

1 Herhaling sequentiële bestanden

1.1 Bestandsbeschrijving

Voorbeeld:

```
File-control.  
select optional nummer-file assign to "nummer.dat"  
organization sequential  
access mode sequential  
file status is nummer-fs.
```

1.2 Recordbeschrijving

Voorbeeld:

```
File section.  
fd nummer-file.  
01 nummer-record.  
    03 nummer-data.  
        05 nummer-boeknr          pic 9(4).  
        05 nummer-auteursnr      pic 9(4).  
  
working-storage section.  
01 nummer-fs                      pic xx.
```

1.3 Bewerkingen

- open $\left. \begin{array}{l} \text{input} \\ \text{output} \\ \text{I-O} \\ \text{extend} \end{array} \right\}$ filename
- read filename next record
at end ...
not at end ...
end-read
- write recordname
end-write
- rewrite recordname
end-rewrite
- close filename

1.4 Opmerkingen

Bij een sequentieel bestand is enkel een sequentiële toegangsfunctie mogelijk, d.w.z. een READ ... is eigenlijk een READ ... NEXT...

De volgorde waarin de records van een sequentieel bestand verwerkt worden is gelijk aan de volgorde waarin de records werden weggeschreven naar het bestand bij creatie.

Uitzonderingssituaties worden opgevangen door AT END.

Tussenvoegen en verwijderen van records is onmogelijk tenzij door gebruik te maken van een nieuw bestand.

1.4.1 5 Overzicht

Wanneer een sequentieel bestand moet worden gecreëerd, dan:

- moet het bestand worden geopend voor OUTPUT;
- mag alleen de WRITE-opdracht worden gebruikt;
- worden de records in dezelfde volgorde in het bestand geplaatst als waarin ze worden weggeschreven.

Wanneer records moeten worden toegevoegd aan het einde van een reeds aanwezig sequentieel bestand, dan:

- moet het bestand worden geopend voor EXTEND;
- mag alleen de WRITE-opdracht worden gebruikt;
- worden de records in dezelfde volgorde aan het bestand toegevoegd als waarin ze worden geschreven, te beginnen na het laatste record op het moment van de OPEN EXTEND uitvoering.

Wanneer een sequentieel bestand moet worden geraadpleegd, dan:

- kan het bestand het beste voor INPUT worden geopend (I-O mag ook);
- mag alleen de READ-opdracht worden gebruikt (READ ... NEXT);
- worden de records gelezen in dezelfde volgorde als waarin ze zijn geschreven.

Wanneer een sequentieel bestand op een niet adresseerbaar geheugenmedium moet worden gemuteerd, dan kan dat alleen door de al dan niet gewijzigde records van dat bestand te schrijven naar een nieuw bestand.

Wanneer een sequentieel bestand op een adresseerbaar geheugenmedium moet worden gemuteerd, dan:

- moet het bestand voor I-O worden geopend;
- moeten alle records worden gelezen;
- kunnen gewijzigde records worden teruggeschreven door de REWRITE-opdracht;
- worden de records verwerkt in dezelfde volgorde waarin ze zijn geschreven.

In de onderstaande tabel vind je een overzicht van de toegestane opdrachten bij de diverse methoden van openen:

open mode van het bestand

	open input	I-O	output	extend
read	+	+	-	-
rewrite	-	+	-	-
write	-	-	+	+

2 Oefeningen

2.1 Herhaling sequentiële bestanden

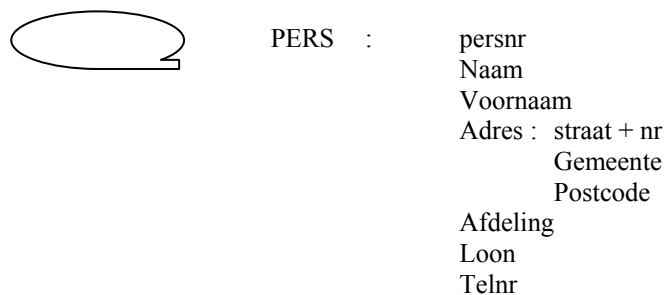
Oefening 1:

Ontwerp een programma in cobol dat een sequentieel bestand met slechts één record aanmaakt. Het sequentiële bestand 'nummer.dat' bevat het laatst gebruikte nummer voor persnr (pic 9(4)). In deze initiële fase wordt het record met nullen gevuld.



Oefening 2 :

Ontwerp een programma in cobol dat een personeelsbestand maakt met volgende gegevens :



Maak in eerste instantie een leeg bestand aan. Wijzig het programma nadien zodat men personeelsleden aan het bestand kunnen toevoegen, met het automatisch toekennen van een nieuw nummer.

2.2 Bewerken van source- teksten

Oefening 3:

Pas oefening 2 aan door select en fd clauses te vervangen via een copy
Maak gebruik van de screeditor om schermen te ontwikkelen.

2.3 Werken met inter-program communication

Oefening 4 :

Ontwerp een menuprogramma in COBOL waar je een keuze kan maken met de pijljestoetsen en met nummers.

Het menuscherm ziet er als volgt uit:

HOGESCHOOL GENT	<TIME> <DATE>
HOOFDMENU	
1. Nummer Initialiseren 2. Nummer bekijken 3. Personeelsrecord toevoegen 4. Personeelsrecord raadplegen	
9. Stoppen	KEUZE :__

2.4 Werken met relatieve bestanden

Oefening 5: We voorzien drie nieuwe opties bij het hoofdmenu van oefening 4:

- keuze 5: Omzetting SEQ → REL
- keuze 6: Zoek VAN - TOT
- keuze 7: Beheer personeel (REL)

Keuze 5: Het sequentiële bestand personeel wordt omgezet tot een relatief bestand(P-REL-S) met toegangsmode **indexed**. De record structuur blijft dezelfde voor het nieuwe bestand. De record-key wordt het personeelsnr.

Keuze 6: Maak voor dit nieuwe bestand een zoekprogramma dat via invoer van een personeelsnr een record opzoekt en toont op het scherm. Nadien bestaat de mogelijkheid het volgende record op het scherm te tonen. Na het laatste record wordt automatisch verdergegaan met het eerste record.

Keuze 7: Het beheer Personeel bestaat uit:

1. Leden toevoegen: dit tot de naam van een lid de waarde 'spaces' krijgt. Het auteurnr wordt automatisch gegenereerd.
2. Leden raadplegen:
 - via persnr
 - via een begin- en eindpersnr
 - via personeelsnaam

3. Leden verwijderen (bevestiging vragen) na invoer van een persnr
4. Leden wijzigen na invoer van persnr of persnaam

Opmerking:

- Elke module wordt voorzien van een programmastructuur (Jackson)
- Maak gebruik van scope terminators, copy voor 'sel' en 'fd' en declaratives (toon een passende boodschap i.f.v. de file status)

3 Bewerken van source-teksten

3.1 Inleiding

De eerste faciliteit biedt de mogelijkheid om gebruik te maken van een programmabibliotheek, waarin COBOL programma's of delen daarvan zijn opgeslagen. Die (delen van) programma's kunnen worden vermeld in een nog te compileren programma en met dat programma mee worden vertaald.

Zo kunt u deze faciliteit bij voorbeeld benutten wanneer een bestand in veel programma's wordt gebruikt en de bestandsbenoeming plus de recordbeschrijvingen steeds dezelfde zijn. Ook is deze faciliteit nuttig om subprogramma's, die al eerder gecodeerd en getest zijn, in een programmablok op te nemen.

Bij het overnemen van de tekst in een COBOL-programma is tevens de mogelijkheid aanwezig om delen van de oorspronkelijke tekst te vervangen door andere tekst.

Een programmabibliotheek moet uiteraard zijn opgebouwd, voordat een COBOL-programma wordt gecompileerd. De leverancier bepaalt hoe een programmabibliotheek moet worden opgebouwd en ingedeeld.

Voor deze faciliteit wordt de **COPY-opdracht** gebruikt.

De tweede faciliteit biedt de mogelijkheid om stukken tekst van een programma rechtstreeks te vervangen door andere tekst, voordat het programma wordt gecompileerd. Dit kan in het gehele programma plaatsvinden, of alleen in één of meer aangegeven delen van een programma.

U kunt deze mogelijkheid bij voorbeeld benutten om in oudere programma's namen te vervangen die inmiddels een COBOL-woord zijn geworden.

Voor deze faciliteit wordt de **REPLACE-opdracht** gebruikt.

3.2 De COPY- opdracht

- Formaatbeschrijving

COPY text-name

REPLACING { identifier-1
literal-1
word-1 } BY { identifier-2
literal-2
word-2 }

Let erop dat de copy-opdracht moet worden afgesloten met een punt.

De tekstnaam moet verwijzen naar een deel van de programma-bibliotheek.

- COPY zonder REPLACING

Als u de REPLACING optie in een COPY niet gebruikt, dan wordt het aangegeven programmadeel letterlijk overgenomen; het komt dan in de plaats van de COPY opdracht.

Een voorbeeld:

Onder de naam 'tekst-1' is in een programmabibliotheek de volgende codering opgenomen:

```
FD bestand,  lineage is a lines,  
                With footing at b,  
                Lines at top c,  
                Lines at bottom d.
```

Als u nu het volgende codeert:

```
FILE SECTION.  
    Copy tekst-1.
```

Dan komt de codering er als volgt uit te zien:

```
FILE SECTION.  
    FD bestand,  lineage is a lines,  
                With footing at b,  
                Lines at top c,  
                Lines at bottom d.
```

De codering uit 'tekst-1' is dus letterlijk overgenomen.

- COPY met REPLACING

Stel dat u in het vorige voorbeeld de diverse waarden in de LINAGE clause wilt veranderen, dan kunt u daarvoor de REPLACING optie in de COPY opdracht benutten. Die optie stelt u in staat, om een aangegeven stuk tekst te vervangen door een andere tekst, of om die aangegeven tekst zelfs geheel niet over te nemen.

Het voorgaande voorbeeld kan er dan als volgt uitzien:

```
FILE SECTION.  
    Copy tekst-1, replacing  a by 58,  
                            b by 56,  
                            c by 6,  
                            d by 2.
```

Het resultaat luidt dan:

```
FILE SECTION.  
    FD bestand,  lineage is 58 lines,  
                with footing at 56,  
                lines at top 6,  
                lines at bottom 2.
```

In feite hebt u hier de identifiers a, b, c en d vervangen door de constanten 58, 56, 6 en 2. Althans, zo ziet de compiler dit.

3.3 De REPLACE- opdracht

De REPLACE opdracht functioneert in feite op dezelfde wijze als de COPY opdracht, met twee verschillen:

- De REPLACE opdracht werkt niet op een tekst uit een programmabibliotheek, maar rechtstreeks op het sourceprogramma of op een gedeelte ervan;
- De REPLACE opdracht heeft pas effect vanaf de plaats in het sourceprogramma waar hij voorkomt; het effect kan eindigen op elke door u aan te geven plaats.

Er zijn twee varianten van de REPLACE opdracht : de REPLACE BY opdracht en de REPLACE OFF opdracht. De formaatbeschrijving voor de REPLACE BY opdracht luidt:

Let erop dat de REPLACE opdracht moet worden afgesloten met een punt.
Door de REPLACE BY opdracht wordt vanaf die plaats elke pseudo-tekst-1 in het programma vervangen door pseudo-tekst-2.

Een voorbeeld :

Verwerk.

```
Replace ==teller== by ==art-index==.  
Perform with test after  
    varying teller from 1 by 1  
    until teller = 100  
    add aantal(teller) to totaal ;  
    display aantal(teller) ;  
end-perform
```

Na de vertaling ziet de codering er als volgt uit :

Verwerk.

```
Perform with test after  
    varying art-index from 1 by 1  
    until art-index = 100  
    add aantal(art-index) to totaal ;  
    display aantal(art-index) ;  
end-perform
```

U mag ook meer dan één pseudo-tekst aangeven in een REPLACE BY opdracht. Die pseudoteksten hebben dan tegelijk effect. Als u in het vorige voorbeeld de REPLACE BY opdracht als volgt wijzigt:

```
Replace ==aantal== by ==art-aantal==,  
    ==teller== by ==art-index==.
```

Dan wordt de codering

Verwerk.

```
Perform with test after  
    varying art-index from 1 by 1  
    until art-index = 100  
    add art-aantal(art-index) to totaal ;  
    display art-aantal(art-index) ;  
end-perform
```

Een REPLACE BY opdracht heeft zijn effect totdat een van de volgende situaties is bereikt:

- Het einde van het huidige source-programma (dus ofwel de laatste programmaregel, ofwel de END PROGRAM clause).
Als in het vorige voorbeeld de namen 'teller' en 'aantal' ook nog verderop in het programma voorkomen, dan worden die dus eveneens vervangen.
- Een andere REPLACE BY opdracht. Als u een REPLACE BY opdracht geeft, wordt automatisch het effect van de voorgaande REPLACE BY opdracht gestopt. Als u in het vorige voorbeeld alleen de vervanging van 'aantal' wilt stopzetten, maar niet die van 'teller', dan kunt u dat als volgt doen op de plek van waaruit (waarvanaf) dat nodig is:

Replace ==teller= by ==art-index=.

Vanaf deze plaats wordt dan alleen nog de naam 'teller' gewijzigd, dus niet meer de naam 'aantal'.

Als u het effect van een REPLACE BY opdracht wilt stoppen zonder een nieuwe REPLACE BY opdracht, dan doet u dat met de tweede variant, de REPLACE OFF opdracht, waarvan de formaatbeschrijving luidt:

REPLACE OFF .

Ook de REPLACE OFF opdracht moet worden afgesloten met een punt.

4 Inter-program communication

4.1 Inleiding

4.1.1 Communicatie tussen programma's

Deze module (IPC) maakt het mogelijk dat twee of meer COBOL programma's met elkaar kunnen communiceren.

Communicatie tussen twee programma's omvat twee aspecten:

- a) De uitvoering moet van het ene programma kunnen worden overgedragen naar het andere programma en weer terug. Dit heet het oproepen van een programma. Men zou dan de betrokken programma's respectievelijk het *oproepende* programma en het *opgeroepen* programma kunnen noemen. Wij noemen het oproepende programma '*hoofdprogramma*' en het opgeroepen programma '*subprogramma*'. Hierbij verstaan we onder een hoofdprogramma elk programma dat een ander programma oproept; een subprogramma is elk programma dat door een ander programma wordt opgeroepen. Als programma A programma B oproept en programma B roept programma C op, dan is dus programma A een hoofdprogramma, programma C een subprogramma en programma B zowel een subprogramma als een hoofdprogramma.
Subprogramma's zijn in alle opzichten een normaal COBOL programma waarin u alle faciliteiten kunt gebruiken die COBOL kent. U mag er dus bestanden benoemen en verwerken, met tabellen manipuleren, andere subprogramma's oproepen, ... Het enige 'speciale' aspect is het feit dat een subprogramma wordt opgeroepen vanuit een ander programma.
- b) Beide programma's moeten gegevens aan elkaar ter beschikking kunnen stellen. Daarbij kan het belangrijk zijn, dat het hoofdprogramma kan aangeven welke gegevens al dan niet door het subprogramma mogen worden gewijzigd.
Het is tevens mogelijk dat er gemeenschappelijke gegevens beschikbaar zijn, dat zijn gegevens die voor elk programma in de run unit of in een gedeelte van de run unit ter beschikking staan en niet expliciet aan elkaar te hoeven worden doorgegeven.

De naam van een subprogramma is de naam uit de PROGRAM-ID paragraaf van dat subprogramma. (in NetExpress is dit de naam van het programma zonder extensie)

Opgelet : hoofd- en kleine letters zijn niet bij elke compiler door mekaar te gebruiken.

4.1.2 Compilatie-varianten

De communicatie tussen twee programma's wordt uiteraard pas tijdens de uitvoering effectief. De daarvoor noodzakelijke instructies moeten echter al beschikbaar zijn tijdens het compileren van de desbetreffende programma's. Hoe meer een compiler weet over beide programma's tegelijkertijd, des te gerichter kan de controle zijn. COBOL kent hiervoor twee varianten:

- a) De eerste variant betreft de situatie dat de programma's apart worden vertaald. De compiler kent dan per vertaling slechts één programma en moet uit dat programma de essentiële kenmerken van het andere programma verkrijgen. Deze situatie stelt hoge eisen aan de programma's, terwijl anderzijds het aantal communicatiefaciliteiten gering is. Zie 4.2.
- b) De tweede variant betreft de situatie dat de programma's te samen worden gecompileerd. Daarbij kunnen de betrokken programma's elkaar omvatten net als bij een blokkendoos (blokstructuur). De compiler kent nu al tijdens de vertaling de programma's die met elkaar gaan communiceren. Er is dan een nauwgezette controle mogelijk, omdat alle relevante instructies beschikbaar zijn. Deze situatie stelt lagere eisen aan de programma's, terwijl tevens meer communicatiefaciliteiten ter beschikking staan. Zie 4.3.

Een combinatie van deze beide varianten is ook mogelijk.

4.1.3 De uitvoering van een run unit

De uitvoering van een run unit begint met de eerste opdracht van het programma dat via het O.S. als eerste programma van de run unit is aangewezen. De uitvoering gaat met dat programma verder totdat een ander programma wordt opgeroepen. De uitvoering daarvan gaat verder totdat ofwel de besturing wordt teruggeven aan het programma waardoor het was opgeroepen, ofwel weer een ander programma wordt opgeroepen, enz...

Als ergens in een run unit een STOP RUN opdracht wordt uitgevoerd dan is daardoor de gehele run unit beëindigd. Het maakt daarbij dus niet uit of die STOP RUN in een hoofdprogramma dan wel in een subprogramma wordt uitgevoerd.

4.2 *Apart-vertaalde programma's*

4.2.1 De overdracht van de besturing

4.2.1.1 Het oproepen van een apart vertaald subprogramma

Om de besturing over te dragen van een hoofdprogramma naar een subprogramma kunt u de CALL opdracht gebruiken, met volgende formaatbeschrijving:

Het USING gedeelte van de CALL opdracht: zie 4.2.2.

Door de uitvoering van een CALL opdracht wordt de besturing overgedragen aan het aangeduide subprogramma, waarbij de aangegeven velden ter beschikking worden gesteld van het subprogramma. De overdracht van de aangegeven velden wordt in 4.2.2 behandeld.

Voorbeeld:

Codering van hoofdprogramma:

```
IDENTIFICATION DIVISION.  
Program-ID.                Hoofd-programma.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01    veld-1                PIC 999 value 150.  
01    veld-2                PIC 999 value 600.  
PROCEDURE DIVISION.  
Hoofd.  
    Display veld-1 space veld-2  
    Call "Sub-programma" using veld-1 veld-2  
    Display veld-1 space veld-2  
    Stop run
```

Codering van subprogramma:

```
IDENTIFICATION DIVISION.  
Program-ID.                Sub-programma.  
DATA DIVISION.  
LINKAGE SECTION.  
01      veld-1             PIC 999.  
01      veld-2             PIC 999.  
PROCEDURE DIVISION using veld-1, veld-2.  
Hoofd.  
        Multiply veld-1 by 2 giving veld-2  
        Exit program
```

Door de uitvoering van de twee DISPLAY opdrachten zal worden afgedrukt.

```
150      600  
150      300
```

In de CALL opdracht moet u aangeven, wélk subprogramma uitgevoerd moet gaan worden. (De naam van een subprogramma staat vermeld in de PROGRAM-ID paragraaf ervan of is de naam van het subprogramma.)

U kunt op twee manieren in de CALL opdracht het gewenste subprogramma aangeven:

- a) In de vorm van een alfanumerieke constante. Vb. Call “Netto-berekening”
- b) In de vorm van een veldnaam. Vb.

```
01      Sub-naam          PIC X(16).  
  
        Move “netto-berekening” to sub-naam  
        Call sub-naam
```

In dit geval kan de compiler tijdens de vertaling niet bepalen, welk subprogramma tijdens de uitvoering zal worden opgeroepen. De inhoud van het veld ‘sub-naam’ kan immers wisselen. Men noemt dit wel een “dynamische oproep”.

Opgelet: vergeet aanhalingstekens niet bij gebruik van vaste naam (statische oproep).

De (NOT) ON EXCEPTION optie en de END-CALL

Tijdens de uitvoering van een CALL opdracht kan blijken, dat het opgeroepen subprogramma niet in de run unit aanwezig is. Hierop kunt u reageren door de ON EXCEPTION optie. Daarin kunt u overigens in de praktijk niet veel anders doen dan via een boodschap melden wat er aan de hand is en vervolgens stoppen met de uitvoering.

```
Call “netto-berekening”  
    on exception  
        display “subprogramma netto-berekening ontbreekt”  
        stop run  
end-call
```

Met de NOT ON EXCEPTION optie kunt u reageren op de situatie dat het subprogramma wel aanwezig blijkt te zijn. De in de NOT ON EXCEPTION genoemde opdrachten worden dan uitgevoerd als het subprogramma de besturing weer heeft teruggegeven aan het hoofdprogramma. U zult de NOT ON EXCEPTION optie dan ook alleen maar nodig hebben als u in de ON EXCEPTION situatie andere opdrachten wilt uitvoeren dan in de NOT ON EXCEPTION situatie *én* in beide gevallen verder wilt gaan na de CALL opdracht.

Na de uitvoering van de (NOT) ON EXCEPTION optie gaat de uitvoering verder met de eerste opdracht na de CALL opdracht.

Bij gebruik van NOT ON EXCEPTION of ON EXCEPTION moet de CALL met END-CALL afgesloten worden.

ON EXCEPTION mag vervangen worden door ON OVERFLOW (COBOL-74).

Opgepast:

- a) Een programma mag nooit zichzelf oproepen, ook niet via een ander subprogramma. In dat geval is het resultaat onvoorspelbaar.
- b) Als het opgeroepen programma niet aanwezig blijkt te zijn en de ON EXCEPTION optie is niet gebruikt, dan wordt het aan de leverancier overgelaten wat er dan dient te gebeuren.
- c) Een CALL opdracht mag ook worden uitgevoerd tijdens de uitvoering van een i-o-error routine.

4.2.1.2 De terugkeer naar het hoofdprogramma

Als een subprogramma gereed is met de uitvoering, moet de besturing worden teruggegeven aan het hoofdprogramma. Daarvoor is de EXIT PROGRAM opdracht bestemd.

Na de uitvoering van een EXIT PROGRAM opdracht gaat de uitvoering in het hoofdprogramma verder, met de eerste opdracht die volgt op de CALL opdracht waardoor het betrokken subprogramma was opgeroepen. De EXIT PROGRAM opdracht mag u overal coderen, mits het de laatste opdracht is in een reeks.

Als de EXIT PROGRAM opdracht wordt uitgevoerd terwijl het programma niet door een hoofdprogramma was opgeroepen, dan wordt het beschouwd als een CONTINUE opdracht.

4.2.2 Het communiceren van gegevens

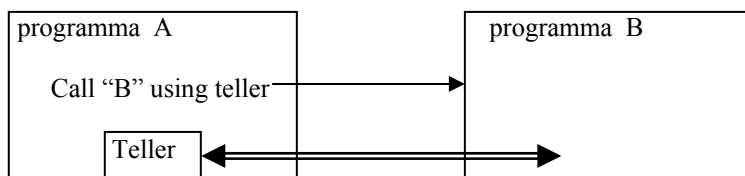
Vrijwel altijd zal het nodig zijn dat hoofdprogramma en subprogramma onderling gegevens uitwisselen. COBOL kent hiervoor drie mogelijkheden: parameters, externe velden en externe bestanden.

In de komende schema's ziet u twee soorten pijlen. Enkele lijn geeft de overdracht van besturing aan, de dubbele lijn de overdracht van gegevens.

- a) Parameters

Bij het gebruik van parameters geeft het hoofdprogramma in de CALL opdracht aan, welke velden beschikbaar zijn voor het subprogramma. Het subprogramma beschrijft die velden in een aparte section en legt via de Procedure Division header de relatie met de velden, die in de CALL opdracht van het hoofdprogramma zijn gespecificeerd. Het subprogramma kan dan over die velden beschikken.

Schema:

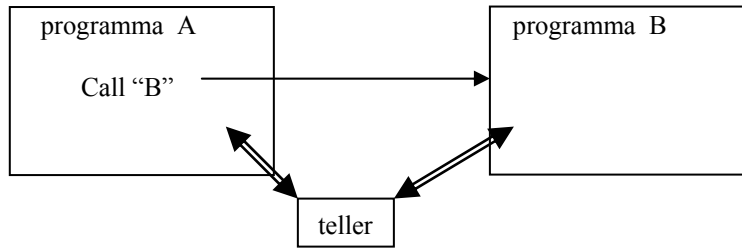


Programma A stelt het veld 'teller' ter beschikking aan programma B; het programma B kan de inhoud van dat veld raadplegen en eventueel wijzigen, afhankelijk van de wijze waarop programma A dat veld ter beschikking heeft gesteld.

- b) Externe velden

Bij het gebruik van externe velden benoemen het hoofdprogramma en het subprogramma elk op identieke wijze een of meer externe velden. Elk extern veld wordt dan een apart veld binnen de run unit, maar is geen onderdeel van het hoofdprogramma noch van het subprogramma.

Schema:

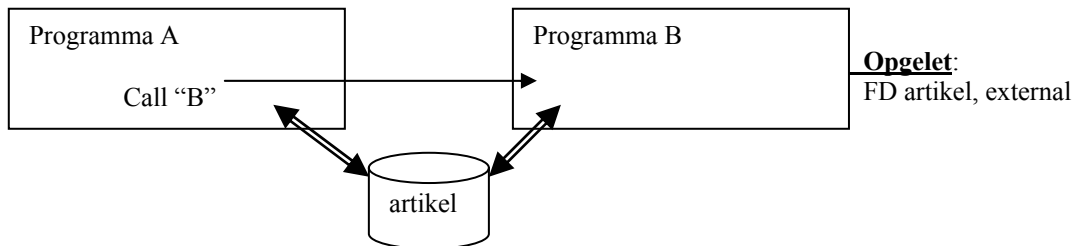


Zowel programma A als programma B kunnen beschikken over het externe veld 'teller'.

c) Externe bestanden

Bij het gebruik van externe bestanden benoemen het hoofdprogramma en het subprogramma elk op identieke wijze één of meer externe bestanden. Elk extern bestand wordt dan een apart bestand binnen de run unit, maar is geen onderdeel van het hoofdprogramma noch van het subprogramma.

Schema:



Zowel programma A als programma B kunnen beschikken over het externe bestand 'artikel'.

Combinaties van deze mogelijkheden zijn ook mogelijk

4.2.2.1 De communicatie via parameters

De communicatie via parameters betekent dat hoofdprogramma en subprogramma aan elkaar gegevens uitwisselen via velden die zich bevinden in het hoofdprogramma. Dit is voor COBOL programma's een vaste afspraak. Als het subprogramma gegevens ter beschikking wil stellen aan het hoofdprogramma – bij voorbeeld het resultaat van een berekening – dan gaat dat altijd via velden die zich in het hoofdprogramma bevinden.

Het hoofdprogramma moet dus melden om welke velden het gaat. De compiler moet dat weten om ervoor te kunnen zorgen dat de juiste adressen en instructies worden gegenereerd. Dit "melden" gebeurt door in de USING clause van de CALL opdracht de betrokken velden te noemen. Deze velden worden de *actuele* parameters genoemd.

Het subprogramma moet aan de compiler laten weten, hoe de relatie is met de velden, die uit het hoofdprogramma zullen worden doorgegeven tijdens de uitvoering. Dit gebeurt door ze te vermelden in de USING clause van de Procedure Division header; bovendien moeten ze worden beschreven in de LINKAGE SECTION. Die velden heten de *symbolische* parameters.

4.2.2.1.1 Parameters in het hoofdprogramma

Het hoofdprogramma moet in de USING clausule van de CALL opdracht vermelden, welke velden worden doorgegeven als het subprogramma wordt aangeroepen. **Alle velden** in het hoofdprogramma zijn daarvoor geschikt: **werkvelden, tellers, records, tabellen, tabelvelden, bestandsvelden, indexvelden.**

Voorbeeld:

```
01 teller.
01 tabel.
   03 element          PIC X(4) occurs 20 times.
.
      Move 12 to teller
      Call "B" using by reference    teller
                                   tabel
```

Programma B kan nu beschikken over de velden 'teller' en 'tabel'.

Als een veld zodanig gemeenschappelijk is dat het subprogramma de inhoud ervan mag wijzigen, dan vermeldt u de naam van dat veld in de BY REFERENCE optie van de CALL opdracht (zie hierboven).

Voorbeeld :

Een loonberekenningsprogramma verwerkt de records van een loonbestand. Per record wordt het subprogramma 'Netto-berekening' opgeroepen. In het hoofdprogramma kan de (gedeeltelijke) codering als volgt luiden:

```
.
FD loonbestand.
01 loon-record.
   03 werknemerkode          PIC X(4).
   03 bruto-loon             PIC 9(5)V99.
   03 nettoloan             PIC 9(5)V99.
   03 ...
.
WORKING-STORAGE SECTION.
01 belasting-tabel.
   03 belasting-regel          occurs 800 times
                               indexed by belasting-index.
       05 belastbaar-inkomen    PIC 9(5)V99.
       05 belasting-bedrag      PIC 9(5)V99.
.
      Call "netto-berekening" using by reference    loon-record
                                                    belasting-tabel
```

Door deze CALL opdracht krijgt het subprogramma de vrije beschikking over de beide groepsvelden 'loon-record' en 'belasting-tabel', inclusief alle velden binnen die groepsvelden. Dat betekent dat het subprogramma dus ook elk van die velden kan wijzigen.

Nu kan het zijn dat u dat risico niet wilt lopen. Voornamelijk in een groot rekencentrum waar diverse (groepen van) medewerkers ieder een deel van een totaal project opleveren, is het vaak beter als bij het coderen van gemeenschappelijke subprogramma's rekening wordt gehouden met velden die niet mogen worden gewijzigd. Dit kunt u bereiken door die velden te noemen in de BY CONTENT optie van de CALL opdracht. Daarmee geeft u aan, dat het subprogramma de inhoud van die velden wel mag raadplegen, maar niet kan wijzigen.

Stel dat in het voorgaande loonprogramma alleen het veld 'netto-loan' door het subprogramma mag worden gewijzigd. Het zou tenslotte wel eens rampzalig kunnen zijn, als ook een van de andere velden zou worden gewijzigd! De codering van de CALL opdracht kan dan als volgt luiden:

Call "Netto-berekening" using	by content	bruto-loon, Belasting-tabel
	By reference	nett-loon

Hierdoor wordt het voor het subprogramma onmogelijk om in een van de velden 'bruto-loon' of 'belasting-tabel' iets te wijzigen.

De vraag is nu: hoe kan worden voorkomen dat in het subprogramma een veld uit het hoofdprogramma ten onrechte wordt gewijzigd? Het subprogramma wordt immers apart vertaald van het hoofdprogramma en de compiler kan dus op dat moment niet weten of volgens het hoofdprogramma dat veld al dan niet mag worden gewijzigd.

Dit probleem is als volgt opgelost. Als een CALL opdracht wordt uitgevoerd zonder de BY CONTENT clause, dan worden aan het subprogramma de adressen van de betrokken velden doorgegeven. Als echter de CALL opdracht wél de BY CONTENT clause bevat, dan wordt eerst de inhoud van elk daarin genoemd veld gekopieerd naar een separaat veld. Vervolgens worden de adressen van die separate velden doorgegeven aan het subprogramma. Dat subprogramma kan dan wel de inhoud van die separate velden wijzigen, maar dat kan geen kwaad, omdat de oorspronkelijke velden in het hoofdprogramma onaangetaast blijven.

Opgelet:

Een actuele parameter moet een elementair veld zijn; een groepsveld is ook toegestaan, mits dat is benoemd met niveaunummer 01.

4.2.2.1.2 Parameters in het subprogramma

Het subprogramma moet in de USING clause van de Procedure Division header de symbolische parameters vermelden als relatie met de actuele parameters van het hoofdprogramma.

De volgorde van de parameters in de USING optie van de Procedure Division header *moet dezelfde zijn* als de volgorde van de parameters in de USING optie van de CALL opdracht waarmee het subprogramma wordt opgeroepen. De verhouding tussen die twee reeksen parameters is namelijk *positioneel* en NIET op naam. Dus de beide eerstgenoemde parameters in de USING clauses horen bij elkaar, dan de beide parameters die daarop volgen, enz...

De namen van de parameters in de beide programma's hoeven niet dezelfde te zijn: de programma's worden immers afzonderlijk vertaald. Hou de namen echter wel gelijk. Dit is immers veel duidelijker, vooral voor een ander die later uw programma's moet onderhouden!

Het voorbeeld van de loonberekening (zie hoger) wordt nu als volgt uitgebreid:

In het hoofdprogramma:

Call "Netto-berekening" using	by content	brutoloon
		belasting-tabel
	by reference	netto-loon

In het subprogramma:

```
IDENTIFICATION DIVISION.
Program-id.          Netto-berekening.
.
.
PROCEDURE DIVISION using bruto-loon
                        belasting-tabel
                        netto-loon.
```


Het hoofdprogramma geeft aan het subprogramma de adressen door van de kopievelden voor 'bruto-loon' en 'belasting-tabel', alsmede het adres van het veld 'netto-loon'. Deze adressen worden gebruikt bij elke verwijzing in het subprogramma naar een van deze velden.

De LINKAGE SECTION

De compiler moet van elke parameter ook de kenmerken kennen, zoals de lengte, decimale positie, enz... De symbolische parameters moeten dus "ergens" in het subprogramma worden beschreven en wel zodanig dat de compiler ze kan onderscheiden van de andere velden.

Dat gebeurt in een aparte section van de Data Division, namelijk in de LINKAGE SECTION.

Elke symbolische parameter moet dus in de LINKAGE SECTION worden benoemd.

Eerste voorbeeld: waarbij een subprogramma een bepaalde berekening uitvoert op twee velden uit het hoofdprogramma.

Hoofdprogramma:

IDENTIFICATION DIVISION.

Program-id. Hoofd.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 A PIC 999 value 200.

01 B PIC 999 value 325.

01 C PIC 9(6).

PROCEDURE DIVISION.

Verwerk.

 Call "Subprogramma" using A, B, C

 Display C

 Stop run.

Subprogramma:

IDENTIFICATION DIVISION.

Program-id. Subprogramma.

DATA DIVISION.

LINKAGE SECTION.

01 A PIC 999.

01 B PIC 999.

01 C PIC 9(6).

PROCEDURE DIVISION using A, B, C.

Verwerk.

 Compute C = A + 3 * B

 Exit program.

Als deze combinatie van programma's wordt uitgevoerd, dan wordt afgedrukt **001175**.

Tweede voorbeeld: van het eerder gegeven voorbeeld van de loonberekening wordt nu het subprogramma geheel gecodeerd.

IDENTIFICATION DIVISION.

Program-id. Netto-berekening.

DATA DIVISION.

LINKAGE SECTION.

```

01   bruto-loon           PIC 9(5)V99.
01   netto-loon          PIC 9(5)V99.
01   belasting-tabel.
      03   belasting-regel   occurs 800 times
                        indexed by belasting-index.
      05   belastbaar-inkomen PIC 9(5)V99.
      05   belasting-bedrag  PIC 9(5)V99.

```

PROCEDURE DIVISION using bruto-loon
belasting-tabel
netto-loon.

Verwerk.

```

Search belastingregel
  When bruto-loon = belastbaar-inkomen (belasting-index)
Or
  (bruto-loon > belastbaar-inkomen (belasting-index)
and
 (bruto-loon < belastbaar-inkomen (belasting-index + 1)))
  subtract belasting-bedrag (belasting-index) from bruto-loon giving netto-loon
end-search
exit program.

```

Bij uitvoering van dit subprogramma is het resultaat dat het veld 'netto-loon' in het hoofdprogramma wordt verminderd met de bij het bruto-loon behorende belasting-bedrag. Daarbij wordt in de belasting-tabel gezocht naar het belastbare bedrag dat gelijk is aan of ligt vlak onder het opgegeven bruto-loon.

In de veldbeschrijvingen in de LINKAGE SECTION mag geen beginwaarde gegeven worden (value). Zorg ervoor dat de kenmerken van een symbolische parameter gelijk zijn aan die van de actuele parameter. Vooral de lengte!! (usage moet gelijk zijn, sign separate moet bij beide voorkomen)

Opgelet!

- a) De LINKAGE SECTION moet in de DATA DIVISION worden gecodeerd na de WORKING-STORAGE SECTION.
- b) De symbolische parameters moeten in de LINKAGE SECTION benoemd worden met niveaunummer 01 of 77. Dit is dus een ander voorschrift dan bij de CALL opdracht: actuele parameters mogen immers elk elementair veld zijn, of een groepsveld op 01-niveau. U kunt met name als actuele parameter wel een specifiek tabelveld doorgeven, maar dat veld moet dan in het subprogramma op 01-niveau (of 77-niveau) benoemd worden. Een voorbeeld:

Hoofdprogramma:

```

01   tabel.
      03   element          PIC X(4) occurs 30 times.

```

Call "B" using by reference element(25)

Subprogramma:

LINKAGE SECTION.

```

01   element          PIC X(4).

```

PROCEDURE DIVISION using element.

- c) U mag in een CALL opdracht wel twee of meer keren dezelfde actuele parameter gebruiken. Doch in een Procedure Division header van een subprogramma mag een symbolische parameter slechts één keer voorkomen. Meer dan één keer zou ook geen zin hebben, want elke referentie ernaar zou altijd op dezelfde actuele parameter betrekking hebben.

Voorbeeld:

Hoofdprogramma:

```
01    h-veld-1          PIC X(5).
01    h-veld-2          PIC X(8).
.
      Call "B" using by reference h-veld-1, h-veld-1, h-veld-2
```

Subprogramma:

```
LINKAGE SECTION.
01    s-veld-1          PIC X(5).
01    s-veld-2          PIC X(5).
01    s-veld-3          PIC X(8).
PROCEDURE DIVISION using s-veld-1, s-veld-2, s-veld-3.
```

Elke verwijzing naar de symbolische parameters 's-veld-1' en 's-veld-2' heeft nu betrekking op dezelfde actuele parameter 'h-veld-1' in het hoofdprogramma.

Deze mogelijkheid lijkt weinig zin te hebben, maar hij kan wel eens benut worden als een subprogramma met "dummy" waarden moet worden opgeroepen, bij voorbeeld tijdens de testfase ervan.

- d) Als een tabel in een hoofdprogramma is benoemd met een index en die tabel wordt doorgegeven aan een subprogramma, dan wordt die index niet doorgegeven. Elke tabel heeft namelijk zijn 'eigen' index, zowel in het hoofdprogramma als in het subprogramma. Als u dus de stand van een index in het subprogramma nodig hebt, dan moet u die via een apart veld doorgeven. Het subprogramma dient dan de waarde daarvan weer over te brengen naar de tabel-index.

Voorbeeld :

Hoofdprogramma:

```
01    index-waarde      PIC 99.
01    tabel.
      03    element     PIC X(4)
                        occurs 60 times
                        indexed by tabel-index.
.
      set index-waarde to tabel-index
      call "B" using by reference tabel, index-waarde
```

Subprogramma:

```
LINKAGE SECTION.
01    index-waarde      PIC 99.
01    tabel.
      03    element     PIC X(4)
                        occurs 60 times
                        indexed by tabel-index.
```

PROCEDURE DIVISION using tabel, index-waarde.

Verwerk.

Set tabel-index to index-waarde

- e) U kunt vanuit een hoofdprogramma een parameter doorgeven aan een subprogramma, die hem dan vervolgens weer kan doorgeven aan een ander subprogramma. Dat doet u door in de CALL opdracht in het eerste subprogramma een parameter te noemen, die ook was genoemd in de Procedure Division header van dat subprogramma.

Voorbeeld :

Hoofdprogramma:

```
01      netto-loon          PIC 9(5)V99.
.
      Call "B" using by reference netto-loon
```

Subprogramma 1:

```
LINKAGE SECTION.
01      netto-loon          PIC 9(5)V99.
PROCEDURE DIVISION using netto-loon.
.
      Call "C" using by reference netto-loon
```

Subprogramma 2:

```
LINKAGE SECTION.
01      netto-loon          PIC 9(5)V99.
PROCEDURE DIVISION using netto-loon.
.
.
      Compute netto-loon = ...
```

Het subprogramma C beschikt op deze manier over het oorspronkelijke veld 'netto-loon' van het hoofdprogramma.

U kunt overigens bij dit soort combinaties van subprogramma's niet de BY CONTENT clausule van het oorspronkelijke hoofdprogramma "overrulen", wel de BY REFERENCE clausule.

CALL hoofdprog	CALL Subpr 1	Subpr 2 kan de velden in hoofd wijzigen
BY CONTENT	BY CONTENT	niet
BY CONTENT	BY REFERENCE	niet
BY REFERENCE	BY CONTENT	niet
BY REFERENCE	BY REFERENCE	wel

