



Programmatie logica

Bestanden

Webleren

School je gratis bij via het internet. Waar en wanneer je wilt.

www.vdab.be/webleren

© COPYRIGHT 2015 VDAB

Niets uit deze syllabus mag worden verveelvoudigd, bewerkt, opgeslagen in een database en/of openbaar gemaakt door middel van druk, fotokopie, microfilm, geluidsband, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van VDAB.

Hoewel deze syllabus met zeer veel zorg is samengesteld, aanvaardt VDAB geen enkele aansprakelijkheid voor schade ontstaan door eventuele fouten en/of onvolkomenheden in deze syllabus en of bijhorende bestanden.

Inhoud

Hoofdstuk 1.	Inleiding	5
1.1.	Algemene informatie.....	5
1.1.1.	Waarom?	5
1.1.2.	Werkwijze	5
Hoofdstuk 2.	Soorten	7
2.1.	Waarom bestanden.....	7
2.1.1.	Traditionele bestanden	7
2.1.2.	Relationele databases	7
2.1.3.	Enkele begrippen	8
2.1.4.	Verschil tussen traditionele en relationele opslag	9
2.2.	Soorten bestanden	10
2.2.1.	Sequentiële bestanden.....	10
2.2.2.	Directe bestanden	12
2.2.3.	Index-sequentiële bestanden	13
2.2.4.	Index-sequentiële bestanden: Voorbeeld 1	14
2.2.5.	Index-sequentiële bestanden: Voorbeeld 2	14
2.2.6.	Samenvatting.....	16
Hoofdstuk 3.	Niet-geformatteerde bestanden	17
3.1.	Een niet-geformatteerd sequentieel bestand maken	17
3.2.	Voorbeeld: Woordenboek maken	18
3.3.	Een niet-geformatteerd sequentieel bestand lezen	19
3.4.	Voorbeeld: Woordenboek tonen	21
3.5.	Samenvatting.....	22
3.6.	Voorbeeld: Gecombineerd	22
Hoofdstuk 4.	Geformatteerde bestanden.....	23
4.1.	Geformatteerde sequentiële bestanden.....	23
4.1.1.	Een bestand maken	23
4.1.2.	Maken: opbouw	23
4.1.3.	Voorbeeld: Personeelsbestand maken	25
4.1.4.	Een bestand lezen.....	26
4.1.5.	Voorbeeld 1: Personeelslijst.....	26
4.1.6.	Voorbeeld 2: Totaal loon	29

4.1.7.	Voorbeeld 3: Hasselt	31
4.1.8.	Samenvatting.....	32
4.1.9.	Een bestand aanpassen	32
4.1.10.	Voorbeeld: Personeelsbestand aanpassen	33
Hoofdstuk 5.	Opgaven.....	35
5.1.1.	Opgave 1 Gemiddelde berekenen.....	35
Hoofdstuk 6.	Tot slot.....	36
6.1.	Eindoefening.....	36
6.2.	Wat nu?	36

Hoofdstuk 1. Inleiding

1.1. Algemene informatie

1.1.1. Waarom?

Waarom leren programmeren? De belangrijkste redenen op een rijtje:

- Programmeren leert je een probleem-oplossende manier van denken aan. Je leert **analytisch denken**.
- Als je kan programmeren kun je een hoop dingen **automatiseren** en ben je waarschijnlijk ook handig met andere ict-gerelateerde zaken.
- Je leert werken met **gegevens**. Informatica is een studie over gegevens en informatie. Bij programmeren leer je hoe je met gegevens om moet gaan en wat je ermee kunt doen.
- Websites maken is waardevol tegenwoordig. Bijna elk bedrijf of project heeft een **bijhorende site** die moet worden onderhouden (Javascript, PHP,...).
- ...

In deze cursus **programmatielogica** leer je gestructureerd programmeren, dwz dat je leert zelfstandig problemen op te lossen met de computer. Je gebruikt Nassi-Shneiderman diagrammen. Deze leggen de basis voor het echte programmeerwerk in één of andere taal.

De cursus programmaticalogica bestaat uit verschillende delen.

Basisstructuren, Procedures en functies en **Arrays en Strings** heb je reeds doorgenomen.

Dit deel gaat over het gebruik van bestanden. Welke soorten zijn er? Hoe kan je bestanden gebruiken?

Als je na dit onderdeel nog zin hebt in meer, kan je je nog verder verdiepen in **Sorteren**.

De nadruk in alle delen ligt op het probleemoplossend denken. Na het tekenen van de diagrammen gaan we deze schema's ook omzetten naar echte programmacode om ze uit te testen.

1.1.2. Werkwijze

In deze cursus gebruiken we een tekentool om Nassi-Shneiderman diagrammen, PSD's of Programma Structuur Diagrammen, te tekenen (Structorizer). Je kan alle diagrammen ook maken met pen en papier. We gebruiken ook het programma **Lazarus** om onze code uit te testen.

In deze cursus zijn heel wat oefeningen voorzien. Je kan ze onderverdelen in twee categorieën:

- *Gewone oefeningen:*
Dit is het elementaire oefenmateriaal dat noodzakelijk is om de leerstof onder de knie te krijgen.
Bij deze oefeningen kan je ook altijd een modeloplossing (van het PSD) terug vinden.
- *Opdrachten voor de coach:*
Per hoofdstuk is er een opdracht voor de coach voorzien. Deze kan als 'test' voor dat hoofdstuk dienen. Dit is een samenvattende oefening van de voorgaande leerstof. Voor deze opdrachten zijn er geen modeloplossingen voorzien.

Als je deze cursus volgt binnen een **competentie centrum**, spreek je met je **instructeur** af hoe de opdrachten geëvalueerd worden.

Anders stuur je je oplossingen van de opdrachten voor de **coach** door aan je coach ter verbetering. Als je een probleem of vraag hebt over één van de gewone oefeningen, kan je ook bij je coach terecht. Stuur steeds zowel het .nsd-bestand als het .pas-bestand dat je gemaakt hebt door. Dit maakt het voor je coach makkelijker om je vraag snel te beantwoorden.

Let op!

Het is niet de bedoeling om met Structorizer te leren werken, wel om een probleem te leren analyseren en in een gestructureerd diagram weer te geven. Structorizer en Lazarus zijn enkel hulpmiddelen om het tekenen en uittesten van die diagrammen te vereenvoudigen.

Hoofdstuk 2. Soorten

2.1. Waarom bestanden

Als we tot nu toe gegevens nodig hadden, dan schreven we in ons programma een Read-opdracht. De gewenste gegevens konden dan via het toetsenbord worden ingevoerd.

Als dat de enige manier zou zijn waarop een programma met gegevens kan werken, dan zou de automatisering nooit zo'n vlucht hebben genomen als nu het geval is. Voor het werken met grote hoeveelheden gegevens is het noodzakelijk dat we ze kunnen opslaan in bestanden.

In deze cursus zullen we die gegevensbestanden verder bestuderen.

2.1.1. Traditionele bestanden

Informatie wordt opgeslagen in **bestanden**. Bestanden kunnen tekst, foto's of bijvoorbeeld gegevens over de medewerkers van een bedrijf bevatten. Een bestand is daarom steeds een verzameling van dezelfde soort gegevens. Bijv.: een personeelsbestand bevat info over personeelsleden, een artikelbestand over artikels, ...

In een bedrijf bestaat er meestal slechts één personeelsbestand, waarvan elk record alle noodzakelijke gegevens over een werknemer bevat.

Aan de hand van dit personeelsbestand, kan een programma bijv. het vakantiegeld van elke werknemer berekenen. Tijdens de verwerking doet het programma een beroep op de gegevens uit het personeelsbestand.

Dit is een voorbeeld van een programma dat **werkt met** gegevens opgeslagen in een bestand.

Gegevens van personeelsleden kunnen ook wijzigen. Een werknemer kan

- in dienst komen,
- verhuizen,
- trouwen,
- een kindje krijgen
- op pensioen gaan
-

Er zijn tal van wijzigingen mogelijk. Het is belangrijk dat het personeelsbestand altijd de juiste gegevens bevat van alle werknemers. Dus moet het personeelsbestand gewijzigd of bijgewerkt kunnen worden. Ook daarvoor bestaan er programma's.

Een dergelijk programma is een programma dat **werkt aan** een gegevensbestand.

2.1.2. Relatieve databases

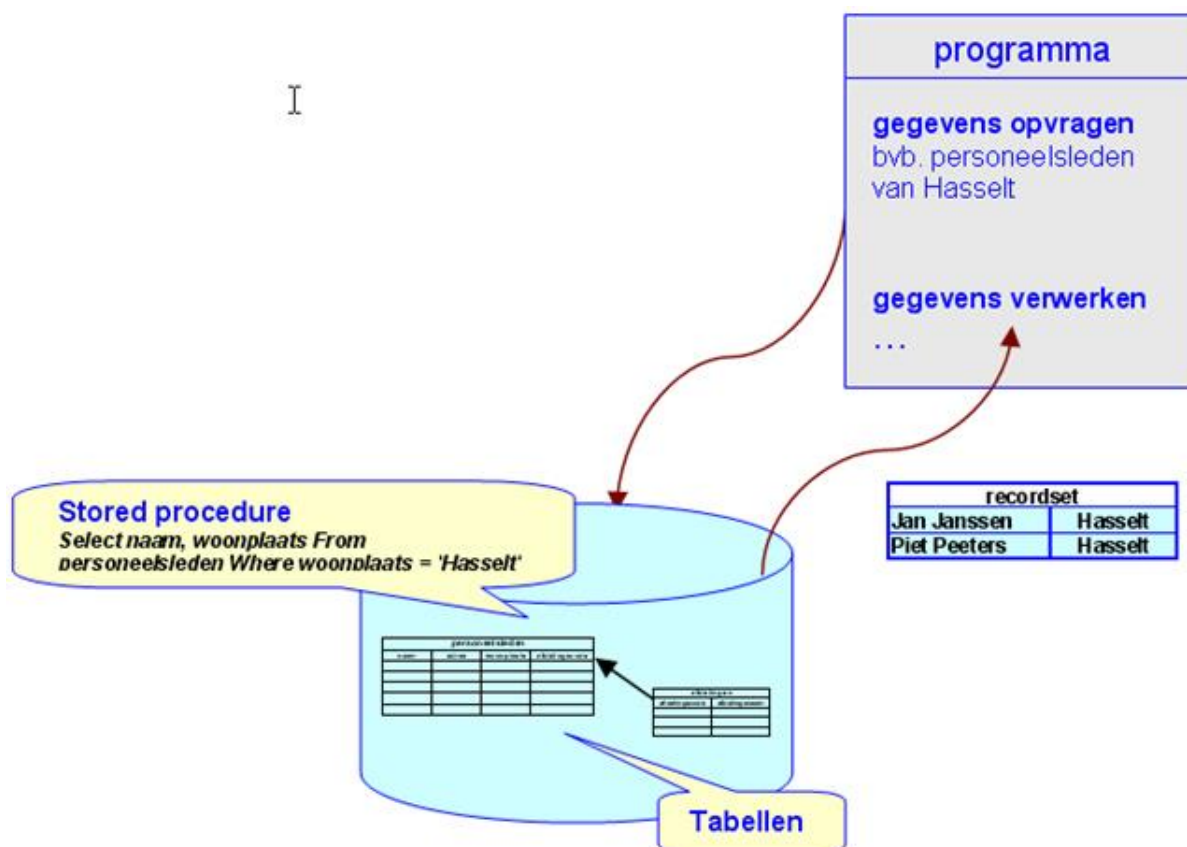
Om grote hoeveelheden gegevens op een gestructureerde manier te bewaren, gebruikt men vaak een **relatieve database** (RDBMS = Relational Database Management System). Het eigenlijke beheer van deze gegevens (opslaan, wijzigen, toevoegen, ...) wordt door het DBMS zelf afgehandeld.

Ook bij het ontwerp van een relationele database speelt de normalisatie van gegevens een belangrijke rol.

Microsoft Access, MySQL, SQL Server, Oracle, DB2,... zijn voorbeelden van zo'n DBMS.

In een relationele database zijn de verschillende tabellen via een sleutel met elkaar verbonden. We spreken van een één-op-één-relatie of een één-op-veel-relatie.

Om de gegevens uit een database te gebruiken in je programma moet je een **SQL-opdracht** of een stored procedure uitvoeren. Het resultaat hiervan is een recordset. Deze recordset kan als een sequentieel bestand verwerkt worden door je programma.

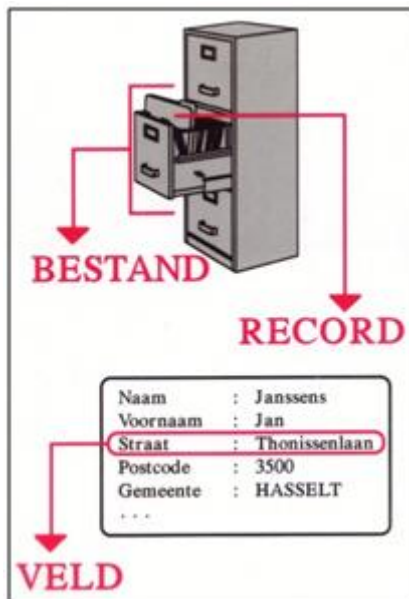


2.1.3. Enkele begrippen

Informatie over één personeelslid uit het personeelsbestand of één artikel uit het artikelbestand noemen we een **record**. Elk record heeft dezelfde structuur. Zo kan een record van het personeelsbestand gegevens bevatten zoals naam, voornaam, adres, woonplaats, tel.nr., geboortedatum, functie,... .

Elk gegeven uit één record noemen we een **veld**. Dus naam is een veld, maar ook voornaam, geboortedatum Elk record van één bestand heeft dezelfde velden.

Elk veld heeft een **eigenschap** of **type**. Het veld kan een getal bevatten, een karakter, een string, een datum,... Die eigenschap moeten we tijdens de verwerking respecteren.



2.1.4. Verschil tussen traditionele en relationele opslag

Bij een traditioneel bestand is er **geen scheiding** tussen het programma en de gegevensopslag. Als de structuur van het bestand wijzigt, moet de programmacode ook aangepast worden.

Bij **databases** is de structuur opgeslagen in het DBMS zelf en dus volledig **gescheiden** van de software die de toegang tot de gegevens beheert.

Voorbeeld

De personeelsdienst van een bedrijf moet een overzicht kunnen maken van de personeelsgegevens en de lonen van de personeelsleden.

Er wordt een gegevensbestand met deze gegevens aangemaakt, evenals een programma om dit personeelsbestand te raadplegen, te tonen en erbij te werken.

Bij de keuze van de bestandsorganisatie wordt rekening gehouden met de wijze waarop dit personeelsbestand zal gebruikt worden.

De R&D afdeling van hetzelfde bedrijf wil uiteraard naast de gegevens en resultaten van de verschillende onderzoeksprojecten ook registreren welke personeelsleden aan welke projecten meewerken.

Deze afdeling heeft dus ook gegevens van de personeelsleden nodig. Ook hier wordt een aangepaste bestandsorganisatie gekozen en een gepast verwerkingsprogramma geschreven.

Enerzijds hebben de verschillende afdelingen dezelfde informatie over de personeelsleden nodig (naam, afdeling, ...). Anderzijds heeft de ene afdeling informatie nodig die de andere afdeling niet gebruikt, bijv. loongegevens voor de personeelsdienst, projectgegevens voor de R&D afdeling. Zowel de personeelsdienst als de R&D afdeling bewaren gegevens over personeelsleden in verschillende bestanden. Dit betekent dat bepaalde informatie meerdere keren opgeslagen wordt, wat meer geheugenruimte vraagt en extra werk betekent bij wijzigingen van de gegevens. Op deze manier verhoogt ook de kans op fouten.

Door gebruik te maken van een database, wordt alle informatie maar één keer opgeslagen. Hierdoor krijgt elke gebruiker te maken met dezelfde gegevensstructuur. Het is de taak van het DBMS om ervoor te zorgen dat de gebruiker de gegevens niet rechtstreeks kan benaderen en dat de gebruiker op een efficiënte manier de nodige informatie krijgt.

2.2. Soorten bestanden

2.2.1. Sequentiële bestanden

Een **sequentieel bestand** is een vorm van een bestand waarbij de gegevens na elkaar (in volgorde van binnenkomen) opgeslagen worden. Dit betekent ook dat deze gegevens, één voor één, na elkaar geraadpleegd worden.

Niet-geformatteerde sequentiële bestanden

Deze tekstbestanden bestaan louter uit tekst. Ze vormen in die zin een opeenvolging van tekens/karakters, over meerdere regels verspreid, waarbij elke regel verschillend van lengte is.

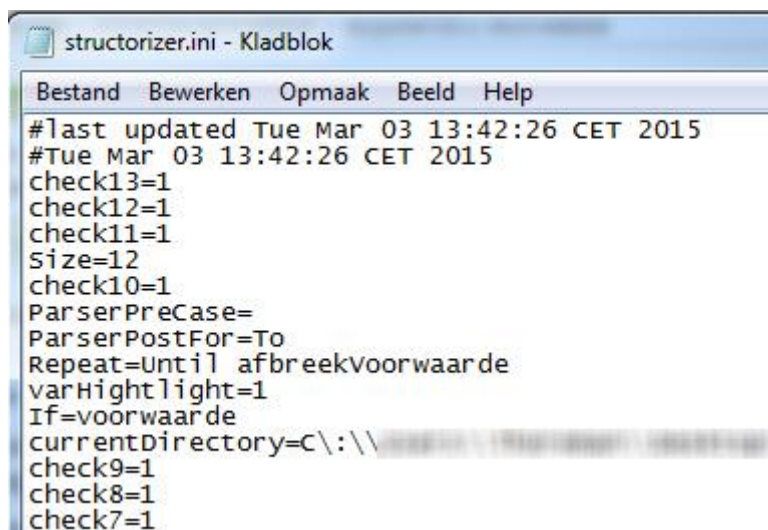
We hebben het hier over **niet-geformatteerde sequentiële bestanden**.

Voorbeelden hiervan zijn tekstbestanden (bijv. een .txt-bestand, een .ini-bestand van windows, enz.), geluidsbestanden (muziekcasette), videobestanden, maar ook gegevensbestanden.

Waarvoor worden deze bestanden zoal gebruikt?

- **Configuration files** worden gebruikt door applicaties: de applicatie dient, voor zijn werking, bepaalde instellingen te weten/kennen. Vaak worden .ini-files dan ook alleen maar gelezen door een applicatie en gebeurt de aanpassing ervan veelal handmatig.
- **Log-files** kunnen gebruikt worden voor bijv. het maken van statistieken (wat is het surfgedrag van de werknemers, welke applicaties geven problemen (errors die gelogd worden), enz.).

Voorbeeld: het ini-bestand van Structorizer



```

structorizer.ini - Kladblok
Bestand  Bewerken  Opmaak  Beeld  Help
#last updated Tue Mar 03 13:42:26 CET 2015
#Tue Mar 03 13:42:26 CET 2015
check13=1
check12=1
check11=1
size=12
check10=1
ParserPreCase=
ParserPostFor=To
Repeat=Until afbreekvoorwaarde
varHighlight=1
If=voorwaarde
currentDirectory=C:\\...
check9=1
check8=1
check7=1

```

Geformateerde sequentiële bestanden

Dit personeelsbestand bestaat uit meerdere records. Elk record bestaat uit 5 velden, nl. personeelsnr, naam, adres, postcode en woonplaats.

Elk record is even groot. We noemen dit een **geformatteerd sequentieel bestand**.

1	Jan	Bosweg 40	3000	Leuven
7	Linda	Dorpstraat 99	9000	Gent
3	Katrien	Kapeldreef 1	1000	Brussel
12	Joris	Kaai 17	8400	Oostende
8	An	Dennenbos 15	3500	Hasselt
15	Piet	Karrenstraat 50	2000	Antwerpen

Om de gegevens van dit bestand te lezen, dient het bestand serieel, dus vanaf het begin tot het einde, record voor record, gelezen te worden.

Verwerking van sequentiële bestanden

Bij het verwerken is het erg belangrijk om te weten wanneer het einde van een bestand bereikt is. Over het algemeen wordt het einde van een sequentieel bestand aangeduid met de Engelse term end-of-file (EOF).

Een sequentieel bestand wordt **vanaf het begin gelezen**, dwz karakter per karakter of regel per regel voor niet-geformatteerde bestanden en record per record voor geformatteerde bestanden, totdat de juiste gegevens gevonden worden of totdat het EOF-teken gelezen wordt.

Als je bijv. record 8 nodig hebt, moet je eerst de 7 voorgaande records lezen.

Dit betekent dat de verwerkingstijd van de gegevens hoog kan oplopen, afhankelijk van de uit te voeren bewerkingen.

Je kan dit vergelijken met een muziekcassette: om het 8e liedje te beluisteren, moet je eerst de 7 voorgaande liedjes doorspoelen.

Je kan ook **niet zomaar** ergens gegevens **tussenvoegen** of gegevens **verwijderen**. Je kan bijv. achteraan gegevens bijvoegen, maar dan moet je je eerst achteraan positioneren, door de reeds aanwezige gegevens eerst in te lezen.

Om gegevens tussen te voegen of te verwijderen bij sequentiële bestanden bestaan er specifieke algoritmen.

Verder in de cursus volgen voorbeelden van de verwerking van tekstbestanden en record-bestanden.

Gebruik van sequentiële bestanden

De **verwerkingstijd** speelt een belangrijke rol bij de keuze voor een sequentiële bestandsorganisatie. Sequentiële bestanden hebben het nadeel dat het lezen en het wijzigen ervan veel tijd kost.

Sequentiële bestanden worden **gebruikt**

- wanneer bij het raadplegen ervan, meestal veel of alle records aan bod moeten komen;
- of wanneer wijzigingen meestal bij een groot aantal of alle records moeten gebeuren.

Als je bijv. enkele keren per dag eenzelfde record of een beperkt aantal dezelfde records moet wijzigen, is deze bestandsorganisatie minder geschikt.

Je dient steeds het volledige record in te lezen, dus ook de velden die eventueel niet nodig zijn/gebruikt gaan worden.

Sequentiële bestandsorganisatie is ook bruikbaar als het niet altijd nodig is dat het bestand op ieder ogenblik de actuele toestand van de gegevens bevat.

Bijv. als de gegevens van een personeelsbestand enkel actueel moeten zijn op het einde van de maand als de lonen berekend worden, kan je voor een sequentiële bestandsorganisatie opteren. Je houdt in de loop van de maand de wijzigingen van de personeelsgegevens bij. Vóór de loonsberekening voer je alle wijzigingen door en beschik je over de juiste gegevens.

2.2.2. Directe bestanden

Bij een reserveringsbureau zou een sequentiële bestandsorganisatie voor de reservaties door de lange verwerkingstijd geen goede keuze zijn.

Aanvragen voor reservaties kunnen op onvoorspelbare momenten via verschillende toestellen binnenkomen. Vrijwel meteen moet nagegaan worden of een reservatie mogelijk is zodat deze reservatie onmiddellijk bevestigd of geannuleerd kan worden.

Voor deze toepassing kan een **directe bestandsorganisatie** gebruikt worden.

Je kan dit vergelijken met een muziek-CD. Je kan direct een bepaald liedje beluisteren. Het is niet nodig om eerst alle voorgaande liedjes te beluisteren.

De records worden niet langer het ene na het andere opgeslagen zoals bij een sequentieel bestand. Elk record krijgt een plaats toegewezen volgens een welbepaalde sleutel. Voor elke sleutel wordt vooraf reeds een plaats voorzien.

De gegevens worden fysiek opgeslagen op schijf waarbij de sleutel van elk record de fysieke plaats op schijf aanduidt. Zo kan een record snel teruggevonden worden op basis van zijn sleutelwaarde.

Als voorbeeld gebruiken we dezelfde gegevens als bij de sequentiële bestanden waarbij het personeelsnr dienst doet als sleutel.

1	Jan	Bosweg 40	3000	Leuven
3	Katrien	Kapeldreef 1	1000	Brussel
7	Linda	Dorpstraat 99	9000	Gent
8	An	Dennenbos 15	3500	Hasselt

Het record met nummer 1 wordt op de eerste plaats opgeslagen, het record met nummer 3 op de derde plaats, enz. De plaatsen 2, 4, 5, 6, enz. worden leeg gelaten.

Een voordeel hiervan is dat men direct de gegevens van een bepaald nummer kan lezen. Als je bijv. het record met nummer 7 wil raadplegen, hoef je niet meer alle voorgaande records eerst in te lezen. De **verwerkingstijd** wordt zo aanzienlijk **korter** in vergelijking met sequentiële bestanden. Mutaties van gegevens worden meestal onmiddellijk uitgevoerd.

Een nadeel van dit soort bestanden is dat er **veel lege ruimte** is tussen de records. Er is meer opslagcapaciteit nodig dan de ruimte die effectief nodig is voor het opslaan van de gegevens.

Bovendien kan directe bestandsorganisatie enkel gerealiseerd worden op een adresseerbaar geheugenmedium zoals een magneetschijf, CD-ROM, ...

2.2.3. Index-sequentiële bestanden

Zoals al gezegd zijn sequentiële bestanden inefficiënt wanneer records in een onvoorspelbare volgorde opgehaald moeten worden. In dergelijke situaties is het belangrijk om snel de locatie van het gewenste record te kunnen vinden. Een index met rangnummers kan dit probleem oplossen. Zo'n bestandssysteem wordt een **geïndexeerd bestand** genoemd. Index-sequentieel georganiseerde bestanden hebben een vaste recordlengte. De sleutel bestaat uit één of meerdere velden uit het bestand.

Een **index** of indextabel voor een bestand bevat een lijst met de in het bestand opgeslagen sleutels en daarnaast het item dat aangeeft waar het record met de bepaalde sleutel is opgeslagen in het originele gegevensbestand (het relatieve recordnr). Om een bepaald record te vinden, hoef je alleen maar de sleutel in de index (index-bestand) te zoeken en vervolgens de informatie, die is opgeslagen op de locatie die aan deze sleutel gekoppeld is, op te halen.

Bij deze bestanden worden eigenlijk twee bestanden bijgehouden door de computer.

- het **fysische bestand**
- bevat de eigenlijke gegevens van het bestand, in hun originele volgorde
- een **indexbestand**
- bevat de sleutel in (meestal stijgende) volgorde en het recordnummer van het fysische record dat bij die sleutel hoort

Het voordeel van deze bestandsorganisatie is dat de sleutel tot het bestand niet meer een recordnummer is zoals bij directe bestanden. Elk veld en elke combinatie van velden uit het bestand kan als sleutel dienen. Je kan bij één fysisch bestand meerdere indexbestanden maken.

Wanneer we een bestand index-sequentieel lezen zorgt de systeemsoftware automatisch voor het sequentieel opzoeken van de sleutel in de index, waarna het overeenkomstig record in het fysisch bestand automatisch wordt opgezocht. Als programmeur hoeven we ons daar niets van aan te trekken. We moeten enkel vermelden welke index we wensen te gebruiken en welke de sleutelwaarde is van het record dat we willen bewerken.

2.2.4. Index-sequentiële bestanden: Voorbeeld 1

Een tabel van personeelsgegevens met daarnaast een indextabel op **personeelsnr.**

Recordopslagruimte		Indextabel	
Recordnummer	Gegevens	Sleutel	Recordnummer
...	
68	20 Peeters Peter
75	19 Janssens Jan ...	18	87
...		19	75
87	18 Pietersen Piet...	20	68
90	22 Jorissen Joris...	22	90
...	
...	

Een tweede indextabel, maar nu in volgorde van de **naam**, die bij dezelfde gegevenstabel hoort:

Indextabel	
Sleutel	Recordnummer
...	...
...	...
Janssen Jan	75
Jorissen Joris	90
Peeters Peter	68
Pietersen Piet	87
...	...
...	...

De gegevens in de indextabel staan altijd gesorteerd op de sleutel. Dit kan zowel een oplopende als een aflopende sortering zijn.

2.2.5. Index-sequentiële bestanden: Voorbeeld 2

Om een nieuw record toe te voegen, plaats je de nieuwe gegevens achteraan in het fysieke bestand. In het indexbestand wordt de sleutel op de juiste plaats tussengevoegd.

Een voorbeeld: voeg *Nelissen Leen* met personeelsnr 21 toe

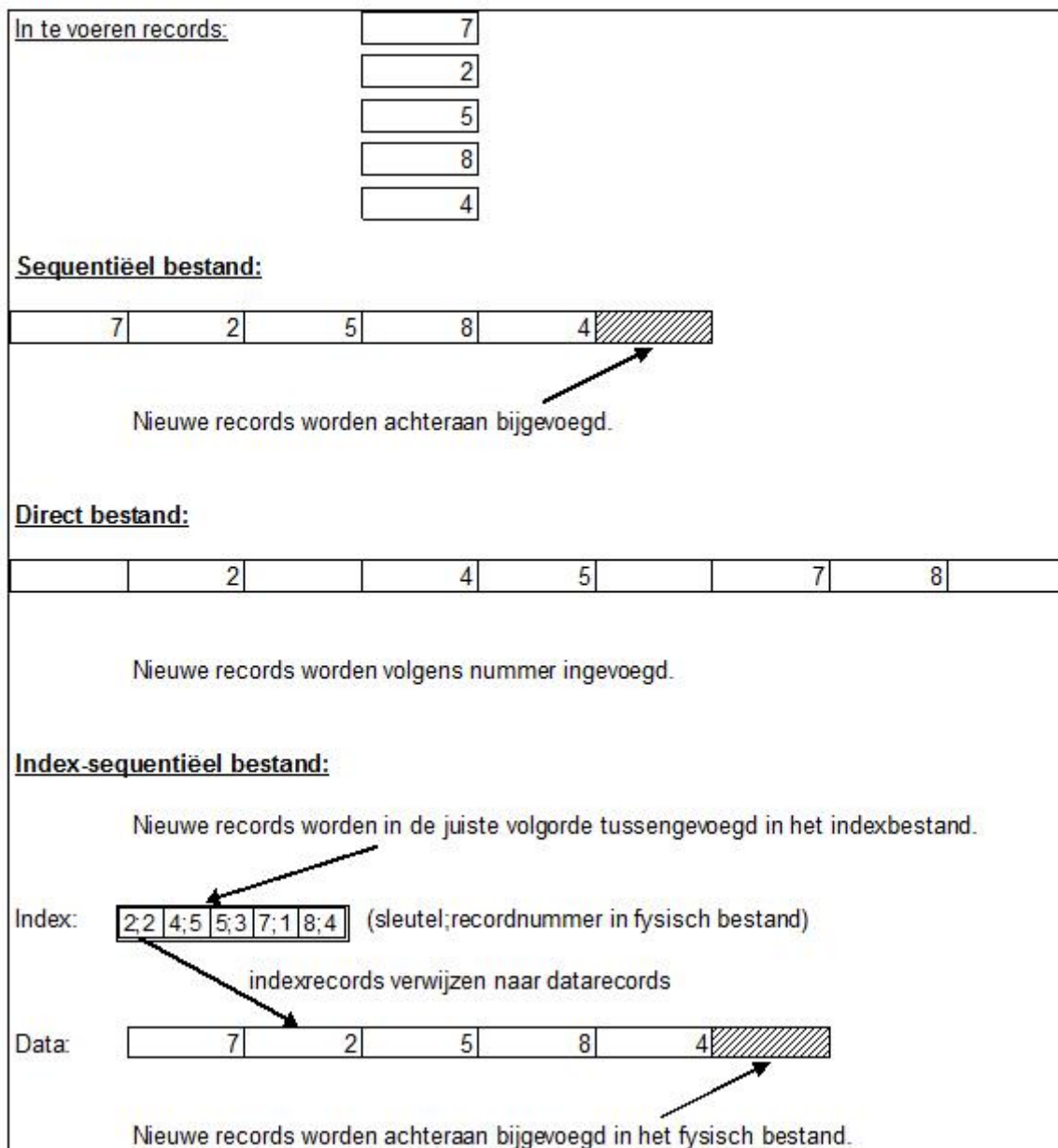
Recordopslagruimte	
<i>Recordnummer</i>	<i>Gegevens</i>
...	
68	20 Peeters Peter ...
75	19 Janssens Jan ...
...	
87	18 Pietersen Piet...
90	22 Jorissen Joris...
91	21 Nelissen Leen
...	

Indextabel	
<i>Sleutel</i>	<i>Recordnummer</i>
...	...
...	...
18	87
19	75
20	68
21	91
22	90
...	...

Indextabel	
<i>Sleutel</i>	<i>Recordnummer</i>
...	...
...	...
Janssen Jan	75
Jorissen Joris	90
Nelissen Leen	91
Peeters Peter	68
Pietersen Piet	87
...	...

Elke wijziging aan het gegevensbestand betekent dus dat ook de indexbestanden moeten bijgewerkt worden. Hoe meer indexbestanden, hoe hoger de verwerkingstijd.

2.2.6. Samenvatting



Hoofdstuk 3. Niet-geformatteerde bestanden

Zoals eerder vermeld sla je de gegevens bij sequentiële bestanden één voor één na elkaar op. Als je één van de records wil raadplegen, moet je het bestand steeds vanaf het begin, record na record inlezen totdat je het gewenste record vindt.

We beginnen met niet-geformatteerde sequentiële bestanden.

3.1. Een niet-geformatteerd sequentieel bestand maken

Voorbeeld: Je wil een woordenboek maken waarbij op de eerste regel het Nederlandse woord en op de volgende regel de vertaling in het Frans staat. Je maakt een programma dat telkens het Nederlandstalige woord en nadien de vertaling vraagt. Dit wordt herhaald tot de letter 'S' of 's' ingegeven wordt.

Om dit op te lossen heb je een aantal speciale instructies nodig:

Instructie	Betekenis
<code>var f: TEXT</code>	definieer een bestand f van het type <code>TEXT</code> . Het bestand f kan zowel cijfers als letters bevatten.
<code>ASSIGN(f, bestandsnaam)</code>	koppel de bestandsvariabele f aan het eigenlijke bestand bestandsnaam . vb <code>ASSIGN(f, 'woordenboek.txt')</code> Bij het uittesten in Lazarus wordt het bestand <code>woordenboek.txt</code> automatisch bewaard in dezelfde map als het pas-bestand.
<code>REWRITE(f)</code>	opent het bestand om te schrijven
<code>Write f, woord</code>	schrijft de inhoud van de variabele woord naar het bestand f
<code>CLOSE(f)</code>	sluit het bestand Op het einde wordt nog een EOF-teken toegevoegd.

3.2. Voorbeeld: Woordenboek maken

Maak onderstaand psd aan. Test het nadien in Lazarus.

BestandMaken
var f: text var naam, woord: string
Write 'Geef de bestandsnaam voor de woordenlijst'
Read naam
ASSIGN(f,naam)
REWRITE(f)
Write 'Typ de woorden in. "S" is stoppen'
Write 'Nederlands:'
Read woord
While (woord<>'S') AND (woord<>'s')
Write f,woord
Write 'Vertaling:'
Read woord
Write f,woord
Write 'Nederlands:'
Read woord
CLOSE(f)
Write
Write 'Druk op <ENTER> om het programma te verlaten'
Read

Mogelijke invoer

```
Geef de bestandsnaam voor de woordenlijst
woordenboek.txt
Typ de woorden in. $ is stoppen
Nederlands:
een
Vertaling:
un
Nederlands:
twee
Vertaling:
deux
Nederlands:
drie
Vertaling:
trois
Nederlands:
$
Druk op <ENTER> om het programma te verlaten
_
```

Mogelijke uitvoer

Het resultaat wordt bewaard in een bestand woordenboek.txt. Dit bestand kan je vinden in dezelfde map als waar je het pas-bestand bewaard hebt.

Als je het tekstbestand woordenboek.txt opent in bijv kladblok, krijg je dit resultaat:



3.3. Een niet-geformatteerd sequentieel bestand lezen

Voorbeeld: Je wil het bestand woordenboek.txt inlezen en tonen op het scherm.

Ook hier heb je een aantal speciale instructies nodig:

Instructie	Betekenis
var f : TEXT	definieer een bestand f van het type TEXT. Het bestand f kan zowel cijfers als letters bevatten.
ASSIGN(f , bestandsnaam)	koppel de bestandsvariabele f aan het eigenlijke bestand bestandsnaam . vb ASSIGN(f , 'woordenboek.txt') Bij het uittesten in Lazarus moet het bestand woordenboek.txt in dezelfde map als het pas-bestand staan.
RESET(f)	opent het bestand om te lezen
Read f , woord	leest het volgende item uit het bestand f en bewaart dit in de variabele woord
CLOSE(f)	sluit het bestand
EOF(f)	End Of File In Lazarus wordt EOF(f) TRUE op het moment dat je het laatste record leest.

3.4. Voorbeeld: Woordenboek tonen

Maak onderstaand psd aan. Test het nadien in Lazarus.

BestandTonen
var f: text var naam, nederlands, frans : string
Write 'Geef de bestandsnaam voor de woordenlijst'
Read naam
ASSIGN(f,naam)
RESET (f)
Write 'Nederlands':10,'Vertaling':10
While NOT EOF(f)
Read f, nederlands
Read f, frans
Write nederlands:10,frans:10
CLOSE(F)
Write
Write 'Druk op <ENTER> om het programma te verlaten'
Read

Mogelijke uitvoer

Als je als invoer het bestand woordenboek.txt kiest, krijg je volgend resultaat:

```
Geef de bestandsnaam voor de woordenlijst
woordenboek.txt
Nederlands Vertaling
    een      un
    twee    deux
    drie     trois
Druk op <ENTER> om het programma te verlaten
```

3.5. Samenvatting

Instructie	Betekenis
var f :text;	definieert f als een tekstbestand
ASSIGN(f , 'naam.txt') ASSIGN(f , naam)	koppelt de bestandsvariabele f en het eigenlijke bestand aan elkaar
RESET(f)	opent het bestand om te lezen
Read f , woord	leest een hele regel uit bestand f en stopt de gelezen waarde in de variabele woord
REWRITE(f)	opent het bestand om te schrijven
Write f , woord	schrijft inhoud van variabele woord naar bestand f
CLOSE(f)	sluit bestand f
EOF(f)	End Of File controleert of je aan het einde van het bestand bent gekomen

3.6. Voorbeeld: Gecombineerd

Maak een programma dat eerst naam en leeftijd inleest in een programma. De invoer moet stoppen als je de letter 's' ingeeft.

Nadien moet het aantal mensen en de gemiddelde leeftijd berekend worden.

Hoofdstuk 4. Geformatteerde bestanden

4.1. Geformatteerde sequentiële bestanden

In voorgaande voorbeelden hebben we gewerkt met bestanden waarbij elk gegeven op een nieuwe regel staat.

Wat als personeelsnummer, naam, adres, postcode, woonplaats en wedde van 1 persoon samen op één regel staan in je gegevensbestand?

In dit onderdeel bekijken we de **geformatteerde sequentiële bestanden**.

4.1.1. Een bestand maken

Eerst moet je de structuur van het bestand vastleggen, m.a.w. je moet beslissen welke gegevens je wil bewaren.

Voor elk gegeven moet je naam en type (getal, tekst, datum,...) bepalen.

Een personeelsrecord bevat de volgende gegevens:

- personeelsnummer
- naam
- adres
- postcode
- woonplaats
- wedde

1	Jan Janssen	Dorpsstraat 1	3600	Genk	1500	2	Piet Peeters	Dorpsplein 2	3500	Hasselt	1000	...
record1						record2						

4.1.2. Maken: opbouw

We moeten een type definiëren waarin we elk veld van het record opnemen en een type geven:

```
TYPE personeel=RECORD
  nr:integer;
  naam, adres, postcode, plaats:string;
  wedde:integer;
END;
```

Met deze regels definiëren we een type **Personeel**. Elk record van dit type bevat de velden **nr**, **naam**, **adres**, **postcode**, **plaats** en **wedde**.

Let hier goed op de volgorde! De gegevens in je bestand moeten in deze volgorde staan.

We hebben een bestandsvariabele van dit type nodig :

```
var f: FILE of personeel
```

We hebben twee variabelen van dit type nodig om alle gegevens tijdelijk in te bewaren:

```
var persoon, leeg: personeel
```

De gegevens van het personeelsbestand worden per werknemer ingelezen via het toetsenbord.

```
Read persoon.nr  
Read persoon.naam  
...
```

Per werknemer wordt één record gecreëerd en weggeschreven naar het personeelsbestand op het extern geheugen.

```
Write f, persoon
```

Write f, persoon wordt in Lazarus vertaald als *writeln(f,persoon)*. Dit moet je veranderen in **write(f, persoon);**

Nadat het record persoon weggeschreven is, moet die terug leeggemaakt worden:

```
persoon:=leeg
```

Belangrijk!

Als je je programma wil testen in Lazarus, moet je bovenaan volgende code toevoegen:

```
{ $mode objfpc }
```

Als je deze code vergeet, zal je programma niet werken!

4.1.3. Voorbeeld: Personeelsbestand maken

Maak onderstaand psd aan. Test het nadien in Lazarus.

PersoneelsgegevensToevoegen
{ \$mode objfpc }
<pre> type personeel=record nr:integer naam, adres, postcode, plaats: string wedde: integer end </pre>
<pre> var f: file of personeel var persoon, leeg: personeel var bestandsnaam: string </pre>
Write 'Geef de bestandsnaam'
Read bestandsnaam
ASSIGN(f, bestandsnaam)
REWRITE(f)
Write 'Geef personeelsnummer (0=stoppen)'
Read persoon.nr
While persoon.nr<>0
Write 'Naam: '
Read persoon.naam
Write 'Adres: '
Read persoon.adres
Write 'Postcode: '
Read persoon.postcode
Write 'Plaats: '
Read persoon.plaats
Write 'Wedde: '
Read persoon.wedde
Write f, persoon
persoon:=leeg
Write 'Geef personeelsnummer (0=stoppen)'
Read persoon.nr
CLOSE(f)
Write
Write 'Druk op <ENTER> om het programma te verlaten.'
Read

4.1.4. Een bestand lezen

Een sequentieel bestand wordt, zoals reeds eerder vermeld, vanaf het begin gelezen, d.w.z. record per record voor geformatteerde bestanden, totdat de juiste gegevens gevonden worden of totdat het EOF-teken gelezen wordt.

Op de volgende pagina's gaan we 3 voorbeelden bekijken:

- Je wil een lijst met enkel naam, adres, postcode en woonplaats.
- Je wil een lijst met naam en loon. Bovendien wil je het aantal personeelsleden en het totale loon.
- Je wil een lijst van de mensen die in Hasselt wonen.

Voor al deze voorbeelden vertrekken we vanuit het gegevensbestand `personeel.txt` dat je uit de webcursus kan downloaden.

4.1.5. Voorbeeld 1: Personeelslijst

Je baas wil een lijst met personeelsgegevens. Deze lijst toont enkel naam, adres, postcode en woonplaats.

PSD

De schrijfo opdracht `Read f, persoon` wordt in Lazarus vertaald als `readln(f, PERSON)`. Dit moet je veranderen in

```
read(f, persoon);
```

Personeelslijst

```
{ $mode objfpc }
```

```
type personeel=record
  nr:integer
  naam, adres, postcode, plaats: string
  wedde: integer
end
```

```
var f: file of personeel
var persoon: personeel
var bestandsnaam: string
```

```
Write 'Geef de bestandsnaam'
```

```
Read bestandsnaam
```

```
ASSIGN(f, bestandsnaam)
```

```
RESET(f)
```

```
While NOT(EOF(f))
```

```
  Read f, persoon
```

```
  Write persoon.naam:20,' ',persoon.adres:20,' ',persoon.postcode:4,' ',persoon.plaats:20
```

```
CLOSE(f)
```

```
Write
```

```
Write 'Druk op <ENTER> om het programma te verlaten.'
```

```
Read
```

Opdracht

Maak bovenstaand PSD en test het uit in Lazarus.

Resultaat:

```
Geef de bestandsnaam
personeel.txt
    Jan Janssen      Dorpsstraat 1    3600      Genk
    Piet Peters      Dorpsplein 1    3500      Hasselt
    Jos Jorissen     Kerkstraat 1    3700      Tongeren
    Hans Anders      Kerkplein 1    3500      Hasselt
    Tom Boonen       Bosstraat 1     3600      Genk
    Eddy Merckx      Zagerijstraat 1 3800      Sint-Truiden
    An Peeters       Bloemendreef 1  3500      Hasselt
    Rita Daemen      Bloemenstraat 1 3600      Genk
    Els Blijen       Groenstraat 1   2000      Antwerpen
    Sofie Janssen    Kleinestraat 1  1000      Brussel

Druk op <ENTER> om het programma te verlaten
```

4.1.6. Voorbeeld 2: Totaal loon

Nu wil je baas een lijst zodat hij kan zien wie wat verdient. Hij wil ook weten aan hoeveel mensen hij een loon uitbetaalt en hoeveel loon er in totaal betaald wordt.

PSD

SomWedde				
{ \$mode objfpc }				
type personeel=record nr:integer naam, adres, postcode, plaats: string wedde: integer end				
var f: file of personeel var persoon: personeel var bestandsnaam: string var aantal, som: integer				
Write 'Geef de bestandsnaam'				
Read bestandsnaam				
ASSIGN(f, bestandsnaam)				
RESET(f)				
aantal:=0				
som:=0				
While NOT(EOF(f))				
<table> <tr> <td>Read f, persoon</td></tr> <tr> <td>aantal:=aantal+1</td></tr> <tr> <td>som:=som+persoon.wedde</td></tr> <tr> <td>Write persoon.naam:20,' ',persoon.wedde:10</td></tr> </table>	Read f, persoon	aantal:=aantal+1	som:=som+persoon.wedde	Write persoon.naam:20,' ',persoon.wedde:10
Read f, persoon				
aantal:=aantal+1				
som:=som+persoon.wedde				
Write persoon.naam:20,' ',persoon.wedde:10				
CLOSE(f)				
Write 'Aantal personen: ', aantal				
Write 'Totaal loon: ', som				
Write				
Write 'Druk op <ENTER> om het programma te verlaten.'				
Read				

Opdracht

Maak bovenstaand PSD en test het nadien uit in Lazarus.

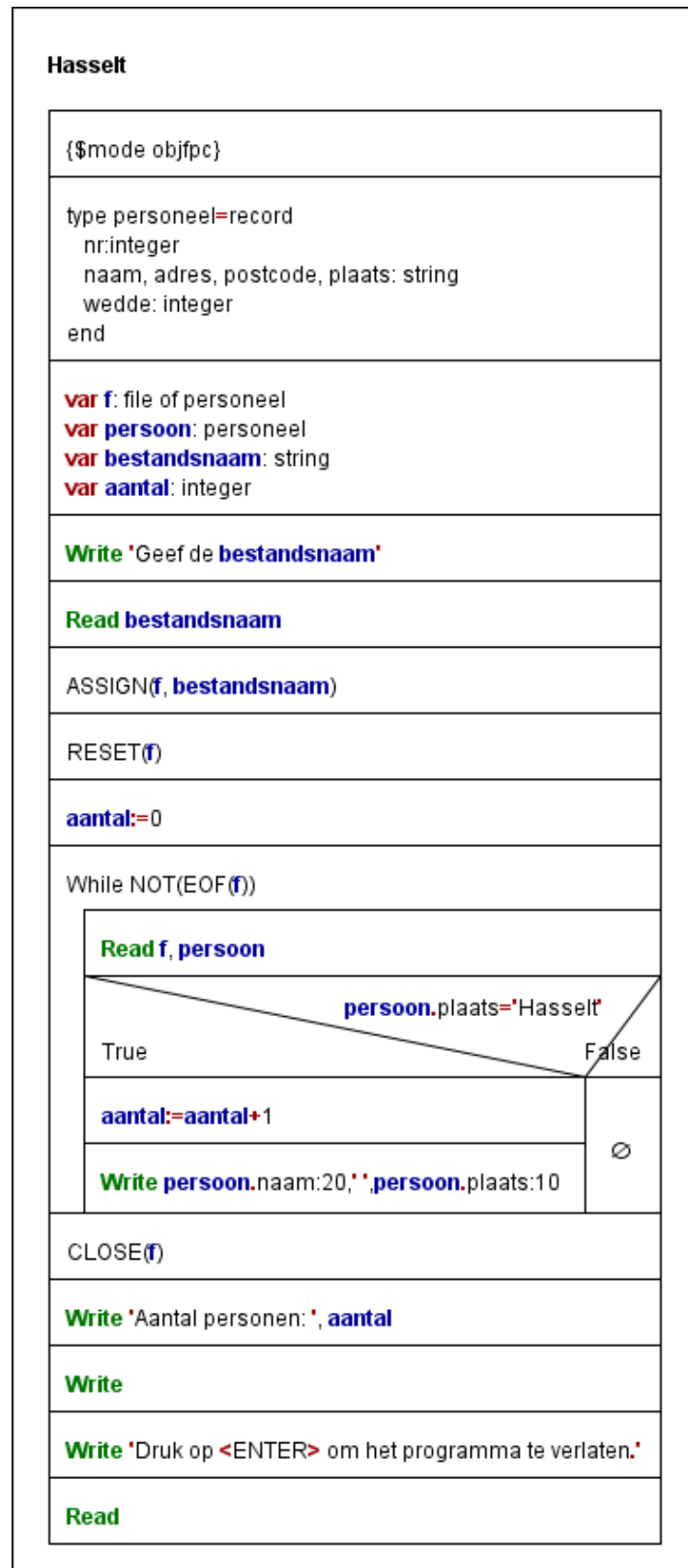
Resultaat:

```
Geef de bestandsnaam
../personeel.txt
      Jan Janssen      1000
      Piet Peters     1500
      Jos Jorissen     2000
      Hans Anders     3000
      Tom Boonen      2500
      Eddy Merckx      3000
      An Peeters      1500
      Rita Daemen     1600
      Els Blijen      2000
      Sofie Janssen    3000
Aantal personen: 10
totaal loon: 21100
Druk op <ENTER> om het programma te verlaten.
```

4.1.7. Voorbeeld 3: Hasselt

Tenslotte wil je baas nog een lijst van alle mensen die in Hasselt wonen. Je hebt alleen de naam en woonplaats nodig. Toon ook hoeveel personeelsleden er in Hasselt wonen.

PSD



Opdracht

Maak bovenstaand PSD aan en test het uit in Lazarus.

Resultaat:

```
Geef de bestandsnaam
../personeel.txt
      Piet Peters      Hasselt
      Hans Anders     Hasselt
      An Peeters       Hasselt
Aantal personen: 3
Druk op <ENTER> om het programma te verlaten.
```

4.1.8. Samenvatting

Instructie	Betekenis
{ \$mode objfpc }	Lazarus heeft deze code nodig om met een geformatteerd bestand te kunnen werken
Type xxx = record veldnamen : type; end;	definieert het type van een record welke velden? welk gegevenstype per veld?
var f : file of xxx ;	definieert een bestand van het gedefinieerde recordtype
var naam : xxx ;	definieert een variabele van het gedefinieerde type
ASSIGN(f , 'bestand.txt') ASSIGN(f , bestand)	koppelt de bestandsvariabele f en het eigenlijke bestand aan elkaar
Read naam.veldnaam	leest de waarde, ingegeven via het toetsenbord, en bewaart deze in een veld (veldnaam) van het record (naam)
Write naam.veldnaam	toont de waarde van het veld veldnaam op het scherm
RESET(f)	opent het bestand f om te lezen
Read f, naam	leest een record naam uit het bestand f Lazarus: read(f, naam) ipv readln(f, naam)
REWRITE(f)	opent het bestand f om te schrijven
Write f, naam	schrijft inhoud van variabele naam naar bestand f Het hele record wordt in één keer, niet veld per veld, weggeschreven naar het bestand. Lazarus: write(f, naam) ipv writeln(f, naam)

4.1.9. Een bestand aanpassen

Het **toevoegen** van records gebeurt steeds achteraan het bestand: eerst worden alle records van het oude bestand overgeschreven naar het nieuwe bestand, vervolgens worden de nieuwe records achteraan toegevoegd aan het nieuwe bestand.

Wat het **wijzigen** van sequentiële bestanden betreft, denken we terug aan een muziekcassette. Een liedje tussenvoegen tussen liedje 6 en 7 kan niet zomaar. Eerst neem je de eerste 6 liedjes over op een nieuwe cassette, vervolgens voeg je het extra liedje toe aan de nieuw cassette. Tenslotte voeg je de rest van de oude cassette toe aan de nieuwe cassette.

De **werkwijze** voor een muziekcassette kan je doortrekken naar een sequentieel bestand: Om een record van een personeelslid tussen te voegen, moet je eerst alle records tot waar het nieuwe record tussengevoegd moet worden naar een nieuw sequentieel bestand schrijven. Vervolgens schrijf je het nieuwe record naar het nieuwe sequentiële bestand. Tot slot schrijf je alle resterende records van het oude sequentiële bestand over naar het nieuwe sequentiële bestand.

Op dezelfde werkwijze **verwijder** je records: kopieer enkel de gewenste records naar een nieuw sequentieel bestand.

Telkens wordt het oude bestand verwijderd en blijft enkel het nieuwe bestand over als enig geldend bestand. Dit bestand kan je dan eventueel hernoemen en de naam geven van het oude bestand.

Meestal verzamelen we de **wijzigingen** in een apart mutatiebestand. Op geregelde tijdstippen worden het originele bestand en het mutatiebestand samengevoegd.

Door de records van een sequentieel bestand te **sorteren** op een sleutel, kan de verwerkingstijd vaak sterk gereduceerd worden.

Voorbeeld: Bij het verwerken van gewerkte uren moet elk record aangepast worden.

Door beide bestanden te sorteren op bijv personeelsnummer kan het verwerken veel sneller gebeuren.

Als de bestanden niet gesorteerd zijn, moet je voor elk personeelsnummer het volledige personeelsbestand doorlopen tot je het juiste nummer gevonden hebt.

Voor het sorteren en samenvoegen van sequentiële bestanden bestaan speciale algoritmen die we in deze cursus niet behandelen.

4.1.10. Voorbeeld: Personeelsbestand aanpassen

In ons bedrijf krijgt iedere werknemer jaarlijks een loonsverhoging van 25 €. Schrijf een programma om het personeelsbestand aan te passen. Je mag terug vertrekken van het bestand personeel.txt.

Denk eraan: Het bedrag van de jaarlijkse loonsverhoging moet makkelijk aan te passen zijn.

Loonsverhoging

```
{ $mode objfpc }
```

```
type personeel=record
  nr:integer
  naam, adres, postcode, plaats: string
  wedde: integer
end
```

```
var f, g: file of personeel
var persoon: personeel
var oudeBestandsnaam, nieuweBestandsnaam: string
var bedrag:integer
```

```
Write 'Geef de oude bestandsnaam'
```

```
Read oudeBestandsnaam
```

```
ASSIGN(f, oudeBestandsnaam)
```

```
RESET(f)
```

```
Write 'Geef de nieuwe bestandsnaam'
```

```
Read nieuweBestandsnaam
```

```
ASSIGN(g, nieuweBestandsnaam)
```

```
REWRITE(g)
```

```
Write 'Geef het bedrag van de loonsverhoging'
```

```
Read bedrag
```

```
While NOT(EOF(f))
```

```
  Read f,persoon
```

```
  persoon.wedde:=persoon.wedde+bedrag
```

```
  Write g,persoon
```

```
CLOSE(f)
```

```
CLOSE(g)
```

```
Write
```

```
Write 'Druk op <ENTER> om het programma te verlaten.'
```

```
Read
```

Hoofdstuk 5. Opgaven

5.1.1. Opgave 1 Gemiddelde berekenen

Lees getallen in uit een niet-geformatteerd sequentieel bestand. Bereken het gemiddelde van deze getallen.

Je kan het gegevensbestand uit de webcursus downloaden.

Het bestand getallen.txt bevat volgende gegevens

5
97
13
104
6
25
49

Uitvoer



```
Geef de bestandsnaam van de getallenlijst
getallen.txt
Het gemiddelde is 42.71
Druk op <ENTER> om het programma te verlaten
```

Hoofdstuk 6. Tot slot

6.1. Eindoefening

Bij deze cursus hoort ook een eindoefening. In die eindoefening komen de belangrijkste zaken van deze cursus terug aan bod.

Stuur een bericht aan je **coach** met als onderwerp "**Opdracht eindoefening Bestanden**".

6.2. Wat nu?

Je hebt nu het Programmatie logica - Bestanden afgerond.

Heb je het deel **Sorteren** nog niet doorgenomen, dan kan je dat nu doen.

Na het doornemen van **Basisstructuren, Procedures en functies** en **Arrays en strings** kan je voor de opleiding Programmatielogica een getuigschrift met resultaatsvermelding krijgen. Hiervoor leg je een eindtest af in één van de VDAB-opleidingscentra. De nodige informatie over de verschillende centra vind je terug in de module **Intro Informatica**.

Ligt je focus meer op **webapplicaties** en wil je graag al beginnen aan een concrete taal, dan zijn onze cursussen **Javascript** of **Inleiding tot PHP** misschien wel interessant voor jou.