

Cobol opgave med Specialisterne

Udarbejdet af Michael Bøgh Nielsen, Danske Bank 2025

Indholdsfortegnelse

Generel introduktion til COBOL.....	4
Teknisk introduktion til COBOL	5
COBOLs kolonnebaserede struktur.....	6
Opgave 1 – Dit første COBOL-program	7
Del 1: Hello World	7
Del 2: Deklarering af variabel	8
Opgave 2 - Variabler og MOVE.....	10
Del 1 – Program med flere variabler og move statement	10
Del 2: Flyt data med MOVE.....	10
Del 3: Udskriv data.....	11
Opgave 3 – Løkker og strenghåndtering	12
Del1 - Kombinér fornavn og efternavn til én variabel	12
Del 2 - Fjern overskydende mellemrum	13
Del 3: Udskrivning af kundeinfo	15
Opgave 4 Introduktion til strukturer i COBOL.....	16
Del 1 - Opret en struktur til kundedata	18
Del 2 – Udskriv hele strukturen.....	18
Opgave 5 Introduktion til Copybooks i COBOL.....	19
Del 1 – Oprettelse af en Copybook.....	20
Del 2 – Inkludér Copybook i et program.....	20
Opgave 6 – Læsning af fil.....	22
Del 1 – Indlæs Kundedata fra en fil	23
Opgave 7 skrivning i fil	24
Del 1 – Skriv samme indhold som der er blevet læst	24
Del 2 – Lav ændringer til skrive strukturen	24
Del 3 - Paragraffer	26
Opgave 8 Læsning af flere filer og skriv i output fil.....	27
Del 1 – Strukturer ny fil med konto oplysninger.....	27
Del 2 – Læs begge filer og udskriv oplysninger	28
Opgave 9 - Introduktion til arrays	29

Del 1 – Brug array filhåndtering	30
Bilag 1 – Datatyper.....	31
Bilag 2 filhåndtering.....	34

Generel introduktion til COBOL

COBOL (Common Business-Oriented Language) er et programmeringssprog, der blev udviklet i **1959** af arbejdsgruppen **CODASYL**, bestående af computerforskere, industrirepræsentanter og regeringsfolk.

Formål: - Skabe et sprog til forretnings- og administrationsopgaver. - Sikre høj læsbarhed, så personer uden dyb teknisk baggrund kunne forstå koden. - Standardisere programmering af dataintensive systemer såsom bank-, løn- og skattesystemer.

Anvendelse: - Finans og bank (konti, transaktioner, regnskaber). - Offentlig administration (lønsystemer, skatter, forsikring). - Store databaser og batch-processer, hvor stabilitet og præcision er vigtig.

Nutidig udbredelse: - COBOL bruges stadig globalt i kritiske finansielle og administrative systemer. - Millioner af transaktioner håndteres dagligt af COBOL-programmer. - Moderne versioner som **GnuCOBOL** gør det muligt at udvikle og teste COBOL på almindelige pc'er og integrere med moderne systemer.

Teknisk introduktion til COBOL

Struktur og syntaks: COBOL er **procedurebaseret** og stærkt struktureret med fire hoveddivisioner:

1. **IDENTIFICATION DIVISION** – Identificerer programmet og metadata.
2. **ENVIRONMENT DIVISION** – Beskriver filsystemer og eksterne ressourcer.
3. **DATA DIVISION** – Deklarerer alle variabler, filer og datastrukturer.
4. **PROCEDURE DIVISION** – Indeholder selve koden og programmeringslogikken.

Nøglepunkter:

- **Læsbart:** COBOL-linjer minder om almindeligt engelsk.
- **Stærkt typet:** Alle variabler og datafelter skal deklarereres præcist.
- **Filhåndtering:** Meget brugt til line-sequential filer og databaser.
- **Kontrolstrukturer:** PERFORM, IF/ELSE, EVALUATE bruges til logik.
- **Variabler og MOVE:** MOVE flytter værdier mellem variabler
- **COMPUTE** udfører aritmetiske operationer.
- **Arrays:** Kan deklarereres med OCCURS og itereres med PERFORM VARYING. –
- **Terminal output/input:** DISPLAY viser data, ACCEPT læser input.
- **Fordele:** - Ekstremt stabilt og pålideligt til store datamængder. - Velegnet til batch-programmering, regnskabsbehandling og rapportgenerering.

Der findes et godt opslagsværk på denne side:

<https://www.tutorialspoint.com/cobol/index.htm>

COBOLs kolonnebaserede struktur

COBOL følger traditionelt et fast kolonneformat, hvilket oprindeligt blev designet til hulkort. Hver kolonne har en specifik funktion, og det er vigtigt at placere koden korrekt, da forkert indplacering kan føre til fejl under kompilering.

Kolonne	Formål
1–6	Sekvensnummer (kan ofte ignoreres i moderne editorer)
7	Kommentar (* = kommentar, - = fortsættelseslinje)
8–11	Area A - Bruges til division-, section- og paragraph-deklarationer.
12–72	Area B – Bruges til statements/instruktioner som MOVE, DISPLAY, IF osv.
73–80	Kommentar eller ID-område (kan bruges til identifikation af linjen).

Vigtige bemærkninger

- Koden må ikke gå ud over 80 karakterer per linje, da dette vil medføre en fejl under kompileringen. Benyt en ny linje, hvis det sker.
- Forkert placering af statements i Area A eller Area B vil få compiler'en til ikke at genkende dem korrekt.

Area A

- Anvendes til strukturelle elementer såsom divisioner, sektioner og paragraffer.
- Eksempler:
 - IDENTIFICATION DIVISION.
 - DATA DIVISION.
 - Paragraffer: MAIN-LOGIC., PROCESS-CUSTOMER.

Area B

- Anvendes til commands/statements, der udfører instruktioner.
- Eksempler:
 - MOVE
 - DISPLAY
 - IF
 - PERFORM

Opgave 1 – Dit første COBOL-program

Formål:

Test, om COBOL er korrekt installeret, og lær grundlæggende struktur for et COBOL-program.

Del 1: Hello World

1. Opret en COBOL-fil

- I den mappe, som blev oprettet tidligere, lav en ny fil i Visual Studio Code.
- Kald filen **Hello.cob**.

2. Skriv din første kode

- Indsæt følgende kode i din Hello.cob-fil. Husk, at **kommandoer skal starte i position 8**, mens kommentarer skal begynde i **position 7**.

Her er et eksempel på gyldigt kodeformat i COBOL:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. HELLO.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
PROCEDURE DIVISION.  
*Nedenfor kommer en display - Cobols måde at skrive i konsollen  
DISPLAY "HELLO, WORLD!"  
STOP RUN.
```

3. Byg programmet

For at bygge programmet til en eksekverbar fil (.exe) skal vi bruge batch-filen cobbuild.bat, som blev oprettet under opsætningsguiden. Følg disse trin:

- Åbn **terminalen** i Visual Studio Code (fx PowerShell eller Command Prompt).
- Naviger til den mappe, hvor din Hello.cob-fil er gemt, ved at køre denne kommando. Husk at udskifte kontonavn med din egen brugerkonto:

```
C:\Users> cd C:\Users\kontonavn\VsCodeWS\COBOL
```

- Byg COBOL-filen til en eksekverbar fil ved at køre følgende kommando. Hvis du ikke bruger PowerShell, kan du undlade "./" foran kommandoen:

```
C:\Users\kontonavn\VsCodeWS\Cobol> ./cobbuild.bat -x hello.cob -o hello.exe -lcob
```

4. Kør programmet

Når byggetrinnet er fuldført, kan du køre din eksekverbare fil for at se resultatet:

```
C:\Users\kontonavn\VsCodeWS\Cobol> ./hello.exe
HELLO, WORLD!
C:\Users\kontonavn\VsCodeWS\Cobol>
```

Hvis alt er korrekt installeret og opsat, vises teksten **HELLO, WORLD!** i terminalen. Tillykke, dit første COBOL-program er nu færdigt!

Del 2: Deklarering af variabel

I denne del skal du lære at arbejde med variabler i COBOL.

Tilføj en variabel i WORKING-STORAGE SECTION

Rediger din eksisterende Hello.cob-kode til følgende:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 VAR-TEXT          PIC X(30) VALUE "HELLO med Variabel".

PROCEDURE DIVISION.
* Nedenfor kommer en DISPLAY - COBOLs måde at skrive i konsollen
  DISPLAY VAR-TEXT
  STOP RUN.
```

Her har vi:

- En variabel VAR-TEXT, der er placeret i **WORKING-STORAGE SECTION**.
- Variablen er defineret som **PIC X(30)**, hvilket betyder, at den kan indeholde op til 30 alfanumeriske tegn (bogstaver og tal).
- Vi giver den en **standardværdi** med VALUE "HELLO med Variabel".

2. Byg og test programmet

- Følg samme trin som vist i **Del 1** for at bygge din nye version af programmet.
- Når du kører programmet, vil terminalen nu vise:

```
C:\Users\kontonavn\VsCodeWS\Cobol> ./hello.exe
```

```
HELLO med Variabel
```

```
C:\Users\kontonavn\VsCodeWS\Cobol>
```

Tillykke! Du har nu lært at oprette et COBOL-program, bygge det til en eksekverbar fil og udskrive både faste strenge og variabler direkte i terminalen.

Opgave 2 - Variabler og MOVE

Formål: Lær hvordan man opretter forskellige typer variabler og flytter data mellem dem i COBOL.

Del 1 – Program med flere variabler og move statement

1. Opret en ny COBOL-fil

Giv filen navnet Opgave2.cob.

2. Opret variabler – Find inspiration i bilaget om datatyper:

Her er de variabler, du skal indarbejde i programmet. Tænk over, hvilken datatype der passer bedst til hver variabel, og find hjælp i bilaget for datatyper.

Kunde-id	Alfanumerisk – 10 karakterer
Fornavn	Alfanumerisk – 20 karakterer
Efternavn	Alfanumerisk – 20 karakterer
Kontonummer	Alfanumerisk – 20 karakterer
Balance	Numerisk, med 2 decimaler – plads til 1 mio.
Valutakode	Alfanumerisk – 3 karakterer

Variablerne oprettes enkeltvis på en linie for sig. De kan initieres med blanke eller nul.

```
01      XXXXX  PIC X(10)  VALUE SPACES.  
01      YYYYY  PIC 9(7)V99 VALUE ZEROS.
```

Del 2: Flyt data med MOVE

Efter variablerne er oprettet, skal du flytte data til dem. Brug MOVE-statement for at tildele værdi til hver variabel. Tænk over data, der passer til variablerne:

KUNDE-ID: fx "1234567890".

FORNAVN: fx "Lars".

EFTERNAVN: fx "Hansen".

KONTONUMMER: fx "DK12345678912345".

BALANCE: fx 2500.75.

VALUTAKODE: fx "DKK".

Del 3: Udskriv data

Brug **DISPLAY**-kommandoen til at udskrive variablene i terminalen. Udskriv dem som følger:

1. **Kunde-ID**
2. **Fornavn og efternavn**
3. **Kontonummer**
4. **Balance og valutakode**

Build og kør programmet

Følg de trin i **Opgave 1** for at bygge programmet med batch-filen cobbuild.bat:

```
C:\Users\kontonavn\VsCodeWS\Cobol> ./cobbuild.bat -x Opgave2.cob -o Opgave2.exe -lcob
```

Kør den genererede eksekverbare fil, og observer resultatet:

```
C:\Users\kontonavn\VsCodeWS\Cobol> ./Opgave2.exe
```

Resultater og refleksion

Når programmet eksekveres, vil du se data udskrevet i terminalen. Bemærk, at nogle variable kan have blanke karakterer. Dette skyldes COBOL's standardadfærds for alfanumeriske felter.

I næste opgave skal vi forbedre programmets udskriftsresultater og fokusere på at fjerne blanke felter. 😊

Opgave 3 – Løkker og strenghåndtering

Formål: Lær at oprette løkker og håndtere/modificere strenge i COBOL. Du vil lære at kombinere data fra flere variabler, fjerne overskydende mellemrum og arbejde med loop-baserede datamodifikationer.

I Cobol er der flere måder at lave løkker på. Se dokumentation på
https://www.tutorialspoint.com/cobol/cobol_loop_statements.htm

Del1 - Kombinér fornavn og efternavn til én variabel

1. Kopier Opgave2.cob og kald den Opgave 3.cob

2. Flyt fornavn og efternavn til en ny variabel

- Opret en ny variabel på 40 karakterer i **WORKING-STORAGE SECTION**, som skal indeholde både fornavn og efternavn.
- Brug **STRING**-kommandoen til at samle indholdet af fornavnet og efternavnet i den nye variabel. Tilføj et enkelt mellemrum mellem fornavn og efternavn.
- Koden kunne fx se sådan ud:

```
01      VAR-1          PIC X(20) VALUE SPACES.  
01      VAR-2          PIC X(20) VALUE SPACES.  
01      VAR-3          PIC X(40) VALUE SPACES.  
  
STRING VAR-1 DELIMITED BY SIZE " "  
      DELIMITED BY SIZE VAR-2  
      DELIMITED BY SIZE  
      INTO VAR-3
```

Eksempel på formålet med **STRING**:

- VAR-1 DELIMITED BY SIZE: Hele indholdet af VAR-1 kopieres til den nye variabel, uanset indhold (mellemrum eller ikke).
- " " DELIMITED BY SIZE: Et enkelt mellemrum tilføjes efter VAR-1.
- VAR-2 DELIMITED BY SIZE: Hele indholdet af VAR-2 tilføjes efter mellemrummet.
- **Resultat**: Kombinationen gemmes som én samlet tekststreng i VAR-3.

3. Display

Før du går videre til næste trin, så udskriv din nye variable VAR-3 ved hjælp af en DISPLAY-kommando for at tjekke resultatet.

Del 2 - Fjern overskydende mellemrum

Nu har du en variabel med navnene kombineret, men den kan også indeholde overskydende mellemrum. For at fjerne unødvendige mellemrum skal du oprette en løkke, der gennemgår strengens indhold og fjerner gentagne mellemrum.

Trin for trin:

1. Opret nødvendige variabler

I **WORKING-STORAGE SECTION** skal du oprette disse variabler:

- **Indeksvariabel:** Bruges til at holde styr på hvilken position i variablen, du arbejder med (fx IX).
- **Indeks til outputvariabel:** Bruges til at holde styr på placeringen af det næste tegn i din rensede variabel (fx IX2).
- **Midlertidige variabler til tegnkontrol:**
 - En variabel til det **nuværende tegn** (fx CURRENT-CHAR).
 - En variabel til det **forrige tegn** (fx PREVIOUS-CHAR).
- **Outputvariabel:** Den variabel, der skal indeholde det rensede navn uden gentagne mellemrum.

2. Lavet loop der går gennem strengen

Du skal lave en loop, der går gennem hver position i variablen med det fulde navn.

- Brug følgende formel for at hente ét tegn ad gangen fra variablen, baseret på en indeksværdi (IX):

Syntax:

```
VARIABEL(IX:1)
```

Her angiver IX positionen i strengen, og :1 angiver, at du kun vil hente ét tegn ad gangen.

3. Fordelingen af processen – Pseudokode

Når du arbejder i løkken, skal du følge denne proces:

- Start løkken fra den første position i variablen.
- Brug en sammensætning af IF-betingelser i løkken til at tjekke hvert tegn:
 1. Tjek, om det **nuværende tegn** (CURRENT-CHAR) er et mellemrum.
 2. Hvis det **nuværende tegn** ikke er et mellemrum, eller det **forrige tegn** (PREVIOUS-CHAR) ikke er et mellemrum:
 - Kopiér det nuværende tegn til den **rensede variabel** på den næste ledige position.
 - Opdatér positionen for den **rensede variabel** (fx brug ADD 1 TO IX2).
 - Ellers spring tegnet over.
- Opdater **PREVIOUS-CHAR**, så du kan sammenligne med den næste karakter, der skal behandles.
- Gentag processen, indtil hele den kombinerede variabel er gennemgået.

Hints til løkkeoprettelse

- Brug **PERFORM VARYING** til at oprette din løkke. Denne struktur er ideel til gentagne operationer over et kendt antal iterationer:

```
PERFORM VARYING IX FROM 1 BY 1 UNTIL IX > LENGTH OF INPUT-VAR
  * Her skal tegn behandles
END-PERFORM
```

- Brug **IF**-betingelsen til at evaluere tegn. Husk, at COBOL understøtter syntaks som:

```
IF CURRENT-CHAR NOT = SPACE OR PREVIOUS-CHAR NOT = SPACE
  * Handling
END-IF
```

- Opdater tælleren for outputvariablen (IX2) efter hvert nyttigt tegn med:

```
ADD 1 TO IX2
```

4. Test dine resultater med DISPLAY

Når løkken og tegnkontrollen er færdig, udskriv den rensede variabel med DISPLAY-kommandoen og sammenlign med den originale FULD-NAV, som du gemte i **Del 1**.

Del 3: Udskrivning af kundeinfo

Til sidst skal du udskrive alle kundeoplysninger på et konsollen:

- Kunde-ID.
- Det rensede navn fra Del 2.
- Kontonummer.
- Balance og valutakode.

Brug eksempelvis en visuel separator for at gøre teksten mere ryddelig, som:

```
DISPLAY "-----"  
DISPLAY "Kunde ID      : " KUNDE-ID  
DISPLAY "Navn (renset) : " RENS-FULD-NAV  
DISPLAY "Kontonummer   : " KONTONUMMER  
DISPLAY "Balance       : " BALANCE " " VALUTAKODE  
DISPLAY "-----"
```

Byg og kør programmet

- Som tidligere skal du bygge programmet ved at eksekvere cobbuild.bat:

```
C:\Users\kontonavn\VsCodeWS\Cobol> ./cobbuild.bat -x Opgave3.cob -o  
Opgave3.exe -lcob
```

- Kør programmet:

```
C:\Users\kontonavn\VsCodeWS\Cobol> ./Opgave3.exe
```

Debugging-tip

Hvis du ikke får det ønskede resultat:

- Tilføj flere DISPLAY-linjer i løkken, fx:
 - Udskriv værdien af CURRENT-CHAR (det aktuelle tegn).
 - Udskriv værdien af PREVIOUS-CHAR (det forrige tegn).
 - Udskriv den rensede variabel (RENS-FULD-NAV) undervejs.

Opgave 4 Introduktion til strukturer i COBOL

Formål: Forstå konceptet bag strukturer i COBOL og hvordan de anvendes til at organisere og arbejde med relaterede data som én enhed. Fokus vil være på opbygning af strukturer, manipulation af data og udskrivning af strukturen som helhed.

Introduktion til strukturer i COBOL

Strukturer i COBOL gør det muligt at:

1. Organisere data under hierarkier ved hjælp af **niveau-numre**.
2. Arbejde med grupper af data som en samlet enhed.
3. Tilgå individuelle felter inden for en struktur.

Nøglebegreber:

- **Niveau-numre:** Bruges til at definere hierarkiet for datastrukturen.
 - 01: Definerer hovedgruppen for en datastruktur.
 - 02-49: Bruges til underfelter inden for hovedgruppen.
 - 77: Definerer selvstændige variabler.
 - 88: Bruges til betingelser for specifikke værdier.
- **Gruppelementer:** Bruges til at samle relaterede data i én logisk enhed.

Eksempel på en struktur

Lad os sige, vi har en situation, hvor vi skal gemme information om en person (fornavn, efternavn, alder og adresse). Dette kan vi organisere i en struktur:

```
DATA DIVISION.  
  WORKING-STORAGE SECTION.  
    01 PERSON-INFO.  
      02 FORNAVN          PIC X(20).  
      02 EFTERNAVN        PIC X(20).  
      02 ALDER            PIC 9(3).  
      02 ADRESSE.  
        03 VEJNAVN         PIC X(30).  
        03 BY-X             PIC X(20).  
        03 POSTNR           PIC X(4).
```

Forklaring af strukturen:

01 PERSON-INFO:

- Dette er hovedgruppen, som repræsenterer hele datastrukturen for en person.

02 FORNAVN:

- Indholder personens fornavn, som kan være op til 20 tegn langt.

02 EFTERNAVN:

- Indholder personens efternavn, som også kan være op til 20 tegn langt.

02 ALDER:

- Indholder personens alder, som er et numerisk felt med op til 3 cifre.

02 ADDRESSE:

- Dette er en undergruppe, der repræsenterer personens adresse.

Felterne under ADDRESSE:

- 03 VEJNAV: Indholder vejnavnet, som kan være op til 30 tegn langt.
 - 03 BY-X: Indholder bynavnet, som kan være op til 20 tegn langt. Da BY er et reserveret ord i Cobol, bliver vi nødt til at omgå dette ved at putte omdøbe det.
 - 03 POSTNR: Indholder postnummeret, som kan være op til 4 tegn langt.
-

Fordele ved strukturer

1. Organisering:

- o Strukturen organiserer relaterede data på en logisk måde. Alle informationer om en person gemmes under én gruppe (PERSON-INFO).

2. Nem adgang til data:

- o Du kan tilgå individuelle felter i strukturen ved at bruge deres navne, f.eks. FORNAVN eller ADDRESSE.VEJNAV.

3. Gruppemanipulation:

- o Hele gruppen kan behandles som én variabel. F.eks. kan du flytte hele PERSON-INFO til en anden struktur af samme type.

Del 1 - Opret en struktur til kundedata

1. Kopier Opgave3.cob

Lav en kopi af din fil fra Opgave 2, og gem den som **Opgave4.cob**.

2. Udskift eksisterende variabler med en struktur

I stedet for at definere variablerne individuelt, grupper data ved hjælp af en 01 struktur kaldet KUNDEOPL. Strukturen skal indeholde de samme felter som i Opgave 2.

Her er variablerne fra opgave 2

01	KUNDE-ID	PIC X(10)	VALUE SPACES.
01	FORNAVN	PIC X(20)	VALUE SPACES.
01	EFTERNAVN	PIC X(20)	VALUE SPACES.
01	KONTONUMMER	PIC X(20)	VALUE SPACES.
01	BALANCE	PIC 9(7)V99	VALUE ZEROS.
01	VALUTAKODE	PIC X(3)	VALUE SPACES.

Nu skal de organiseres under hovedgruppen 01 KUNDEOPL, hvor du skal putte KUNDE-ID, FORNAVN og EFTERNAVN ind under som 02 niveauer.

For de sidste 3 variabler kan du putte dem ind under et nyt 02 niveau KONTOINFO.

3. Build og kør programmet

Som tidligere kan du bygge og køre programmet for at sikre, at udskriften matcher resultatet fra Opgave 2.

Del 2 – Udskriv hele strukturen

Man kan også skrive hele strukturen ud med et statement – nemlig ved at lave en DISPLAY på 01 niveauet.

1. Prøv dette og check output i terminalen

Den vil så skrive alle variabler ud på 1 linje i en række efter hinanden.

Opgave 5 Introduktion til Copybooks i COBOL

Copybooks er en vigtig del af COBOL-programmering og bruges til at genbruge kode og opretholde konsistens i større programmer. De fungerer som små stykker kode, der kan inkluderes i andre COBOL-programmer ved hjælp af en **COPY**-kommando. Copybooks bruges primært til at definere **datastrukturer**, **konstante værdier** eller **proceduresektioner**, der ofte genbruges.

Hvad er en Copybook?

En **Copybook** er en fil, der indeholder en del af koden, som du kan inkludere i flere COBOL-programmer. Den gør det muligt at undgå gentagelse af den samme kode i forskellige programmer, hvilket sparer tid og gør vedligeholdelse lettere.

Typisk anvendes copybooks til:

1. **Datastrukturer:**

Definition af felter og grupper, som bruges flere steder i programmet.

Eksempel: Definition af en medarbejderstruktur, som kan bruges i både lønudbetaling og rapportering.

2. **Konstante værdier:**

Brug af faste værdier, såsom landekoder eller statuskoder.

Hvorfor bruge Copybooks?

1. **Genbrug af kode:**

Copybooks gør det muligt at genbruge den samme kode i flere programmer, hvilket reducerer duplikering.

2. **Konsistens:**

Ved at bruge copybooks sikrer du, at datastrukturer eller procedurer er ens på tværs af flere programmer.

3. **Let vedligeholdelse:**

Hvis du skal ændre noget i en struktur, kan du opdatere copybooken ét sted, og ændringen vil automatisk gælde i alle programmer, der refererer til den.

4. **Tidsbesparelse:**

Copybooks sparer tid, da du ikke behøver at skrive den samme kode igen og igen.

Del 1 – Oprettelse af en Copybook

1. Opret en ny fil i samme mappe, hvor dine programmer ligger.
 - Giv filen navnet **KUNDER.cpy**.
2. Kopiér datastrukturen fra **Opgave 4** (uden 01 niveauet!) og tilføj de nedenstående nye variabler.

```
* Den tidligere struktur + disse variabler
 02 ADRESSE.
  03 VEJNAVN      PIC X(30).
  03 HUSNR        PIC X(5).
  03 ETAGE         PIC X(5).
  03 SIDE          PIC X(5).
  03 BY            PIC X(20).
  03 POSTNR        PIC X(4).
  03 LANDE-KODE   PIC X(2).

 02
  03 TELEFON      PIC X(8).
  03 EMAIL         PICX(50).
```

Gem filen.

KUNDER.cpy er nu din Copybook-fil, som du kan inkludere i alle programmer, hvor denne struktur er nødvendig.

Del 2 – Inkludér Copybook i et program

1. **Kopier Opgave4.cob:**
Omdøb kopien til **Opgave5.cob**.
2. **Inkludér Copybook i PROGRAM-ID**
Erstat strukturen i **WORKING-STORAGE SECTION** med følgende kode for at inkludere Copybooken:

```
01 KUNDEOPL.
  COPY "KUNDEOPL.cpy".
```

3. **Tilføj data til de nye felter**
Fyld noget data i de nye variabler fra copybook'en.
4. **Display de nye felter**
For at teste, at din Copybook fungerer, skal du udskrive de nye variable fra strukturen. Du kan gøre dette individuelt eller som en samlet udskrift:

Extra info - Hvis man har dublet variabler i sin kode, eksempelvis hvis man har brug for 2 ens strukturer, skal man kvalificere dem med strukturnavnet

```
01 KUNDEOPL-01.  
      COPY "KUNDEOPL.cpy".  
01 KUNDEOPL-02.  
      COPY "KUNDEOPL.cpy".  
  
DISPLAY FORNAVN OF KUNDEOPL-01.  
DISPLAY FORNAVN OF KUNDEOPL-02.
```

Prøv dette 😊

Opgave 6 – Læsning af fil

Formål:

Lær, hvordan man håndterer filer i COBOL, herunder hvordan man åbner, læser og arbejder

Hvad er filhåndtering i COBOL?

Filhåndtering er en grundlæggende funktion i COBOL, der gør det muligt at læse og skrive data fra og til eksterne filer. Processen involverer fire trin:

1. **Definering:** Angivelse af filens placering og struktur (f.eks. dens layout).
2. **Åbning:** Åbning af filen, så der kan læses eller skrives data.
3. **Læse:** Læse poster fra filen en ad gangen.
4. **Lukning:** Lukning af filen, når arbejdet er færdigt.

Filhåndtering i COBOL kræver nøje strukturering af inputdata, da placeringen af data i filen skal matche de definerede variabler i programmet præcist.

Et simpelt eksempel

Forestil dig, at vi har en tekstfil kaldet datafile.txt, der ser sådan ud:

Position:

1234567

John 30
Jane 25
Alice28
Bob 35

Hver linje i filen har:

- Et **navn** (1-5 tegn) efterfulgt af...
- En **alder** (2 cifre).

I COBOL defineres filens datastruktur sådan her:

```
01 FILE-RECORD.  
  * Name variablen indeholder position 1 - 5  
  05 NAME    PIC X(5).  
  * AGE variablen indeholder position 6 - 7  
  05 AGE    PIC 99.
```

Hver linje indeholder et navn (5 tegn) og en alder (2 cifre). Positioner er meget vigtigt i denne fil, da de skal svare overens med den struktur man læser filen ind i.

Del 1 – Indlæs Kundedata fra en fil

Følg disse trin for at oprette et program, der læser kundedata fra en tekstfil, hvor data passer til din tidligere oprettede **KUNDEOPL**-struktur.

1. Opret tekstfilen **Kundedata.txt**

1. Lav en ny tekstfil i din editor, og gem den i samme mappe som din COBOL-programkode.
2. Kald filen **Kundedata.txt**.
3. Indsæt 5 rækker i filen, hvor data svarer til den **KUNDEOPL**-struktur, du oprettede i det tidligere program. Du skal selv finde på noget data, og sikre det passer til strukturem
4. Kopier programmet fra opgave 5 til Opgave6.cob
5. Kig i **Bilag 2** for at se hvordan du læser filen ind i programmet, og brug dette til at indlæse kundedata
6. Udskriv oplysninger i konsollen og check at filen er rigtig læst.

Opgave 7 skrivning i fil

Skrivning til en fil fungerer langt hen ad vejen som læsning af fil:

1. Definere filen (hvor den findes og dens struktur).
2. Åbne filen.
3. Skrive poster til filen.
4. Lukke filen.

Bilag 2 indeholder et programeksempel – der både læser og skriver i filer.

Del 1 – Skriv samme indhold som der er blevet læst

1. Kopier programmet fra opgave 6 og kald det Opgave7.cob
2. Opret en tom txt fil i samme mappe som programmet. Kald filen Kundetekst.txt
3. Ret i koden, så programmet skriver det der er læst – i samme struktur ned i den nye fil. Alt hvad du skal bruge, er i Bilag 2

Del 2 – Lav ændringer til skrive strukturen

1. Opret en ny struktur til skrivning i filen

- For at skrive data til filen skal vi definere en ny struktur i programmet.

2. Den nye struktur

- Strukturen skal repræsentere én linje i output-filen og kan se således ud:

```
01  KUNDE-ADR.  
    02  NAVN-ADR          PIC X(100).
```

3. Behandling af data og skrivning i fil

- Når oplysningerne fra inputfilen er læst, skal de behandles linje for linje for at skabe pænt formaterede adresselinjer i outputfilen.
- Hver datalinje flyttes til variablen NAVN-ADR i KUNDE-ADR og skrives til outputfilen.

- Efter hver skriving **initialiseres NAVN-ADR med tomme værdier** ved hjælp af følgende kommando:

```
MOVE SPACES TO NAVN-ADR
```

Fremgangsmåde til behandling og skrivning af data

1. **Flyt KUNDE-ID til NAVN-ADR og skriv linjen:**
 - KUNDE-ID flyttes direkte til NAVN-ADR.
 - Skriv derefter resultatet i filen.
2. **Sammensæt FORNAVN og EFTERNAVN og skriv linjen:**
 - Kombinér fornavn og efternavn til én samlet linje, fx med STRING, og flyt det derefter til NAVN-ADR.
 - Skriv resultatet til filen.
3. **Sammensæt VEJNAVN, HUSNR, ETAGE og SIDE, og skriv linjen:**
 - Formater adresselinjen med vejnavn, husnummer, etage og side samlet til én linje.
 - Flyt resultatet til NAVN-ADR og skriv linjen.
4. **Sammensæt POSTNR og BY, og skriv linjen:**
 - Kombinér postnummer og bynavn til én linje og flyt det til NAVN-ADR.
 - Skriv linjen til filen.
5. **Skriv TELEFON og EMAIL hver for sig:**
 - Flyt telefonnummer til NAVN-ADR og skriv linjen.
 - Flyt e-mail-adressen til NAVN-ADR og skriv linjen.
6. **Tilføj en blank linje mellem kunder:**
 - Initialisér hele KUNDE-ADR ved at flytte SPACES til NAVN-ADR, og skriv en blank linje i outputfilen.
 - Dette forbereder programmet på næste kundes data.

På denne måde sikrer vi, at outputfilen bliver struktureret og letlæselig med hver kundes oplysninger arrangeret i en pæn format: **En kunde vises som en blok af linjer, adskilt af en blank linje.**

Del 3 - Paragraffer

Man kan med fordel lave nogle nye paragraffer(metoder) til at behandle de enkelte punkter. Herved for man en bedre læsbarhed i koden. Nedenfor er et eksempel på en paragraf og hvordan den kaldes:

```
* nu er data indlæst fra filen
    PERFORM FORMAT-NAVN
    DISPLAY "Formateret navn: " FULD-NAVN
    * fortsæt med andre paragraffer
    PERFORM FORMAT-VEJ
    PERFORM FORMAT-BY

    STOP RUN.

    * Paragraf til behandling af navn
    FORMAT-NAVN.
        STRING FORNAVN DELIMITED BY SPACE
            " " DELIMITED BY SIZE
            EFTERNAVN DELIMITED BY SPACE
            INTO FULD-NAVN
        END-STRING
    EXIT.
    * Paragraf slut
```

I hver paragraf – formater de enkelte dele, så der ikke er overskydende mellemrum. Som ved navn ovenfor.

Opgave 8 Læsning af flere filer og skriv i output fil

Del 1 – Strukturer ny fil med konto oplysninger

Kopier program fra Opgave 7

1. Åbn dit program fra Opgave 7, og lav en kopi af filen.
2. Gem kopien som Opgave8.cob
3. Lav en ny tom fil til konto data – kald filen KontoOpl.txt
4. Lav en ny copybook fil – kald den KONTOOPL.cpy
- 5.

Nu skal det være lidt sværere – vi skal læse fra flere filer og kombinere i en output fil

1. Kopier programmet fra opgave 7 – og kald det Opgave8.cob
2. Lav en ny fil med konto data – kald filen
3. Lav en ny copybook fil – kald den KONTOOPL.cpy

```
02 KUNDE-ID          PIC X(10).  
02 KONTO-ID          PIC X(10).  
02 KONTO-TYPE        PIC X(20).  
02 BALANCE           PIC ZZZZZ9V99.  
02 VALUTA-KD         PIC X(3).  
.....
```

4. Indsæt nogle linjer i KontoOpl.txt, der passer til ovenstående struktur

VIGTIGT – KUNDE-ID skal matche de KUNDE-ID'er der er i den anden input fil.

I programmet kan du bruge dette 01 niveau for copybook:

```
01 KONTO-REKORD.  
    COPY "KONTOOPL.cpy".
```

5. Opret en ny struktur – hvor der kan skrives ud i – og en tom txt fil kaldet KUNDEKONTO.txt

Strukturen skal "bare" være en lang streng – der kan holde oplysningerne fra opgave 7 – samt konto informationer der fylder 53 bytes – samt lidt mellemrum for læsbarheden:

```
01  KUNDEKONTO.  
02  OUTPUT-TEXT          PIC X(100).
```

Del 2 – Læs begge filer og udskriv oplysninger

Logikken skal være følgende:

1. Læs rækker fra kundefilen i en løkke
2. For hver række der læses i kundefilen – Skrives kunde data ud i filen, som i opgave8
3. Hvis kunden har nogen konti, skal disse også skrives ud. Dette gøres ved at løbe hele kontofilen igennem i en løkke, og for hver række læst, checkes, om der er et match imellem KUNDE-ID der både findes i Kunde og kontofilen
Hvis der er match – skrives kontooplysninger også ud i filen – i en række (01 Niveau)
Når du rammer "end of file" er det vigtigt at den lukkes. Og så skal den åbnes igen ved næste kunde. På kan filen læses igen og igen.

Opgave 9 - Introduktion til arrays

Hvad er arrays i COBOL?

Et array i COBOL kaldes en **tabel** eller **forekomst** (på engelsk: "table" eller "occurs"). Det er en datastruktur, der bruges til at gemme flere værdier af samme type i én variabel. I stedet for at definere mange individuelle variabler, kan man bruge et array til at organisere og gemme data i en liste.

Eksempel på situation

Forestil dig, at du har en liste med navne på 10 kunder, som du vil gemme. I stedet for at definere 10 separate variabler som KUNDE-NAVN-1, KUNDE-NAVN-2, osv., kan du definere ét array, der indeholder alle navnene.

I Cobol defineres det således:

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
01 KUNDE-NAVNE.  
    05 KUNDE-NAVN PIC X(20) OCCURS 10 TIMES.
```

Dette kan også laves på øverste niveau, hvis man eksempelvis bruger en copybook. Så hvis man vil læse alle rækker ind fra kontofilen, kan man gøre dette i et array:

```
01 KONTO-ARRAY OCCURS 20 TIMES.  
    COPY "KONTOOPL.cpy".
```

Så nu kan man starte med at læse kontofilen ind i dette array. For hver række skal man bruge en tæller – som er en numerisk variabel der starter på 1 og lægges 1 til for hver række der læses.

```
* Loop start until end of file  
* Læs række fra fil  
MOVE KONTOOPL-IN TO KONTOOPL-AR(IX)  
ADD 1 to IX  
* Loop slut
```

Del 1 – Brug array filhåndtering

1. Kopier opgave 8 – og gem den som Opgave9.cob
2. I stedet for at læse kontofilen hver gang i loop'et – indlæses filen nu en gang i array
– før læsning af kundefilen
3. Brug array til at matche KUNDE-ID i stedet for fillæsningen.
4. Gem og kør programmet – skulle gerne ligne opgave 8 output.

Bilag 1 – Datatyper

Introduktion til COBOL-datatyper og MOVE

COBOL har en struktureret måde at håndtere data på. Alt data deklarereres i DATA DIVISION, typisk i WORKING-STORAGE SECTION, og hver variabel får en datatype via PIC-mønsteret.

MOVE-sætningen bruges til at kopiere eller konvertere data mellem variabler.

1. Alfanumeriske variabler (PIC X)

- Beskrivelse: Indeholder tekst, bogstaver, tal som tekst eller symboler.
- Deklaration og initialisering:

01 WS-NAME PIC X(20) VALUE SPACES. *-> Initialiseret til blanke

- MOVE-eksempel:

MOVE "Michael" TO WS-NAME

DISPLAY WS-NAME

- Output: Michael (resten af 20 tegn er blanke)
-

2. Numeriske variabler (PIC 9)

- Beskrivelse: Indeholder tal, heltal eller decimaltal.
- Heltal:

01 WS-AGE PIC 9(3) VALUE ZEROS. *-> Initialiseret til 000

- Decimaler:

01 WS-BALANCE PIC 9(5)V99 VALUE ZEROS. *-> Initialiseret til 00000.00

- MOVE-eksempler:

MOVE 28 TO WS-AGE

MOVE 1234.56 TO WS-BALANCE

DISPLAY WS-AGE WS-BALANCE

3. Boolean-lignende variabler (88-level)

- COBOL har ikke indbygget boolean, men kan definere 88-level conditions:

01 WS-STATUS PIC X VALUE "N".

 88 STATUS-OK VALUE "Y".

 88 STATUS-NOT-OK VALUE "N".

- MOVE-eksempel:

MOVE "Y" TO WS-STATUS

IF STATUS-OK

 DISPLAY "Status: OK"

END-IF

4. Initialisering med SPACES og ZEROS

- Alfanumerisk: SPACES

01 WS-TEXT PIC X(10) VALUE SPACES.

- Numerisk: ZEROS

01 WS-NUM PIC 9(5) VALUE ZEROS.

5. Konvertering mellem typer med MOVE

- COBOL kan automatisk konvertere mellem PIC X og PIC 9, hvis data er gyldige:

01 WS-AGE-NUM PIC 9(3) VALUE ZEROS.

01 WS-AGE-TXT PIC X(3) VALUE SPACES.

MOVE WS-AGE-NUM TO WS-AGE-TXT *> Numerisk til tekst

MOVE "028" TO WS-AGE-NUM *> Tekst til numerisk

- Bemærkning: Hvis tekst indeholder ikke-numeriske tegn, kan konvertering til numerisk give fejl.
-

6. Eksempelprogram med MOVE

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. DEMO.  
  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 WS-NAME      PIC X(20) VALUE SPACES.  
01 WS-AGE       PIC 9(3)  VALUE ZEROS.  
01 WS-BALANCE  PIC 9(5)V99 VALUE ZEROS.  
01 WS-STATUS    PIC X      VALUE "N".  
    88 STATUS-OK     VALUE "Y".  
    88 STATUS-NOT-OK VALUE "N".  
  
PROCEDURE DIVISION.  
    MOVE "Michael" TO WS-NAME  
    MOVE 28 TO WS-AGE  
    MOVE 1234.56 TO WS-BALANCE  
    MOVE "Y" TO WS-STATUS  
  
    DISPLAY WS-NAME WS-AGE WS-BALANCE  
  
    IF STATUS-OK  
        DISPLAY "Status: OK"  
    ELSE  
        DISPLAY "Status: Ikke OK"  
    END-IF
```

Dette eksempel viser initialisering, MOVE, konvertering og 88-level boolean i praksis.

Bilag 2 filhåndtering

Programeksempel der læser fil ind i struktur, og skriver i en anden fil – samme data:

```
IDENTIFICATION DIVISION.  
  PROGRAM-ID. ReadWriteFile.  
  
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
  SELECT INPUT-FILE ASSIGN TO "datafile.txt"  
    ORGANIZATION IS LINE SEQUENTIAL.  
  SELECT OUTPUT-FILE ASSIGN TO "outputfile.txt"  
    ORGANIZATION IS LINE SEQUENTIAL.  
  
DATA DIVISION.  
FILE SECTION.  
FD INPUT-FILE.  
01 INPUT-RECORD.  
  05 NAME      PIC X(5).  
  05 AGE       PIC 99.  
  
FD OUTPUT-FILE.  
01 OUTPUT-RECORD.  
  05 NAME      PIC X(5).  
  05 AGE       PIC 99.  
  
WORKING-STORAGE SECTION.  
01 END-OF-FILE  PIC X VALUE "N".  
  
PROCEDURE DIVISION.  
MAIN-PROCEDURE.  
  OPEN INPUT INPUT-FILE  
  OPEN OUTPUT OUTPUT-FILE  
  
  PERFORM UNTIL END-OF-FILE = "Y"  
    READ INPUT-FILE INTO INPUT-RECORD  
    AT END  
      MOVE "Y" TO END-OF-FILE  
    NOT AT END  
      MOVE INPUT-RECORD TO OUTPUT-RECORD  
      WRITE OUTPUT-RECORD  
      DISPLAY "Name: " NAME ", Age: " AGE  
    END-READ  
  END-PERFORM  
  
  CLOSE INPUT-FILE
```

CLOSE OUTPUT-FILE
STOP RUN.