Programmeren in COBOL

Sephiroth

14 maart 2008

Inhoudsopgave

I Bewerkingen **7**

1 Inleiding **9**
1.1 Indeling van een programma . . . 9
1.2 Hello World! (DISPLAY) . . . 10
1.3 Commentaar . . . 10
1.4 Velden(variabelen) . . . 11
1.4.1 Numerieke velden . . . 11
1.4.2 Alfanumerieke velden . . . 11
1.5 Een simpel programma . . . 12
1.6 ACCEPT . . . 12
1.7 MOVE . . . 13
1.8 Decimale getallen . . . 13
1.9 Negatieve getallen . . . 14
1.10 Oefeningen . . . 15
1.10.1 Opgaven . . . 15
1.10.2 Praktijk . . . 16

2 Berekeningen **19**
2.1 COMPUTE . . . 19
2.1.1 Wortel . . . 20
2.2 ADD . . . 20
2.3 SUBTRACT . . . 20
2.4 MULTIPLY . . . 21
2.5 DIVIDE . . . 21
2.6 Speciale gevallen . . . 22
2.7 Conclusie . . . 22

3 IF en lussen **23**
3.1 Condities . . . 23
3.1.1 Relatiecondities . . . 23
3.1.2 Tekencondities . . . 23
3.1.3 OR en AND . . . 23
3.2 IF-statement . . . 23
3.3 EVALUATE . . . 24
3.4 Lussen . . . 24
3.4.1 TIMES . . . 24
3.4.2 UNTIL . . . 24

4 Deelprogramma's **27**
4.1 Gebruik van deelprogramma's . . . 27
4.2 De simpele manier . . . 27
4.3 Deelprogramma met times . . . 28
4.4 Deelprogramma met UNTIL . . . 28

II Velden **31**

5 Groepsvelden en conditie met voorwaardenaam **33**
5.1 Groepsvelden . . . 33
5.1.1 FILLER . . . 34
5.2 Conditie met voorwaardenaam . . . 35
5.2.1 De simpele manier . . . 35
5.2.2 Gebruik van THRU of THROUGH . . . 36
5.2.3 Een boolean . . . 36

III Bestandsorganisatie **39**

6 Sequentiële bestanden **41**
6.1 Voorbeeld van een sequentieel bestand . . . 41
6.2 Sequentiële bestanden lezen . . . 41
6.3 Sequentiële bestanden schrijven . . . 43
6.4 Record achteraan toevoegen . . . 44
6.5 Record herschrijven . . . 45
6.6 Record verwijderen . . . 46

7 Relatieve bestanden zonder hash **47**
7.1 Relatief bestand aanmaken . . . 47
7.2 Relatief bestand lezen . . . 49
7.2.1 Relatief bestand sequentieel lezen . . . 49
7.2.2 Relatief bestand random lezen . . . 50
7.3 Record toevoegen . . . 52
7.4 Record herschrijven . . . 53
7.5 Record verwijderen . . . 53

8 Relatieve bestanden met hash **55**
8.1 Relatief bestand met hash aanmaken . . . 55
8.2 Relatief bestand met hash lezen . . . 59
8.3 Record herschrijven . . . 61
8.4 Record verwijderen . . . 62

IV Gebruikersomgevingen **65**

9 Character-based interfaces **67**
9.1 BLANK SCREEN . . . 67
9.2 LINE en COLUMN . . . 67
9.2.1 LINE . . . 68
9.2.2 COLUMN . . . 69

9.2.3 Een andere manier 69
9.3 Combineren met velden 69
9.3.1 FROM 69
9.3.2 USING 70
9.4 Werken met kleuren 70
9.4.1 BACKGROUND-COLOR 71
9.4.2 FOREGROUND-COLOR 71
9.4.3 REVERSE-VIDEO, HIGHLIGHT en BLINK 71

Deel I

Bewerkingen

Hoofdstuk 1

Inleiding

Dit is de inleiding van Programmeren in COBOL. COBOL is niet hoofdlettergevoelig dus een "a"betekent hetzelfde als een "A". Voor de overzichtelijkheid wordt aangeraden om bepaalde regels te volgen, sleutelwoorden(syntax) worden in hoofdletters geschreven en velden(variabelen) met kleine letters. Natuurlijk is de inhoud van een veld wel hoofdlettergevoelig.

1.1 Indeling van een programma

Een programma in COBOL wordt ingedeeld in 4 delen of divisies.

Indeling

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

Met de volgende betekenis:

IDENTIFICATION DIVISION geeft informatie over het programma. Bijvoorbeeld wat de naam is van het programma of wie het gemaakt heeft.

De *ENVIRONMENT DIVISION* is voornamelijk voor het gebruiken van bestanden door het programma.

In de *DATA DIVISION* komen al de verschillende soorten velden(variabelen) die het programma gebruikt.

In de *PROCEDURE DIVISION* staat de werkelijke programmacode.

Let op: de divisies moeten worden geschreven vanaf de 8ste positie. M.a.w. als je dit zou typen in een editor moet je voor je bijvoorbeeld *IDENTIFICATION DIVISION* typt, 7 spaties laten. Dit gebruik stamt nog uit de tijd toen COBOL op papier werd geschreven. Achter deze divisies moet een punt komen. Wat ook belangrijk is om te weten, is dat alleen *IDENTIFICATION DIVISION* en *PROCEDURE DIVISION* verplicht zijn in een programmacode. De andere twee moet je alleen schrijven als je het werkelijk gebruikt.

Positie

```
         1         2         3
1234567890123456789012345678901234567890
       IDENTIFICATION DIVISION.
```

1.2 Hello World! (DISPLAY)

Laten we ons eerste programma schrijven en het traditionele "Hello World!öp het scherm tonen.

Hello World!

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. Hello World.
       PROCEDURE DIVISION.
       hoofd.
           DISPLAY "Hello World!"
           STOP RUN
           .
```

Zoals eerder werd gezegd geeft *IDENTIFICATION DIVISION* informatie over het programma zelf. *PROGRAM-ID.* (let op de punt) is een onderdeel van deze divisie, die geeft aan hoe het programma heet. Na een naam te hebben gegeven, moet ook een punt komen.

Programma's kunnen worden ingedeeld in deelprogramma's (zie hoofdstuk deelprogramma's). *hoofd.* kun je in dit geval vergelijken met *main()* in bijvoorbeeld java. Dit geeft gewoon aan dat dit het belangrijkste stukje code is van het programma. Je kunt *hoofd.* echter vervangen door een andere naam.

De werkelijke programmacode wordt pas geschreven vanaf de 12de positie.

DISPLAY laat iets zien op het scherm. Wil je zoals in dit programma gewoon tekst tonen, dan moet je de tekst tussen aanhalingstekens plaatsen ("").

STOP RUN. (let op de punt) geeft aan dat je aan het einde van het programma zit.

1.3 Commentaar

Het zou natuurlijk handig zijn als je commentaar kon plaatsen in een programma. Daarmee kun je later zien wat bijvoorbeeld bepaalde delen van een programma doen.

Hello World!

```
         1         2         3
1234567890123456789012345678901234567890
      *Dit is commentaar
```

Om commentaar te vermelden, moet je gebruik maken van een asterisk (*) op de 7de positie (dus voor de asterisk moet je 6 spaties laten). De hele regel wordt dan gezien als commentaar.

1.4 Velden(variabelen)

Variabelen worden in *COBOL* velden genoemd en gedeclareerd in de *DATA DIVISION*. Meer bepaald bij *WORKING-STORAGE SECTION*.

1.4.1 Numerieke velden

Numerieke velden

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77   getal PIC 9 VALUE 1.
```

Dit is een voorbeeld van een numeriek veld. Een veld begint op de 8ste positie. Het begint met het getal 77. 77 is het zogenaamde niveaunummer. Nu uitleggen wat het is zou zinloos zijn, aangezien dit nog maar de basis is. Naast 77 zou je ook 01 kunnen gebruiken. Het niveaunummer wordt gevolgd door de naam die je wilt geven aan het veld. Vaak wordt de naam van het veld vanaf de 12de positie geschreven. Daarna wordt het gevolgd door *PIC*(of *PICTURE*) dat bepaalt welk soort veld het is. De "9"staat voor een numeriek veld. Het aantal negens bepaalt hoeveel cijfers het veld maximum mag bevatten, 10 keer een 9 schrijven betekent dus een numeriek veld van 10 tekens lang. Als je een waarde aan een veld wilt toevoegen, kun je *VALUE* gebruiken, maar dit is niet verplicht. Vergeet opnieuw de punt op het einde niet.

Numerieke velden

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77   getal PIC 9(10).
```

In plaats van bijvoorbeeld *"77 getal PIC 9999999999."*öm de lengte te bepalen, kan je evengoed gewoon *"PIC 9(10)"* schrijven om aan te geven dat het getal uit 10 cijfers zal bestaan. Beide hebben hetzelfde effect.

Als je bijvoorbeeld het getal 7 in een variabele zet met *"PIC 999"* dan krijg je op het scherm "007"te zien. Dus waar geen getal komt wordt 0 geplaatst. Later zullen we zien hoe je dit kunt wegwerken(zie hoofdstuk editing).

1.4.2 Alfanumerieke velden

Dit is natuurlijk te vergelijken met een String in de programmeertaal Java. Je kunt zo'n veld declareren met *"PIC X"*. De X geeft opnieuw het soort variabele en zijn maximale lengte aan.

Alfanumeriek velden

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77   tekst PIC X(10).
```

Ook hier kan *VALUE* worden gebruikt, alleen moet de inhoud nu tussen aanhalingstekens komen ("stringöf 'string').

Alfanumeriek velden

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77   tekst PIC X(13) value "dit is tekst!".
```

1.5 Een simpel programma

Getal

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. getal.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77   tekst PIC X(18) VALUE "Dit is een getal:".
       77   getal PIC 9(3) VALUE 7;

       PROCEDURE DIVISION.
       hoofd.
            DISPLAY tekst " " getal
            STOP RUN
            .
```

```
Dit is een getal: 007
```

Je kunt dus makkelijk 2 velden in dezelfde *DISPLAY* tonen. Dit zorgt er wel voor dat de twee velden naast elkaar komen te staan. Om een spatie tussen de twee velden te hebben, kun je gewoon doen. Wil je dat ze onder elkaar komen dan zet je ze gewoon in 2 *DISPLAY*Šs.

DISPLAY

```
       PROCEDURE DIVISION.
       hoofd.
            DISPLAY tekst
            DISPLAY getal
            STOP RUN.
```

```
Dit is een getal:
007
```

1.6 ACCEPT

Het zou natuurlijk leuk zijn moest de gebruiker input kunnen geven. Dit gebeurt via *ACCEPT*, na *ACCEPT* moet je het veld geven waarin de gebruiker iets voor moet ingeven.

ACCEPT

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. getal.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77   getal PIC 9(3).

       PROCEDURE DIVISION.
```

```
        hoofd.
            DISPLAY "Geef een getal:"
            ACCEPT getal
            DISPLAY "Dit is uw getal: " getal
            STOP RUN.
```

```
Geef een getal:
7
Dit is uw getal: 007
```

Het vreemde aan *ACCEPT* is dat wanneer de gebruiker iets ingegeven heeft, de computer een biep-geluid geeft. Om dit te vermijden kunt je *NO BEEP* gebruiken.

NO BEEP

```
            ACCEPT getal NO BEEP
```

1.7 MOVE

Met behulp van *MOVE* kun je makkelijk data overbrengen van het ene veld naar een andere. Maar je kunt ook gewoon een getal of tekst via *MOVE* naar een veld overbrengen.

MOVE

```
        PROCEDURE DIVISION.
        hoofd.
            ACCEPT input NO BEEP
            MOVE input TO output
            MOVE 6 TO getal
            MOVE "Dit is tekst" TO tekst
            STOP RUN
            .
```

1.8 Decimale getallen

Het stukje over numerieke velden ging alleen over gehele getallen en dus geen decimale. In COBOL moet je als je een decimaal getal wilt gebruiken dat ook zo declaren.

COBOL-code: Decimale getallen

Numerieke velden

```
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        77  getal PIC 99V9.

        PROCEDURE DIVISION.
        hoofd.
            DISPLAY "Geef een getal:"
            ACCEPT getal NO BEEP
            DISPLAY "Hier is uw getal: " getal
            STOP RUN
            .
```

De *V* na *PIC* geeft aan waar je de decimale punt wilt hebben. Als je een decimaal getal wilt ingeven, moet je een punt(.) gebruiken en geen komma(,). Dit is omdat COBOL ontwikkeld is in Amerika, waar een decimaal getal wordt aangegeven met een punt. Als je het getal echter toont op het scherm, komt er geen punt te staan. De *V* is alleen voor het inlezen van een getal en om er mee te rekenen. Wil je toch een punt hebben, dan moet je een extra variabele declareren (zie hoofdstuk editing).

Decimale getallen

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77   invoer   PIC 99V9.
       77   uitvoer  PIC 99.9.
       PROCEDURE DIVISION.
       hoofd.
            DISPLAY "Geef een getal:"
            ACCEPT invoer NO BEEP
            MOVE invoer TO uitvoer
            DISPLAY "Hier is uw getal:"
            DISPLAY uitvoer
            STOP RUN.
```

Het veld üitvoer"zal gebruikt worden om het op het scherm te laten zien. Het verschil is dat de V vervangen werd door een punt. Bij dit veld moet je de inhoud kopiëren vanuit het veld "getal"door middel van MOVE.

1.9 Negatieve getallen

Er geldt ook iets gelijkaardigs voor negatieve getallen, alleen is het hier erger. Als je negatieve getallen wilt gebruiken, moet je die ook eerst declareren.

Negatieve getallen

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77   getal PIC S999.

       PROCEDURE DIVISION.
       hoofd.
            DISPLAY "geef een getal"
            ACCEPT getal NO BEEP
            DISPLAY " "
            DISPLAY getal
            STOP RUN
            .
```

Als je dus een negatief getal wilt, moet je na PIC S999 zetten. S betekent sign (teken) en geeft aan dat het een negatief getal mag zijn. Dit geeft echter een probleem, zoals hieronder te zien is in de schermtekst als je het programma uitvoert.

```
Geef een getal
-888

88Q
```

88Q is dus niet het getal dat we hebben ingegeven. Hier kun je echter perfect mee rekenen, maar zoals bij een decimaal getal moet je er iets speciaal mee doen. Je hebt weer een extra veld nodig.

Negatieve getallen

```
77 uitvoer PIC -999.
```

De "-"geeft aan waar je het teken wilt (zie hoofdstuk Editing). Opgelet, dit betekent niet dat er altijd een - zal staan; bij een positief getal zal er geen teken staan. Gebruik je daarentegen PIC +999 dan staat er altijd een teken, hetzij +, hetzij -, afhankelijk van de inhoud. Gebruik opnieuw MOVE om de inhoud van getal naar uitvoer te kopiëren.

1.10 Oefeningen

1.10.1 Opgaven

Theorie

1. Geef de vier devisies.
2. Hoe toon je iets op het scherm?
3. Hoe schrijf je commentaar en op welke positie begin je te schrijven?
4. Waar declareer je een veld?
5. Hoe lees je iets in?
6. Hoe verplaats je inhoud van één veld naar een ander veld?

Praktijk

Opgave 1: Tekst op het scherm tonen Toon de tekst "Dit is een test voor opgave 1öp het scherm.

[bewerk] Opgave 2: Een getal inlezen en daarna tonen op het scherm De gebruiker wordt gevraagd een getal te geven. Dat getal kan maximaal 2 tekens lang zijn. Wanneer dat getal gegeven is komt de tekst "Dit is uw getal: öp het scherm met daarachter het getal.

[bewerk] Opgave 3: Tekst in lezen en tonen op het scherm Lees 2 verschillende namen in beide 20 tekens lang. Toon die vervolgens dan op het scherm naast elkaar met daartussen het woord ën".

[bewerk] Opgave 4: Negatieve en decimale getallen Lees een negatief en decimaal getal in en toon ze vervolgens op het scherm.

[bewerk] Oplossingen

[bewerk] Theorie

1. Geef de vier divisies. IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, DATA DIVISION en PROCEDURE DIVISION.
2. Hoe toon je iets op het scherm? Met DISPLAY gevolgd door een veld en/of tekst tussen aanhalingstekens().
3. Hoe schrijf je commentaar en op welke positie begin je te schrijven? Je typt eerst een asterisk(*) en je begint op de 7de positie
4. Waar declareer je een veld? Bij de WORKING-STORAGE SECTION in de DATA DIVISION.

5. Hoe lees je iets in? Met ACCEPT gevolgd door een veld. Als er een ACCEPT wordt gedaan maakt de computer een geluid om dit te vermijden kun je NO BEEP achteraan toevoegen.

6. Hoe verplaats je inhoud van één veld naar een ander veld? Via MOVE.

1.10.2 Praktijk

Opgave 1: Tekst op het scherm tonen

Opgave 1

```
IDENTIFICATION DIVISION.
PROGRAM-ID. Opgave1.
PROCEDURE DIVISION.
hoofd.
    DISPLAY "Dit is een test voor opgave 1"
    STOP RUN
    .
```

[bewerk] Opgave 2: Een getal inlezen en daarna tonen op het scherm COBOL-code: Opgave 2

Groepsvelden

```
IDENTIFICATION DIVISION.
PROGRAM-ID. Opgave2.

DATA DIVISION.
WORKING-STORAGE SECTION.
77  getal PIC 99.

PROCEDURE DIVISION.
hoofd.
    DISPLAY "Geef een getal:"
    ACCEPT getal NO BEEP
    DISPLAY "Dit is uw getal: " getal
    STOP RUN
    .
```

[bewerk] Opgave 3: Tekst in lezen en tonen op het scherm

Opgave 3

```
IDENTIFICATION DIVISION.
PROGRAM-ID. Opgave3.

DATA DIVISION.
WORKING-STORAGE SECTION.
77  naam1 PIC x(20).
77  naam2 PIC x(20).

PROCEDURE DIVISION.
hoofd.
    DISPLAY "Geef de eerste naam:"
    ACCEPT naam1 NO BEEP
    DISPLAY "Geef de tweede naam:"
    ACCEPT naam2 NO BEEP
    DISPLAY naam1 " en " naam2
    STOP RUN
    .
```

[bewerk] Opgave 4: Negatieve en decimale getallen

Opgave 4

```
IDENTIFICATION DIVISION.
```

```
PROGRAM-ID. Opgave4.

DATA DIVISION.
WORKING-STORAGE SECTION.
77   getal1in PIC S999.
77   getal2in PIC 99V9.

77   getal1uit PIC -999.
77   getal2uit PIC 99.9.

PROCEDURE DIVISION.
hoofd.
     DISPLAY "Geef een negatief getal:"
     ACCEPT getal1in NO BEEP
     DISPLAY "Geef een decimaal getal:"
     ACCEPT getal2in NO BEEP

     MOVE getal1in To getal1uit
     MOVE getal2in To getal2uit

     DISPLAY "Hier is uw negatief getal: " getal1uit
     DISPLAY "Hier is uw decimaal getal: " getal2uit
     STOP RUN
     .
```

Hoofdstuk 2

Berekeningen

Dit is een hoofdstuk waar je kunt leren hoe je moet iets moet berekenen in COBOL.

2.1 COMPUTE

COMPUTE is de makkelijkste manier waarmee je iets kunt berekenen in COBOL. De volgende delen van deze pagina zijn hierdoor eigenlijk onnodig, maar het kan best leuk zijn om te weten dat je het ook anders kunt doen. Rekenen wordt gedaan met +, -, *, /, **(machten). Om die tekens te gebruiken, moet je ze tussen 2 spaties zetten. Nooit bijvoorbeeld "1+1"maar altijd "1 + 1".

COMPUTE

```
COMPUTE uitkomst = getal + 2
```

De variabelen waar de uitkomst moet komen te staan aan de linkerkant van de "="de berekening staat altijd aan de rechterkant. "="lees je dus best als "wordt"(en niet ïs"). Het is mogelijk om af te ronden in COBOL dankzij *ROUNDED*: Een 1, 2, 3 en 4 worden naar beneden afgerond; 5, 6, 7, 8 en 9 worden naar boven afgerond: Stel dat de uitkomst 4.5 is, maar het komt terecht in een variabele met *"PIC 9"* die alleen 4 opslaat. Een wetenschappelijke/wiskundige afronding naar boven kan je uitvoeren met *ROUNDED*.

COMPUTE

```
COMPUTE uitkomst ROUNDED = getal / 2
```

Ook kun je de uitkomst in meerdere variabelen zetten en kun je de ene niet laten afronden en de andere dan weer wel.

COMPUTE

```
COMPUTE uitkomst uitkomst2 ROUNDED = 5 / (getal + 2 - getal2) * .5 ** 25
```

Wanneer er een complexe berekening wordt gedaan, hebben bepaalde tekens voorrang op andere, net zoals in de wiskunde. Haakjes hebben altijd voorrang. Daarna komt **, gevolgd door * en / (evenwaardig) en tenslotte + en -. Gelijk(waardig)e tekens worden van links naar rechts uitgevoerd.

2.1.1 Wortel

Hoewel er geen teken bestaat om een wortel te doen met *COMPUTE*, is er toch een simpele manier om het te doen. Er bestaat namelijk een alternatieve notatie voor een wortel.

$$\sqrt{a} = a^{\frac{1}{2}} \tag{2.1}$$

Links staat een gewone vierkantswortel en rechts staat 2 tot de 1/2 macht. Die zijn namelijk gelijk. De algemene notatie is:

$$\sqrt[n]{a} = a^{\frac{1}{n}} \tag{2.2}$$

Dit maakt dat je een wortel kan berekenen door het volgende te doen.

COMPUTE

```
COMPUTE uitkomst = a ** (1 / n)
```

2.2 ADD

Naast *COMPUTE* zijn er nog ander manieren om te rekenen. Het enige voordeel dat deze manier heeft is dat het zeer logisch kan overkomen.

ADD

```
ADD verkoop TO omzet
```

Hier wordt dan de inhoud van verkoop toegevoegd aan de inhoud van omzet. Het is mogelijk om met meerdere variabelen bij *ADD* te gebruiken.

ADD

```
ADD getal1 getal2 TO getal3 getal4 ROUNDED
```

Getal1 en getal2 worden zowel bij getal3 als bij getal4 opgeteld. Net zoals C*OMPUTE* kun je gebruik maken van *ROUNDED*. Het woord *GIVING* zorgt dat het resultaat in een andere variabele gezet wordt.

COBOL-code: ADD

Indeling

```
ADD prijs TO btw GIVING totaleprijs afgerondeprijs ROUNDED
```

2.3 SUBTRACT

SUBTRACT

```
SUBTRACT 1 FROM getal
```

SUBTRACT is verschilt niet erg van *ADD*: alleen wordt hier *FROM* gebruikt in de plaats van *TO*. Ook hier kan je *GIVING* en *ROUNDED* gebruiken.

SUBTRACT

```
SUBTRACT 1 getal1 FROM geta2 GIVING getal3 getal4 ROUNDED
```

Waar je wel voor moet opletten is de declaratie van de variabelen. Als je *SUBTRACT* doet, kun je een negatief getal krijgen. Daarom is het best dat je *S* gebruikt bij de variabelen, bijvoorbeeld *"PIC S999"*. (data editing).

2.4 MULTIPLY

MULTIPLY

```
MULTIPLY getal1 BY getal2 getal3 ROUNDED
```

Hier worden getal2 en getal3 vermenigvuldigd met getal1. Ook hier kun je *ROUNDED* en *GIVING* gebruiken.

MULTIPLY

```
MULTIPLY getal1 BY getal2 GIVING getal3 getal4 ROUNDED
```

2.5 DIVIDE

DIVIDE

```
DIVIDE getal1 INTO getal2 getal3 ROUNDED
```

Opgelet, hier worden getal2 en getal3 gedeeld door getal1 (niet omgekeerd). Hier kun je ook *ROUNDED* en *GIVING* gebruiken.

DIVIDE

```
DIVIDE getal1 INTO getal2 GIVING getal3 getal4 ROUNDED
```

Naast *"DIVIDE ... INTO"* kun je ook *"DIVIDE ... BY"* gebruiken.

Indeling

```
DIVIDE getal1 BY getal2 GIVING getal3 getal4 ROUNDED
```

Dit zorgt ervoor dat de berekening wordt omgedraaid. getal 1 zal worden gedeeld door getal2 in de plaats van getal2 door getal1. Je kunt bij *DIVIDE* ook de rest opslaan via *REMAINDER*.

DIVIDE

```
DIVIDE getal1 BY getal2 GIVING getal3 REMAINDER getal4
```

Hier zal de rest worden opgeslagen in getal4.

2.6 Speciale gevallen

Hoewel de notatie ingewikkeld is, kan met bovenstaande operaties bondiger programmacode geschreven worden. Onderstaande codelijnen zijn equivalent aan elkaar:

Speciale gevallen

```
ADD k l m TO a b
COMPUTE a b = a + k + l + m

SUBTRACT k l m FROM a b
COMPUTE a = a - ( k + l + m )
COMPUTE b = b - ( k + l + m )

MULTIPLY p BY q GIVING r s
COMPUTE r s = p * q

DIVIDE 15 BY k GIVING l REMAINDER m
COMPUTE l = 5 / k
COMPUTE m = 5 - k * l
```

Foutmeldingen kunnen opgevangen worden met *ON SIZE ERROR*. Als er de uitkomst niet in de *PIC* (*PICTURE*) past, kan een subroutine opgeroepen worden.

Speciale gevallen

```
COMPUTE a = b * c
   ON SIZE ERROR PERFORM subroutine
END-COMPUTE

MULTIPLY b BY c GIVING a
   ON SIZE ERROR PERFORM subroutine
END-COMPUTE
```

2.7 Conclusie

COMPUTE is de eenvoudigste manier om berekeningen uit te voeren. Het gaat veel sneller en je kunt snel complexe berekeningen uitvoeren zonder al teveel programmeerwerk. Wil je toch *ADD*, *SUBTRACT*, *MULTIPLY* en *DIVIDE* gebruiken dan is het aangeraden om goed te oefenen.

Hoofdstuk 3

IF en lussen

In dit hoofdstuk leer je over conditionele expressies (IF) en herhalingen (PERFORM n TIMES, PERFORM UNTIL ...).

3.1 Condities

3.1.1 Relatiecondities

Via een relatieconditie vergelijk je een veld, constante of een rekenkundige uitdrukking met een andere: Groter, kleiner of gelijk. Dit kan met de operatoren >, <, =, >= en <=.

3.1.2 Tekencondities

Dit zijn POSITIVE, NEGATIVE en ZERO. Hiermee vergelijk je een veld, constante of een rekenkundige uitdrukking om respectievelijk te zien of ze positief, negatief of nul zijn.

3.1.3 OR en AND

Hiermee kun je meerdere condities gebruiken. Bij OR moet één van de relatie condities waar zijn bij AND moeten beide waar zijn.

3.2 IF-statement

In programmeren kun je 2 of meer dingen met elkaar vergelijken en op basis daarvan iets al of niet uitvoeren. Dit kan met IF ... END-IF. In onderstaand programma kan de gebruiker een getal raden. Als het getal juist is, dan krijgt hij "goed gegoktöp het scherm.

COBOL-code: IF-statement

Groepsvelden

```
DISPLAY "geef een getal van 0 tot 9"
ACCEPT getal

IF getal = 5
  DISPLAY "goed gegokt"
END-IF
```

IF zal de inhoud van de variabele vergelijken met 5. Als het overeenkomt, zal wat er tussen IF en END-IF staat, uitgevoerd worden. Met ELSE kan je ook iets doen in het andere geval. (THEN hoeft niet geschreven te worden, maar komt de leesbaarheid wel ten goede.)

COBOL-code: IF-statement

Groepsvelden

```
IF getal = gezochtgetal
   THEN DISPLAY "goed gegokt"
   ELSE DISPLAY "slecht gegokt"
END-IF
```

THEN geeft aan wat er gebeurt als het goed is en ELSE als het fout is. Natuurlijk kun je ook 2 variabelen met elkaar vergelijken zoals hier de IF getal vergelijkt met gezochtgetal.

Een IF moet altijd worden afgesloten met END-IF, anders leest de compiler tot bij het volgende punt. Moderne COBOL-conventies raden het gebruik van END-IF aan.

3.3 EVALUATE

3.4 Lussen

Er zijn twee manieren om lussen te gebruiken: door middel van TIMES of UNTIL. TIMES laat een lus een aantal keren uitvoeren. Bij UNTIL moet er aan een bepaalde voorwaarde voldaan worden voordat de lus stopt. In beide gevallen moet er wel eerst PERFORM worden gebruikt.

3.4.1 TIMES

COBOL-code: Lus met times

Groepsvelden

```
PERFORM 60 TIMES
   DISPLAY "times"
END-PERFORM
```

Met dit programma wordt "times"60 keer getoond op het scherm. Er kan natuurlijk ook gebruik gemaakt worden van een variabele die bepaalt hoe vaak de lus moet worden uitgevoerd.

3.4.2 UNTIL

COBOL-code: Lus met UNTIL

Groepsvelden

```
DISPLAY "Hoe vaak moet de lus worden uitgevoerd"
ACCEPT getal NO BEEP

PERFORM UNTIL getal = 0
   COMPUTE getal = getal Ũ 1
   DISPLAY getal
END-PERFORM
```

Hoewel dit voorbeeld sterk overeenkomt met het vorige, is het toch fundamenteel verschillend: als je de - vervangt door een +, dan zal de lus nooit eindigen!(Mainframes stoppen gelukkig vanzelf na een seconde (dure) rekentijd.) Om aan deze lus een einde te brengen moet er voldaan worden aan een bepaalde voorwaarde hier is dat wanneer het getal 0 bevat. Er kunnen ook meerdere condities bij UNTIL worden gebruikt.

COBOL-code: Meerdere condities.

Groepsvelden

```
PERFORM UNTIL (getal1 = 0 OR getal2 > 0) AND getal 3 < 0
```

VARYING

Deze lus kan vergeleken worden met de for-lus in andere talen. Maar moet echter gecombineerd worden met UNTIL.

COBOL-code: Lus met VARYING

Groepsvelden

```
IDENTIFICATION DIVISION.
PROGRAM-ID. performvarying.

DATA DIVISION.
WORKING-STORAGE SECTION.
77  i pic 99.

PROCEDURE DIVISION.
hoofd.
    PERFORM VARYING i FROM 1 BY 1 UNTIL i > 20
      DISPLAY "I is: " i
    END-PERFORM
    STOP RUN.
```

Deze lus begint ook met PERFORM gevolgd door VARYING. i is de teller in deze lus en wordt gevolgd door FROM. FROM bepaalt de start positie van de teller i in de plaats van een cijfer kan er ook een veld staan. Dan komt BY en betekent hoeveel er wordt opgeteld of afgetrokken(je kan dus ook een negatief getal gebruiken) van de teller per keer dat de lus wordt uitgevoerd. Dan komt UNTIL die werkt net zoals bij de gewone UNTIL als er aan een bepaalde voorwaarde wordt voldaan eindigt de lus.

Hoofdstuk 4

Deelprogramma's

Dit is een hoofdstuk waar je kunt leren hoe je deelprogramma's kunt maken in COBOL.

4.1 Gebruik van deelprogramma's

Deelprogramma's zijn deeltjes code die apart worden gezet en gebruikt worden door het hoofdprogramma. Dit wordt gedaan zodat de programmacode overzichtelijker wordt en stukjes code opnieuw kan worden gebruikt. Een deelprogramma voert slechts één taak uit. Daardoor is het belangrijk om een goede naam te geven aan het deelprogramma dat zegt, liefst in één woord, wat het doet.

4.2 De simpele manier

Deelprogramma

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. naam.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 naam PIC X(20).

       PROCEDURE DIVISION.
       hoofd.
            DISPLAY "wat is uw naam?"
            ACCEPT naam
            PERFORM toon
            STOP RUN
            .

       toon.
            DISPLAY naam
            .
```

Dit is een simpel voorbeeld van een deelprogramma. Een deelprogramma wordt opgeroepen door middel van *PERFORM*, dus hetzelfde als bij een lus. Na *PERFORM* komt de naam van het deelprogramma in dit geval toon. Een deelprogramma wordt na de *STOP RUN.* van het hoofdprogramma geschreven. De naam van het deelprogramma wordt geschreven vanaf de 8ste positie, gevolgd door een punt. De code van het deelprogramma wordt, net zoals bij het hoofdprogramma, geschreven vanaf de 12de positie. Het is belangrijk om te

onthouden dat na de code van het deelprogramma een punt komt dat aangeeft dat er een einde is gekomen aan het deelprogramma. Na een deelprogramma kunnen eventueel nog andere worden geschreven. Ook kan er van één deelprogramma worden verwezen naar een ander.

4.3 Deelprogramma met times

Deelprogramma met TIMES

```
IDENTIFICATION DIVISION.
PROGRAM-ID. dlprogmettimes.

DATA DIVISION.
WORKING-STORAGE SECTION.
77 getal pic 9(3).

PROCEDURE DIVISION.
hoofd.
    PERFORM tel 80 TIMES
    STOP RUN.

tel.
    COMPUTE getal = getal + 1
    PERFORM toon
    .

toon.
    DISPLAY getal
    .
```

Zoals bij lussen, kan er ook gebruik gemaakt worden van *TIMES*. *PERFORM* wordt gevolgd door de naam van het deelprogramma (tel) en ... *TIMES*, het aantal keer dat het uitgevoerd moet worden. Dit is ook een voorbeeld van een verwijzing vanuit een deelprogramma naar een ander.

4.4 Deelprogramma met UNTIL

Deelprogramma met UNTIL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. dlprog.

DATA DIVISION.
WORKING-STORAGE SECTION.
77 getal PIC 9(3).

PROCEDURE DIVISION.
hoofd.
    PERFORM tel UNTIL getal = 80
    STOP RUN.

tel.
    COMPUTE getal = getal + 1
    PERFORM toon
    .

toon.
    DISPLAY getal
    .
```

Dit programma doet eigenlijk hetzelfde als het vorige maar hier wordt gebruik gemaakt van *UNTIL*. Dit is grotendeels hetzelfde. *PERFORM* wordt gevolgd door de naam van het deelprogramma en vervolgens *UNTIL* met de voorwaarde waaraan moet voldaan worden om te stoppen.

Deel II

Velden

Hoofdstuk 5

Groepsvelden en conditie met voorwaardenaam

In dit hoofdstuk leer je over groepsvelden en condities met voorwaardenaam. Dit gaat een beetje dieper in op velden.

5.1 Groepsvelden

Een groepsveld kan gebruikt worden om via één *ACCEPT* data te geven aan meerdere variabelen.

Groepsvelden

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 getal.
          02 a PIC 9.
          02 b PIC 99.

       PROCEDURE DIVISION.
       HOOFD.
            DISPLAY $geef een getal: $
            ACCEPT getal NO BEEP
            DISPLAY $-----T
            DISPLAY a
            DISPLAY b
            STOP RUN.
```

In dit programma wordt een getal gelezen en dat getal zal worden opgesplitst in 2 getallen. In tegenstelling tot een gewone variabele, waar level "77"wordt gebruikt bij het declareren, moet je voor een groepsveld "01"gebruiken: "01 getal.". Getal zal dan de naam zijn van het groepsveld. Bij de naam van een groepsveld wordt geen PIC gebruikt. Het groepsveld wordt onderverdeeld in andere variabelen. In dit geval is dat a en b. Bij deze variabelen moet wel een PIC worden gebruikt en dit is hetzelfde als bij andere variabelen. De lengte van het groepsveld getal hangt af van de som van de lengtes van variabelen die bij het groepsveld horen. Hier is dit dus 3 tekens lang (a heeft er 1 en b heeft er 2). Als je dus een getal leest met het groepsveld getal zal die worden opgesplitst.

```
geef een getal:
193
-----
```

```
1
93
```

Om de variabelen van een groepsveld apart op het scherm te tonen, hoef je dus alleen maar *DISPLAY* te gebruiken, gevolgd door de naam van de variabele. Het omgekeerde is ook waar. Stel, je vraagt eerst de voornaam van de gebruiker en daarna de achternaam en je wilt die heel makkelijk opnieuw op het scherm weergeven.

Groepsvelden

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 naam.
   02 voornaam PIC X(10).
   02 achternaam PIC X(10).

PROCEDURE DIVISION.
HOOFD.
     DISPLAY "geef uw voornaam:"
     ACCEPT voornaam NO BEEP
     DISPLAY "geef uw achternaam:"
     ACCEPT achternaam NO BEEP

     DISPLAY "-----"
     DISPLAY naam
     STOP RUN.
```

Heel simpel dus: je leest apart de variabelen voornaam en achternaam. Omdat die bij groepsveld naam horen kun je ze via *DISPLAY* beide in één keer tonen op het scherm. Maar wat al je voornaam exact 10 letters lang is? Dan wordt de voornaam en achternaam aan elkaar geschreven. Hiervoor bestaat de *FILLER*.

5.1.1 FILLER

Een *FILLER* kan voor twee zaken worden gebruikt. De eerste werd hierboven al aangehaald: Als je 2 variabelen in een groepsveld hebt en die gebruiken de totale voorziene lengte (bepaald door *PIC*). Met een extra variabele, de *FILLER*, kan je een spatie tussen die 2 variabelen zetten.

FILLER

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 naam.
   02 voornaam PIC X(10).
   02 PIC X VALUE " ".
   02 achternaam PIC X(10).

PROCEDURE DIVISION.
HOOFD.
     DISPLAY "geef uw voornaam:"
     ACCEPT voornaam
     ACCEPT achternaam
     DISPLAY "-----"
     DISPLAY naam
     STOP RUN.
```

Hier is tussen die variabelen voornaam en achternaam nog een derde geplaatst: 02 PIC X VALUE Omdat je de FILLER toch niet gebruikt, moet je hem niet benoemen. Sommige compilers geven wel een warning als je hem niet FILLER noemt (02 FILLER PIC X VALUE). Tussen achternaam en voornaam zal nu een spatie komen wanneer je het groepsveld naam toont. Je kunt natuurlijk naast een spatie ook alle andere tekens gebruiken -, +, *, /,... Maar let op als je een FILLER gebruikt: Lees dan nooit het hele groepsveld in, in dit geval naam, anders overschrijf je de FILLER. Dus nooit "ACCEPT naam".

De FILLER kun je ook op een andere manier gebruiken. Wanneer je gegevens inleest kan het zijn dat je uiteindelijk bepaalde informatie niet nodig hebt. Je leest bijvoorbeeld namen in en je weet dat de eerste 10 tekens de voornaam is, maar die wil je niet. Dan kun je ook een FILLER gebruiken.

FILLER

```
01 naam.
  02 PIC X(10).
  02 achternaam PIC X(10).
```

Als je dan naam inleest, zal de voornaam worden opgeslagen in de FILLER. Maar omdat je die toch niet nodig hebt, geef je de FILLER geen naam. De gegevens gaan toch niet verloren: Als je het groepsveld naam toont, krijg je nog steeds de voornaam te zien. Het is gewoon een simple truuk om minder (overbodige) variabelen te hebben bij grote programma's.

5.2 Conditie met voorwaardenaam

Conditie met voorwaardenaam is een manier om een bepaalde variabele te activeren als er voldaan is aan een bepaalde voorwaarde.

5.2.1 De simpele manier

Conditie met voorwaardenaam

```
IDENTIFICATION DIVISION.
PROGRAM-ID. conditiemetvoorwaarden.

DATA DIVISION.
WORKING-STORAGE SECTION.
77 getal PIC 9.
  88 even VALUE 0, 2, 4, 6, 8.
  88 oneven VALUE 1, 3, 5, 7, 9.

PROCEDURE DIVISION.
HOOFD.
    DISPLAY "geef een getal:"
    ACCEPT getal NO BEEP

    IF even
      THEN DISPLAY "getal is even."
    END-IF
    IF oneven
      THEN DISPLAY "getal is oneven."
    END-IF
    STOP RUN.
```

De variabele getal is onderverdeeld in 2 andere variabelen: even en oneven. Als het getal dat wordt opgeslagen in de variabele getal, even is, dan wordt de variabele even geactiveerd. En met oneven wordt de variabele oneven geactiveerd.

5.2.2 Gebruik van THRU of THROUGH

Als veel opvolgende cijfers als voorwaarde gelden, kun je THRU gebruiken.

Conditie met voorwaardenaam

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. conditiemetvoorwaarden.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 getal PIC 99.
          88 nietgeslaagd VALUE 0 THRU 49.
          88 geslaagd VALUE 50 THRU 99.

       PROCEDURE DIVISION.
       HOOFD.
            DISPLAY "Geef het resultaat:"
            ACCEPT getal NO BEEP

            IF nietgeslaagd
              THEN DISPLAY "U bent nietgeslaagd."
            END-IF

            IF geslaagd
              THEN DISPLAY "U bent geslaagd."
            END-IF
            STOP RUN.
```

Dit programma kijkt of een leerling al of niet geslaagd is. Wanneer het getal lager dan 50 is, zal de variabele niet geslaagd geactiveerd worden. Als het 50 of hoger is, zal de variabele geslaagd geactiveerd worden. Naast de Amerikaanse spelling THRU kan, je ook het Engelse THROUGH gebruiken.

5.2.3 Een boolean

Je kan dankzij conditie met voorwaardenaam dus ook een boolean maken. Hier is een simpel voorbeeld hoe het moet.

Boolean

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. conditiemetvoorwaarden.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 boolean PIC X.
          88 ok VALUE "j", "J".

       PROCEDURE DIVISION.
       HOOFD.
            DISPLAY "Is het OK?(j/n)"
            ACCEPT boolean NO BEEP

            IF ok
              THEN DISPLAY "Het is ok"
              ELSE DISPLAY "Het is niet ok"
            END-IF
            STOP RUN.
```

De gebruiker wordt gevraagd of het ok is. Antwoordt hij met een "j", dan wordt de variabele ok geactiveerd; wanneer hij met iets anders antwoordt gebeurt er niets.

Deel III

Bestandsorganisatie

Hoofdstuk 6

Sequentiële bestanden

Hier zul je leren hoe je sequentiële bestanden kunt lezen.

6.1 Voorbeeld van een sequentieel bestand

Een sequentieel bestand is een bestand waar iedere lijn een record vormt. Een record zal gegevens opslaan voor bijvoorbeeld één persoon. Een sequentieel bestand kan verschillende datatypes (of extensies) hebben. Wij zullen .txt gebruiken. Maar dat kan natuurlijk ook .dat of .in of iets anders zijn.

Sequentieel bestand

Peeters	Jan	Kerkplein 8	ZAVENTEM	M
Jansen	Peter	Stationstraat 124	KAMPENHOUT	M

In dit sequentieel bestand worden de naam, adres en geslacht van een persoon opgeslagen.

6.2 Sequentiële bestanden lezen

We gaan een klein programma schrijven dat de gegevens inleest en alleen de achternaam, voornaam en geslacht op het scherm toont.

Sequentieel bestand lezen

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. seqbestanlezen.

       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT invoer ASSIGN TO "invoer.txt"
           LINE SEQUENTIAL.

       DATA DIVISION.
       FILE SECTION.
       FD invoer.
       01 persoon.
        02 achternaam   PIC X(20).
        02 voornaam     PIC X(15).
        02 geslacht     PIC X.

       WORKING-STORAGE SECTION.
       77        PIC X  VALUE "n".
```

```
           88 eof          VALUE "j".

       PROCEDURE DIVISION.
       HOOFD.
            OPEN INPUT invoer
            READ invoer AT END SET eof TO TRUE END-READ
            PERFORM lees UNTIL eof
            CLOSE invoer
            STOP RUN.

       LEES.
            DISPLAY voornaam " " achternaam " " geslacht
            READ invoer AT END SET eof TO TRUE END-READ
            .
```

```
Jan             Peeters             M
Peter           Jansen              M
```

Nu gaan we dat een beetje uitleggen.

Sequentieel bestand lezen

```
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
            SELECT invoer ASSIGN TO "invoer.txt"
            LINE SEQUENTIAL.
```

Als er gewerkt wordt met bestanden wordt dit stukje toegevoegd. Het bestand invoer.txt wordt gekoppeld met invoer wat belangrijk is in de *DATA DIVISION*. *LINE SEQUENTIAL* geeft aan dat het een sequentieel bestand is.

Sequentieel bestand lezen

```
       DATA DIVISION.
       FILE SECTION.
       FD invoer.
       01 persoon.
        02 achternaam PIC X(20).
        02 voornaam PIC X(15).
        02 geslacht PIC X.
```

In de *DATA DIVISION* komt er de *FILE SECTION* bij. Er wordt gebruik gemaakt van groepsvelden om de gegevens in op te slaan. In dit geval is dat persoon.

Sequentieel bestand lezen

```
            OPEN INPUT invoer
            READ invoer AT END SET eof TO TRUE END-READ
```

In de programmacode moet het bestand eerst worden geopend. Dit doe je met *OPEN INPUT*, gevolgd door de naam van het bestand. Daarna doe je een READ. De boolean eof is nodig voor een lus. Als je op het einde van een bestand zit, wordt deze op *TRUE* gezet. Daardoor kan een lus weten wanneer je op het einde zit. Een *READ* leest slechts één record in één keer. Al de gegevens worden dan in het groepsveld persoon gezet. Wanneer je daarna nog een *READ* doet, gaat hij naar het volgende record en gaan de gegevens van het vorige verloren.

Sequentieel bestand lezen

```
            PERFORM lees UNTIL eof
```

Dit is een verwijzing naar een deelprogramma maar ook een lus. Dit zal het deelprogramma blijven uitvoeren tot het aan het einde is van het bestand. Hiervoor wordt de variabele eof gebruikt, als die op true gezet wordt, eindigt de lus.

Sequentieel bestand lezen

```
LEES.
    DISPLAY voornaam "␣" achternaam "␣" geslacht
    READ invoer AT END SET eof TO TRUE END-READ
    .
```

Dit is het deelprogramma. Eerst toont het op het scherm wat het de vorige keer uit het bestand heeft gelezen. Daarna leest het een nieuwe record.

Sequentieel bestand lezen

```
    CLOSE invoer
```

Nadat je gedaan hebt met het lezen van het bestand, moet je het bestand terug sluiten met *CLOSE*.

6.3 Sequentiële bestanden schrijven

Een bestand schrijven volgt natuurlijk het omgekeerde principe van een bestand lezen. Je verwerkt eerst de gegevens je plaatst die in het groepsveld van de uitvoer en vervolgens schrijf je het bestand. We zullen in dit voorbeeld een bestand openen en daarna lezen wat er in zit om dat te verwerken en uit te schrijven. Je moet natuurlijk niet een bestand lezen; je kan gewoon de gegevens inlezen via een *ACCEPT*, maar dat neemt tijd in beslag. We gebruiken in dit voorbeeld hetzelfde sequentieel bestand om te lezen als in het vorige.

Sequentieel bestand schrijven

```
IDENTIFICATION DIVISION.
PROGRAM-ID. seqbestanlezen.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT invoer ASSIGN TO "invoer.txt"
    LINE SEQUENTIAL.

    SELECT uitvoer ASSIGN to "uitvoer.txt"
    LINE SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD invoer.
01 persoon.
 02 achternaam PIC X(20).
 02 voornaam   PIC X(15).
 02 geslacht   PIC X.


FD uitvoer.
01 regel.
 02 uitvn PIC X(15).
 02       PIC X.
 02 uitan PIC X(20).
 02       PIC X.
 02 uitge PIC X.

WORKING-STORAGE SECTION.
```

```
77          PIC X  VALUE "n".
  88 eof           VALUE "j".

PROCEDURE DIVISION.
HOOFD.
     OPEN INPUT invoer
     OPEN OUTPUT uitvoer
     READ invoer AT END SET eof TO TRUE END-READ
     PERFORM lees UNTIL eof

     CLOSE invoer
     CLOSE uitvoer
     STOP RUN.

LEES.
     MOVE achternaam TO uitan
     MOVE voornaam TO uitvn
     MOVE geslacht TO uitge
     WRITE regel END-WRITE
     DISPLAY voornaam "␣" achternaam "␣" geslacht
     READ invoer AT END SET eof TO TRUE END-READ
     .
```

Een uitvoer-bestand wordt op dezelfde manier gekoppeld met een groepsveld en ook op dezelfde manier gesloten. Het bestand openen gebeurt op een andere manier *"OPEN OUTPUT uitvoer"*. In COBOL wordt namelijk een bestand gelezen, terwijl een record geschreven wordt.

Sequentieel bestand schrijven

```
     MOVE achternaam TO uitan
     MOVE voornaam TO uitvn
     MOVE geslacht TO uitge
     WRITE regel END-WRITE
```

Het enige waar er moet op gelet worden, is het overzetten van de variabelen van het éne groepsveld naar het andere en ook de *WRITE*-commando gevolgd door de groepsveld van de uitvoer en de *END-WRITE*. Hieronder bevindt zicht het resultaat van dit programma.

```
Jan             Peeters         M
Peter           Jansen          M
```

6.4 Record achteraan toevoegen

Telkens we *OPEN OUTPUT* doen en we schrijven daar iets in wordt er eigenlijk een nieuw bestand opgeslagen m.a.w. als er al een bestand bestond met dezelfde naam gaan de gegevens in dat bestand verloren en wordt het vervangen door andere gegevens. Daardoor kun je geen record toevoegen met *OPEN OUTPUT* hievoor moet je *OPEN EXTEND* gebruiken. *OPEN EXTEND* betekent dat je automatisch vanachter gegevens kunt toevoegen. In dit volgende programma gaat de gebruiker gegevens van een persoon opgeven en die worden achteraan toegevoegd.

Record achteraan toevoegen.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. RecordToevoegen.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
```

```
       SELECT uitvoer ASSIGN to "uitvoer.txt"
       LINE SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD uitvoer.
01 regel.
 02 uitvn PIC X(15).
 02       PIC X.
 02 uitan PIC X(20).
 02       PIC X.
 02 uitge PIC X.

WORKING-STORAGE SECTION.
77         PIC X  VALUE "n".
  88 eof          VALUE "j".

PROCEDURE DIVISION.
HOOFD.
     OPEN EXTEND uitvoer
     DISPLAY "Geef de voornaam:"
     ACCEPT uitvn NO BEEP
     DISPLAY "Geef de achternaam:"
     ACCEPT uitan NO BEEP
     DISPLAY "Geef het geslacht:"
     ACCEPT uitge NO BEEP
     WRITE regel END-WRITE
     CLOSE uitvoer
     STOP RUN.
```

Niet zo moeilijk dus gewoon in de plaats van *OPEN OUTPUT* moet je *OPEN EXTEND* gebruiken.

6.5 Record herschrijven

Het is mogelijk bij een sequentieel bestand een record te herschrijven. Dit is echter onpraktisch. Als je één bepaald record wilt aanpassen heb je twee keuzes ofwel ga je alle records één voor één af tot je de juiste record hebt gevonden ofwel schrijf je een programma die zelf alle records afgaat en vind wat je zoekt. Daardoor zijn sequentiële bestanden niet de goede oplossing als je weet dat je uw records vaak zult moeten aanpassen. Hiervoor zijn betere bestandsorganisaties die later nog aan bod komen. In het volgende programma krijgt de gebruiker ieder record te zien en wordt hem gevraagd of hij ze wilt herschrijven.

Record herschrijven.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. seqbestanlezen.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT bestand ASSIGN TO "uitvoer.txt"
     LINE SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD bestand.
01 regel.
 02 voornaam    PIC X(15).
 02             PIC X.
 02 achternaam  PIC X(20).
 02             PIC X.
 02 geslacht    PIC X.

WORKING-STORAGE SECTION.
```

```
77          PIC X  VALUE "n".
  88 eof           VALUE "j".
77 invoer PIC X.

PROCEDURE DIVISION.
HOOFD.
     OPEN I-O bestand
     READ bestand AT END SET eof TO TRUE END-READ
     PERFORM herschrijf UNTIL eof
     CLOSE bestand
     STOP RUN.

HERSCHRIJF.
     DISPLAY voornaam " " achternaam " " geslacht
     DISPLAY "Wilt u de gegevens van deze persoon aanpassen"
             "(0=neen)"
     ACCEPT invoer NO BEEP
     IF invoer = 0
       THEN DISPLAY "Geen gegevens aangepast"
       ELSE
         DISPLAY "Geef de voornaam:"
         ACCEPT voornaam NO BEEP
         DISPLAY "Geef de achternaam:"
         ACCEPT achternaam NO BEEP
         DISPLAY "Geef het geslacht:"
         ACCEPT geslacht NO BEEP
         REWRITE regel END-REWRITE
     END-IF
     READ bestand AT END SET eof TO TRUE END-READ
     .
```

Een record herschrijven op zich is niet zo moeilijk. Er zijn twee dingen waar je wel voor moet opletten. *I-O* of *INPUT-OUTPUT* dit betekent dat een bestand zal worden gezien als invoer en als uitvoer. Dit komt omdat je eers de records moet lezen voor je ze kan herschrijven. Je moet dus altijd *I-O* of *INPUT-OUTPUT* gebruiken als je iets wilt herschrijven. Voor je iets kunt herschrijven moet je eerst iets lezen met de READ en vervolgens herschrijf je het met de *REWRITE*. *REWRITE* werkt hetzelde als *WRITE*.

6.6 Record verwijderen

Je kunt bij sequentiële bestanden geen records verwijderen. Wat je echter wel kunt doen is de record herschrijven en opvullen met spaties. Je kunt dan een beveiliging inbouwen zodat wanneer de record wordt gelezen die bijvoorbeeld wordt overgeslagen. Je hoeft natuurlijk je record niet opvullen met spaties dit mag natuurlijk ook iets anders zijn of in ieder geval één of ander kenmerk zodat je programma weet dat deze record moet worden overgeslagen.

Hoofdstuk 7

Relatieve bestanden zonder hash

Bij sequentiële bestanden is het moeilijk om een bepaald record terug te vinden. De enige optie is om dan record per record te lezen tot je het vind. Dat is natuurlijk niet echt handig en efficiënt. Daarom bestaan er relatieve bestanden. Een record in een relatief bestand zal een nummer krijgen de zogenaamde relatieve sleutel van 1 tot n. Via een bepaalde berekening kan COBOL bepalen waar deze record zich in het bestand bevind. Een relatief bestand kan op 3 manieren worden gelezen. *SEQUENTIAL* zoals een sequentieel bestand. *RANDOM* wat betekent dat je een nummer ingeeft en de record wordt opgehaald met dat nummer. En *DYNAMIC* dit is een combinatie van de twee vorige.

7.1 Relatief bestand aanmaken

Een relatief bestand kun je niet in een tekstverwerker aanmaken. Je moet het via een programma aanmaken die je zelf hebt geschreven.

Relatief bestand aanmaken

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. relatief.

       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT persoon ASSIGN TO "persoon.dat"
               ORGANIZATION LINE SEQUENTIAL
               ACCESS SEQUENTIAL.

           SELECT persoonrel ASSIGN TO "persoonrel.dat"
               ORGANIZATION RELATIVE
               ACCESS SEQUENTIAL
               RELATIVE KEY persoonnummer.

       DATA DIVISION.
       FILE SECTION.
       FD persoon.
       01 gegevens PIC X(50).

       FD persoonrel.
       01 gegevensrel PIC X(50).

       WORKING-STORAGE SECTION.
       77                  PIC X.
        88 eof              VALUE "j", "J".
       01 persoonnummer PIC 99.

       PROCEDURE DIVISION.
```

```
HOOFD.
    OPEN INPUT persoon
    OPEN OUTPUT persoonrel

    DISPLAY "persoonnummer␣␣␣␣persoonnaam"
    READ persoon AT END SET eof TO TRUE END-READ

    PERFORM UNTIL eof
        MOVE gegevens TO gegevensrel
        WRITE gegevensrel END-WRITE
        DISPLAY persoonnummer "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" gegevensrel
        READ persoon AT END SET eof TO TRUE END-READ
    END-PERFORM
    STOP RUN.
```

Dit is dus een voorbeeld van een programma dat relatieve bestanden aanmaakt. We lezen hier een sequentieel bestand om vervolgens het relatief bestand aan te maken.

```
Jan         Peeters
Peter       Pieters
Piet        Jansen
```

Dit is het sequentieel bestand. Je kunt natuurlijk ook de gegevens inlezen via *ACCEPT* maar als uw programma onverwacht fouten bevat moet je de gegevens opnieuw ingeven. Via een sequentieel bestand te lezen vermijd je onnodig werk. Nu gaan we een beetje de code uitleggen.

Relatief bestand aanmaken

```
FILE-CONTROL.
    SELECT persoon ASSIGN TO "persoon.dat"
        ORGANIZATION LINE SEQUENTIAL
        ACCESS SEQUENTIAL.

    SELECT persoonrel ASSIGN TO "persoonrel.dat"
        ORGANIZATION RELATIVE
        ACCESS SEQUENTIAL
        RELATIVE KEY persoonnummer.
```

Bij *FILE-CONTROL* zijn veel dingen toegevoegd. *ORGANIZATION* geeft aan wat voor een bestand het is, *LINE SEQUENTIAL* voor sequentiële bestanden en *RELATIVE* voor relatieve bestanden. Het is niet verplicht om *ORGANIZATION* te schrijven maar aangezien we met twee verschillende soorten bestanden werken kan het duidelijker zijn. *ACCES* geeft aan hoe er met het bestand zal worden gewerkt. Sequentieel is dus record per record. Je zult je waarschijnlijk wel afvragen waarom we het relatieve bestand sequentieel gaan gebruiken en niet *RANDOM*. Dit is omdat het bestand alleen maar zal worden aangemaakt en zullen we dus record per record aanmaken en achter elkaar plaatsen. *RELATIVE KEY* zal de sleutel bepalen bij elke record, hier koppelen we deze sleutel met de variabele persoonnummer.

COBOL-code: Relatief bestand aanmaken

Indeling

```
DATA DIVISION.
FILE SECTION.
FD persoon.
01 gegevens      PIC X(50).

FD persoonrel.
01 gegevensrel PIC X(50).

WORKING-STORAGE SECTION.
```

```
77                PIC X.
 88 eof           VALUE "j", "J".
01 persoonnummer  PIC 99.
```

Er valt niet echt veel te zeggen over de *DATA DIVISION*. We gaan toch de gegevens niet verwerken dus is het indelen in groepsvelden niet nodig, je zou het wel kunnen doen. Waar je moet opletten is de lengte van de records. Die is hier 50 tekens lang als je dit relatief bestand wilt lezen zal je ook 50 tekens moeten gebruiken anders krijg je foutmeldingen. Waar je ook moet opletten is persoonnummer dat is de relatieve sleutel. Het aantal records dat je kunt opslaan hangt af van deze sleutel de *PIC* is hier 99 dus kun je maximaal 99 records hebben.

Relatief bestand aanmaken

```
        READ persoon AT END SET eof TO TRUE END-READ

        PERFORM UNTIL eof
            MOVE gegevens TO gegevensrel
            WRITE gegevensrel END-WRITE
            DISPLAY persoonnummer "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" gegevensrel
            READ persoon AT END SET eof TO TRUE END-READ
        END-PERFORM
```

Hier valt ook niet veel te vertellen. Het is grotendeels zoals bij sequentiële bestanden. Het sequentieel bestand wordt gelezen. De gegevens worden overgebracht en de record wordt geschreven in het relatief bestand.

7.2 Relatief bestand lezen

7.2.1 Relatief bestand sequentieel lezen

Relatief bestand lezen

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. relatief.

       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
            SELECT persoon ASSIGN TO "persoonr.dat"
                ORGANIZATION RELATIVE
                ACCESS SEQUENTIAL
                FILE STATUS bestandstatus.

       DATA DIVISION.
       FILE SECTION.
       FD persoon.
       01 gegevens PIC X(50).

       WORKING-STORAGE SECTION.
       77 bestandstatus PIC XX.
        88 eof          VALUE 10.

       PROCEDURE DIVISION.
       HOOFD.
            OPEN INPUT persoon
            READ persoon AT END CONTINUE END-READ

            PERFORM UNTIL eof
                DISPLAY gegevens
                READ persoon AT END CONTINUE END-READ
            END-PERFORM
            CLOSE persoon
            STOP RUN.
```

We gaan eerst aantonen dat je een relatief bestand ook gewoon sequentieel kunt lezen. Hoewel we een zogenaamde relatieve sleutel hebben zullen we die nog niet gebruiken.

Relatief bestand lezen

```
SELECT persoon ASSIGN TO "persoonr.dat"
    ORGANIZATION RELATIVE
    ACCESS SEQUENTIAL
    FILE STATUS bestandstatus.
```

Het enige interessante hier is *FILE STATUS* die geeft aan wat de status is van het bestand. Bijvoorbeeld of het open of gesloten is, of je iets uit het bestand wil lezen terwijl je het hebt geopend om er alleen in te schrijven of zoals we nu gaan zien of het aan het einde is van het bestand en zoverder.

Relatief bestand lezen

```
77 bestandstatus PIC xx.
 88 eof          VALUE 10.
```

FILE STATUS is gekoppeld aan de variabele bestandstatus. De variabele bestandstatus zal een nummer krijgen naar gelang de status van het bestand. Als het op het einde van het bestand is krijgt het nummer 10. Hierdoor wordt eof(end of file) geactiveerd. Dat is nodig voor de volgende lus

Relatief bestand lezen

```
PERFORM UNTIL eof
    DISPLAY gegevens
    READ persoon AT END CONTINUE END-READ
END-PERFORM
```

Hier is niet speciaals te zien behalve de *READ*. Als het op het einde van het bestand komt gaat het gewoon verder met de programmacode.

7.2.2 Relatief bestand random lezen

COBOL-code: Relatief bestand lezen

Indeling

```
IDENTIFICATION DIVISION.
PROGRAM-ID. relatief.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

    SELECT persoon ASSIGN TO "persoonr.dat"
        ORGANIZATION RELATIVE
        ACCESS RANDOM
        RELATIVE KEY persoonnummer.

DATA DIVISION.
FILE SECTION.
FD persoon.
01 gegevens PIC X(50).

WORKING-STORAGE SECTION.
77 persoonnummer PIC 99.
 88 stoppen      VALUE 0.
```

```
PROCEDURE DIVISION.
HOOFD.
    OPEN INPUT persoon
    DISPLAY "Geef nummer van de persoon(0 om te stoppen):"
    ACCEPT persoonnummer NO BEEP

    PERFORM UNTIL stoppen
        READ persoon
        DISPLAY "persoonnummer | gegevens"
        DISPLAY persoonnummer "                " gegevens
        DISPLAY " "
        DISPLAY "Geef nummer van de persoon(0 om te stoppen):"
        ACCEPT persoonnummer NO BEEP
    END-PERFORM
    STOP RUN.
```

Een relatief bestand *RANDOM* lezen is vrij makkelijk. Je vraagt aan de gebruiker om de persoonnummer in te geven en dan zoek je de gegevens van die persoon op.

Relatief bestand lezen

```
    SELECT persoon ASSIGN TO "persoonr.dat"
        ORGANIZATION RELATIVE
        ACCESS RANDOM
        RELATIVE KEY persoonnummer.
```

Dit keer gebruiken we dusaccess random we kunnen om het evenwelke record lezen. Opnieuw is de relatieve sleutel persoonnummer.

Relatief bestand lezen

```
DATA DIVISION.
FILE SECTION.
FD persoon.
01 gegevens PIC X(50).

WORKING-STORAGE SECTION.
77 persoonnummer PIC 99.
 88 stoppen      VALUE 0.
```

Zoals er in het vorige stukje werd gezegd moet je opletten met de lengte van de record. Als je de record hebt weggeschreven met een lengte van 50 tekens moet je die ook met die lengte weer lezen. Met de variabele persoonnummer gaan we de records lezen. Als persoonnummer 0 is wordt stoppen actief die komt van pas bij de lus dat we gaan gebruiken.

Relatief bestand lezen

```
    DISPLAY "Geef nummer van de persoon(0 om te stoppen):"
    ACCEPT persoonnummer NO BEEP

    PERFORM UNTIL stoppen
        READ persoon
        DISPLAY "persoonnummer | gegevens"
        DISPLAY persoonnummer "                " gegevens
        DISPLAY " "
        DISPLAY "Geef nummer van de persoon(0 om te stoppen):"
        ACCEPT persoonnummer NO BEEP
    END-PERFORM
```

Om een record op te halen is het enige wat je moet doen een nummer zetten in persoonnummer er moet wel een record zijn met dat nummer. Vervolgens doe je gewoon *READ* en de gegevens worden opgehaald. Is persoonnummer 0 dan stopt de lus.

Fouten

De kans is natuurlijk heel groot dat er fouten zullen gebeuren wanneer je een nummer vraagt. De gebruiker kan het fout in typen of het record bestaat gewoonweg niet. Om te vermijden dat je aan raar fout bericht krijgt of het programma plotseling beïndigd kun je in COBOL een klein beveiliging inbouwen. Dit kost weinig moeite en het is best dat je het erbij zet.

Foute sleutel

```
        PERFORM UNTIL stoppen
            READ persoonrel
              INVALID KEY DISPLAY "U heeft een fout nummer gegeven."
              DISPLAY " "

              NOT INVALID KEY
              DISPLAY "persoonnummer | gegevens"
              DISPLAY persoonnummer "               " gegevensrel
              DISPLAY " "
            END-READ
            DISPLAY "Geef nummer van de persoon (0 om te stoppen):"
            ACCEPT persoonnummer NO BEEP
        END-PERFORM
```

Dit gaat dus via *INVALID KEY* en *NOT INVALID KEY*. Achter *INVALID KEY* kun je dus een stukje programmacode schrijven voor het geval dat het nummer fout is. Achter *NOT INVALID KEY* kun je dus een stukje code schrijven voor als het wel juist is.

7.3 Record toevoegen

Hier zullen we zien hoe je makkelijk een record kunt toevoegen aan een relatief bestand.

Record toevoegen

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID. relatief.

        ENVIRONMENT DIVISION.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
            SELECT persoon ASSIGN TO "persoonr.dat"
                ORGANIZATION RELATIVE
                ACCESS SEQUENTIAL
                RELATIVE KEY persoonnummer.

        DATA DIVISION.
        FILE SECTION.
        FD persoon.
        01 gegevens PIC X(50).

        WORKING-STORAGE SECTION.
        77 persoonnummer PIC 99.

        PROCEDURE DIVISION.
        HOOFD.
            OPEN EXTEND persoon
            DISPLAY "Geef de gegevens van de persoon van de persoon:"
            ACCEPT gegevens NO BEEP

            WRITE gegevens
            DISPLAY "De nummer van de persoon is: " persoonnummer
            CLOSE persoon
            STOP RUN.
```

Een record toevoegen vraagt weinig programmeerwerk. *OPEN EXTEND* wilt zeggen dat je het bestand opent met de bedoeling om achteraan in het bestand iets toe te voegen. Dus je kunt alleen maar een record vanachter toevoegen niet meer niet minder. Je laat de gegevens inlezen en schrijft die in het bestand d.m.v. *WRITE*. Dan kun je nog als extra de relatieve sleutel laten zien. Dan kan de gebruiker later de gegevens nog terugvinden d.m.v. die sleutel.

7.4 Record herschrijven

Nu gaan we zien hoe je een record moet herschrijven.

Record herschrijven

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. relatief.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
            SELECT persoon ASSIGN TO "persoonr.dat"
                 ORGANIZATION RELATIVE
                 ACCESS RANDOM
                 RELATIVE KEY persoonnummer.

       DATA DIVISION.
       FILE SECTION.
       FD persoon.
       01 gegevens PIC X(50).

       WORKING-STORAGE SECTION.
       77 persoonnummer PIC 99.
       77 aanpassen     PIC X.
        88 ja           VALUE "J", "j".

       PROCEDURE DIVISION.
       HOOFD.
            OPEN I-O persoon
            DISPLAY "Geef nummer van de persoon die je wil aanpassen:"
            ACCEPT persoonnummer NO BEEP

            READ persoon
            DISPLAY "persoonnummer | gegevens"
            DISPLAY persoonnummer "               " gegevens
            DISPLAY " "
            DISPLAY "Wilt u de gegevens aanpassen(j/n)"
            ACCEPT aanpassen NO BEEP
            IF ja
              THEN DISPLAY "geef de nieuwe gegevens"
              ACCEPT gegevens
              REWRITE gegevens
            END-IF
            CLOSE persoon
            STOP RUN.
```

OPEN I-O is nieuw, *I-O* is afgekort van *INPUT-OUTPUT* en betekent dus dat je het bestand zowel kunt lezen als schrijven. In dit programma vragen we eerst de nummer van de persoon op. Vervolgens tonen we die gegevens op het scherm en vragen we of deze gegevens moet worden aangepast. Je plaatst de nieuwe gegevens in de variabel gegevens en gebruikt *REWRITE* om ze weg te schrijven. Bij *REWRITE* kun je ook gebruik maken van *INVALID KEY* en *NOT INVALID KEY*.

7.5 Record verwijderen

Het laatste dat we nog moeten zien is een record verwijderen.

Record verwijderen

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. relatief.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
            SELECT persoon ASSIGN TO "persoonr.dat"
                 ORGANIZATION RELATIVE
                 ACCESS RANDOM
                 RELATIVE KEY persoonnummer.

       DATA DIVISION.
       FILE SECTION.
       FD persoon.
       01 gegevens PIC X(50).

       WORKING-STORAGE SECTION.
       77 persoonnummer PIC 99.

       PROCEDURE DIVISION.
       HOOFD.
            OPEN I-O persoon
            DISPLAY "Geef nummer van de persoon die je wil verwijderen:"
            ACCEPT persoonnummer NO BEEP
            DELETE persoon
            STOP RUN.
```

Als je iets uit een relatief bestand wilt verwijderen moet je die open via *INPUT-OUTPUT* anders werkt het niet. Dit programma vraagt het de nummer en verwijderd dan de record met dat nummer via *DELETE*. Bij *DELETE* kun je ook *INVALID KEY* en *NOT INVALID KEY* gebruiken.

Hoofdstuk 8

Relatieve bestanden met hash

Relatieve bestanden kunnen via een sleutel makkelijk een record terugvinden. Dit kan via een cijfer (van 1 tot n). Maar een cijfer is niet altijd handig, en vaak weinig informatief. Een naam of een beschrijvend woord zouden makkelijker zijn. COBOL laat het gebruik van namen toe via een omweg: een hash-functie zet de naam om in een cijfer waarmee je makkelijk het record kunt terugvinden. Het grote nadeel van een relatief bestand met hash is dat je op voorhand moet weten hoeveel records je maximaal zult hebben. Als je eerst 10 records wou hebben en je maakt dat bestand aan moet je wanneer je plotseling 20 records wilt opslaan een nieuw relatief bestand aanmaken.

8.1 Relatief bestand met hash aanmaken

Zoals een gewoon relatief bestand moeten we vertrekken vanuit een sequentieel bestand. We lezen dus eerst een sequentieel bestand in, en maken daarvan een relatief bestand met hash.

Sequentieel bestand

```
Wikipedia    De vrije encyclopedie
Wiktionary   Vrij woordenboek met definities en uitleg
Wikimedia    Website over de projecten van Wikimedia
Wikiquote    Verzameling citaten
Wikibooks    Handleidingen en vrije boeken
Wikinews     Vrije nieuwsbron
Wikisource   Documenten vanuit publiek domein
Wikispecies  Catalogus van alle soorten
Commons      Vrije mediabestanden
```

Dit bestand gaan we omzetten in een relatief bestand. Het bevat de namen van wiki's met hun omschrijvingen. De namen van de wiki's zullen we gebruiken als sleutel. In het relatief bestand zal zowel de naam als de omschrijving van de wiki staan. Hieronder is een voorbeeld van een programma die dit voor ons zal doen.

Het programma dat het relatief bestand zal aanmaken.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. Wiki.

       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT wikiseq ASSIGN TO "wikiseq.txt"
```

```
        ORGANIZATION LINE SEQUENTIAL
        FILE STATUS seqstatus.

      SELECT wikirel ASSIGN TO "wikirel.txt"
        ORGANIZATION RELATIVE
        ACCESS RANDOM
        RELATIVE KEY nummer
        FILE STATUS relstatus.

DATA DIVISION.
FILE SECTION.
FD wikiseq.
01 seqwiki.
 02 seqnaam PIC X(11).
 02 PIC X.
 02 seqomschr PIC X(50).

FD wikirel.
01 relwiki.
 02 relnaam PIC X(11).
 02 relomschr PIC X(50).

WORKING-STORAGE SECTION.
01 seqstatus PIC XX.
 88 eofseq VALUE '10'.
01 relstatus PIC XX.
 88 eofrel VALUE '10'.
 88 slechtesleutel VALUE '23'.

01 nummer PIC 99.
01 wikinaam PIC X(11).
01 wikinummer PIC 9(11).
01 kleineletters PIC X(27) VALUE 'abcdefghijklmnopqrstuvwxyz '.
01 hoofdletters PIC X(27) VALUE 'ABCDEFGHIJKLMNOPQRSTUVWXYZ '.
01 cijfers PIC X(27) VALUE '012345678901234567890123456'.
01 teller PIC 99.

PROCEDURE DIVISION.
HOOFD.
     OPEN OUTPUT wikirel
     CLOSE wikirel
     OPEN INPUT wikiseq I-O wikirel
     READ wikiseq NEXT AT END CONTINUE END-READ
     PERFORM omzetten UNTIL eofseq
     CLOSE wikiseq wikirel
     STOP RUN.

OMZETTEN.
     PERFORM zoeken
     IF slechtesleutel
       THEN
            MOVE seqnaam TO relnaam
            MOVE seqomschr TO relomschr
            WRITE relwiki
     END-IF
     READ wikiseq NEXT AT END CONTINUE END-READ.

ZOEKEN.
     PERFORM hash
     READ wikirel INVALID KEY CONTINUE END-READ
     PERFORM VARYING teller FROM 1 BY 1 UNTIL slechtesleutel
     OR seqnaam = relnaam OR teller > 9
       PERFORM botsing
       READ wikirel INVALID KEY CONTINUE END-READ
     END-PERFORM.

HASH.
     MOVE seqnaam TO wikinaam
     INSPECT wikinaam CONVERTING kleineletters to hoofdletters
     INSPECT wikinaam CONVERTING hoofdletters TO cijfers
     DIVIDE wikinummer BY 9 GIVING wikinummer REMAINDER nummer
     ADD 1 TO nummer.

BOTSING.
```

```
ADD 1 TO nummer.
IF nummer = 10
  MOVE 1 TO nummer
END-IF.
```

Een relatief bestand aanmaken met hash kan zeer moeilijk zijn om te doen en te begrijpen. Nu gaan we stap voor stap voor stap uitleggen wat dit programma doet.

COBOL-code: De bestanden benaderen.

Sequentieel bestand

```
SELECT wikiseq ASSIGN TO "wikiseq.txt"
  ORGANIZATION LINE SEQUENTIAL
  FILE STATUS seqstatus.

SELECT wikirel ASSIGN TO "wikirel.txt"
  ORGANIZATION RELATIVE
  ACCESS RANDOM
  RELATIVE KEY nummer
  FILE STATUS relstatus.
```

Als u de twee vorige hoofdstukken hebt gelezen dan zal dit niet moeilijk zijn om te begrijpen. wikiseq is het bestand dat hier helemaal bovenaan werd beschreven en we zullen gebruiken om een relatieve bestand met hashfunctie aan te maken. wikirel zal natuurlijk dat bestand zijn, bij dit bestand is er wel iets waar je voor moet opletten. In het vorige hoofdstuk Relatieve bestanden zonder hash gebruikten we *ACCESS SEQUENTIAL* maar door de hash-functie moeten we dit bestand met een *RANDOM* benaderen, hier gaan we later nog op in.

hoofd.

```
hoofd.
    OPEN OUTPUT wikirel
    CLOSE wikirel
    OPEN INPUT wikiseq I-O wikirel
    READ wikiseq NEXT AT END CONTINUE END-READ
    PERFORM omzetten UNTIL eofseq
```

In het begin moeten we natuurlijk de bestanden openen, het relatief bestand wikirel moet als *I-O*(*INPUT-OUTPUT*) worden gelezen want er wordt in dit bestand zowel geschreven als gelezen. Waarom wordt ook later uitgelegd. Maar we moeten echter eerst een keer die relatief bestand openen als *OUTPUT*, dit zorgt ervoor dat het bestand wordt aangemaakt als het nog niet bestaat en voorkomt dat er fouten zijn als we het vervolgens openen met I-O. We lezen vervolgens het eerste lijntje in van het sequentiële bestand. Om vervolgens naar het deelprogramma omzetten te gaan als het op het einde van het sequentiële bestand is zal het deelprogramma stoppen.

Als we in omzetten zijn aangekomen gaan we meteen naar het deelprogramma zoeken en van daaruit meteen naar de hash-functie.

COBOL-code: Hash-functie

Sequentieel bestand

```
HASH.
    MOVE seqnaam TO wikinaam
    INSPECT wikinaam CONVERTING kleineletters to hoofdletters
    INSPECT wikinaam CONVERTING hoofdletters TO cijfers
```

```
MOVE wikinaam TO wikinummer
DIVIDE wikinummer BY 9 GIVING wikinummer REMAINDER nummer
ADD 1 TO nummer.
```

Dit is waar alles om draait, dit klein stukje code zorgt ervoor dat je via een woord of een naam een record zult kunnen plaatsen of vinden. We slaan eerst de naam van de wiki op in een hulpveld. Met dit hulpveld zullen we de hash-functie gebruiken. Eerst zorgen we ervoor dat alles in hoofdletters staat via de *INSPECT*(zie Werken met tekst). Als dat gebeurt is moeten we alles omzetten in een cijfer dit doen we ook via *INSPECT*. Wikinaam is nogsteeds een alfanumeriek veld en hiermee kan je niet rekenen dit moeten we eerst in een numeriek veld overzetten. Het getal dat we nu hebben is 11 tekens lang goed voor 100 miljard mogelijke combinaties. Dit is natuurlijk teveel van het goede we hebben maar 9 records dat we willen opslaan. De oplossing is het getal delen door het aantal records dat we willen opslaan. De uitkomst ervan interesseert ons niet want dat is nogsteeds meer dan 10 miljard. Wat ons wel interesseert is de restwaarde dat zal een getal zijn van 0 tot 8. Dat zijn 9 verschillende combinaties. Maar aangezien COBOL niet zero-bases is in vergelijking met bijvoorbeeld Java moeten we er één bij optellen. Dit geeft ons dan een getal van 1 tot 9 en slaan we op in het veld dat dient als *RELATIVE KEY* in dit geval nummer. Vervolgens gaan we terug naar het deelprogramma zoeken.

Opmerking: Hoe je precies aan het eindgetal komt maakt eigenlijk niet uit zolang je maar aan een getal komt die gebaseerd is op de naam of het woord en die genoeg kans loopt om te verschillen met de getallen van andere namen of worden.

Zoeken

```
ZOEKEN.
    PERFORM hash
    READ wikirel INVALID KEY CONTINUE END-READ
    PERFORM VARYING teller FROM 1 BY 1 UNTIL slechtesleutel
    OR seqnaam = relnaam OR teller > 9
      PERFORM botsing
      READ wikirel INVALID KEY CONTINUE END-READ
    END-PERFORM.
```

De hash functie is juist gebeurt. We hebben dus een sleutel en we lezen vervolgens het relatief bestand. Dit is meteen de reden waarom we het geopend hebben als *I-O* we moeten het bestand ook lezen omdat we moeten zien of er al geen record op die plaats is. Als er geen record is op die plaats dan krijgen we een *INVALID KEY* hier slechtesleutel genaamd. *INVALID KEY* betekent hier ironisch genoeg dat het goed is want op de plaats van onze sleutel is er geen record dus kunnen we daar een record wegschrijven. Maar wat als er wel een record is? Hiervoor kunnen er 3 oorzaken zijn ofwel heeft toevallig een ander record dezelfde sleutel gekregen van de hashfunctie of hebben we het record al een keer weggeschreven of we hebben al 9 records weggeschreven. Aan de twee laatste kunnen we niet veel doen als het record al weggeschreven is maakt het niet veel uit en als we al 9 records hebben weggeschreven moeten we het programma aanpassen zodat we toch meer dan 9 records kunnen gebruiken. Als er toevallig een andere record hetzelfde sleutel heeft gekregen noemen we dat een botsing of collision. Dit is niet erg en het gebeurd vrijwel altijd. Het simpelste wat we kunnen doen om het op te lossen is gewoon één bij tellen bij de sleutel en het opnieuw te proberen, dit gebeurt in het deelprogramma botsing. Het is bijna gedaan want nu gaan we al terug naar het deelprogramma omzetten.

Omzetten

```
OMZETTEN.
     PERFORM zoeken
     IF slechtesleutel
        THEN
             MOVE seqnaam TO relnaam
             MOVE seqomschr TO relomschr
             WRITE relwiki
     END-IF
     READ wikiseq NEXT AT END CONTINUE END-READ.
```

Onthoud *INVALID KEY* betekende in dit geval goed en daardoor is het veld slechtesleutel geactiveerd. Vervolgens gebeurd net hetzelfde als bij een gewoon relatief bestand. Schrijf alles weg en lees het volgende record van het sequentiële bestand. Als alle records zijn gelezen sluit de twee bestand en het programma heeft zijn werk gedaan.

8.2 Relatief bestand met hash lezen

Als je een relatief bestand met hash aanmaken onder de knie hebt moet dit niet zo moeilijk meer zijn. Jammer genoeg als je zeker wilt weten of je zo'n bestand perfect hebt kunnen maken moet je eerst ze ook kunnen lezen. Dit volgend programma leest dat bestand van het vorige stukje in. De gebruiker geeft de naam van de wiki en het programma geeft zijn omschrijving.

Een relatief bestand met hash lezen

```
IDENTIFICATION DIVISION.
PROGRAM-ID. Wiki.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

     SELECT wikirel ASSIGN TO "wikirel.txt"
       ORGANIZATION RELATIVE
       ACCESS RANDOM
       RELATIVE KEY nummer.

DATA DIVISION.
FILE SECTION.
FD wikirel.
01 relwiki.
 02 relnaam PIC X(11).
 02 relomschr PIC X(50).

WORKING-STORAGE SECTION.
01 nummer PIC 99.
01 wikinaam PIC X(11).
01 hulp PIC x(11).
01 wikinummer PIC 9(11).
01 kleineletters PIC X(27) VALUE 'abcdefghijklmnopqrstuvwxyz '.
01 hoofdletters PIC X(27) VALUE 'ABCDEFGHIJKLMNOPQRSTUVWXYZ '.
01 cijfers PIC X(27) VALUE '012345678901234567890123456'.
01 teller PIC 99.

PROCEDURE DIVISION.
HOOFD.
     OPEN INPUT wikirel
     ACCEPT wikinaam NO BEEP
     PERFORM zoeken
     CLOSE wikirel
     STOP RUN.

ZOEKEN.
     PERFORM hash
```

```
           PERFORM varying teller FROM 1 BY 1 UNTIL teller > 9
             READ wikirel END-READ
             IF relnaam = wikinaam
                THEN
                  DISPLAY relomschr
                  MOVE 10 TO teller
                ELSE
                  PERFORM botsing
             END-IF
           END-PERFORM.

       HASH.
           MOVE wikinaam TO hulp
           INSPECT hulp CONVERTING kleineletters to hoofdletters
           INSPECT hulp CONVERTING hoofdletters TO cijfers
           MOVE hulp TO wikinummer
           DIVIDE wikinummer BY 9 GIVING wikinummer REMAINDER nummer
           ADD 1 TO nummer.

       BOTSING.
           ADD 1 TO nummer.
           IF nummer = 10
             MOVE 1 TO nummer
           END-IF.
```

De hash-functie en botsingzijn in dit programma nogsteeds hetzelfde, dus het enige waar je moet voor opletten is de hoofd en zoeken.

hoofd

```
       hoofd.
           OPEN INPUT wikirel
           ACCEPT wikinaam NO BEEP
           PERFORM zoeken
           CLOSE wikirel
           STOP RUN.
```

De hoofd is niet echt moeilijk. Je moet het relatief gewoon open als input en vervolgens lees je de naam van de wiki in waarvan je de omschrijving wilt hebben. Vervolgens voer je zoeken uit.

Zoeken

```
       zoeken.
           PERFORM hash
           PERFORM varying teller FROM 1 BY 1 UNTIL teller > 9
             READ wikirel END-READ
             IF relnaam = wikinaam
                THEN
                  DISPLAY relomschr
                  MOVE 10 TO teller
                ELSE
                  PERFORM botsing
             END-IF
           END-PERFORM.
```

Zoeken is het enige waar je misschien nog moet opletten. Eers voor je natuurlijk de hash-functie uit en die geeft dan een sleutel. Vervolgens start je met een lus die maximaal het aantal records dat er in het bestand zit zal draaien. Je voert een *READ* uit met de sleutel die je hebt van de hash-functie. Nu moeten we eerst controleren of het wel het juist record is. Dit doen we door de naam da we hebben gekregen van de gebruiker te verglijken met dat van de record. Als het juist is toont dit programma gewoon de omschrijving van de wiki. Als het fout is moet botsing worden uitgevoerd tot het record is gevonden.

8.3 Record herschrijven

Een record herschrijven is niet moeilijk en je kunt de code van het vorige stukje gebruiken en gewoon een beetje aanpassen.

Record herschrijven

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. Wiki.

       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.

           SELECT wikirel ASSIGN TO "wikirel.txt"
             ORGANIZATION RELATIVE
             ACCESS RANDOM
             RELATIVE KEY nummer.

       DATA DIVISION.
       FILE SECTION.
       FD wikirel.
       01 relwiki.
        02 relnaam PIC X(11).
        02 relomschr PIC X(50).

       WORKING-STORAGE SECTION.
       01 nummer PIC 99.
       01 wikinaam PIC X(11).
       01 hulp PIC x(11).
       01 wikinummer PIC 9(11).
       01 kleineletters PIC X(27) VALUE 'abcdefghijklmnopqrstuvwxyz '.
       01 hoofdletters PIC X(27) VALUE 'ABCDEFGHIJKLMNOPQRSTUVWXYZ '.
       01 cijfers PIC X(27) VALUE '012345678901234567890123456'.
       01 teller PIC 99.

       PROCEDURE DIVISION.
       HOOFD.
           OPEN I-O wikirel
           DISPLAY "Geef de naam van de wiki die je wilt aanpassen:"
           ACCEPT wikinaam NO BEEP
           PERFORM zoeken
           CLOSE wikirel
           STOP RUN.

       ZOEKEN.
           PERFORM hash
           PERFORM varying teller FROM 1 BY 1 UNTIL teller > 9
             READ wikirel END-READ
             IF relnaam = wikinaam
               THEN
                 DISPLAY "Het originele inhoud: " relomschr
                 DISPLAY "Geef de nieuwe inhoud: "
                 ACCEPT relomschr NO BEEP
                 REWRITE relwiki END-REWRITE
                 MOVE 10 TO teller
               ELSE
                 PERFORM botsing
             END-IF
           END-PERFORM.

       HASH.
           MOVE wikinaam TO hulp
           INSPECT hulp CONVERTING kleineletters to hoofdletters
           INSPECT hulp CONVERTING hoofdletters TO cijfers
           MOVE hulp TO wikinummer
           DIVIDE wikinummer BY 9 GIVING wikinummer REMAINDER nummer
           ADD 1 TO nummer.

       BOTSING.
           ADD 1 TO nummer.
           IF nummer = 10
             MOVE 1 TO nummer
```

```
        END-IF.
```

Niet zoveel verschil dus. Er zijn slechts twee zaken die echt anders zijn. Ten eerste hoe je het bestand moeten openen. Net zoals een gewoon relatief bestand moet je dit openen met *I-O(INPUT-OUTPUT)*. Vervolgens doe je nadat je het record hebt gevonden die je wilt aanpassen een *REWRITE* net zoals een gewoon relatief bestand. Hieronder is er een schermafdruk van dit programma.

```
Geef de naam van de wiki die je wilt aanpassen:
Wikipedia
Het originele inhoud: De vrije encyclopedie
Geef de nieuwe inhoud:
VRIJE ENCYCLOPEDIE
```

Als je dan opnieuw wikipedia opzoekt zal je zien dat er "VRIJ ENCYCLOPEDIE"staat.

8.4 Record verwijderen

Dit volgende programma is gebaseerd op het vorige maar in de plaats van een record te herschrijven wordt die gedelete.

Record verwijderen

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. RecordVerwijderen.

       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
            SELECT wikirel ASSIGN TO "wikirel.txt"
              ORGANIZATION RELATIVE
              ACCESS RANDOM
              RELATIVE KEY nummer.

       DATA DIVISION.
       FILE SECTION.
       FD wikirel.
       01 relwiki.
        02 relnaam PIC X(11).
        02 relomschr PIC X(50).

       WORKING-STORAGE SECTION.
       01 nummer PIC 99.
       01 wikinaam PIC X(11).
       01 hulp PIC x(11).
       01 wikinummer PIC 9(11).
       01 kleineletters PIC X(27) VALUE 'abcdefghijklmnopqrstuvwxyz '.
       01 hoofdletters PIC X(27) VALUE 'ABCDEFGHIJKLMNOPQRSTUVWXYZ '.
       01 cijfers PIC X(27) VALUE '012345678901234567890123456'.
       01 teller PIC 99.

       PROCEDURE DIVISION.
       HOOFD.
            OPEN I-O wikirel
            DISPLAY "Geef de naam van de wiki die je wilt verwijderen:"
            ACCEPT wikinaam NO BEEP
            PERFORM zoeken
            CLOSE wikirel
            STOP RUN.

       ZOEKEN.
            PERFORM hash
            PERFORM varying teller FROM 1 BY 1 UNTIL teller > 9
              READ wikirel END-READ
```

```
            IF relnaam = wikinaam
              THEN
                DELETE wikirel
                DISPLAY "Record verwijderd"
                MOVE 10 TO teller
              ELSE
                PERFORM botsing
            END-IF
          END-PERFORM.

      HASH.
          MOVE wikinaam TO hulp
          INSPECT hulp CONVERTING kleineletters to hoofdletters
          INSPECT hulp CONVERTING hoofdletters TO cijfers
          MOVE hulp TO wikinummer
          DIVIDE wikinummer BY 9 GIVING wikinummer REMAINDER nummer
          ADD 1 TO nummer.

      BOTSING.
          ADD 1 TO nummer.
          IF nummer = 10
            MOVE 1 TO nummer
          END-IF.
```

Niet zo veel verschil met het vorige. *DELETE* in de plaats van *REWRITE* en natuurlijk altijd openen met *I-O*(*INPUT-OUTPUT*). Als je een record wilt lezen die je hebt gedelete geeft hij een foutmelding.

Deel IV

Gebruikersomgevingen

Hoofdstuk 9

Character-based interfaces

Er zijn twee manieren om in COBOL uw scherm op te maken de eerste is character gebaseerd en gebruikt alleen maar tekens en een paar kleuren om het scherm op te maken. De tweede manier werkt met pixels. Nu zien we character gebaseerd. Hoewel het grafisch minder mooi is als de andere manier heeft het zijn voordelen. Het is simpel en kan makkelijk met variabelen werken.

9.1 BLANK SCREEN

BLANK SCREEN laat toe om uw scherm volledig leeg te halen. Hiervoor moet je bij de *DATA DIVISION* een nieuwe sectie toevoegen. Deze sectie heet *SCREEN SECTION*.

BLANK SCREEN

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BLANKSCREEN.

DATA DIVISION.
WORKING-STORAGE SECTION.
77 toets PIC X.
SCREEN SECTION.
77 leeg BLANK SCREEN.

PROCEDURE DIVISION.
HOOFD.
    DISPLAY leeg
    ACCEPT toets
    STOP RUN.
```

SCREEN SECTION werkt zoals *WORKING-STORAGE SECTION*. In *SCREEN SECTION* kun je velden declareren. Deze velden kun je bepaalde eigenschappen geven. In dit geval geef je de eigenschap *BLANK SCREEN* aan de veld leeg. Wanneer leeg wordt opgeroepen via *DISPLAY* zal je een blanco scherm krijgen.

9.2 LINE en COLUMN

Via de attributen LINE en COLUMN je kun tekst plaatsen op bepaalde posities. Dit kan op 25 lijnen en 80 kolomen.

9.2.1 LINE

Je kunt ook tekst opslaan in de screen section en bepalen op welk lijntje die tekst komt te staan.

LINE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. lijn.

DATA DIVISION.
WORKING-STORAGE SECTION.
77 toets PIC x.
SCREEN SECTION.
77 leeg BLANK SCREEN.
77 tekst1 LINE 1 VALUE "dit is tekst 1 op regel 1".
77 tekst2 LINE 3 VALUE "dit is tekst 2 op regel 3".
77 tekst3 LINE 2 VALUE "dit is tekst 3 op regel 2".
77 tekst4 LINE 8 VALUE "dit is tekst 4 op regel 8".
77 tekst5 LINE 3 VALUE "dit is tekst 5 ik overschrijf regel 3".

PROCEDURE DIVISION.
HOOFD.
     DISPLAY leeg
     DISPLAY tekst1
     DISPLAY tekst2
     DISPLAY tekst3
     DISPLAY tekst4
     ACCEPT toets NO BEEP
     DISPLAY tekst5
     ACCEPT toets NO BEEP
     STOP RUN.
```

Verschillende stukjes tekst oproepen kan langdradig worden. Je kunt in de plaats van meerdere DISPLAY'Ste gebruiken ook één gebruiken. Je zet daarvoor de verschillende tekstjes in één groepsveld. Dit gaat veel sneller en komt overzichtelijkker over.

LINE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. lijn.

DATA DIVISION.
WORKING-STORAGE SECTION.
77 toets PIC X.
SCREEN SECTION.
01 scherm.
 02 leeg BLANK SCREEN.
 02 tekst1 LINE 1 VALUE "dit is tekst 1 op regel 1".
 02 tekst2 LINE 3 VALUE "dit is tekst 2 op regel 3".
 02 tekst3 LINE 2 VALUE "dit is tekst 3 op regel 2".
 02 tekst4 LINE 8 VALUE "dit is tekst 4 op regel 8".


PROCEDURE DIVISION.
HOOFD.
     DISPLAY scherm
     ACCEPT toets NO BEEP
     DISPLAY leeg
     DISPLAY tekst4
     ACCEPT toets NO BEEP
     STOP RUN.
```

Via scherm kun je alles in één keer op het scherm tonen. Maar je kunt ook nogsteeds alles apart tonen. Als je weet dat je nooit de verschillende deeltjes apart zult oproepen moet je die dan ook geen naam geven.

9.2.2 COLUMN

Via dit kun je bepalen in welke kolom je de verschillende tekstjes wilt plaatsen.

COLUMN

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. kolom.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 toets PIC X.
       SCREEN SECTION.
       01 scherm.
        02 leeg BLANK SCREEN.
        02 tekst1 LINE 1 COLUMN 1 VALUE "lijn 1 kolom 1".
        02 tekst2 LINE 2 COLUMN 10 VALUE "lijn 2 kolom 10".
        02 tekst3 LINE 5 COLUMN 66 VALUE "lijn 5 kolom 66".
        02 tekst4 LINE 1 COLUMN 20 VALUE "lijn 1 kolom 20".

       PROCEDURE DIVISION.
       HOOFD.
            DISPLAY scherm
            ACCEPT toets NO BEEP
            STOP RUN.
```

COLUMN is daardoor heel goed geschikt om verschillende stukjes tekst achter elkaar te plaatsen. Natuurlijk zijn LINE en COLUMN perfect te combineren en kun je ze tegelijkertijd gebruiken.

Dit is het resultaat:

9.2.3 Een andere manier

LINE en COLUMN zijn niet enkel en alleen voor de SCREEN SECTION je kun ze perfect gebruiken middenin je programma code via DISPlAY.

Een andere manier.

```
            DISPLAY "dit is tekst" LINE 5 COLUMN 35
```

Dit is vooral handig als je slechts een paar lijntjes slechts een paar keer laat zien. Maar niet wanneer je een heel scherm moet opmaken en altijd opnieuw moet oproepen.

9.3 Combineren met velden

Het gebruik van velden is essentieel in een programma. Je moet gegevens inlezen en verwerken en dan terug kunnen tonen op het scherm. In dit stuk gaan we leren hoe je de SCREEN SECTION kan laten comuniceren met andere secties van de DATA DIVISION.

9.3.1 FROM

FROM is een attribuut en laat toe om een veld te laten zien op het scherm via de SCREEN SECTION. FROM laat enkel toe dat het veld wordt getoond dus je kunt er niet mee werken.

FROM

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. veldtonen.
```

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77 tekst PIC X(12)  value "Dit is tekst".
77 toets PIC X.
SCREEN SECTION.
01 scherm.
 02 leeg BLANK SCREEN.
 02 LINE 5 COLUMN 10 PIC x(12) FROM tekst.

PROCEDURE DIVISION.
HOOFD.
    DISPLAY scherm
    ACCEPT toets NO BEEP
    STOP RUN.
```

Wanneer je FROM gebruikt moet je ook PIC gebruiken.

9.3.2 USING

USING laat dan weer toe om gegevens in te voeren of aan te passen. USING wordt net zoals FROM gebruikt en je moet ook gebruik maken van PIC.

USING

```
IDENTIFICATION DIVISION.
PROGRAM-ID. invoer.

DATA DIVISION.
WORKING-STORAGE SECTION.
77 voornaam    PIC X(20).
77 achternaam PIC X(20).
77 toets       PIC X.
SCREEN SECTION.
01 scherm.
 02 leeg BLANK SCREEN.
 02 LINE 1              VALUE "************************".
 02 LINE 2              VALUE "*   Geef de gegevens   *".
 02 LINE 3              VALUE "************************".
 02 LINE 5 COLUMN 1   VALUE "Geef de voornaam".
 02 LINE 5 COLUMN 20 PIC X(20) USING voornaam.
 02 LINE 6 COLUMN 1   VALUE "Geef de achternaam".
 02 LINE 6 COLUMN 20 PIC X(20) USING achternaam.

PROCEDURE DIVISION.
HOOFD.
    DISPLAY scherm
    ACCEPT scherm
    STOP RUN.
```

Dit programma leest de voornaam en achternaam van een persoon. Let wel op wanneer je gegevens wilt lezen via SCREEN SECTION moet je ook ACCEPT gebruiken gevolgd door de naam van het scherm dat je gebruikt.

Dit is het resultaat:

9.4 Werken met kleuren

In COBOL kunnen normaal slechts 8 kleuren worden gebruikt genummerd van 0 tot 7.

9.4.1 BACKGROUND-COLOR

BACKGROUND-COLOR laat toe om de achtergrond kleur te bepalen. Niet alleen van het scherm maar ook van stukjes tekst.

BACKGROUND-COLOR

```
01 scherm.
 02 leeg BLANK SCREEN BACKGROUND-COLOR 1.
 02 LINE 1 COLUMN 1 VALUE "dit is tekst" BACKGROUND-COLOR 4.
 02 LINE 2 COLUMN 1 VALUE "nog tekst" BACKGROUND-COLOR 3.
```

BACKGROUND-COLOR wordt dus gevolgd door een nummer van 0 tot 7.

Dit is het resultaat:

9.4.2 FOREGROUND-COLOR

FOREGROUND-COLOR laat dan weer toe om de kleur van de tekst zelf aan te passen.

FOREGROUND-COLOR

```
01 scherm.
 02 leeg BLANK SCREEN.
 02 LINE 1 COLUMN 1 VALUE "dit is tekst"  FOREGROUND-COLOR 4.
 02 LINE 2 COLUMN 1 VALUE "nog tekst" FOREGROUND-COLOR 3.
```

Dit is het resultaat:

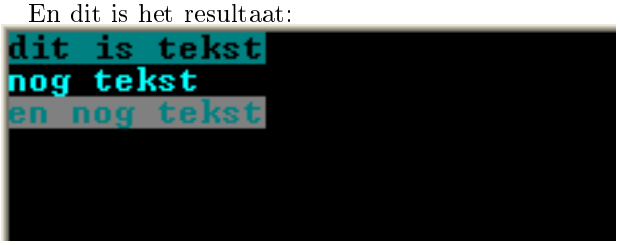
9.4.3 REVERSE-VIDEO, HIGHLIGHT en BLINK

Dit zijn drie atributen die speciale dingen doen met de kleuren. REVERSE-VIDEO verwisselt de background en foreground van kleur. HIGHLIGHT maakt de kleur van de tekst feller. En BLINK maakt de kleur van de achtergrond feller.

REVERSE-VIDEO, HIGHLIGHT en BLINK.

```
01 scherm.
 02 leeg BLANK SCREEN FOREGROUND-COLOR 3.
 02 LINE 1 COLUMN 1 VALUE "dit is tekst"  REVERSE-VIDEO.
 02 LINE 2 COLUMN 1 VALUE "nog tekst" HIGHLIGHT.
 02 LINE 3 COLUMN 1 VALUE "en nog tekst" BLINK.
```

En dit is het resultaat:



Index

COMPUTE, 19

DELETE, 53

groepsveld, 33

READ, 49
REWRITE, 53

Wortel, 20
WRITE, 52