTERNAVN   19    20  A
*
 DEFINE   KJØNN        W     1  A
```

I dette eksemplet er det definert en fil ved navn FOLK. Denne fila har 4 felt definert; FNR som starter i posisjon 1, er 11 posisjoner langt og dessuten numerisk med 0 desimaler. Videre er det KOMMUNE som starter i posisjon 12, er 4 posisjoner langt og alfanumerisk definert. Så HØYDE med start i posisjon 16, lengde 3 posisjoner, numerisk definert med 2 desimaler. Til slutt er ETTERNAVN definert. Det starter i posisjon 19, har lengde på 20 posisjoner og er alfanumerisk. I tillegg til denne fila er det definert et arbeidsfelt, KJØNN. Det har W istedenfor startposisjon, er 1 posisjon og alfanumerisk.

I DEFINE-instruksjonen kan vi utelate DEFINE! Resultatet blir det samme, men programteksten vil se slik ut:

```
FILE FOLK
          FNR          1    11  N  0
          KOMMUNE     12     4  A
          HØYDE       16     3  N  2
          ETTERNAVN   19    20  A
*
          KJØNN        W     1  A
```

Det var kort om DEFINE-instruksjonen med de viktigste parametrene. Med mer avansert bruk av Easytrieve Plus vil vi sikkert få bruk for noen av disse:

- HEADING
- MASK
- VALUE
- OCCURS

HEADING brukes til å lage kolonneoverskrifter til rapporter. Når vi skal skrive ut et felt i en rapport, vil det settes av en kolonne til feltet, og det er overskriften til denne kolonnen vi lager med HEADING. Hvis vi ikke bruker HEADING, vil felt-navnet bli kolonneoverskrift. HEADING kan også skrives som en instruksjon under REPORT.

```
FILE FOLK
          FNR            1    11  N  0 HEADING ('FØDSELSNR')
          KOMMUNE       12     4  A    HEADING ('KOMMUNE-' 'NUMMER')
JOB
PRINT
REPORT
TITLE 'UTLISTING AV RECORDS'
LINE  FNR KOMMUNE
```

MASK brukes til å lage masker til numeriske felt. Disse maskene vil bli brukt i REPORT og DISPLAY.

```
FILE FOLK
          FNR            1    11  N  0 MASK '99/99-99 99999'
          KOMMUNE       12     4  A
JOB
PRINT
REPORT
TITLE 'UTLISTING AV RECORDS'
LINE  FNR KOMMUNE
```

Hvis vi har en person med fødselsnr 30036247243 så vil det bli slik i rapporten som lages i programmet over: 30/03-62 47243. Dette gjør det som vi ser lettere å lese! Masken vi lager kan godt være lenger enn det feltet er definert, men antall siffer i masken skal være lik lengden til feltet.

Med VALUE kan vi gi et arbeidsfelt en initialverdi (dvs. en verdi feltet har når programmet starter).

Eksempel:

```
TELLER    W  4 P VALUE 1
TEKST     W 10 A VALUE 'STARTTEKST'
```

Som vi ser må verdier til alfanumeriske felt stå i fnutter

OCCURS bruker vi når vi skal lage indekserte felt. Indekserte felt vil si at en record eller en del av en record består av felt som er like lange og inneholder samme type data.

Hvorfor indeksere felt? Hvis den samme operasjon skal gjøres på mange "like" felt, kan vi utføre operasjonene mye enklere ved å bruke indekserte felt. La meg vise dette med et program. Programmet skal erstatte alle blanke tegn med 0 (tallfelt som inneholder blanke kan nemlig skape problemer for Easytrieve Plus). De blanke tegnene kan dukke opp hvorsomheltst i fila. Programmet blir slik uten indeksering:

```
//0414KRL  JOB 8019,'LE NEUX',MSGLEVEL=(2,0),
//         MSGCLASS=X,CLASS=A,NOTIFY=0414KRL,TIME=1
//NULLTULL EXEC PGM=EZTPA00
//STEPLIB  DD DSN=SSB2.EASYPL60.LOAD,DISP=SHR
//INNFIL   DD *
1 1
 2
33
4
  7
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//UTFIL    DD SYSOUT=*
//SYSIN    DD *
FILE INNFIL
   INN1          1   1 A
   INN2          2   1 A
   INN3          3   1 A
FILE UTFIL F 3
   UT1           1   1 A
   UT2           2   1 A
   UT3           3   1 A
JOB INPUT (INNFIL)
          IF INN1 = ' '
             UT1 = '0'
          ELSE
              UT1 = INN1
          END-IF
          IF INN2 = ' '
             UT2 = '0'
          ELSE
              UT2 = INN2
          END-IF
          IF INN3 = ' '
             UT3 = '0'
          ELSE
              UT3 = INN3
          END-IF
   PUT UTFIL
```

Som vi ser må vi gjøre IF-testen for hvert eneste felt (dvs. hver eneste posisjon) i recorden. Hvis recorden var lengre enn 3, og det er den jo som oftest, ville det bli fryktelig mye unødvendig skriving.

I slike situasjoner bruker vi OCCURS. Da trenger vi bare å skrive selve testen en gang. Riktignok må vi ha med et arbeidsfelt som fungerer som teller, og putte hele testen inn i en DO WHILE-løkke. Men programmet blir mye bedre og dessuten mer fleksibelt. Hvis recordlengden ikke er 3, men 1476, må vi i det første eksemplet lage 1476 IF-tester, mens ved å bruke indeksering (som i eksemplet under) trenger vi bare å endre alle 3-tall til 1476. Eksemplet med OCCURS kommer her (resultatet av de to programmene vil bli akkurat det samme).

```
//0414KRL  JOB 8019,'LE NEUX',MSGLEVEL=(2,0),
//         MSGCLASS=X,CLASS=A,NOTIFY=0414KRL,TIME=1
//NULLHULL EXEC PGM=EZTPA00
//STEPLIB  DD DSN=SSB2.EASYPL60.LOAD,DISP=SHR
//INNFIL   DD *
1 1
 2
33
4
  7
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//UTFIL    DD SYSOUT=*
//SYSIN    DD *
FILE INNFIL
   INN           1   1 A OCCURS 3
FILE UTFIL F 3
   UT            1   1 A OCCURS 3
   TELLER        W   4 P 0
*
JOB INPUT (INNFIL)
   TELLER = 1
     DO WHILE TELLER <= 3
         IF INN(TELLER)= ' '
            UT(TELLER) = '0'
         ELSE
             UT(TELLER) = INN(TELLER)
         END-IF
         TELLER = TELLER + 1
      END-DO
   PUT UTFIL
END
```

Når TELLER er lik 1 og vi sier at vi skal behandle INN(TELLER), vil programmet behandle det første av de indekserte feltene. Når TELLER er 2 vil INN(2) behandles osv. Det betyr at når vi bruker OCCURS, må vi også ha definert arbeidsfelt som skal holde rede på tellingen av de indekserte feltene.

8.3. IF, ELSE OG END-IF

IF brukes når en instruksjon bare skal utføres hvis en betingelse er oppfylt.

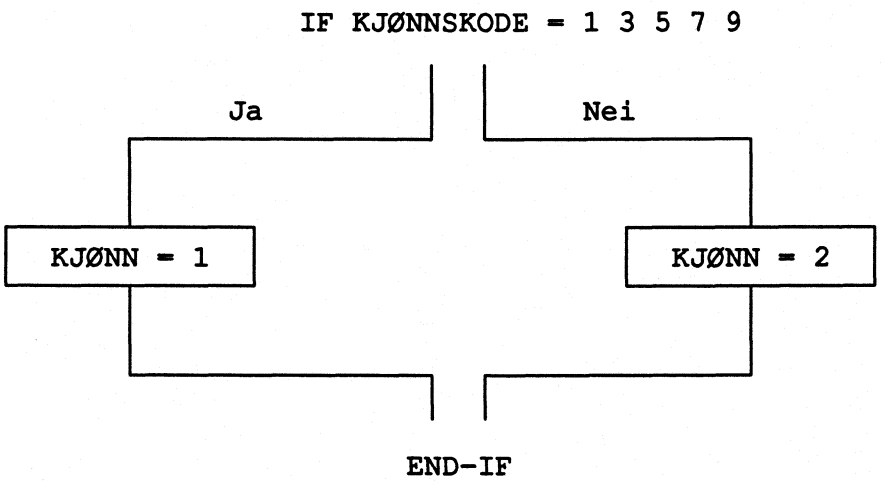
Syntaks:

```
IF betingelse(r)
 instruksjon(er)
ELSE
 instruksjon(er)
END-IF
```

Hvis betingelsen slår til utføres instruksjonen(e) etter IF, hvis ikke utføres instruksjonen(e) etter ELSE. Det er frivillig å ta med ELSE, men END-IF må alltid være med.

Eksempel:

```
IF KJØNNSKODE = 1 3 5 7 9
 KJØNN = 1
ELSE
 KJØNN = 2
END-IF
```



Vi kan ha IF inne i en annen IF. Dette kalles nestede IF-instruksjoner. Vi kan neste så mange IF vi vil, men jo flere vi nester jo lettere mister vi oversikten over det vi gjør.

8.4. DISPLAY

DISPLAY kan brukes til å:

- Lage feillister
- Sjekke om IF eller DO WHILE-tester har slått til
- Liste ut tellerverdier og andre resultater ved avslutning av programmet

Det vi velger å DISPLAYe har vanligvis blitt skrevet til SYSPRINT (samme sted som programmet kommer ut). Dette bør vi unngå. Isteden bør vi skrive det til en eller flere utskriftsfiler. Da kan vi lett velge (i JCL) om DISPLAYs skal skrives til SYSOUT (kjørerapporten) eller til en fil.

Hva kan vi DISPLAYe? Vi DISPLAYer tekster og feltverdier (både arbeidsfelt og fra filer). Tekstene vi DISPLAYer brukes til å fortelle hvilket felt vi DISPLAYer. Disse må stå i fnutter ('tekst').

Det som skal DISPLAYes bør alltid skrives til en utskriftsfil. Utskriftsfila defineres både i JCL og i programmet:

```
//DISP     DD SYSOUT=*
:
FILE DISP PRINTER ASA
:
JOB INPUT (INN)
:
DISPLAY DISP 'FNR ' FNR ' MED RECNR ' INN:RECNR ' FINNES IKKE I KATALOGEN!'
:
```

Her DISPLAYes det til utskriftsfila DISP. Teksten "FNR ", feltet FNR, teksten " MED RECNR ", feltet INN:RECORD-COUNT og teksten " FINNES IKKE I KATALOGEN!" hver gang DISPLAY-instruksjonen utføres.

Når vi DISPLAYer har vi mulighet til å legge inn blanke linjer. Disse kommer foran det vi DISPLAYer. Til dette bruker vi SKIP og et tall som forteller hvor mange blanke linjer vi skal ha. Programmet vil sette inn en printkarakter i posisjon 1 på utskriftsfila.

```
DISPLAY DISP SKIP 1 'FEIL I RECORD NR ' INN:RECORD-COUNT '!!!!'
```

Vi kan også få satt inn printkarakter for sideskift når vi DISPLAYer. Da bruker vi NEWPAGE.

```
DISPLAY DISP NEWPAGE 'OVERSIKT OVER FEIL'
```

Til plassering av tekster og/eller felt kan vi bruke COL. Dessuten kan vi forskyve plassering ved å bruke +n eller -n (n er et heltall). +n og -n kan ikke plasseres foran første felt eller konstant som skal DISPLA'es. Det vil gi syntaksfeil.

```
DISPLAY DISP 'FEIL I RECORD NR:' COL 30 INN:RECORD-COUNT +1 'PÅ INNFILA!'
```

Programmet på neste side vise mange eksempler på bruk av DISPLAY.

```
//O414KRLÆ JOB 8019,'KRL',MSGCLASS=X,CLASS=A,NOTIFY=O414KRL
//         EXEC PGM=EZTPA00
//STEPLIB  DD   DSN=SSB2.EASYPL60.LOAD,DISP=SHR
//INN1     DD   *
10 A GEPARD
11 A
12   EKORN
//SYSPRINT DD  SYSOUT=*
//DISP1    DD  SYSOUT=*
//DISP2    DD  SYSOUT=*
//SYSIN    DD  *
FILE INN1
   KODE          1   2 A HEADING 'KODE INN1'
   NR-1          4   1 A
   RASE          6  15 A HEADING 'RASE INN1'
FILE DISP1 PRINTER ASA          . * Utskriftsfil for 1. feilliste
FILE DISP2 PRINTER ASA          . * utskriftsfil for 2. feilliste
     T1          W   2 N VALUE 0 . * Teller antall feil til liste 1
     T2          W   2 N VALUE 0 . * Teller antall feil til liste 2
     DISP1TELL   W   2 N VALUE 0 . * Teller for sideskift til liste 1
     DISP2TELL   W   2 N VALUE 0 . * Teller for sideskift til liste 2
JOB INPUT (INN1) START OVERSKRIFT FINISH OPPSUMMERING
* Tester om nr-1 er blank. Hvis den er det skrives det til feilliste 1
IF  NR-1 SPACE
  DISP1TELL = DISP1TELL + 1
  IF DISP1TELL = 43
     DISPLAY DISP1 NEWPAGE
     PERFORM OVERSKR1
     DISP1TELL = 0
  END-IF
  DISPLAY DISP1 'NR-1 IKKE UTFYLT FOR POST NUMMER' +
                +1 RECORD-COUNT +1 COL 55 'KODE:' +1 KODE
  T1 = T1 + 1
END-IF
* Tester om rase er blanke. Hvis den er det skrives det til feilliste 2
IF RASE SPACES
  DISP2TELL = DISP2TELL + 1
  IF DISP2TELL = 43
     DISPLAY DISP2 NEWPAGE
     PERFORM OVERSKR2
     DISP1TELL = 0
  END-IF
  DISPLAY DISP2 'RASENAVN IKKE UTFYLT FOR POST NUMMER ' +
                 RECORD-COUNT COL 55 'KODE: ' KODE
  T2 = T2 + 1
END-IF
*  Prosedyrer som brukes til å lage overskrifter
OVERSKRIFT. PROC
 DISPLAY DISP1 'OVERSIKT OVER DE POSTENE SOM MANGLER NR-1.'
 DISPLAY DISP1 SKIP 1
 DISPLAY DISP2 'OVERSIKT OVER DE POSTENE SOM MANGLER RASENAVN.'
 DISPLAY DISP2 SKIP 1
END-PROC
OVERSKR1. PROC
 DISPLAY DISP1 'OVERSIKT OVER DE POSTENE SOM MANGLER NR-1. (FORTS.)'
 DISPLAY DISP1 SKIP 1
END-PROC
OVERSKR2. PROC
 DISPLAY DISP2 'OVERSIKT OVER DE POSTENE SOM MANGLER RASENAVN. (FORTS.)'
 DISPLAY DISP2 SKIP 1
END-PROC
* Prosedyre som skriver ut antall feil hver feilliste inneholder
OPPSUMMERING. PROC
 DISPLAY DISP1 SKIP 1 'ANTALL POSTER SOM MANGLER NR-1:' COL 50 T1
 DISPLAY DISP2 SKIP 1 'ANTALL POSTER SOM MANGLER RASENAVN:' COL 50 T2
END-PROC
```

8.5. PUT

Instruksjonen PUT brukes til å skrive til en fil. Det er to måter å gjøre det på. Den enkleste er PUT fil2 FROM fil1. Denne syntaksen gjør at fil2 får den samme filbeskrivelsen som fil1. Vi kopierer altså fil1 til fil2.

Eksempel:

```
FILE INN
FILE UT
JOB INPUT (INN)
PUT UT FROM INN
```

Dette er vel omtrent det korteste programmet vi kan lage i Easytrieve Plus. Programmet kopierer fil INN til fil UT. Legg merke til at vi ikke behøver å definere et eneste felt i de to filene.

Dette enkle programmet er det sjelden vi får bruk for. Derimot hender det oftere at vi skal lage en ny fil som består av noen spesielt utvalgte poster (records) fra innfila. Da kan vi bruke et program a la programmet under. Vi trenger fortsatt ikke å definere felt som ikke benyttes til testing.

```
FILE INN
   KOMMNR       6  4 N
FILE UT
JOB INPUT (INN)
IF KOMMNR = 0301 THRU 1154
    PUT UT FROM INN
END-IF
```

Vær klar over at utfila her får samme filbeskrivelse som innfila!

Hvis vi ønsker at utfila vi skal lage skal ha en annen filbeskrivelse enn innfila, må vi gjøre en omredigering. Da kan vi ikke bruke PUT UT FROM INN, for med PUT ... FROM ... kan vi ikke omredigere postene.

For å omredigere må vi først flytte verdiene fra feltene i den ene fila til feltene i den nye. Dette kan blant annet gjøres med MOVE. Deretter bruker vi PUT, slik som i eksemplet under.

```
FILE INN
   KOMMNR       6  4 N
   DIV          7 12 A
   DIVDIV      25 32 A
FILE UT
   KOMMNR       1  4 N
   DIVDIV       5 32 A
   DIV         37 12 A
JOB INPUT (INN)
IF KOMMNR = 0301 THRU 1154
    MOVE LIKE INN TO UT
    PUT UT
END-IF
```

Vi kan skrive til flere filer i det samme programmet. Disse filene kan ha forskjellige filbeskrivelser. Nytt eksempel:

```
FILE INN
   KOMMNR       6  4 N
   BYLAND       8  1 N
   DIV          7 12 A
   DIVDIV      25 32 A
FILE BYER
   KOMMNR       1  4 N
   DIVDIV       5 32 A
   DIV         37 12 A
FILE LANDKOMM
   KOMMNR       1  4 N
   DIV          5 12 A
JOB INPUT (INN)
IF BYLAND = 0
    MOVE LIKE INN TO BYER
    PUT BYER
ELSE
    MOVE LIKE INN TO LANDKOMM
    PUT LANDKOMM
END-IF
```

Det er altså bare to måter å bruke PUT på:

- PUT fil2 FROM fil1 (Kopiering)
- PUT fil (Omredigering/ny fil)

8.6. STOP og STOP EXECUTE

Med STOP-instruksjonen avslutter vi en aktivitet (JOB, SORT). Har vi bare en aktivitet i programmet vårt, avsluttes det. STOP brukes når vi bare skal ibehandle records i begynnelsen av en fil. Når alle records vi er interessert i er ferdig behandlet, avslutter vi aktiviteten med STOP. Videre kan STOP benyttes for å avslutte et program vi ikke vil skal fortsette hvis en bestemt situasjon (betingelse) oppstår. Hvis du ikke benytter automatisk innlesing av innfila (JOB INPUT NULL), må du bruke STOP for å avslutte programmet.

Eksempel.

```
* Program som lister ut 100 første records fra en fil
FILE INN
 REC 1 113 A
FILE DISP PRINTER ASA
JOB INPUT INN
 IF RECORD-COUNT <= 100
  DISPLAY DISP REC
 ELSE
  STOP
 END-IF
```

Programmet avsluttes etter 100 leste records. Så lenge vi bare er interessert i de 100 første records, stopper vi når disse er ferdig behandlet. Det sparer oss for gjennomlesing av de resterende records på fila (kan være utrolig mange) og jobben blir ferdig mye fortere.

Hvis en jobb har flere aktiviter og vi ønsker å avslutte hele jobben før alle aktivitetene er utført, bruker vi STOP EXECUTE. Aktiviteter som måtte komme etter at STOP EXECUTE er utført vil ikke bli gjort. Når STOP brukes vil aktiviteten avsluttes, og programmet vil starte på den neste.

8.7. MOVE og assignment

Instruksjonen MOVE brukes til å flytte data fra et felt til et annet. Flyttingen som blir utført er såkalt alfanumerisk, hvilket har betydning hvis lengden på feltene er forskjellige.

```
Frafelt > Tilfelt ==> Tilfeltet kuttes fra høyre det antall
                      karakterer det er kortere enn frafeltet
Frafelt < Tilfelt ==> Tilfeltet fylles med det antall blanke
                      karakterer tilfeltet er lengre enn
                      frafeltet til høyre for (etter) dataene.
```

Dette gjelder selv om feltene er definert som numeriske!

Vi kan bruke parameteren FILL for å fylle på med en spesiell karakter hvis tilfeltet er lengre enn frafeltet.

Eksempel:

```
ALFA-10   W  10 A
ALFA-2    W   2 A VALUE 'AB'
                                          Resultat:
MOVE '*' TO ALFA-10 FILL '*'              '**********'
MOVE ALFA-2 TO ALFA-10                    'AB        '
MOVE ALFA-2 TO ALFA-10 FILL '*'           'AB********'
MOVE ALFA-10 TO ALFA-2                    'AB'
```

Assignment instruksjonen gir en verdi til et felt, enten ved en regneoperasjon, eller ved en flytting av data.

Flytting av data foregår slik:

TILFELT = FRAFELT

Data flyttes fra FRAFELT til TILFELT

En regneoperasjon lager en numerisk verdi ved å bruke en eller flere av disse:

```
*     (multiplikasjon)
/     (divisjon)
+     (addisjon)
-     (subtraksjon)
```

Resultatet av en regneoperasjon legges ut i et felt. Dette feltet må være numerisk definert. Vi kan regne med felt som er definert numeriske (type N, P, B eller U). I tillegg til at de må være definert som numeriske, må de inneholde tall, ellers vil Easytrieve Plus avbryte programmet.

En regneoperasjon utføres i en denne rekkefølgen:

1. Multiplikasjoner og divisjoner
2. Addisjoner og subtraksjoner

Når det er flere multiplikasjoner og/eller divisjoner, vil disse bli utført fra venstre mot høyre. Etter at alle multiplikasjoner og divisjoner er utført, vil addisjoner og subtraksjoner bli utført, også fra venstre mot høyre. Eksemplet på neste side viser dette bedre enn ord.

```
FILE DISP3 PRINTER ASA
    N1            W 3 N 0 VALUE 4
    N2            W 3 N 0 VALUE 48
    N3            W 3 N 0 VALUE 8
    N4            W 3 N 0 VALUE 11
    N5            W 3 N 0
 JOB INPUT (NULL)
   N5 ROUNDED = N4 + 5 * N3 - N2 / 16 + N1
   DISPLAY DISP3 'N5 BLIR NÅ:' COL 15 N5
   STOP EXECUTE
```

Regneoperasjonen vi skal utføre er: N5 = N4 + 5 * N3 - N2 / 16 + N1. Slik vil regningen bli utført:

```
1.        N5 = 11 + 5 * 8 - 48 / 16 + 4
                    -----

2.        N5 = 11 +  40   - 48 / 16 + 4
                          -------

3.        N5 = 11 +  40   -    3    + 4
               --------

4.        N5 =    51      -    3    + 4
                  ----------------

5.        N5 =          48          + 4
                        ---------------

6.        N5 =                  52
```

Hvis vi bruker parenteser vil det som står inne i dem vil bli utført først.

```
 FILE DISP3 PRINTER ASA
    N1            W 3 N 0 VALUE 4
    N2            W 3 N 0 VALUE 48
    N3            W 3 N 0 VALUE 8
    N4            W 3 N 0 VALUE 11
    N5            W 3 N 1
 JOB INPUT (NULL)
   N5 ROUNDED = N4 + 5 * ((N3 - N2) / 16 + N1)
   DISPLAY DISP3 'N5 BLIR NÅ:' COL 15 N5
   STOP EXECUTE

   1.        N5 = 11 + 5 * ((8 - 48) / 16 + 4)
                           --------

   2.        N5 = 11 + 5 * (  -40    / 16  + 4)
                           -----------

   3.        N5 = 11 + 5 * (       -2.5    + 4)
                                  -----------

   4.        N5 = 11 + 5 *             1.5
                       -------------------

   5.        N5 = 11 +           7.5
                  -----------------

   6.        N5 =       18.5
```

> NB! Ta alltid med ROUNDED når vi utfører divisjoner.
> Hvis vi utelater ROUNDED vil resultatet av divisjonen bli kuttet, ikke avrundet.

Når Easytrieve Plus utfører en regneoperasjon med desimaltall, vil lengden og antall desimaler resultatet får avhenge av hvilken operasjon som er utført. Tabellen under viser reglene Easytrieve Plus bruker:

Regneoperasjon	Desimaler	Lengde
RES = FELT1 + FELT2	Det samme som den av FELT1 og FELT2 som har flest desimaler	En mer enn det feltet som er lengst
RES = FELT1 - FELT2	Det samme som den av FELT1 og FELT2 som har flest desimaler	En mer enn det feltet som er lengst
RES = FELT1 * FELT2	Summen av antall desimaler i FELT1 og i FELT2	Summen av lengden av FELT1 og FELT2
RES = FELT1 / FELT2	Den største av: a: Antall desimaler i FELT1 minus antall desimaler i FELT2 b: Antall desimaler i RES minus 1	Samme som FELT1

Ved mellomresultater kan lengden være på 30 tegn. Hvis resultatfeltet er definert kortere enn det Easytrieve Plus bruker i sine regneoperasjoner, vil feltet bli kuttet etter at alle regneoperasjonene er utført. Kuttingen skjer fra venstre, bortsett fra ved multiplikasjon. Da blir overflødige desimaler kuttet fra høyre først, så kuttes det fra venstre hvis nødvendig.

Ved Assignment med alfanumeriske felt kan ikke regneoperasjoner utføres, kun flytting av data fra et felt til et annet er lov. Reglene ved flyttingen er som for MOVE (se dette).

8.8. MOVE kontra assignment av tall til numeriske variable

Når vi bruker MOVE vil det alltid bli flyttet alfanumerisk. Ved å bruke assignment vil det kunne variere, alt etter hvordan til-felt og fra-felt er definert og hvilke verdier de har (Se tabellen under).

Fra		Til	Type flytting
Type	Verdi	Type	
Alfanumerisk	Alfabetisk	Alfanumerisk	Alfanumerisk
Alfanumerisk	Numerisk	Alfanumerisk	Numerisk
Konstant	Alfabetisk	Alfanumerisk	Alfanumerisk
Konstant	Numerisk	Alfanumerisk	Alfanumerisk
Alfanumerisk	Alfanumerisk	Numerisk	Flytting ikke lov
Alfanumerisk	Numerisk	Numerisk	Flytting ikke lov
Numerisk	Alfanumerisk	Numerisk	Abend
Numerisk	Numerisk	Numerisk	Numerisk
Konstant	Alfabetisk	Numerisk	Flytting ikke lov
Konstant	Numerisk	Numerisk	Numerisk

En alfanumerisk flytting fylles fra venstre mot høyre. Eventuelle ledige posisjoner fylles ut med blanke og eventuell avkutting (trunkering) skjer mot høyre.

Ved numerisk flytting fylles feltet fra høyre mot venstre og eventuelle uutfylte posisjoner fylles med nuller. Avkutting skjer mot venstre.

8.9. MOVE LIKE

Hvis vi har mange felt som skal flyttes fra en fil til en annen, kan det lønne seg å bruke MOVE LIKE. Med denne instruksjonen flytter vi alle felt som har like navn på begge filene. Vær forsiktig med å flytte når tilfelt er kortere enn frafelt (se MOVE). Reglene for flytting er som ved assignment (se dette).

Eksempel (alfanumerisk definerte felt på innfila):

```
 PARM DEBUG (NOCLIST NODMAP FLOW NOXREF) LIST (NOPARM) SORT (MSG NO)
 FILE INNFIL
   N1       1      3  A
   N2       4      4  A
   N3       8      3  A
   N4      11      4  A
 FILE UTFIL F 16
   N1       1      2  A
   N2       3      5  A
   N3       8      2  A
   A4      11      2  A
 JOB INPUT (INNFIL)
     MOVE LIKE INNFIL TO UTFIL
     PUT UTFIL
```

Hvis N1 = '123', N2 = '4567', N3 = '891' og N4 = '234' på innfila blir feltene på utfila slik: N1 = '12', N2 = '4567 ', N3 = '89' og A4 = ' '.

Eksempel (numerisk definerte felt på innfila):

```
 PARM DEBUG (NOCLIST NODMAP FLOW NOXREF) LIST (NOPARM) SORT (MSG NO)
 FILE INNFIL
   N1       1      3  N 0
   N2       4      4  N 0
   N3       8      3  N 0
   N4      11      4  N 0
 FILE UTFIL F 16
   N1       1      2  A
   N2       3      5  N 0
   N3       8      2  N 0
   A4      11      2  N 0
 JOB INPUT (INNFIL)
     MOVE LIKE INNFIL TO UTFIL
     PUT UTFIL
```

Hvis N1 = 123, N2 = 4567, N3 = 891 og N4 = 234 på innfila blir feltene på utfila slik: N1 = '23', N2 = 04567, N3 = 91 og A4 = 00.

8.10. SEARCH

Search er en meget nyttig instruksjon. Den henter data fra en oppslagsfil og legger resultatet i et arbeidsfelt eller på et felt til en fil. Før vi kan bruke SEARCH må vi ha definert oppslagsfila med FILE og DEFINE.

Eksempel:

```
 FILE LANDKAT TABLE 200
  ARG       1    3  A
  DESC      4   20  A
 FILE INN
  FNR       1   11  A
  LAND     12    3  A
 FILE UT
  FNR       1   11  A
  LANDTXT  12   20  A
 FILE DISP PRINTER ASA
 JOB INPUT (INN)
  MOVE LIKE INN TO UT
  SEARCH LANDKAT WITH LAND GIVING LANDTXT
  IF NOT LANDKAT
   MOVE LAND TO LANDTXT FILL ' '
   DISPLAY DISP 'LAND ' LAND ' IKKE FUNNET FOR RECORD NR: ' INN:RECORD-COUNT
  END-IF
  PUT UT
```

For å si at det er en tabell vi definerer, må vi oppgi TABLE og maks. antall records det er i tabellen. Tallet vi oppgir kan være større enn det faktiske antall, men ikke mindre.

```
FILE LANDKAT TABLE 200
```

LANDKAT defineres som oppslagsfil (TABLE) med maks. 200 records i eksemplet over.

Når vi har definert en oppslagsfil, må vi definere to felt; ARG og DESC. ARG er søkebegrepet, DESC er dataene som skal hentes fra oppslagsfila. Disse to feltene **skal** hete ARG og DESC. Fila **MÅ** være sortert stigende på ARG-feltet, og **IKKE** inneholde dubletter.

Selve oppslaget utføres med SEARCH-instruksjonen:

```
SEARCH LANDKAT WITH LAND GIVING LANDTXT
```

Her slås det opp i fila LANDKAT med LAND som søkebegrep, og resultatet legges i feltet LANDTXT. Programmet vil prøve å søke i oppslagsfila for å finne en record der ARG har den samme verdi som LAND. Når dette er tilfelle flyttes verdien DESC har for den samme recorden i oppslagsfila til feltet LANDTXT. Hvis verdien av LAND ikke finnes i oppslagfila, beholder LANDTXT sin verdi fra forrige oppslag.

Vi bør alltid ha med en IF-test som sjekker om oppslaget har vært vellykket, dvs. at argumentet finnes i oppslagsfila. Hvis det ikke finnes, må vi nemlig selv sørge for å legge noe i feltet (LANDTXT i eksemplet over), ellers beholder feltet verdien fra forrige oppslag. Videre bør vi (som i eksemplet over) lage en feilliste når oppslaget ikke har vært vellykket.

Et oppslag i en oppslagsfil vil (så lenge oppslagsfila ikke er for stor) være raskere enn å matche de to filene. Dessuten trenger bare oppslagsfila (den minste) å være sortert.

8.11. PRINT OG REPORT.

Instruksjonen PRINT bruker for å gi beskjed om å lage en rapport. Denne rapporten må defineres senere i programmet med REPORT-instruksjonen (se denne). Hver gang Easytrieve Plus-programmet utfører PRINT-instruksjonen, vil det behandle dataene slik rapporten er definert i REPORT-instruksjonen.

REPORT brukes for å lage rapporter. I REPORT-instruksjonen definerer vi rapportens utseende. For å få rapporten slik vi vil bruker vi følgende instruksjoner (kan bare brukes under REPORT):

- SEQUENCE
- CONTROL
- SUM
- TITLE
- HEADING
- LINE

De av instruksjonene vi velger å bruke, MÅ stå i samme rekkefølge som over her. For SEQUENCE, CONTROL OG SUM skal det bare være en av hver, mens det kan være flere av TITLE, HEADING og LINE.

Til selve REPORT-instruksjonen er det mange parametere. De viktigste er:

- rapportnavn
- PRINTER filnavn (styrer rapporten til utskriftsfil)
- PAGESIZE antall linjer (sidestørrelse i linjer)
- LINESIZE antall posisjoner (linjebredde)
- NOADJUST (hindrer Easytrieve Plus å justere kolonnene i forhold til arkets bredde)
- NODATE (dagens dato skrives ikke ut i overskriften)
- SUMMARY (summerer felt som ikke er forspaltekriterier)

SEQUENCE bruker vi til å fortelle hvordan rapporten vår skal sorteres. Vi kan sortere flere felt enten stigende eller synkende (gjerne ett felt stigende og et annet synkende).

CONTROL brukes når vi vil aggregere felt. De felt vi ramser opp i CONTROL-instruksjonen vil bli rapportens forspaltekriterier (dvs. alle records med samme verdi i de feltene som er med i CONTROL summeres til en linje. Det blir altså en linje for hver kombinasjon av forspaltekriteriene. Dessuten vil det bli egne sumlinjer for alle nivåer av forspalten). Det første feltet i CONTROL blir det høyeste nivået, mens det siste blir det laveste).

SUM brukes når vi skal ha ut summer av bare noen få av de aggregerte feltene i rapporten (og det skal vi jo sjelden). Det skrives bare ut summer for de feltene som er med i SUM-instruksjonen. Utelates SUM får vi summer på alle de aggregerte feltene i rapporten (hvis SUMMARY er med).

TITLE må vi bruke for å få overskrift på rapporten vår. Maksimalt antall overskriftslinjer er 99. Vi må ha minst 1 TITLE-instruksjon i rapporten vår.

HEADING gir overskrifter til kolonner i rapporten. Hvert felt i rapporten kan få sin HEADING. Hvis HEADING ikke er oppgitt i DEFINE-instruksjonen eller i rapporten, blir feltnavnet overskrift).

LINE er for å si hvilke felt som skal skrives ut i rapporten og i hvilken rekkefølge de skal komme.

Det kan hende at den plassen som er avsatt for å lage rapporten er for liten, og i så fall setter vi inn denne linja i JCL:

```
//EZTVFM   DD  UNIT=WORK,SPACE=(4096,(100,200),,,ROUND)
```

Eksempel på en enkel rapport:

```
 PARM DEBUG (NOCLIST NODMAP FLOW NOXREF) LIST (NOPARM) SORT (MSG NO)
 FILE INNFIL
      INR                 5       7   A
      KOMMNR             33       4   A
      KJØNN              47       1   A
      ALDER              64       3   A
 FILE RAPP PRINTER ASA
 JOB INPUT INNFIL
  IF INR = '1877577'
   PRINT RAPP
  END-IF
  REPORT RAPP PRINTER RAPP NOADJUST
   TITLE 1 ' '
   TITLE 2 'OVERSIKT OVER PASIENTER UTSKREVET FRA RIKSHOSPITALET (1877577)'
   LINE INR KJØNN ALDER KOMMNR
```

Eksempel på et program som lager to aggregerte rapporter:

```
 PARM DEBUG (NOCLIST NODMAP FLOW NOXREF) LIST (NOPARM) SORT (MSG NO)
 FILE INN
      FYLKE              27       2   A
      KOMMUNE            27       4   A
      LIGGEDAGER-MENN    44       4   N 0
      LIGGEDAGER-KVIN    66       4   N 0
 FILE RAPP1 PRINTER ASA
 FILE RAPP2 PRINTER ASA
 JOB INPUT INN
  IF FYLKE = '02' THRU '04'
   PRINT RAPP1
  END-IF
  IF FYLKE = '20'
   PRINT RAPP2
  END-IF
  REPORT RAPP1 PRINTER RAPP1 NOADJUST SUMMARY
   SEQUENCE FYLKE
   CONTROL FYLKE
   TITLE 1 ' '
   TITLE 2 'ANTALL LIGGEDAGER FORDELT PÅ FYLKE FOR AKERSHUS, OSLO OG HEDMARK'
   HEADING TALLY ('ANTALL' 'OBSERVASJONER')
   HEADING LIGGEDAGER-MENN ('MENN')
   HEADING LIGGEDAGER-KVIN ('KVINNER')
   LINE FYLKE TALLY LIGGEDAGER-MENN LIGGEDAGER-KVINNER
  REPORT RAPP2 PRINTER RAPP2 NOADJUST SUMMARY
   SEQUENCE KOMMUNE
   CONTROL KOMMUNE
   TITLE 1 ' '
   TITLE 2 'ANTALL LIGGEDAGER FOR KOMMUNENE I FINNMARK'
   HEADING LIGGEDAGER-MENN ('MENN')
   HEADING LIGGEDAGER-KVIN ('KVINNER')
   LINE KOMMUNE LIGGEDAGER-MENN LIGGEDAGER-KVINNER
```

I forbindelse med REPORT har Easytrieve Plus noen spesialprosedyrer som kan kalles opp. Disse kan brukes til å modifisere rapportens utseende og innhold under oppbyggingen av rapporten.

- RECORD-INPUT (brukes til å tilpasse dataene som skal brukes i rapporten)
- BEFORE-LINE (brukes til å pynte på rapporten før en detaljlinje skrives ut, dataene i linja kan ikke endres her. Kun for rapporter uten CONTROL-instruksjon)
- AFTER-LINE (brukes til å pynte på rapporten etter en detaljlinje er skrevet ut, dataene i linja kan ikke endres her. Kun for rapporter uten CONTROL-instruksjon)
- BEFORE-BREAK (brukes til å regne ut prosenter og til å legge sammen felt i rapporten. Utføres på detaljlinja før den skrives ut (i rapporter som har med CONTROL-instruksjonen))
- AFTER-BREAK (brukes til å pynte på rapporten etter at totallinjer er skrevet ut)
- ENDPAGE (brukes for å lage fotnoter)

- TERMINATION (brukes for å lage slutt-tekst på rapporten)

Av disse prosedyrene er det først og fremst BEFORE-BREAK og ENDPAGE som er nyttige. De andre kan vi glemme i første omgamg ihvertfall.

Eksempel på bruk av BEFORE-BREAK og ENDPAGE:

```
 PARM DEBUG (NOCLIST NODMAP FLOW NOXREF) LIST (NOPARM) SORT (MSG NO)
 FILE INN
      FYLKE              27       2   A
      KOMMUNE            27       4   A
      LIGGEDAGER-MENN    44       4   N 0
      LIGGEDAGER-KVIN    66       4   N 0
 FILE RAPP1 PRINTER ASA
      LIGGEDAGER          W       8   N 0 VALUE 0
      LIGGEDAGER-M-PST    W       4   N 2 VALUE 0
      LIGGEDAGER-K-PST    W       4   N 2 VALUE 0
 JOB INPUT INN
  IF FYLKE = '02' THRU '04'
   PRINT RAPP1
  END-IF
  REPORT RAPP1 PRINTER RAPP1 NOADJUST SUMMARY
   SEQUENCE FYLKE
   CONTROL FYLKE
   TITLE 1 ' '
   TITLE 2 'ANTALL LIGGEDAGER FORDELT PÅ FYLKE FOR AKERSHUS, OSLO OG HEDMARK'
    HEADING LIGGEDAGER-MENN ('MENN')
    HEADING LIGGEDAGER-KVIN ('KVINNER')
    HEADING LIGGEDAGER ('BEGGE KJØNN')
    HEADING LIGGEDAGER-M-PST ('MENN I PROSENT' 'AV TOTALEN')
    HEADING LIGGEDAGER-K-PST ('KVINNER I PROSENT' 'AV TOTALEN')
   LINE FYLKE LIGGEDAGER LIGGEDAGER-MENN LIGGEDAGER-KVINNER +
        LIGGEDAGER-M-PST LIGGEDAGER-K-PST
 BEFORE-BREAK. PROC
  LIGGEDAGER = LIGGEDAGER-MENN + LIGGEDAGER-KVIN
  LIGGEDAGER-M-PST ROUNDED = LIGGEDAGER-MENN * 100 / LIGGEDAGER
  LIGGEDAGER-K-PST ROUNDED = LIGGEDAGER-KVIN * 100 / LIGGEDAGER
 END-PROC
 ENDPAGE. PROC
  DISPLAY '-----------------------'
  DISPLAY 'FORELØPIGE TALL'
 END-PROC
```

Legg merke til at selv om rapporten styres til utskriftsfil (RAPP1), skal vi ikke angi filnavnet i DISPLAY-instruksjonen i disse prosedyrene, det vil GI syntaksfeil.

8.12. COPY

COPY-instruksjonen brukes for å kopiere filbeskrivelsen til en fil. Dataene på fila blir IKKE kopiert.

```
FILE INN
 FØDSELSDATO     1  6 A
 PERSONNR        7  5 A
 INNTEKT        12  8 N 0
 FORMUE         20  8 N 0
FILE UT
 COPY INN
```

Fila UT får her samme filbeskrivelse som inn. Vi skiller feltene i de to filene på denne måten:

```
INN:FØDSELSDATO er fødselsdato fra fila INN
UT:FØDSELSDATO er fødselsdato fra fila UT
```

Vi kan definere nye felt for en fil selv om vi har brukt COPY

```
FILE INN
 FØDSELSDATO     1  6 A
 PERSONNR        7  5 A
 INNTEKT        12  8 N 0
 KOMMNR         20  4 A
FILE UT
 COPY INN
 KJØNN          20  1 A
```

Fila UT vil inneholde alle felt fra fila INN samt feltet KJØNN.

8.13. PARM

Vi har muligheter til å påvirke programmets omgivelser. Det innebærer blant annet at vi selv kan bestemme hvor mye informasjon som skal komme ut på kjørerapporten. Stort sett får vi mye informasjon som vi ikke har bruk for, tfordi den er beregnet på debuggere og programmerere som liker å grave seg dypt ned i interne registre og andre skumle steder. For de som ikke vil ha ut mer informasjon enn nødvendig, foreslås følgende PARM-instruksjon:

```
PARM DEBUG (NOCLIST NODMAP FLOW NOXREF) LIST (NOPARM) SORT (MSG NO)
```

De parametrene som starter med NO, sørger for at kjørerapporten blir mindre.

Parameteren FLOW er viktig å ha med når et program går i dass, for den forteller hvilken linje problemet oppsto i. Dette vil forenkle feilsøkingen. Derfor bør vi alltid ha med FLOW.

PARM-instruksjonen legger vi som første instruksjon i Easytrieve Plus programmet vårt.

Hvis vi vil sjekke syntaksen i programmet vårt, uten å kjøre det, bruker vi:

```
PARM (SYNTAX)
```

Easytrieve Plus vil da sjekke syntaksen for oss. Når den er iorden, må vi endre PARM-instruksjonen og kjøre på nytt.

8.14. DO WHILE, END-DO

DO WHILE bruker vi hvis vi skal gjenta en eller flere instruksjoner mange ganger. Programmet går i løkke og utfører instruksjonene mellom DO WHILE og END-DO så lenge betingelsen for DO WHILE er oppfylt. Hvis betingelsen alltid er oppfylt, vil programmet gå i en evig løkke, og dette er det viktig å forhindre. Derfor må vi være våken og vite hva vi gjør når vi skal bruke DO WHILE-instruksjonen.

Syntaks:

```
DO WHILE betingelse(r)
 instruksjon(er)
END-DO
```

Instruksjonen(e) utføres så lenge betingelsen for DO WHILE er oppfylt. END-DO må alltid være med.

Eksempel:

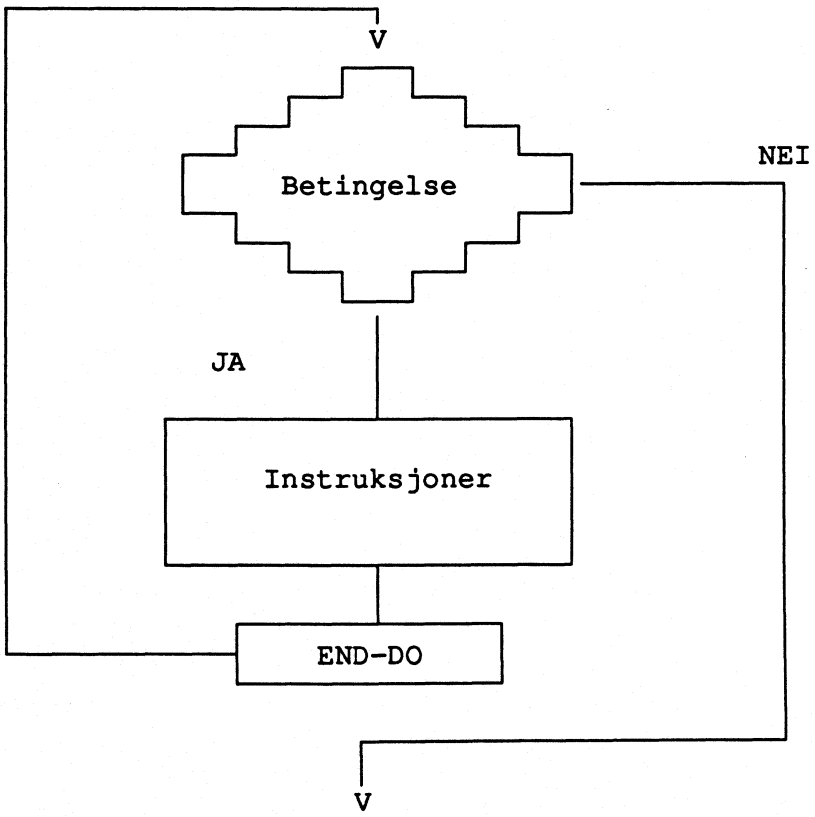
```
TELLER = 0
DO WHILE TELLER <= 100
 TELLER = TELLER + 1
 FELT = WFELT(TELLER)
 PUT UTFIL
END-DO
```

DO WHILE brukes ofte sammen med indekserte felt. Det er gjerne slik at vi skal gjøre samme operasjon på alle de indekserte feltene. Da gjør vi som i eksemplet over. WFELT bør være definert omtrent slik:

```
WFELT   W   5  N 0  OCCURS 100
```

Denne DEFINE-instruksjonen gir oss 100 felt som heter henholdsvis WFELT(1), WFELT(2) ... WFELT(100).

Slik virker DO WHILE:



8.15. GOTO

GOTO er en instruksjon som brukes for å hoppe i programmet. Det betyr at vi hopper over noen instruksjoner. Vi kan hoppe enten til et bestemt sted i programmet eller til behandling av neste record (programmet hopper til toppen av programmet og leser neste record fra innfila. Skrives GOTO JOB).

Problemet med GOTO er at vi kan hoppe på kryss og tvers i programmet, noe som kan gjøre programmet meget uoversiktlig. Derfor bør vi helst ikke bruke GOTO, med unntak av GOTO JOB. Vi kan oppnå det samme som med GOTO ved å bruke prosedyrer. Programmet blir dessuten mer oversiktlig med prosedyrer enn med GOTO.

Eksempel:

```
JOB INPUT INN
 IF KJØNN NE '1' '2'
  DISPLAY DISP KJØNN ' ER UGYLDIG KJØNN FOR RECORD NR ' RECORD-COUNT
  GO TO JOB
 END-IF
 MOVE LIKE INN TO UT
 PUT UT
```

8.16. PERFORM og bruk av prosedyrer

Instruksjonen PERFORM brukes for å kalle opp en prosedyre. Denne prosedyren må vi ha laget nederst i programmet vårt. Bruk av prosedyrer kan bidra til strukturert programmering, noe som er en fordel både for den som programmerer og den som eventuelt må gå inn å endre på en annen persons program.

Eksempel:

```
//O414KRLT JOB 8019,'O1D',MSGLEVEL=(0,0),
//           MSGCLASS=X,CLASS=A,NOTIFY=O414KRL
//O1D      EXEC PGM=EZTPA00
//STEPLIB  DD   DSN=SSB2.EASYPL60.LOAD,DISP=SHR
//INN      DD DSN=TAB.DIV(DATA),DISP=SHR
//SYSPRINT DD SYSOUT=*
//UT       DD SYSOUT=*
//DISP     DD SYSOUT=*
//SYSIN    DD *
FILE INN
     INR              1       7   A
     KJØNN           27       1   A
     LIGGEDAGER      56       4   A
FILE UT FB (59 23187)

     ANTALL-L         W       2   N 0 VALUE 0
     TELLER-L         W       2   N 0 VALUE 0
     FEILPROSENT      W       5   N 2 VALUE 0
FILE DISP PRINTER ASA
JOB INPUT INN START OVERSKRIFT FINISH OPPSUMMERING
 IF LIGGEDAGER <= '0000'
  PERFORM FEIL-LIGGEDAGER
 ELSE
  PUT UT FROM INN
 END-IF

FEIL-LIGGEDAGER. PROC
   TELLER-L = TELLER-L + 1
   IF TELLER-L = 43
    PERFORM OVERSKRIFT
    TELLER-L = 0
   END-IF
   ANTALL-L = ANTALL-L + 1
   DISPLAY DISP 'RECORD: ' RECORD-COUNT ' MED INR: ' INR +
                ' HAR UGYLDIGE LIGGEDAGER: ' LIGGEDAGER
END-PROC

OVERSKRIFT. PROC
 DISPLAY DISP NEWPAGE 'LISTE OVER OBSERVASJONER MED FEIL LIGGEDAGER'
 DISPLAY DISP SKIP 1
END-PROC

OPPSUMMERING. PROC
 DISPLAY DISP SKIP 1 'OPPSUMMERING:'
 DISPLAY DISP SKIP 1 'ANTALL OBSERVASJONER MED FEIL LIGGEDAGER: ' +
              ANTALL-L
 FEILPROSENT ROUNDED = ANTALL-L * 100 / INN:RECORD-COUNT
 DISPLAY DISP 'ANDEL OBSERVASJONER MED FEIL: ' FEILPROSENT
END-PROC
```

8.17. MSTART, MACRO, MEND og bruk av makroer

Makroer brukes for å slippe å skrive den samme programbiten om og om igjen.

Makroen vi lager skal stå aller øverst i programmet (før eventuelle kommentarer også!). Makroen skal begynne med instruksjonen MSTART med navnet på makroen som parameter. Neste instruksjon må være MACRO. I MACRO-instruksjonen forteller vi først hvor mange variabler makroen skal ha, deretter ramser vi opp navnene vi vil gi disse variablene.

```
MACRO 3 MNDTELL DAGTELL UT
```

I denne instruksjonen har vi definert 3 makrovariable. Det betyr at det er 3 variabler vi skal gi verdier når makroen kalles opp i selve programmet.

Den neste delen av makroen består av helt vanlige Easytrieve Plus-instruksjoner, med unntak av at vi skal bruke makrovariablenes navn på det som skal endres for hvert kall av makroen. Makrovariablene skal ha & foran seg når de brukes i en makro. Makroen avsluttes med MEND.

Når vi skal kalle opp en makro skriver vi slik:

```
%ACTION 1 31 NORMÅR
```

%ACTION kaller opp makroen ACTION, 1 er verdien den første makrovariabelen får, 31 er verdien den andre makrovaribelen får og NORMÅR er verdien den tredje makrovariablen får. Det som skjer etter at makroen er kalt opp er at Easytrieve Plus går igjennom hele makroen og bytter ut alle makrovariablene med de verdiene vi har gitt dem i kallet av makroen. Deretter utføres instruksjonene i makroen.

Vi tenker oss at vi har en makro som ser slik ut:

```
MSTART ACTION
  MACRO 3 MNDTELL DAGTELL UT
     MNDTELLER = MNDTELLER + 1
     DAGTELLER = 0
     IF MNDTELLER = &MNDTELL
        DO WHILE DAGTELLER < &DAGTELL
            DAGNRTELLER = DAGNRTELLER + 1
            DAGTELLER = DAGTELLER + 1
            MND (&UT)= MNDTELLER
            DAG (&UT)= DAGTELLER
            DAGNR (&UT) = DAGNRTELLER
            PUT (&UT)
        END-DO
     END-IF
MEND
```

Når vi kaller opp makroen med kommandoen %ACTION 1 31 NORMÅR bytter Easytrieve Plus ut makrovariabelnavnene med verdiene vi gir dem i kallet, slik at programbiten som skal eksekveres blir slik:

```
     MNDTELLER = MNDTELLER + 1
     DAGTELLER = 0
     IF MNDTELLER = 1
        DO WHILE DAGTELLER < 31
            DAGNRTELLER = DAGNRTELLER + 1
            DAGTELLER = DAGTELLER + 1
            MND (NORMÅR)= MNDTELLER
            DAG (NORMÅR)= DAGTELLER
            DAGNR (NORMÅR) = DAGNRTELLER
            PUT (NORMÅR)
        END-DO
     END-IF
```

```
//O414KRLW JOB (0144),'LØNØ',MSGLEVEL=(2,0),
//           CLASS=A,MSGCLASS=X,NOTIFY=O414KRL
//GOEY     EXEC PGM=EZTPA00,TIME=(0,4)
//STEPLIB  DD DSN=SSB2.EASYPL60.LOAD,DISP=SHR
//NORMÅR   DD  SYSOUT=*
//SKUDDÅR  DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
MSTART ACTION
  MACRO 3 MNDTELL DAGTELL UT
     MNDTELLER = MNDTELLER + 1
     DAGTELLER = 0
     IF MNDTELLER = &MNDTELL
        DO WHILE DAGTELLER < &DAGTELL
            DAGNRTELLER = DAGNRTELLER + 1
            DAGTELLER = DAGTELLER + 1
            MND (&UT)= MNDTELLER
            DAG (&UT)= DAGTELLER
            DAGNR (&UT) = DAGNRTELLER
            PUT (&UT)
        END-DO
     END-IF
MEND
FILE NORMÅR FB (7 17297)
    DAG   1 2 N 0
    MND   3 2 N 0
    DAGNR 5 3 N 0
FILE SKUDDÅR FB (7 17297)
    COPY NORMÅR
MNDTELLER   W 2 N 0 VALUE 0
DAGTELLER   W 2 N 0 VALUE 0
DAGNRTELLER W 3 N 0 VALUE 0
JOB  INPUT (NULL)
  DO WHILE MNDTELLER <= 12
     %ACTION    1 31 NORMÅR
     %ACTION    2 28 NORMÅR
     %ACTION    3 31 NORMÅR
     %ACTION    4 30 NORMÅR
     %ACTION    5 31 NORMÅR
     %ACTION    6 30 NORMÅR
     %ACTION    7 31 NORMÅR
     %ACTION    8 31 NORMÅR
     %ACTION    9 30 NORMÅR
     %ACTION   10 31 NORMÅR
     %ACTION   11 30 NORMÅR
     %ACTION   12 31 NORMÅR
  END-DO
  MNDTELLER = 0
  DAGNRTELLER = 0
  DO WHILE MNDTELLER <= 12
     %ACTION    1 31 SKUDDÅR
     %ACTION    2 29 SKUDDÅR
     %ACTION    3 31 SKUDDÅR
     %ACTION    4 30 SKUDDÅR
     %ACTION    5 31 SKUDDÅR
     %ACTION    6 30 SKUDDÅR
     %ACTION    7 31 SKUDDÅR
     %ACTION    8 31 SKUDDÅR
     %ACTION    9 30 SKUDDÅR
     %ACTION   10 31 SKUDDÅR
     %ACTION   11 30 SKUDDÅR
     %ACTION   12 31 SKUDDÅR
  END-DO
STOP EXECUTE
```

9. FEILSØKING

Vi deler grovt feilene som kan gjøres i tre deler; Syntaksfeil, logiske feil og avbrutte program. For syntaksfeilene har Easytrieve Plus greie meldinger. Meldingene legges på linja under den der Easytrieve Plus oppdaget feilen (vanligvis blir det under linja feilen er i).

Logiske feil er feil programmereren (DU) har gjort og disse greier ikke Easytrieve Plus å si fra om. Dette fordi programmet er kjørbart og det er ikke programmets oppgave å sjekke vår logikk, programmet gjør bare det vi ber om. Derfor er det viktig å programmere slik at vi har muligheter til å sjekke at vi har programmert riktig. Dette gjør vi for eksempel vha DISPLAY-instruksjonen.

Å finne og å rette feil når Easytrieve Plus har avbrutt kjøringen (ABEND) er ofte et mareritt. Easytrieve Plus er meget dårlig til å forklare oss hva vi har gjort galt når programmet skjærer seg. Hvis vi ikke er proff DEBUGGER, og det er det ingen her i huset som er, synker vi nå ned i en hengemyr av uforståelige, kryptiske koder.

Hvis vi ikke har med PARM-instruksjonen med parameter DEBUG (FLOW) vil ikke engang programmet si hvilken instruksjon feilen oppsto i. Derfor bør vi alltid ha med DEBUG (FLOW) i PARM-instruksjonen øverst i programmet.

Når PARM-instruksjonen er i orden, vil vi finne ut hvilken instruksjon programmet skar seg i. Dette er en god start. Nå må vi se hva denne instruksjonen gjør, og spesielt sjekke de feltene som brukes i instruksjonen. Som oftest er feilen at vi har prøvd å regne på verdier som ikke er numeriske. Hvis det ikke viser seg å være dette som er feilen, forvandler hengemyra seg til kvikksand. Det beste vi kan gjøre da vil ofte være å prøve å forenkle programmet og/eller dele det opp i flere deler.

Når programmet skjærer seg vil vi finne en linje i kjørerapporten som ligner på denne:

```
12 *******A006 PROGRAM INTERRUPT - CODE 7 (DATA EXCP)
```

Tallet til venstre på linja forteller hvilken linje det skar seg på. Feilmeldingen PROGRAM INTERRUPT forteller akkurat det vi vet; at noe gikk galt og CODE 7 (DATA EXCP) prøver å fortelle oss hva. Kryptisk, ikke sant?

Lenger ned i kjørerapproten (ofte helt nederst) vil vi finne en linje som ligner denne:

```
INN                 9354   INPUT      SAM  FIX     BLK         ASA    80  3120
```

Den forteller oss hvilken fil det skar seg i (INN), og hvilken record (9354). Da er det bare å se på fila, finne recorden det skar seg på og se på det feltet som ble brukt i instruksjonen programmet stoppet på. Det er sannsynlig at det er verdien dette feltet innehar som programmet ikke svelget.

Disse forklaringene har jeg erfart:

```
CODE B (DEC DIVIDE EXCP) ===> DELE PÅ NULL!       (WORKFELT)
CODE B (DEC DIVIDE EXCP) ===> DELE PÅ INGENTING!  (WORKFELT)
CODE B (DEC DIVIDE EXCP) ===> 0/0                 (WORKFELT)
CODE 7 (DEC DIVIDE EXCP) ===> DELE PÅ INGENTING!  (FILFELT)
CODE 7 (DEC DIVIDE EXCP) ===> 0/0                 (FILFELT)
CODE B (DEC DIVIDE EXCP) ===> DELE PÅ NULL!       (FILFELT)

CODE 7 (DATA EXCP)       ===> REGNE PÅ FELT SOM IKKE ER TILGJENGELIG
                              (F. EKS. ETTER END OF FILE)
CODE 7 (DATA EXCP)       ===> REGNE PÅ FELT SOM IKKE ER NUMERISKE
                              (F. EKS. BLANKE)
CODE 7 (DATA EXCP)       ===> REGNE PÅ FELT SOM HAR FÅTT NUMERISK
                              VERDI MED MOVE, IKKE ASSIGNMENT!

CODE 4 (PROTECTION EXCP) ===> DU HAR GLEMT PUT ETTER Å HA FLYTTET
                              (MOVE) DATA TIL EN FIL
```

Det er ikke sikkert noen av disse passer med det vi måtte ha gjort feil, og da er det kanskje på tide å gå hjem?

For å unngå at programmet skjærer seg bør vi gjøre programmene dine så enkle som mulig. Det kan ofte være lurt å dele et program i to eller flere deler fremfor å lage et digert program. Jo mindre programmet er, desto enklere er det å finne feil.

10. MATCHING AV FILER

Matching av filer vil si å lese inn to eller flere filer samtidig for eventuelt å koble sammenhørende data. For å få til dette må filene inneholde nøkkelfelt. Ved å sammenligne nøkkelfeltene, finner vi sammenhørende data. Hvis nøkkelfeltene er like matcher filene, ellers matcher de ikke.

Dette høres kanskje enkelt ut, men dessverre er det ofte litt mer komplisert enn som så. Det stilles nemlig noen krav til de filene som skal matches, og dessuten er det flere forskjellige måter filer kan matches på (mer om dette senere).

Hvorfor matche filer? Poenget med matching er å finne fram til sammenhørende data fra forskjellige filer og behandle dem samtidig. Dataene må være forbundet med nøkkelfelt som ligger på alle filene. Matching kan blant annet brukes til å:

- hente data fra 2 forskjellige registre til et nytt register
- oppdatere et register eller en fil
- koble forskjellige datafiler

Følgende krav stilles til filene som skal matches:

- Alle filene må være SORTERT på nøkkelfeltene i STIGENDE rekkefølge
- ANTALL nøkkelfelt i hver fil må være likt
- Korresponderende nøkler må enten begge være numeriske eller begge alfanumeriske

NB!	Hvis filene ikke er sortert vil vi få helt gal matching. Easytrieve Plus gir oss ikke melding hvis dataene ikke er sortert. Det er derfor VI som må sørge for at filene er sortert.

Vi kan ikke matche to filer der den ene har et numerisk definert nøkkelfelt og den andre har definert nøkkelfeltet alfanumerisk. Derimot kan nøkkelfeltene ha forskjellig lengde, og hvis nøkkelfeltene er numerisk definert kan de ha forskjellige data-typer (N, P, U, B).

Når nøkkelfeltene er definert numerisk, vil det sammenlignes fra høyre mot venstre, mens det sammenlignes fra venstre mot høyre ved alfanumerisk definering av feltene. Hvis nøkkelfeltene har forskjellig lengde og de er definert alfanumerisk, må vi være oppmerksom på hvilke resultater matchingen gir. På de neste sidene følger noen eksempler på hvilke resultater matchinger med ulik lengde på nøkkelfeltene gir.

Eksempel 1: Alfanumeriske felt med forskjellig lengde matches (Fila med lengst nøkkelfelt leses først)

```
FILE A
  ALFA3       1 3 A
FILE B
  ALFA2       1 2 A
FILE C
  ALFA2       1 2 A
  ALFA3       3 3 A
JOB INPUT (A KEY(ALFA3) +
           B KEY(ALFA2))
IF MATCHED
   MOVE LIKE A TO C
   MOVE LIKE B TO C
   PUT C
END-IF
```

Denne tabellen viser hvilke poster (records) som matcher fra fil a og fil b når vi kjører programmet over (En post er en samling av data som hører sammen og behandles som en enhet). Verdiene står i fnutter (' ') slik at vi skal kunne se de blanke tegnene.

		F I L B			
	Verdi	' '	' 0'	'0 '	'00'
	' '	match	*	*	*
F	' 0'	*	*	*	*
I	' 0 '	*	match	*	*
L	' 00'	*	*	*	*
	'0 '	*	*	match	*
A	'00 '	*	*	*	match
	'000'	*	*	*	*

* = ikke match

Eksempel 2: Alfanumeriske felt med forskjellig lengde matches (Fila med kortest lengde på nøkkelen leses først)

```
FILE A
  ALFA3       1 3 A
FILE B
  ALFA2       1 2 A
FILE C
  ALFA2       1 2 A
  ALFA3       3 3 A
JOB INPUT (B KEY(ALFA2) +
           A KEY(ALFA3))
IF MATCHED
   MOVE LIKE A TO C
   MOVE LIKE B TO C
   PUT C
END-IF
```

		F I L B			
	Verdi	' '	' 0'	'0 '	'00'
	' '	match	*	*	*
F	' 0'	match	*	*	*
I	' 0 '	*	match	*	*
L	' 00'	*	match	*	*
	'0 '	*	*	match	*
A	'00 '	*	*	*	match
	'000'	*	*	*	match

* = ikke match

Som vi ser er det ikke helt enkelt å forutsi resultatet når vi matcher 2 filer med alfanumeriske nøkkelfelt som har forskjellig lengde. Derfor bør vi sørge for at nøkkelfeltene har samme lengde.

Eksempel 3: Numeriske felt med forskjellig lengde matches

```
FILE A
  NUM3       1 3 N
FILE B
  NUM2       1 2 N
FILE C
  NUM2       1 2 N
  NUM3       3 3 N
JOB INPUT (A KEY(NUM3) +
           B KEY(NUM2))
IF MATCHED
   MOVE LIKE A TO C
   MOVE LIKE B TO C
   PUT C
END-IF
```

Dette programmet vil bli avbrutt (ABEND) fordi vi har en del felt på filene som Easytrieve Plus ikke godtar som numeriske. I tabellen under er disse merket med *. De andre er merket som om kjøringene hadde gått i orden.

		F I L B			
	Verdi	' '	' 0'	'0 '	'00'
	' '	*	*	*	*
F	' 0'	*	match	*	match
I	' 0 '	*	*	*	*
L	' 00'	*	match	*	match
	'0 '	*	*	*	*
A	'00 '	*	*	*	*
	'000'	*	match	*	match

Tabellen over viser også hvilke verdier Easytrieve Plus regner som numeriske og hvilke som ikke blir regnet som det. De verdiene som har match er numeriske.

Matching av filer og matching av filer er ikke nødvendigvis det samme. Det er nemlig mange typer matching. Først og fremst er det et stort skille mellom å matche to filer, og det å matche tre eller flere.

Som en hovedregel anbefaler jeg å aldri matche mer enn to filer på en gang.

Matching av flere enn to filer kan lett føre til problemer. Spesielt hvis det er dubletter på flere av filene (en dublett har vi hvis to eller flere poster i EN fil har samme verdi for nøkkelfeltet). Derfor bør vi holde oss til matching av to filer.

Når vi matcher to filer kan 8 mulige situasjoner oppstå:

	Fil1		Fil2		
1.	En	post	matcher en	post	(En-til-en)
2.	En	post	matcher mange	poster	
3.	En	post	matcher ingen	poster	
4.	Mange	poster	matcher en	post	<==== må snus
5.	Mange	poster	matcher mange	poster	<==== må unngås
6.	Mange	poster	matcher ingen	poster	
7.	Ingen	poster	matcher en	post	
8.	Ingen	poster	matcher mange	poster	(Ingen-til-mange)

Easytrieve Plus greier ikke å håndtere alle disse situasjonene: Nr. 4 og nr. 5 vil skape problemer. Men hvis vi snur relasjonene, ser vi at relasjon nr. 4 blir lik nr. 2 (mange-til-en blir en-til-mange). I praksis snur vi relasjonene ved å bytte rekkefølgen filene leses i. Det er altså bare mange-til-mange relasjonen Easytrieve Plus ikke kan greie.

Ut fra dette er det lett å skjønne at det er svært viktig å kjenne filer som skal matches så godt at vi vet hvilke av situasjonene ovenfor som kan oppstå.

Hvis vi prøver en mange-til-en relasjon i Easytrieve Plus vil den bli gjort om til en en-til-en og en mange-til-ingen relasjon. Det betyr at utfallet ikke vil bli som vi hadde ment.

Eksempel: Hvis vi har 5 poster som har lik verdi for nøkkelfeltene i fil1 og 1 post i fil2 som har denne verdien for sine nøkkelfelt har vi en mange-til-en situasjon. Easytrieve Plus vil da finne at den første posten fra fil1 vil matche med posten fra fil2 (en-til-en). De 4 andre postene fra fil1 vil ikke matche med noen fra fil2 (mange-til-ingen). Snur vi rekkefølgen filene leses i vil vi få at den ene posten på fil2 vil matche med alle 5 fra fil1 (en-til-mange).

En mange-til-mange situasjon bli gjort om til en en-til-mange og en mange-til-ingen situasjon.

Dette betyr at vi må vite noe om dataene våre før vi begynner å matche. Det vi må vite først er:

- Finnes det dubletter på fil 1?
- Finnes det dubletter på fil 2?

Disse to spørsmålene gir oss svar på hvilke relasjoner vi kan risikere å møte. De problematiske matchingene vil vi få hvis vi har dubletter på fil 1 eller på begge filene. Hvis vi har dubletter bare på fil 1 vil vi løse det problemet ved å lese fil 2 før fil 1. Men er det dubletter på begge filene vil vi få problemer uansett.

Med alle disse mulige relasjonene som kan oppstå når vi sammenligner to nøkkelfelt må vi ha mulighet til å gjøre tester på hvilket resultat sammenligningen har gitt, både i forhold til hverandre og til postene før og etter på de to filene. Til dette har Easytrieve Plus laget noen spesielle IF-tester:

- IF MATCHED
- IF NOT MATCHED
- IF DUPLICATE FIL#
- IF NOT DUPLICATE FIL#
- IF FIRST-DUP FIL#
- IF NOT FIRST-DUP FIL#
- IF LAST-DUP FIL#
- IF NOT LAST-DUP FIL#
- IF FIL#
- IF NOT FIL#
- IF EOF FIL#

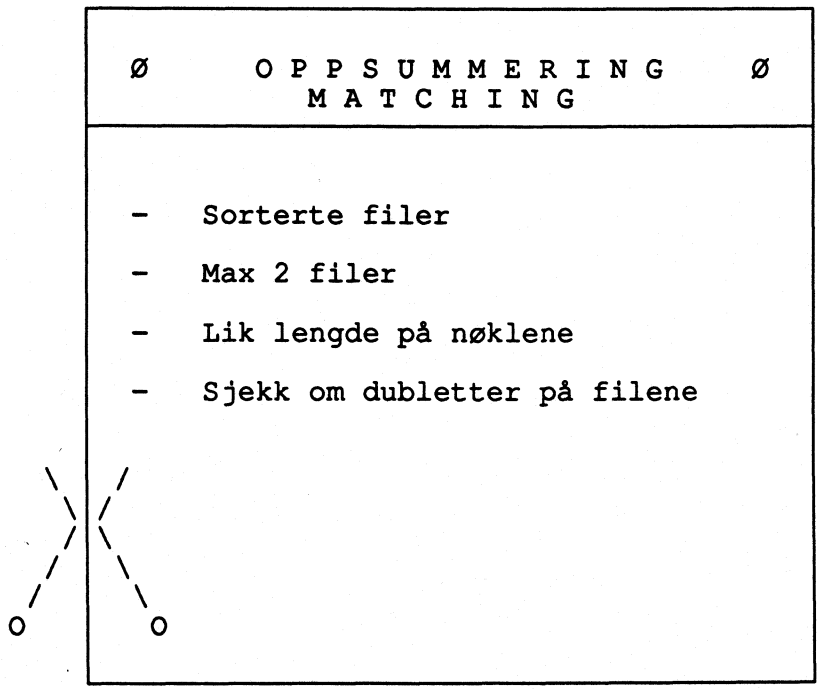
Fil# vil si fil1 eller fil2.

Disse testene kan vi selvsagt kombinere akkurat slik vi vil.

IF MATCHED og IF NOT MATCHED bruker vi til å test om filene matcher. Med IF DUPLICATE kan vi sjekke om det er dubletter på noen av filene, mens IF FIRST-DUP og IF LAST-DUP sjekker om det er første eller siste dublett vi behandler.

IF fil 1 og IF fil 2 brukes til å finne ut hvilken av filene som er 'aktive' når de ikke matcher. Den fila som har den laveste verdien for nøkkelfeltet, er den aktive fila. Når filene matcher, er begge aktive. Hvis filene ikke matcher og fil 1 har den laveste verdien for nøkkelfeltet er denne aktiv og vi kan behandle disse dataene. Fil 2, som altså ikke er aktiv, er heller ikke tilgjengelig for behandling. Hvis vi prøver det, vil Easytrieve Plus gi feilmelding.

Eksempler på matching og bruk av de spesielle IF-testene ligger på datasettet EZT.DIV. Member match1 og match2 viser de fleste muligheter ved matching av to filer. Kopier dem over til din bruker og kjør dem. Da vil du se matching i praksis!



Eksempel på hvordan sammenligningen av to filer utføres. Programmet som skal kjøres ser slik ut:

```
PARM DEBUG (NOCLIST NODMAP FLOW NOXREF) LIST (NOPARM) SORT (MSG NO)
FILE FIL1
     FELTA           1   1 N  0
     FELTB           3   1 A
FILE FIL2
     FELTA           1   1 N  0
     FELTC           3   1 A
FILE UT
     FELTA           1   1 N  0
     FELTB           3   1 A
     FELTC           5   1 A
JOB INPUT (FIL1 KEY(FELTA) +
           FIL2 KEY(FELTA))
IF MATCHED
   MOVE LIKE FIL1 TO UT
   MOVE LIKE FIL2 TO UT
   PUT UT
END-IF
```

Med inputfiler som i rammen under til venstre vil programmet behandle filene som vist i rammen under til høyre:

FIL1		FIL2	
FELTA	FELTB	FELTA	FELTC
1	C	0	A
2	D	1	E
2	E	2	A
2	F	2	B
3	Y	2	C
5	R	3	T
6	W	4	Q
6	X	4	P
7	J	5	K
		5	L
		5	D

Behandles slik:		
1	1 C	0 A
2	1 C	1 E
3	2 D	2 A
4	2 D	2 B
5	2 D	2 C
6	2 E	3 T
7	2 F	3 T
8	3 Y	3 T
9	5 R	4 Q
10	5 R	4 P
11	5 R	5 K
12	5 R	5 L
13	5 R	5 D
14	6 W	
15	6 X	
16	7 J	

Forskjellige tester vil slå til som vist under:

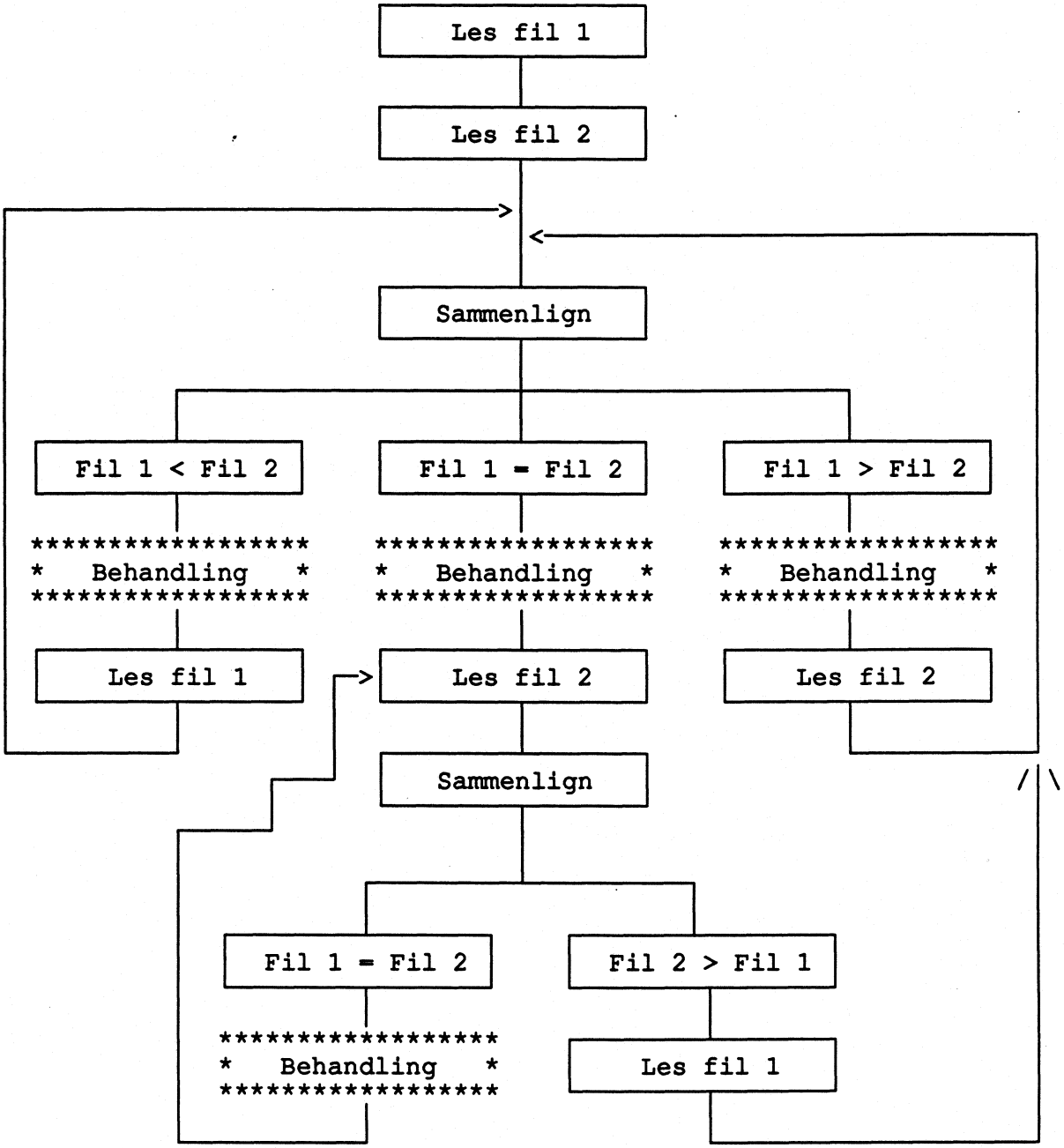
IF MATCHED: 2, 3, 4, 5, 8, 11, 12 og 13
IF NOT MATCHED: 1, 6, 7, 9, 10, 14, 15, og 16
IF DUPLICATE FIL1: 3, 4 , 5, 6, 7, 14 og 15
IF DUPLICATE FIL2: 3, 4, 5, 9, 10, 11, 12 og 13
IF FIRST-DUP FIL1: 3, 4, 5 og 14
IF FIRST-DUP FIL2: 3, 9 og 11
IF LAST-DUP FIL2: 5, 10 og 13
IF FIL1: 2, 3, 4, 5, 6, 7, 8, 11, 12, 13, 14, 15 og 16
IF FIL2: 1, 2, 3, 4, 5, 8, 9, 10, 11, 12 og 13
IF EOF FIL2: 14, 15 og 16

Testene kan kombineres. Hvis vi f.eks sier

IF MATCHED AND FIRST-DUP FIL1

vil betingelsen være oppfylt for nr. 3, 4 og 5.

Flytdiagram for sammenligning av to filer, Easytrieve Plus



Der det er ruter med behandling i kan vi gjøre testing, skriving og andre programmeringsfunksjoner. Resten skjer automatisk.

VEDLEGG

I. Sortering i Easytrieve Plus kontra sortering med Syncsort.
II. NUMERIC-testen.
III: IBM's overpunch for lagring av negative tall

VEDLEGG I. Sortering i Easytrieve Plus kontra sortering med Syncsort.

Som tidligere nevnt vil en sortering bruke mer tid når vi utfører den i Easytrieve Plus enn om vi bruker Syncsort direkte (i et eget steg). Dette selv om Easytrieve Plus kaller opp Syncsort. Jeg har ikke fått noen fornuftig forklaring på hvorfor, vi får bare godta det som det er. Under er det et eksempel på et program som først sorterer en fil, for deretter å matche den mot en annen fil. Dette kan gjøres enten ved å sortere filen fra Easytrieve Plus, eller å sortere filen med i et eget steg med Syncsort først, for deretter å utføre matchingen i Easytrieve Plus.

Sortering og matcing i Easytrieve Plus:

```
//O414KRLT JOB 8019,'SORTEZT',MSGLEVEL=(2,0),
//           MSGCLASS=X,CLASS=A,NOTIFY=O414KRL,REGION=6144K
/*JOBPARM  L=30
//SORTEZT  EXEC PGM=EZTPA00,REGION=6144K
//STEPLIB  DD   DSN=SSB2.EASYPL60.LOAD,DISP=SHR
//EZTVFM   DD  UNIT=WORK,SPACE=(4096,(100,200),,,ROUND)
//SORTWK01 DD  UNIT=WORK,SPACE=(4096,500,,,ROUND)
//*************************************************
//* PROGRAM SOM SORTERER I EASYTRIEVE PLUS
//*************************************************
//INN      DD DSN=TAB.DIV(DATA),DISP=SHR
//TFYLKE   DD DSN=EZT.DIV(TFYLKE),DISP=SHR
//SYSPRINT DD SYSOUT=*
//UT       DD SYSOUT=*
//SYSIN    DD *
PARM DEBUG (NOCLIST NODMAP NOFLOW NOXREF) LIST (NOPARM) +
     SORT (MSG NO)
FILE INN
     INR                 1       7   A
     KOMMNR             23       4   A
     REC                 1      59   A
FILE SORT1 VIRTUAL FB (59 23187)
 COPY INN
FILE UT FB (61 23180)
     INR                 1       7   A
     REC                 1      59   A
     TFYLKE             60       2   A
FILE TFYLKE
     INR                 1       7   A
     TILKNYTNINGSFYLKE  14       2   A
SORT INN TO SORT1 USING INR BEFORE SELEKSJON
SELEKSJON. PROC
 IF KOMMNR = '0301'
  SELECT
 END-IF
END-PROC
JOB INPUT (TFYLKE KEY (INR) +
           SORT1  KEY (INR))
 IF MATCHED
  MOVE LIKE TFYLKE TO UT
  MOVE LIKE SORT1 TO UT
  PUT UT
 END-IF
```

Tabellen på neste side viser at selve sorteringen bruker ca. 25 % av tiden når Syncsort gjør det i et eget steg istedenfor å la Easytrieve Plus kalle opp Syncsort. Det skulle derfor være all mulig grunn til å advare mot å bruke Easytrieve Plus til større sorteringer.

Ekstern eller intern sortering	Antall records inn og ut	CPU-tid sortering i 100-dels minutter	CPU-tid matching i 100-dels minutter	CPU-tid totalt i 100-dels minutter	Ekstern CPU-tid i % av intern	Ekstern sortering i % av intern
EZT+ (MED SORT)	20 000	3	2	5		
SYNCSORT & EZT+	20 000	1	2	3	60	33
EZT+ (MED SORT)	60 000	9	4	13		
SYNCSORT & EZT+	60 000	3	4	7	54	33
EZT+ (MED SORT)	121 000	18	7	25		
SYNCSORT & EZT+	121 000	4	7	12	48	22
EZT+ (MED SORT)	242 000	35	14	49		
SYNCSORT & EZT+	242 000	8	14	23	47	23
EZT+ (MED SORT)	483 000	63	28	91		
SYNCSORT & EZT+	483 000	15	28	44	48	24

Programmet med Syncsort i eget steg ser slik ut:

```
//O414KRL4 JOB 8019,'KRL',MSGLEVEL=(2,0),
//            MSGCLASS=X,CLASS=A,NOTIFY=O414KRL,REGION=6144K
//SORTSYNC EXEC SYNC4,CYL=1,REGION=6144K
//SORTIN   DD DSN=TAB.DIV(DATA),DISP=SHR
//SORTOUT  DD DSN=&&SORTINN,
//            DISP=(NEW,PASS,DELETE),UNIT=WORK,
//            DCB=(LRECL=80,DSORG=PS,RECFM=FB,BLKSIZE=3120),
//            SPACE=(3120,(111,111),RLSE)
//SYSOUT   DD SYSOUT=*
//PARM     DD  *
    SORT FIELDS=(1,7,A),FORMAT=CH
//SORTEZT  EXEC PGM=EZTPA00
//STEPLIB  DD   DSN=SSB2.EASYPL60.LOAD,DISP=SHR
//EZTVFM   DD  UNIT=WORK,SPACE=(4096,(100,200),,,ROUND)
//INN      DD DSN=&&SORTINN,DISP=(OLD,DELETE)
//TFYLKE   DD DSN=EZT.DIV(TFYLKE),DISP=SHR
//SYSPRINT DD SYSOUT=*
//UT       DD SYSOUT=*
//SYSIN    DD *
FILE INN
     INR                  1       7   A
     KOMMNR              23       4   A
     REC                  1      59   A
FILE UT FB (61 23180)
     INR                  1       7   A
     REC                  1      59   A
     TFYLKE              60       2   A
FILE TFYLKE
     INR                  1       7   A
     TILKNYTNINGSFYLKE   14       2   A
JOB INPUT (TFYLKE KEY (INR) +
           INN    KEY (INR))
 IF MATCHED
  MOVE LIKE TFYLKE TO UT
  MOVE LIKE INN TO UT
  PUT UT
 END-IF
```

VEDLEGG II. NUMERIC-testen.

Vi kan teste om verdien av et felt er numerisk. Det gjør vi med NUMERIC. Testen ser slik ut; IF feltnavn NUMERIC. Denne testen oppfører seg litt rart når vi skal teste på verdier som har den IBM'ske overpunch i siste posisjon. Poenget med denne overpunchen er å kunne representere negative tall uten å bruke en posisjon av feltet til fortegn. I steden for å sette fortegn først i feltet, blir det siste sifferet overpunchet med en bokstav. Denne bokstaven reprensenterer det tallet den overskriver og dessuten sier den om tallet er positivt eller negativt. F. eks. vil tallet -903 bli skrevet 90L, der L betyr 3 og at hele tallet er negativt! Liste over overpunchbokstaver med tilhørende verdier finnes i vedlegg III.

Nå viser det seg at NUMERIC-testen ikke alltid slår til når verdiene er overpunchet! Dette er utrolig nok avhengig av hvordan feltet vi skal teste på er definert. Hvis vi definerer feltet som alfanumerisk eller numerisk uten å angi antall desimaler, vil ikke NUMERIC-testen slå til når tallene er overpunchet. Det vil den derimot når vi angir at feltet har desimaler.

Programmet på neste side gir mange morsomme (?) resultater. Ut fra disse kan vi konkludere med følgende:

Verdier som har overpunch og skal testes med NUMERIC-testen skal:

- Ha overpunchbokstaven i siste posisjon (vil normalt være tilfelle)
- Være definert som numerisk felt
- Ha angitt antall desimaler i feltdefinisjonen (selv om det er 0)

De samme regler gjelder også når vi skal regne med disse verdiene! Hvis vi glemmer det kan vi få enda morsommere resultater.

Programmet ligger på stormaskinen på Kongsvinger under navnet EZT.DIV(NUMERIC). Alt som ligger på dette datasettet kan alle se på og kopiere.

```
//0414KRLÅ JOB 8019,'LE NEUX',MSGCLASS=X,CLASS=A,NOTIFY=0414KRL
//SPESS    EXEC PGM=EZTPA00
//STEPLIB  DD   DSN=SSB2.EASYPL60.LOAD,DISP=SHR
//INN1     DD   *
100æ7
100A3
111å5
11111
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  DUMMY
//UT1      DD  SYSOUT=*
//SYSIN    DD  *
MSTART NUMTEST
  MACRO 1 FELT
IF &FELT NUMERIC
  DISPLAY UT1 COL 5 '&FELT NUMERISK:' COL 30 &FELT
ELSE
  DISPLAY UT1 COL 5 '&FELT IKKE NUMERISK: ' COL 30 &FELT
END-IF
MEND
LIST NOMACROS
PARM DEBUG (NOCLIST NODMAP FLOW NOXREF) LIST (NOPARM) SORT (MSG NO)
FILE INN1
   NUM                  1   4 N
   NUM0                 1   4 N 0
   NUM2                 1   4 N 2
   ALFA                 1   4 A
   NUMSPES              1   5 N
   NUMSPES0             1   5 N 0
   NUMSPES4             1   5 N 4
FILE UT1 PRINTER ASA
JOB INPUT (INN1)
%NUMTEST NUM
%NUMTEST NUM0
%NUMTEST NUM2
%NUMTEST NUMSPES
%NUMTEST NUMSPES0
%NUMTEST NUMSPES4
```

VEDLEGG III: IBM's overpunch for lagring av negative tall

For å slippe å bruke én posisjon av et felt til oppbevaring av fortegn, bruker IBM i stormaskinverdenen noe som kalles overpunch. Istedenfor å ha med fortegnet, endrer de heller det siste tallet i feltet. Dette gjøres om til en bokstav. Denne bokstaven forteller både hvilket tall det bokstaven står for, og hvilket fortegn tallet skal ha. For eksempel vil tallet -1162 bli lagret som 116K, der K betyr 2 og at hele tallet er negativt.

Her følger en oversikt over hvilke bokstaver som kan brukes og hvilket tall og fortegn de representerer.

ü	+1
s	+2
t	+3
u	+4
v	+5
w	+6
x	+7
y	+8
z	+9
æ	+0
A	+1
B	+2
C	+3
D	+4
E	+5
F	+6
G	+7
H	+8
I	+9
å	-0
J	-1
K	-2
L	-3
M	-4
N	-5
O	-6
P	-7
Q	-8
R	-9
\	+0
S	+2
T	+3
U	+4
V	+5
W	+6
X	+7
Y	+8
Z	+9

STIKKORDSREGISTER