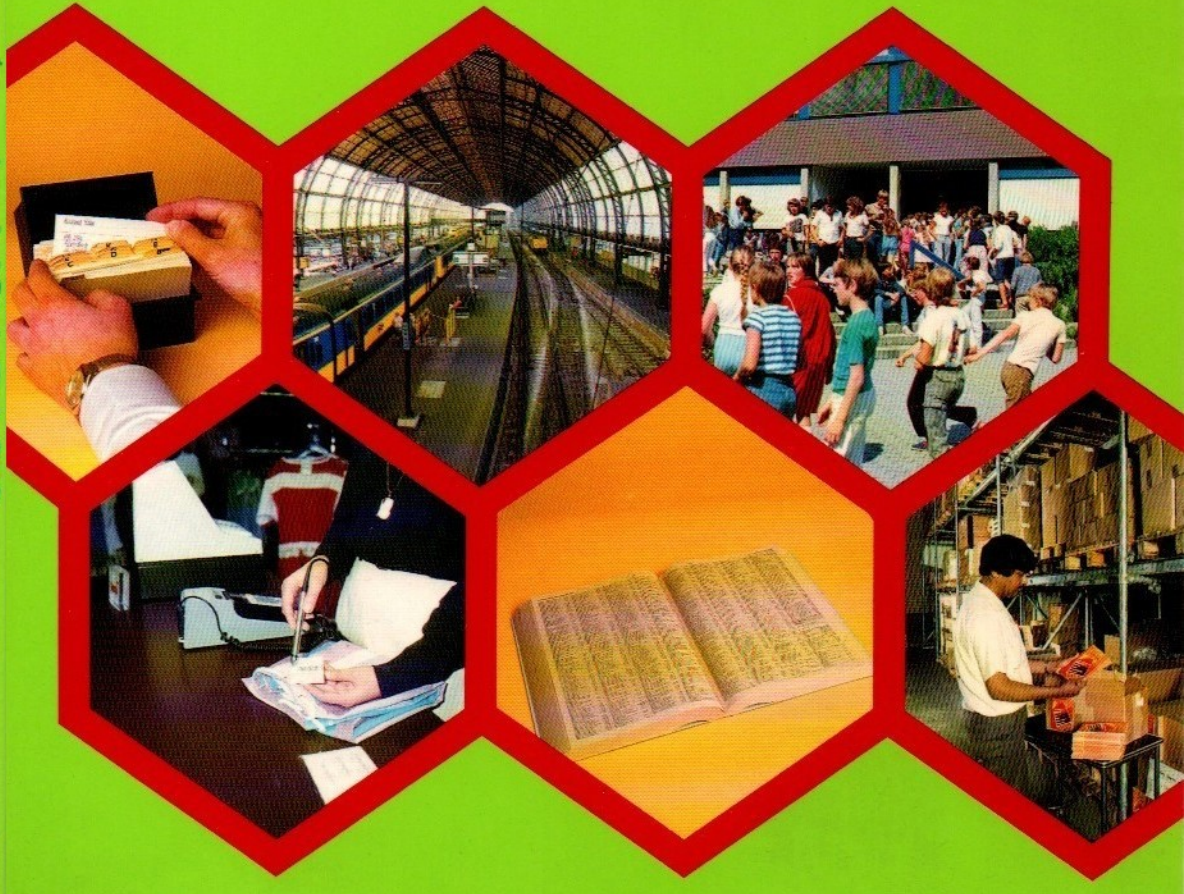
Werken met bestanden in MSX BASIC

een zelfstudiegids gebaseerd op MSX BASIC

LeRoy Finkel en Jerald R. Brown

Academic Service



Finkel en Brown

Werken met bestanden in MSX BASIC

een zelfstudiegids gebaseerd op MSX BASIC

WERKEN MET BESTANDEN IN MSX BASIC

Andere boeken in deze reeks:

Werken met bestanden in BASIC - Finkel & Brown
Werken met bestanden op de Commodore 64 - Fisher, Finkel & Brown
Werken met bestanden op de Apple - Finkel & Brown
Werken met bestanden in IBM- en GW-BASIC - Brown & Finkel

Andere boeken voor MSX-computers:

Programmeercursus MSX BASIC - Van Veen
40 grafische programma's in MSX BASIC - Sutter

Werken met bestanden in MSX BASIC

een zelfstudiegids gebaseerd op MSX BASIC

LeRoy Finkel en Jerald R. Brown

Academic Service

Oorspronkelijke titel:
Data File Programming in BASIC - A Self Teaching Guide
Verschenen bij: John Wiley & Sons, Inc., New York
Copyright © 1981 by John Wiley & Sons, Inc.
All rights reserved.
Authorized translation from English language edition published by John Wiley & Sons, Inc.
© Nederlandse vertaling: 1986, Academic Service.

Vertaling en bewerking: Ing. J.M. den Haan & P.J. Smith

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Finkel, Leroy

Werken met bestanden in MSX BASIC : een zelfstudiegids gebaseerd op MSX BASIC / Leroy Finkel, Jerald R. Brown ; vert. [uit het Engels door] J.M. den Haan ; [bew. door P.J. Smith]. - Den Haag : Academic Service
Vert. van: Data file programming in BASIC. - New York [etc.] : Wiley, 1981. - (A Self-teaching guide). Met index.
ISBN 90-6233-215-3
SISO 365.3 SVS 7.12.3 UDC 681.3.06+800.92 MSX BASIC UGI 650
Trefw.: programmeren (computer) / MSX BASIC (programmeertaal).

Uitgegeven door: Academic Service
Postbus 81
2870 AB Schoonhoven
Druk: Krips Repro Meppel
Bindwerk: Meeuwis, Amsterdam
Omslagontwerp: JAM Gauw
ISBN 90 6233 215-3

Niets uit deze uitgave mag worden verveelvoudigd en/of openbaar gemaakt door middel van druk, fotokopie, microfilm, geluidsband, elektronisch of op welke andere wijze ook en evenmin in een retrieval system worden opgeslagen zonder voorafgaande schriftelijke toestemming van de uitgever.

ENKELE OPMERKINGEN VOORAF

Dit boek is een zelfstudiegids waarin u zelf nogal actief wordt betrokken. U zult bijvoorbeeld regelmatig toetsvragen en opgaven voorgeschoteld krijgen. Hoewel de antwoorden daarop steeds direct worden gegeven, verdient het uiteraard aanbeveling eerst zelf een antwoord op te schrijven. Een eenvoudig stukje papier, met enige zelfdiscipline gehanteerd, zal in dit opzicht wonderen doen ...

Elk hoofdstuk wordt voorafgegaan door een opsomming van doelstellingen, en afgesloten met een afzonderlijke serie toetsvragen. Aan de hand hiervan kunt u zelf controleren in hoeverre u de desbetreffende leerstof hebt begrepen en actief kunt toepassen. Voor de meer gevorderde gebruiker biedt dit de mogelijkheid snel te kunnen vaststellen of een gegeven hoofdstuk voor hem of haar relevant is. Het hoofdstuk kan dan eventueel worden overgeslagen. Het boek wordt afgesloten met een eindtoets waarin uw algemeen begrip van de gehele materie wordt getest.

We zijn er van uitgegaan dat u bij het doornemen van dit boek de beschikking hebt over een computer (met cassette- of schijfgeheugen). Programmeren is namelijk een vaardigheid die alleen te leren is door 'doen'. U wordt dan ook aangemoedigd de gegeven taalelementen, constructies en programma's zelf te draaien, eventueel aan te passen, en aan de hand van de behandelde leerstof eigen toepassingen te maken. Om dat laatste gaat het tenslotte. We wensen u daarbij veel succes!

INHOUD

VOORWOORD

Een *bestand* is een verzameling bij elkaar behorende gegevens die op een bepaalde wijze in een extern geheugen van een computer – bijvoorbeeld cassette of floppy disc – wordt opgeslagen, en die door de computergebruiker als een eenheid kan worden aangesproken. Bij een *gegevens*bestand kunnen we bijvoorbeeld denken aan verzamelingen boekhoudkundige gegevens, fiscale gegevens, gegevens over bedrijfsvoering, verkoopcijfers, ledenlijsten, adreslijsten, statistische gegevens, enzovoort. Dergelijke gegevensverzamelingen kunnen ontstaan bij grote, uiterst professionele computertoepassingen, maar ook bij toepassingen die in de sfeer van kleinere bedrijfsadministraties, persoonlijk gebruik, hobby of onderwijs liggen. Vooral de opkomst van de microcomputer heeft dit soort toepassingen binnen het bereik van een groter publiek gebracht.

Dit boek heeft als doelstelling u te leren deze gegevensbestanden op te bouwen en te onderhouden op cassettes en/of schijfgeheugens zoals floppy discs of harde schijven. We doen dit vanuit de programmeertaal MSX BASIC. Als uitgangspunt nemen we aan dat u al het een en ander van die taal weet. U kunt bijvoorbeeld programmateksten (*listings*) lezen, en zelf eenvoudige programma's schrijven. U bent bijvoorbeeld ook al aardig vertrouwd met begrippen zoals stringvariabelen, stringfuncties en arrays. Voor alle zekerheid komen deze en soortgelijke basisbegrippen in het begin van dit boek wel aan de orde, maar het accent ligt daarbij meer op opfrissen, verdiepen en nieuwe toepassingswijzen dan op een uitvoerige onderwijskundige presentatie.

Met het werken met gegevensbestanden in BASIC of welke taal dan ook hoeft u geen enkele ervaring te hebben. Daarover gaat – zoals gezegd – dit boek. Stap voor stap, en aan de hand van praktische voorbeelden, wordt u in deze materie ingewijd. U zult daarin actief betrokken worden door zelf programma's en/of programmadelen te schrijven. Mocht u al enige ervaring hebben met gegevensbestanden, dan zult u met dit boek uw kennis waarschijnlijk kunnen verdiepen.

WOORD VOORAF BIJ DE NEDERLANDSTALIGE UITGAVE

Van geen enkele programmeertaal bestaan zoveel dialecten als van de programmeertaal BASIC. Veel fabrikanten van computers ontwikkelden hun eigen BASIC-versie, met als gevolg dat bijna geen enkel programma op een computer van 'het andere merk' draait. Enkele Japanse firma's besloten daarin verandering te brengen. Zij brachten de eerste MSX-computers op de markt. Amerikaanse en Europese computergiganten, waaronder Philips, volgden hun voorbeeld. Randapparatuur en programma's zijn volkomen uitwisselbaar, een eis die de homecomputermarkt terecht stelt.

Nu geven de meeste boeken over BASIC niet veel meer dan een wat meer uitgebreide handleiding bij de computer. Ze geven een alfabetische opsomming van de toegestane 'statements' en functies, soms toegelicht met kleine voorbeeldprogramma's. Hoe met bestanden moet worden gewerkt blijkt in de meeste gevallen totaal niet.

Moge dit boek daarom een welkome aanvulling zijn op uw bibliotheek. Het is een bewerking van het bekende boek van Finkel en Brown: "Data File Programming in BASIC". Het oorspronkelijke boek is vooral geschikt voor oudere Microsoft BASIC-versies, geïmplementeerd op veel microcomputers, werkende onder de besturingssystemen CP/M en MS-DOS. Microsoft, een der grootste en belangrijkste softwarehuizen ter wereld, is ook de ontwerper van de nieuwe standaard voor homecomputers MSX BASIC, waarmee in dit boek gewerkt wordt.

Het boek veronderstelt min of meer enige kennis over het schrijven van eenvoudige programma's in MSX BASIC. Het boek "Programmeercursus MSX BASIC", eveneens uitgegeven door Academic Service, vormt hiervoor een prima basis.

J.M. den Haan,
P.J. Smith.

1 DUIDELIJKHEID, LEESBAARHEID EN LOGISCHE OPBOUW VAN BASIC PROGRAMMA'S

1.0 DOELSTELLINGEN

Na bestudering van dit hoofdstuk is het de bedoeling dat u:
- kunt aangeven hoe een programma volgens de 'top-to-bottom' methode geschreven moet worden;
- met behulp van REMARK-opdrachten een inleidende module kunt schrijven;
- zes manieren kunt aangeven voor de verzorging van programmatekst en -uitvoer;
- zeven methoden kunt aangeven om geheugenruimte te besparen.

1.1 INLEIDING

In dit boek behandelen we het werken met gegevensbestanden in de programmeertaal MSX BASIC. We gaan ervan uit dat u al het een en ander van de taal weet. U hebt bijvoorbeeld een beginnerscursus gevolgd of een inleidend boek over BASIC gelezen, waardoor u in ieder geval in staat bent programmateksten (Engels: listings) te lezen en zelf eenvoudige programma's te schrijven. Van gegevensbestanden, of het werken daarmee in BASIC of welke taal dan ook, hoeft u op dit moment nog niets te weten. Daarover gaat – zoals gezegd – dit boek.

De technieken die hierbij aan de orde komen zijn gebaseerd op bepaalde versies van BASIC. Over de verschillende BASIC-versies straks meer. We vermelden hier alvast dat we ons in eerste instantie hebben gebaseerd op zogenaamd MICROSOFT EXTENDED BASIC, zoals die op vele MSX-computers wordt toegepast.

Alle voorbeeldprogramma's die in dit boek voorkomen zijn op een Philips VG 8020 MSX-computer daadwerkelijk uitgetest. Werkt u met andere versies van BASIC, dan zullen de daarbij te gebruiken opdrachten en technieken vergelijkbaar, maar niet in alle gevallen dezelfde zijn. U zult dan het meeste profijt van dit boek trekken door het samen met de handleiding van uw computersysteem te gebruiken.

Een van de eerste dingen waarmee we ons gaan bezighouden is de opbouw en duidelijkheid van het programma. Dat lijkt misschien overbodig, maar: programma's die met gegevensbestanden werken kunnen behoorlijk lang en ingewikkeld worden. En: hoe langer en ingewikkelder, des te groter de kans op fouten. Door een goede programmaopbouw zal de kans op fouten kleiner zijn. Mochten ze onverhoopt tóch voorkomen, dan zullen ze door een goede opbouw en uiterlijke verzorging van het programma gemakkelijker op te sporen en te verwijderen zijn. Het 'mes' snijdt dus aan twee kanten. Maar voordat we hierop verder ingaan eerst iets over de verschillende versies van BASIC.

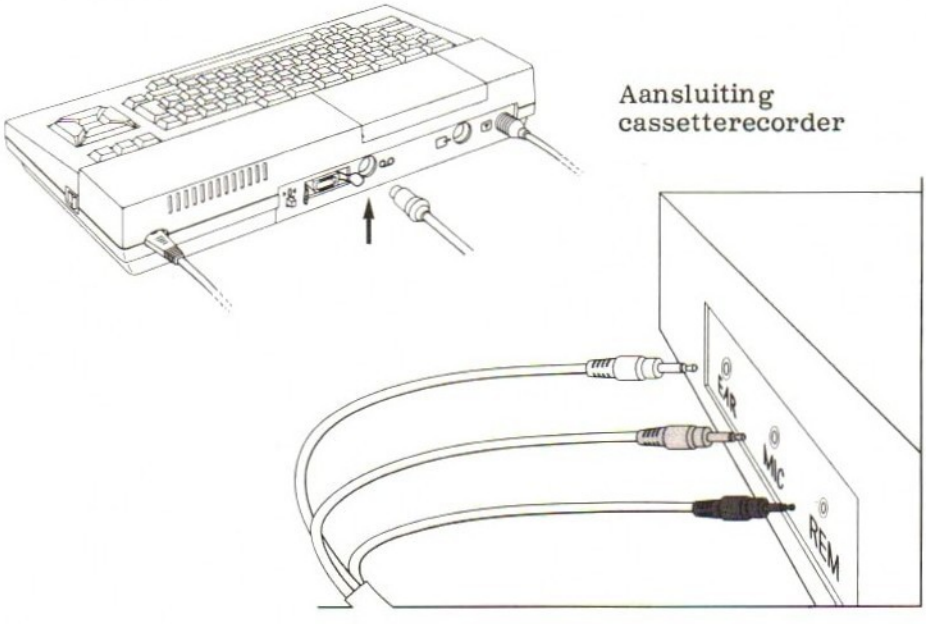
1.2 DE TAAL BASIC

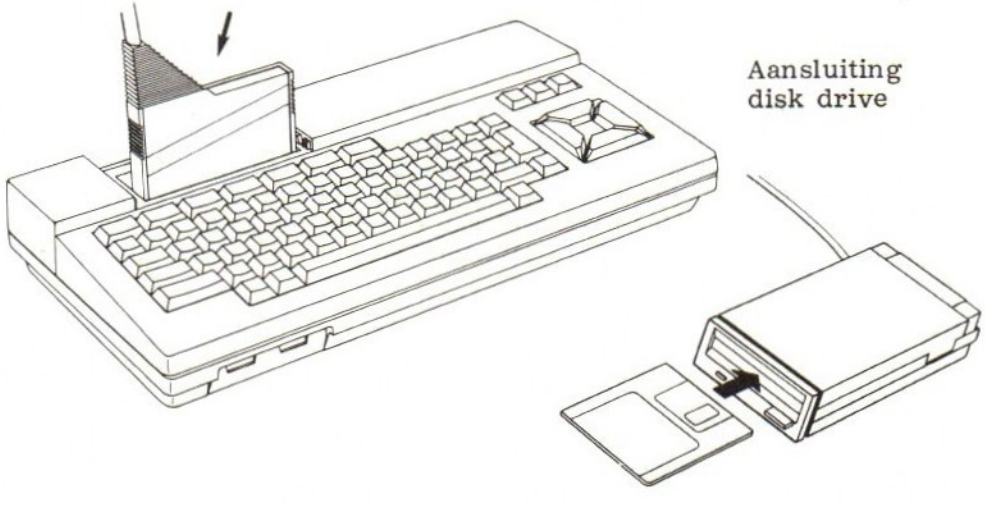
BASIC werd in het begin van de jaren zestig ontwikkeld op het Dartmouth College in de Verenigde Staten. De taal was bedoeld voor studenten met weinig of geen kennis op het gebied van computers en wiskunde. In het oorspronkelijke BASIC waren daarom slechts die taalelementen opgenomen die een beginnende programmeur nodig zou hebben. Toen andere onderwijsinstellingen, computerfabrikanten en rekencentra ook BASIC gingen gebruiken, werden vaak extra elementen toegevoegd om de taal voor het eigen toepassingsgebied geschikter te maken. Zo kwamen in korte tijd vele BASIC 'dialecten' tot stand, zoals bijvoorbeeld Extended BASIC, Expanded BASIC, SUPERBASIC, XBASIC en BASIC PLUS.

In 1978 trachtte het American National Standards Institute (ANSI) in deze Babylonische spraakverwarring enige orde te scheppen. Er werd een industriële norm voor BASIC vastgelegd – een gekortwiekte vorm van de meest gangbare versies, waaraan alle BASIC-systemen minimaal moesten voldoen. Men sprak daarom van 'minimal BASIC'. Ondanks die ANSI-norm bestaat vandaag-de-dag nog steeds een groot aantal BASIC-versies. De meeste daarvan lijken veel op elkaar, maar hebben toch hun eigen kenmerken en (eigen) aardigheden.

Op het gebied van microcomputers werden de meest gebruikte versies van BASIC ontwikkeld door de Microsoft Company. Daarbij wordt in het algemeen van MICROSOFT BASIC gesproken.

De programma's die in de hoofdtekst van dit boek voorkomen zijn gebaseerd op MICROSOFT EXTENDED BASIC (MSX). Het paraderen met de toeters en bellen van MSX BASIC is bewust vermeden. Liever hebben we ons willen richten op het presenteren van gemakkelijk te begrijpen programma's die zonder meer op alle MSX-computers kunnen draaien.





Bron: Philips MSX thuiscomputer VG 8080, gebruiksaanwijzing.

1.3 TAALGEBRUIK

Door dit eenvoudige taalgebruik worden de programma's leesbaarder en zijn ook voor anderen gemakkelijker te begrijpen. Ook zijn fouten gemakkelijker op te sporen en te verbeteren.

De gebruikte MSX BASIC-opdrachten zijn allemaal standaard. Zij 'doen' het op elke MSX-computer, inclusief de nieuwe MSX 2-computers. Dit geldt natuurlijk ook voor de opdrachten, waarmee met cassetteband en diskette gewerkt kan worden. Diverse 'basis'-MSX BASIC-opdrachten worden slechts kort in hoofdstuk 2 toegelicht. Wie hierover een uitgebreidere inleiding nodig heeft verwijzen we naar het boek "Programmeercursus MSX BASIC", uitgegeven door Academic Service. In dit boek vindt u ook een inleidend hoofdstuk over het werken met sequentiële bestanden op cassetteband en diskette.

De programmaregels in dit boek zijn veelal 'breder' dan 40 tekens. Werkt u met een MSX 2-computer dan kunt u de 80-kolommenstand inschakelen. MSX 1-gebruikers raden we aan het scherm in te stellen op 40 tekens (WIDTH 40 in SCREEN 0) en de opdrachten over meerdere beeldschermregels te verdelen.

1.4 LEESBARE PROGRAMMA'S SCHRIJVEN

Mede door eenvoudig taalgebruik zijn de voorbeeldprogramma's van dit boek betrekkelijk gemakkelijk te lezen en te begrijpen. De BASIC-code is simpel, rechttoe-rechtaan. Fraaie trucs, bijzondere taalconstructies en dergelijke ontbreken. De programma's zijn als het ware met het oog op de (menselijke) lezer, in plaats van de computer, tot stand gekomen.

Goed leesbare BASIC-programma's schrijven vereist dat men van te voren nadenkt over een logische programmaopbouw, en dat men ook aan het uiterlijk van de programmatekst de nodige aandacht besteedt. Door deze combinatie van goede opbouw en uiterlijke verzorging verklaart het programma min of meer zichzelf. De structuur wordt doorzichtiger, en het ontluizen* en het aanpassen aan andere computersystemen gemakkelijker.

* Onder 'ontluizen' verstaan we het verwijderen van fouten ('luizen') uit een programma. De term is afgeleid van het Engels-Amerikaans jargon (bug = luis, ongedierte; to debug = ontluizen).

De programmeerstijl waarmee we in dit boek werken is voor een belangrijk deel gebaseerd op het zogenaamde gestructureerd programmeren, hoewel dit niet in alle opzichten streng wordt doorgevoerd. Er wordt vooral gewerkt met *modules*: kleine, logische programma-onderdelen, die elk een bepaalde hoofdfunctie of duidelijk afgebakende activiteit verzorgen. We maken ook gebruik van technieken die al jaren als goede programmeerpraktijk zijn aanvaard, maar om de een of andere reden de laatste tijd veelal in onbruik zijn geraakt.

De meeste suggesties die we doen hebben *niet* tot gevolg dat er geheugenruimte wordt bespaard, of dat de executietijd (Engels: runtime = de computertijd nodig voor het uitvoeren van het programma) korter wordt. Integendeel. We zijn primair geïnteresseerd in leesbaarheid, desnoods ten koste van zaken zoals geheugengebruikt en executietijd. Aan het einde van dit hoofdstuk zullen we echter wel laten zien hoe hieraan tegemoet gekomen kan worden.

1.5 DE TOP-TO-BOTTOM BENADERING

Bij het opbouwen van een programma gaan we uit van de hoofdactiviteiten die moeten plaatsvinden. Deze zouden kunnen bestaan uit enkele, of wellicht alle van de volgende elementen:
- gegevensinvoer (Engels: data entry);
- gegevensanalyse (Engels: data analysis);
- berekening (Engels: computation);
- bestandmutatie (Engels: file update);
- tekstverzorging (Engels: editing);
- uitvoerpresentatie (Engels: report generation).

Door het programma te verdelen in modules die elk één van deze activiteiten verzorgen, en de modules in volgorde uit te voeren – eerst module 1, dan module 2, enz. – wordt het programma gemakkelijk te volgen. We noemen dit een 'top-to-bottom' (= van boven naar beneden) opbouw.

Modules kunnen op hun beurt worden onderverdeeld in kleinere 'blokken', die elk een deelactiviteit of berekening verzorgen. De omvang van zo'n programmablok binnen een module hangt af van de uit te voeren taak, maar is ook een kwestie van smaak. Van persoon tot persoon zal dit verschillen, en wellicht ook van programma tot programma.

Bouw programma's modulair op en ga daarbij uit van een top-to-bottom benadering.

1.6 REMARK-OPDRACHTEN

Programmamodules en -blokken kunnen voor de duidelijkheid van elkaar gescheiden worden door REMARK-opdrachten of lege programmaregels. Sommige BASIC-systemen accepteren geen lege, genummerde regels, en verwijderen die bij het listen (= afdrukken) van het programma. In dat geval kan de functie van een lege regel worden vervuld door een regelnummer gevolgd door REMARK of REM, bijvoorbeeld:

```
150  REM
```

Soms kan worden volstaan met een regelnummer gevolgd door een apostrof (') of een dubbele punt als alternatief voor REMARK of REM. Test dit op uw eigen systeem uit, maar denk hierbij ook aan de overdraagbaarheid.

voorbeelden:

```
100 REM *** GEGEVENSINVOERMODULE ***
110 REM     LEES GEGEVENS VAN DATA-OPDRACHTEN 9000-9090
120 '
130 :
140 REM
.
.
.
190 '
200 REM *** REKENMODULE ***
```

Laat elk programmamodule of -blok beginnen met REMARK's waarin de uit te voeren activiteit wordt vermeld, en zet deze af van de daarop volgende programmatekst met behulp van een lege regel of een 'lege' REMARK (zie bijvoorbeeld regel 190 hierboven). Wees daarbij consequent: gebruik òf een volledig lege regel, òf REM, òf REMARK, òf een apostrof of dubbele punt, maar niet verschillende mogelijkheden door elkaar.

Om REMARK-opdrachten die commentaar of toelichting bevatten te onderscheiden van andere REMARK's gebruiken sommigen het sterretje (Engels: asterisk), zoals in regel 100 hierboven. Weer anderen slaan een aantal posities over voordat ze met het eigenlijke commentaar beginnen (regel 110).

REMARK's kunnen op dezelfde regel als andere BASIC-code worden geplaatst door zogenaamde multi-opdrachtregels (Engels: multiple statement lines) toe te passen. De REMARK moet dan wel de laatste opdracht op de regel zijn (waarom?). Dankzij zo'n multi-opdracht-

regel kan de desbetreffende opdracht ter plaatse worden toegelicht. Vaak wordt daarbij enige ruimte tussen de 'echte' opdracht en de bijbehorende REMARK gelaten.

voorbeeld:

```
220 LET S(X) = S(X) + N:    REM *** SOMMEER EENHEDEN IN ARRAY S
```

In het algemeen zijn goed leesbare programma's doorspekt met toelichtende REMARK's. Maar ook hier: overdaad schaadt. De opdracht:

```
123 LET C = A + B
```

is bijvoorbeeld zo duidelijk dat we met een toelichtende REMARK een open deur zouden intrappen. Voegt een REMARK geen zinnige informatie toe, dan kan hij beter achterwege blijven.

1.7 GOTO-OPDRACHTEN

De onvoorwaardelijke GOTO (bijvoorbeeld GOTO 210) is misschien wel de meest aangevochten opdracht in BASIC, of in welke taal dan ook. Puristen zouden het gebruik daarvan onder alle omstandigheden vermijden. Realistischer is het ons inziens alle GOTO's (en de daaraan verwante GOSUB's) naar verderop gelegen punten in het programma te laten verwijzen, en terugverwijzingen dus uit te sluiten. Dit zou overigens volledig in overeenstemming zijn met de 'top-to-bottom' opzet. Hetzelfde geldt in principe ook voor het gebruik van 'IF...THEN' opdrachten, hoewel er voor de hand liggende uitzonderingen zullen zijn.

voorbeelden:

```
140 GOTO 210
150 IF X < Y THEN 800
160 GOSUB 8000
```

Nog een suggestie: laat een GOTO, GOSUB of IF...THEN nooit verwijzen naar een REMARK-opdracht. Komt u op een gegeven moment geheugenruimte tekort, dan is het namelijk niet onwaarschijnlijk dat u enkele REMARK's zult moeten opofferen om geheugenruimte te winnen. Dit zou tot gevolg hebben dat alle betrokken GOTO's enz. aangepast moeten worden. Een riskante aangelegenheid!

voorbeeld:

```
150 GOTO 300
.
.                                          } zo niet
.
300 REM *** GEGEVENSINVOER ***
310 LINE INPUT "NAAM: "; N$
```

```
150 GOTO 300
.
.                                          } zo wel
.
299 REM *** GEGEVENSINVOER ***
300 LINE INPUT "NAAM: "; N$
```

1.8 DE INLEIDENDE MODULE

De eerste module kan uit verschillende 'rubrieken' bestaan. Zo'n rubriek kan bijvoorbeeld een korte omschrijving van het programma zijn, of kan bestaan uit een gebruiksaanwijzing, een lijst van alle voorkomende variabelen en hun functie, of de initialisatie* van constanten, variabelen en arrays. Regelnummers worden gekozen uit de serie 100 t/m 199 of 1000 t/m 1999. We gaan nu nader op deze rubrieken in.

Omschrijving van het programma

Deze rubriek begint met een REMARK die de naam van het programma aangeeft. Gebruik daarvoor bij voorkeur niet zomaar een willekeurige naam, maar een naam die iets weergeeft van wat het programma doet. Vermeld vervolgens de naam van de auteur of programmeur, en de datum. Daarna kunt u een REMARK opnemen ten behoeve van anderen die wellicht van uw programma gebruik zullen maken, en waarin u aangeeft voor welk computer- en/of softwaresysteem het programma werd ontworpen. Ook latere veranderingen, aanpassingen en dergelijke kunnen hier worden vermeld. De rubriek kan afgesloten worden met een korte uiteenzetting van de doelstelling van het programma.

* Initialiseren is het toekennen van een beginwaarde aan een programmagrootheid, bijvoorbeeld een variabele. Verder in deze paragraaf komen we hierop terug.

voorbeeld:

```
100 REM *** SUBSYSTEEM LOONBEREKENING ***
110 REM
120 REM     AUTEURSRECHT BLITS SOFTWARE B.V., DEN HAAG, 02/84
130 REM     HP 2000 BASIC
140 REM
150 REM     AANGEPAST VOOR MSX BASIC
160 REM     DOOR PIET BITJESVRETER, 08/84
170 REM
```

Gebruiksaanwijzing

In deze rubriek kunt u een toelichting opnemen over het gebruik van het programma. Eventueel kunt u die toelichting door het programma zelf laten produceren op het moment dat het gestart wordt. Dit kan ook nog facultatief – op verzoek van de gebruiker, oftewel als 'optie' – gebeuren. De gebruiker krijgt dan de vraag voorgeschoteld of hij de gebruiksaanwijzing wil inzien, en kan de desbetreffende tekst dan door een simpel 'ja' of 'neen' in te toetsen te voorschijn toveren. Is de tekst te uitgebreid, plaats dan alleen de vraag zelf in de inleidende module. De bijbehorende PRINT-opdrachten kunnen dan in een aparte module (subroutine) aan het einde van het programma worden opgenomen.

Dit 'achteraan' plaatsen van de tekst kan natuurlijk ook worden toegepast als de tekst in de vorm van REMARK's in het programma wordt verwerkt. In beide gevallen voorkomt men dat bij een listing van het programma eerst ellenlange verhalen te voorschijn komen, wat vooral tijdens de ontwikkeling van een programma lastig kan zijn.

vervolg voorgaand voorbeeld:

```
180 REM     DIT PROGRAMMA MAAKT EEN LOONLIJST, GEBASEERD OP
190 REM     GEGEVENS DIE DOOR DE GEBRUIKER ZELF WORDEN INGEVOERD
200 REM
210 LINE INPUT "HEBT U GEBRUIKSAANWIJZINGEN NODIG (J/N): "; A$
220 IF A$ = "J" THEN GOSUB 800
230 REM
```

Programmagrootheden

De gebruiksaanwijzingen, althans het oproepen daarvan als het een subroutine betreft, kunnen gevolgd worden door REMARK-opdrachten waarin de gebruikte grootheden zoals variabelen, stringvariabelen, arrays, constanten en bestanden worden opgesomd en ver-

klaard. Deze informatie is vooral van belang bij het later aanpassen van het programma, en wordt meestal ingebouwd als het eigenlijke programmeerwerk klaar is. Pas dan zijn alle variabelen, arrays, enz. definitief bekend.

Een opmerking over het gebruik van constanten is hier op zijn plaats. Men kan soms de behoefte hebben bij opeenvolgende runs (= uitvoeringen) van een programma een constante te veranderen. De constante is dus slechts tijdens een programmarun werkelijk constant. In zo'n geval doet men er beter aan daarvoor een variabele te gebruiken. Bij de opeenvolgende runs hoeft dan slechts één (initialisatie-)opdracht veranderd te worden.

Overigens kan het nuttig zijn – vooral ten behoeve van deze rubriek – tijdens het programmeren regelmatig aantekeningen te maken, zodat belangrijke details die u te binnen schieten, niet verloren gaan. Als het eigenlijke programma klaar en uitgetest is, neem het geheel dan nog eens door, en werk aan de hand van uw aantekeningen de REMARK's bij.

voorbeeld:

```
220 REM   GEBRUIKTE VARIABELEN:
230 REM
240 REM   BL        = BRUTOLOON
250 REM   NL        = NETTOLOON
260 REM   LB        = LOONBELASTING
270 REM   IN        = INHOUDING
280 REM   AO        = PREMIE AOW
290 REM   AW        = PREMIE AWW
300 REM   X,Y,Z     = LUSVARIABELEN
310 REM   T(X)      = ARRAY VOOR GEWERKTE UREN
320 REM   NS        = ARRAY VOOR NAMEN (20)
330 REM   SNS       = ARRAY VOOR SALARISNUMMERS (5)
340 REM
350 REM   CONSTANTEN:
360 REM
370       LET C1 = 0.112:   REM *** PERCENTAGE AOW
380       LET C2 = 0.015:   REM *** INHOUDINGSPERCENTAGE
390 REM
400 REM   GEBRUIKTE BESTANDEN:
410 REM
420 REM   TB1       = TABEL LOONBELASTING
430 REM   TB2       = TABEL LOONSCHALEN
440 REM
```

Merk op: 1. hoe de lengte van strings wordt aangegeven in regels 320 en 330;
2. het gebruik van REMARK's in de multi-opdrachtregels 370 en 380 (multi-opdrachtregels zijn behandeld in § 1.6).

Initialisatie

De inleidende module wordt afgesloten met een initialisatierubriek. Hier worden ook de dimensies van één- en meerdimensionale arrays vermeld, ook als dit in MSX BASIC niet strikt noodzakelijk is.

In MSX BASIC krijgen niet-geïnitialiseerde numerieke variabelen automatische de waarde nul. Maak ter wille van de duidelijkheid en de overdraagbaarheid ook hier geen gebruik van, en geef zulke variabelen expliciet de waarde nul met behulp van een LET-opdracht.

Eventuele user-defined functies (= functies die door de programmeur zelf worden gedefinieerd) worden ook in het initialisatiedeel – dus vóór het eigenlijke gebruik – opgenomen.

vervolg voorgaand voorbeeld:

```
450 REM   INITIALISATIES
460 REM
470       DIM H(7), N$(30)
480       S = 0
490 REM
500
```

1.9 ANDERE MODULES

Na de inleidende module komen de andere modules, eventueel op hun beurt gevolgd door subroutines, DATA-opdrachten en PRINT USING format IMAGE-opdrachten*. Met behulp van al dan niet lege REMARK opdrachten worden alle modules van elkaar gescheiden en van een kop voorzien. De modules zelf kunnen eventueel worden opgedeeld in blokken, die onderling ook weer gescheiden worden door lege REMARK's.

Gegevensinvoer

In de meeste gevallen zal de tweede module bestemd zijn voor de invoer van gegevens. Die gegevens kunnen door de gebruiker rechtstreeks via het toetsenbord worden ingevoerd. Ze kunnen ook afkomstig zijn van DATA-opdrachten, bestanden (files) of andere invoermedia. In hoofdstuk 3 gaan we daar verder op in. Daarbij

* Dat wil zeggen opdrachten die betrekking hebben op de opmaak of layout van de uitvoer.

komen ook de zogenaamde *gegevensverificatie* en *-integriteit* ter sprake, dat wil zeggen hoe de aangeboden invoer op geldigheid kan worden gecontroleerd.

Voorlopig is het belangrijk er rekening mee te houden dat ook onervaren gebruikers met uw programma kunnen werken. Programma's moeten zoveel mogelijk 'foolproof' zijn, oftewel 'gekbestendig'. Gebruik mede daarom altijd even een 'aanzetje' (Engels: prompt), compleet met voorbeeld, als een gebruiker iets moet intypen.

voorbeeld:

```
240 INPUT "VOER DATUM IN (DD/MM/JJ): "; D$
```

DATA-opdrachten

DATA-opdrachten kunnen het beste collectief aan het einde van het programma worden geplaatst. Maak daarbij gebruik van REMARK's om de aard van de gegevens en hun volgorde binnen de DATA-opdrachten vast te leggen.

voorbeeld:

```
9400 REM   ARRAYGEGEVENS: JUISTE ANTWOORDEN IN VOLGORDE VAN VRAAG.
9410 REM   WAARDEBEREIK 1-5
9420 REM
9430       DATA 4,5,1,3,2,1,1,4,4,5
9440 REM
9450 REM   ANTWOORDEN VAN RESPONDENTEN: PER DATA-OPDRACHT EEN
9460 REM   IDENTIFICATIENUMMER GEVOLGD DOOR 10 ANTWOORDEN, 1-5
9470 REM
9480       DATA 17642, 4,5,1,3,2,2,1,4,4,4
9490       DATA 98126, 3,5,2,3,2,1,5,4,5,2
.
.
.
```

DATA-opdrachten kunnen eventueel als afzonderlijke module aan het einde van het programma worden ondergebracht. Voorafgaande modules verzorgen dan de berekeningen, het gegevenstransport (data handling), het lezen en beschrijven van bestanden en de uitvoerpresentatie.

Uitvoer

Met uitzondering van foutmeldingen kan alle uitvoer eveneens vanuit één module worden verzorgd. Alle PRINT- en 'PRINT USING'-opdrachten zijn dan op één plaats te vinden. Eventueel kunnen ook verschillende 'print' modules worden gebruikt. Deze opzet heeft het voordeel dat latere aanpassingen, bijvoorbeeld door wijzigingen in formulieren, gemakkelijker uit te voeren zijn. Berekeningen – – behalve triviale gevallen – horen in deze module(s) niet thuis, maar moeten vóór de uitvoering van de printmodule(s) plaatsvinden. Hierdoor wordt weliswaar een zwaardere wissel getrokken op het gebruik van variabelen en arrays – gegevens moeten immers langer vastgehouden worden – maar omdat alles zijn eigen plaats heeft, wordt het programma doorzichtiger en gemakkelijker te 'ontluizen'.

1.10 SUBROUTINES

Goed ontworpen modules hebben in principe één ingang en één uitgang, te weten aan het begin en aan het einde van de module. Ze worden in principe na elkaar uitgevoerd. Daarop is één uitzondering: de aanroep van een subroutine tijdens de uitvoering van de module.

voorbeeld:

```
290 :
300 REM *** REKENMODULE ***
310 :
320         LET T =  (V * X) / Q
330         LET T9 = T9 + T
340 :
350         GOSUB 800
360 :
400 REM *** UITVOERMODULE ***
410 :
```

Na uitvoering van de subroutine wordt teruggekeerd naar de volgende opdracht van de aanroepende module.

Technisch bekeken zijn subroutines alleen nodig om het coderen van reeksen identieke opdrachten te voorkomen. Een subroutine zou dan ook vanuit verschillende plaatsen in het programma opgeroepen moeten worden. Subroutines hebben echter bovendien het voordeel – ook al worden ze maar één keer aangeroepen – dat ze de leesbaarheid en de doorzichtigheid van een programma ten goede komen.

Het gebruik van subroutines is dus wenselijk, maar moet niet overdreven worden. Sommigen gaan zover, dat het hele hoofdprogramma uit niets anders bestaat dan GOSUB-opdrachten die reeksen verderop gelegen subroutines aanroepen. Naar onze mening is het verstandiger te streven naar de gulden middenweg tussen beide extremen: subroutineloze programma's enerzijds, en programma's die uit niets anders dan subroutines bestaan anderzijds. Als criterium kan worden gehanteerd dat het gebruik van een subroutine een duidelijker besturingsstroom (= de uitvoeringsvolgorde van opdrachten) en een verbetering van de leesbaarheid tot gevolg moet hebben. De leesbaarheid wordt overigens ook hier gediend door het gebruik van REMARK's om de routine snel te kunnen herkennen, en ze van de rest van de programmatekst af te zetten.

Wat de fysieke plaats van subroutines betreft zijn er twee mogelijkheden. Ze kunnen of direct na de aanroepende module worden geplaatst (behalve als er verschillende aanroepen zijn), of in een gemeenschappelijk deel aan het einde van het programma.

voorbeeld: alternatief 1

```
300 REM *** REKENMODULE ***
310 ...
320 ...
330         GOSUB 410
340         GOSUB 460
:
:
400 REM *** SUBROUTINE VOOR GETALCONVERSIE ***
410 ...
:
:
450 REM *** SUBROUTINE VOOR BEREKENINGEN ***
460 ...
:
:
```

alternatief 2

```
330         GOSUB 810
340         GOSUB 910
:
:
800 REM *** SUBROUTINE VOOR GETALCONVERSIE ***
810 ...
:
:
900 REM *** SUBROUTINE VOOR BEREKENINGEN ***
910 ...
:
:
```

1.11 HET OOG WIL OOK WAT

Er zijn verschillende mogelijkheden om het uiterlijk en de duidelijkheid van uw programma te verbeteren. Enkele van de hieronder te bespreken 'kneepjes' daarvoor zijn misschien niet toe te passen op uw computer. Andere worden wellicht automatisch uitgevoerd. Hoe dan ook: probeer ze , en maak er zo mogelijk gebruik van.

Lay-out

Een manier om het uiterlijk van een BASIC-programma te verbeteren is alle regelnummers evenlang te maken. Gebruik voor kleine programma's bijvoorbeeld de nummers 100 t/m 999, en voor grotere 1000 t/m 9999. Bij het afdrukken van uw programma staat dan alles netjes onder elkaar.

Uw MSX-computer heeft een commando (RENUM) om regels automatisch te 'hernummeren'. U kunt dan de regelnummers laten oplopen met een vaste stapgrootte van bijvoorbeeld 10. De regelmatige nummering komt het uiterlijk ten goede, en op deze wijze wordt bovendien de mogelijkheid opengelaten om later opdrachten tussen te voegen. Aangezien dit laatste de regelmaat verstoort, zou dan eigenlijk opnieuw genummerd moeten worden. Zonder hernummercommando is dit echter nauwelijks te doen. Meestal zullen we dan genoegen moeten nemen met die afwijkende nummers. Hopelijk zullen ze tot de uitzonderingen behoren, en het totaalbeeld zal dan toch regelmatig zijn.

Tot de lay-out technieken behoort ook het toevoegen van spaties in de programmaopdrachten zelf. Daardoor worden ze beter leesbaar. In MSX BASIC is dit overal toegestaan - wat u op het toetsenbord intypt wordt precies zo vastgelegd en weergegeven bij het afdrukken van het programma. Voor vele computers geldt dit niet. Toegevoegde spaties worden daarbij buiten beschouwing gelaten, en de computer gaat ongestoord zijn gang met een eigen, automatische lay-out methode.

Enkele suggesties

1. Gebruik spaties om het verschil tussen REMARK's en overige BASIC-code te laten uitkomen.

goed:

```
100 REM *** SUBROUTINE VOOR HET TESTEN VAN GEGEVENSINVOER ***
110 REM
120 LET D$ = N$
130 LET Z = LEN(D$)
```

beter:

```
100 REM *** SUBROUTINE VOOR HET TESTEN VAN GEGEVENSINVOER ***
110 REM
120         LET D$ = N$
130         LET Z = LEN(D$)
```

2. Gebruik spaties voor en na rekenkundige operatoren en vergelijkingsoperatoren.

   voorbeeld:

```
140 LET C = (A * B) / D
150 IF  D$ <> C$ THEN PRINT "FOUT IN INVOER"
160 IF  C  <= D THEN 700
```

3. Gebruik een spatie vóór elk gegeven in een DATA-lijst.

   voorbeeld:

```
340 INPUT A, B, C$
 :
900 DATA 36, 14, "BLABLA"
```

4. Gebruik spaties tussen BASIC-sleutelwoorden (Engels: keywords) en variabelen.

   voorbeeld:

```
150 LET X = 4 * Y + 10
160 INPUT "VOER RUBRIEKNAAM IN"; R$
170 IF R$ = "STOP" THEN 999
```

5. FOR/NEXT lussen

   a. Spring enkele posities in bij de statements tussen FOR en NEXT.

      voorbeeld:

```
100 FOR X = 1 TO 40
110     LET Y = 2 * X
120     PRINT X, Y
130 NEXT X
```

b. Spring bij geneste FOR/NEXT lussen* eveneens enkele posities in, of gebruik dubbele punten om het niveau aan te geven.

voorbeeld:

```
100 FOR X = 1 TO 10
110     FOR Y = 1 TO 5
120         LET A(X,Y) = 0
130     NEXT Y
140 NEXT X
```

of:

```
100 : FOR X = 1 TO 10
110 :: FOR Y = 1 TO 5
120     LET A(X,Y) = 0
130 :: NEXT Y
140 : NEXT X
```

6. Spring enkele posities in bij geneste 'IF...THEN'-opdrachten.

voorbeeld:

```
100 IF A$ = "STOP" THEN 910
110    IF B$ = "EINDE" THEN 920
120       IF C = 0 THEN 930
```

7. Spring ook in bij een groep opdrachten die in een bepaalde 'tak' van een 'IF...THEN'-constructie thuishoren.

voorbeeld:

```
140 IF A$ <= B$ THEN 180
150    LET H$ = A$
160    LET A$ = B$
170    LET B$ = H$
180 IF B$ <= C$ THEN 220
```

Andere technieken om uiterlijk en leesbaarheid te verbeteren

Er zijn ook nog andere mogelijkheden om een programma voor uzelf en anderen duidelijker te maken. Het gebruik van LET bijvoorbeeld – ook indien niet strikt nodig – kan verhelderend werken, vooral bij multi-opdrachtregels. Verwarrend is bijvoorbeeld:

---

* Een geneste FOR/NEXT is een lus die binnen een andere FOR/NEXT-lus (uiteraard met een andere besturingsvariabele) is opgenomen. In § 2.14 wordt hierop nader ingegaan.

```
260 X = Y: C = X*Y: IF X = N THEN X = C
```

Een betere oplossing zou zijn:

```
260 LET X = Y: C = X * Y: IF X = N THEN X = C
```

of:

```
260 LET X = Y: LET C = X * Y: IF X = N THEN LET X = C
```

Opdrachten zo rangschikken dat ze op een natuurlijke manier van links naar rechts te lezen zijn, is ook een manier om de leesbaarheid te verbeteren. Dit houdt bijvoorbeeld in dat een A – zo mogelijk – vóór een B komt, en een 1 vóór een 2. 'IF...THEN'-opdrachten kunnen aan duidelijkheid winnen als de variabele die het minst verandert achteraan staat (zie regels 270 en 300) in voorbeeld b hieronder).

voorbeelden:

a.
```
150 READ A, B, C
```

b.
```
260 FOR X = 1 TO 8
270     IF M(X) <> N THEN 290
280        LET M(X) = N
290 NEXT X
300 IF D$ = "STOP" THEN 999
```

Opdrachten die wat aan de lange kant zijn – vooral wiskundige uitdrukkingen – zijn meestal moeilijk hanteerbaar, en kunnen daarom beter opgesplitst worden in twee of meer delen. Een handige manier overigens om lange opdrachten op leesbaarheid te testen is ze gewoon hardop te lezen.

verwarrend:

```
250 LET T = (N * 3.75) + ((N - 40) * 3.25) + ((N - 60) / 3) / ((D * N) * A)
```

duidelijker:

```
250 LET T = (N * 3.75) + ((N - 40) * 3.25)
255 LET T = T + ((N - 60) / 3) / ((D * N) * A)
```

of:

```
250 LET T1 = (N * 3.75) + ((N - 40) * 3.25)
252 LET T2 = ((N - 60)/3) / ((D * N) * A)
254 LET T = T1 + T2
```

De fraaiste uitvoer wordt bereikt met behulp van de 'PRINT USING' opdracht, of met behulp van de speciale opmaakmogelijkheden (Engels: formatted output capability) die in MSX BASIC opgenomen zijn. Schermopmaak kan het beste plaatsvinden met de eigen grafische mogelijkheden van MSX BASIC.

Philips VG 8020-toetsenbord.

1.12 GEHEUGENGEBRUIK EN EXECUTIETIJD

Hopelijk zal het voorgaande u overtuigd hebben van de noodzaak duidelijke, goed leesbare programma's te schrijven. U zult waarschijnlijk niet alle behandelde foefjes onthouden, laat staan toepassen. Hoe dan ook, we hopen dat u – gevoelig gemaakt voor dit aspect van het programmeren – een eigen, 'zindelijke' programmeerstijl, gebaseerd op dit soort ideeën, ontwikkelt en dat daardoor de kwaliteit van uw programma's er op vooruitgaat.

Bijna al onze suggesties hebben een belangrijk nadeel: ze kunnen er toe leiden dat extra geheugenruimte wordt gebruikt. Het is zelfs mogelijk dat uw geheugenruimte, mede hierdoor, niet toereikend is voor het hele programma. Er zijn dan twee mogelijkheden: of uw algoritme (= oplossingsmethode) herzien, of het uit de kluiten gewassen programma op de een of andere manier snoeien.
Dit laatste kan inhouden dat enkele van de door u genomen maatregelen, bedoeld om de leesbaarheid te verbeteren, helaas ongedaan gemaakt moeten worden. Enkele mogelijkheden hiervoor zullen we – met een bloedend hart – op een rijtje zetten.

Geheugenbesparende maatregelen die relatief veel zoden aan de dijk zetten:
1. Gebruik verscheidene opdrachten per regel.
2. Verwijder alle REMARK-opdrachten, te beginnen met de inleidende module.

Geheugenruimte kan verder worden bespaard door:
1. Namen van variabelen te beperken tot één letter.
2. Onnodige haakjes te verwijderen.
3. Variabelen waar mogelijk opnieuw te gebruiken (normaal gesproken een afschuwelijke gewoonte).

4. Spaties in opdrachten waar mogelijk te verwijderen.
5. Zo mogelijk INTEGER-variabelen te gebruiken, bijvoorbeeld:

```
FOR X% = 1 TO 10
```

6. Arrays zo zuinig mogelijk te dimensioneren.
7. GOTO in plaats van GOSUB te gebruiken bij subroutines die slechts vanuit één plaats in het programma worden aangeroepen.

Door de volgende maatregelen kan zonodig de executietijd met enkele microseconden of zelfs seconden worden verkort. Enkele hiervan komen overeen met de reeds besproken maatregelen voor het besparen van geheugenruimte.

1. Verwijder alle REMARK-opdrachten, of laat de inleidende module naar het einde van het programma verhuizen.
2. Gebruik liever variabelen dan constanten (zoals eerder aanbevolen).
3. Gebruik multi-opdrachtregels.
4. Definieer de meest gebruikte variabelen het eerst.
5. Gebruik bij voorkeur INTEGER-variabelen.
6. Zet subroutines vóór het hoofdprogramma.
7. Gebruik FOR/NEXT-lussen indien enigszins mogelijk.
8. Verwijder onnodige haakjes.
9. Beperk het gebruik van GOSUB's.

Houd er echter rekening mee dat, hoewel ze uw programma weliswaar kunnen versnellen, deze maatregelen over het algemeen als slechte programmeerstijl aan te merken zijn, en indruisen tegen bijna alles wat we in dit hoofdstuk te berde hebben gebracht.

Om ruimte te besparen en de aandacht minder af te leiden, hebben we in dit boek overigens ook zelf niet altijd onze eigen suggesties opgevolgd. Niettemin zijn de programma's ons inziens goed leesbaar en redelijk zelfverklarend gebleven.

1.13 TOETSVRAGEN

1. Werkt een goedlopend MSX BASIC-programma zonder meer op andere systemen die ook BASIC kennen?
   Waarom wel/niet?

2. Hoe krijgt u de grootste zekerheid dat een door u geschreven programma ook op een andere computer zal functioneren?

3. Wat wordt bedoeld met het begrip 'overdraagbaarheid' (Engels: portability) van computerprogramma's?

4. Noem minstens drie rubrieken die thuis horen in de met behulp van REMARK's opgebouwde inleidende module.

5. Beschrijf de 'top-to-bottom' methode voor het opbouwen van programma's.

6. Naar welke opdrachten kan beter niet verwezen worden bij het gebruik van sprongopdrachten zoals GOTO en GOSUB?
   Waarom?

7. Wat wordt bedoeld met 'initialiseren'?

8. Wat is de belangrijkste (technische) reden voor het coderen van een deel van een programma als een subroutine?

9. Welk(e) offer(s) moet(en) gebracht worden om zelfverklarende, goed leesbare programma's te schrijven?

10. Hoeveel opdrachten van een multi-regelopdracht worden uitgevoerd als de eerste opdracht een REMARK is.

1.14 ANTWOORDEN OP DE TOETSVRAGEN

1. Als de betrokken computers geen MSX-computers zijn kan het voorkomen dat het programma niet zal functioneren.

2. Door 'conservatief te programmeren, dat wil zeggen geen gebruik te maken van de 'features' van MSX BASIC.

3. De mate waarin een programma zonder of met minimale veranderingen geschikt is voor verwerking door verschillende computers.

4. Drie van de volgende mogelijkheden: gebruikte variabelen en hun betekenis of functie; gebruikte bestanden; programmanaam; beschrijving van het doel van het programma; auteur; datum van totstandkoming of revisie; voor welke BASIC-versie en/of computersysteem werd het programma ontwikkeld.

5. Het programma wordt voor zover mogelijk zo ingericht, dat uitvoering bij de opdracht met het laagste regelnummer begint, en in volgorde van regelnummer wordt voortgezet met zo weinig mogelijk vertakkingen onderweg.

6. REMARK-opdrachten, omdat deze bij gebrek aan geheugenruimte door de programmeur verwijderd kunnen worden.

7. Het voor de eerste keer in een programma toekennen van waarden (veelal nullen of spaties) aan variabelen of array-elementen, en het dimensioneren van arrays.

8. Door opdrachten in de vorm van een subroutine te gieten wordt voorkomen dat die opdrachten opnieuw moeten worden gecodeerd als ze op een andere plaats in het programma nodig zijn.

9. Meer geheugengebruik en eventueel langere executietijd.

10. Geen. De computer beschouwt de tweede en eventueel volgende opdrachten op de regel als onderdeel van de REMARK-opdracht, dus als commentaar.

2 OVERZICHT VAN DE BELANGRIJKSTE TAALELEMENTEN

2.0 DOELSTELLINGEN

De bedoeling van dit hoofdstuk is een overzicht te geven van de belangrijkste taalelementen van MSX BASIC. Na bestudering ervan moet u in staat zijn de volgende MSX BASIC-elementen te gebruiken:
- LET
- READ en DATA
- INPUT en LINE INPUT
- IF...THEN
- FOR/NEXT
- GOSUB en RETURN
- ON... GOTO
- LEN, ASC en INSTR
- MID$, LEFT$ en RIGHT$

2.1 INLEIDING

Zoals eerder gezegd gaan we er in dit boek van uit dat u al eerder programma's in BASIC hebt geschreven, en dat u een BASIC-programma redelijk kunt lezen en begrijpen. In dit hoofdstuk krijgt u de gelegenheid uw geheugen op te frissen. Naast een behandeling van de taalelementen zelf komen ook enkele basistechnieken ter sprake waarbij die taalelementen worden toegepast. Vele daarvan zult u nodig hebben bij het werken met bestanden. Door daar nu aandacht aan te besteden kunnen we ons straks concentreren op het 'echte' (bestands-)werk.

2.2 NAMEN VAN VARIABELEN

In de eerste versies van BASIC mocht de naam van een variabele maar uit één letter bestaan, of een letter gevolgd door een cijfer. Voor zogenaamde stringvariabelen moest bovendien als laatste teken een dollarteken worden toegevoegd. A, A1, Z7, Z0, B$ en B1$ waren dus correcte variabelenamen. AA, A25, SALARIS en NAAM$, bijvoorbeeld, waren dat niet. BASIC-80 en andere nieuwe BASIC-dialecten staan echter het gebruik toe van namen die uit meer dan één letter bestaan. De laatstegenoemde namen zijn daarbij wel geldig. Zo ook namen als SUBTOTAAL, NETTOLOON, KEUZE, STRAAT$ enz. De verleiding om lange variabelenamen te gebruiken kan groot zijn, maar pas op! MSX BASIC gebruikt namelijk alleen de eerste twee letters van een naam; eventueel volgende letters worden eenvoudig buiten beschouwing gelaten. SALARIS en SALADE zijn dus in feite een en dezelfde variabele. Voorzichtigheid is dus geboden bij het kiezen van namen voor variabelen. Er kunnen fouten optreden die moeilijk te achterhalen zijn. Bovendien is voor langere namen meer geheugenruimte nodig, wat bij grotere of complexere programma's wel eens problemen kan opleveren.

Een andere beperking bij het gebruik van langere namen is dat lettercombinaties, die ook in BASIC-opdrachten, commando's of functies kunnen voorkomen, verboden zijn. De 'Reserved Word List' (= lijst van gereserveerde woorden) in uw gebruikershandleiding geeft aan welke woorden geen deel mogen uitmaken van een lange variabelenaam. Voorbeelden zijn:

FOR, DATA, OPEN, CLOSE, PRINT, KILL, IF, THEN

Veel narigheid kan dus worden voorkomen door eenvoudige namen te kiezen, bijvoorbeeld A, T1 en Y$. Daardoor is het wel moeilijker om echt 'suggestieve' namen te bedenken, dat wil zeggen namen die iets weergeven van de functie van een variabele en daardoor bijdragen aan de duidelijkheid van het programma. Met een beetje fantasie lukt dit echter aardig, bijvoorbeeld T voor totaal, ET voor eindtotaal, S voor salaris.

Sommige tekens geven gemakkelijk aanleiding tot misverstanden, en moeten daarom met beleid worden gebruikt. De letter O, bijvoorbeeld, kan aangezien worden voor het cijfer 0; één van beide (bijna altijd het cijfer, maar soms ook de letter) wordt daarom meestal doorgestreept (Ø), maar dit kan op zijn beurt weer worden verward met de letter Q. Ook de letter I en het cijfer 1 geven vaak problemen.

Sommige programmeurs kiezen voor bepaalde functies steevast dezelfde variabelen in alle programma's die ze schrijven. X, Y en Z worden dan bijvoorbeeld gebruikt als besturingsvariabelen in FOR/NEXT-lussen. T (teller) of I (uit de wiskunde) worden veel gebruikt in telconstructies zoals LET I = I + 1. In dit laatste voorbeeld valt de mogelijke verwarring tussen letter en cijfer overigens mee, juist wegens het bewuste gebruik.

Variabelenamen – ook wel labels genoemd – duiden eigenlijk geheugenplaatsen aan waar gegevens worden opgeslagen. De gegevens kunnen numeriek zijn (een getal), of alfanumeriek (een stuk tekst, ook wel 'string' genoemd); deze laatste categorie wordt in de volgende paragraaf uitvoeriger behandeld. Beide soorten waarden worden in de desbetreffende geheugenplaatsen opgeslagen met behulp van toekenningsopdrachten (LET, READ, INPUT). Bij verdere verwijzingen naar die variabelen gebruikt de computer de toegekende waarde. De genoemde toekenningsopdrachten worden in dit hoofdstuk behandeld.

Toetsvraag

(a) Geef twee redenen om eenvoudige variabelenamen, zoals A, X3 en Y$, te gebruiken.

(a) Twee van de volgende mogelijkheden:
1. Bespaart geheugenruimte.
2. Voorkomt het gebruik van gereserveerde woorden, met name als onderdeel van een langere naam.
3. Geen overdraagbaarheidsprobleem bij het gebruik van verschillende BASIC-versies.

2.3 STRINGVARIABELEN

Stringvariabelen zijn herkenbaar aan het dollarteken aan het einde van hun naam. A$ is dus een stringvariabele, A een numerieke variabele. Een string (= tekenreeks) bestaat uit één of meer letters, cijfers of bijzondere tekens. Omdat deze tekenverzameling (voornamelijk) uit alfabetische en numerieke tekens bestaat, spreekt men bij een string van een *alfanumeriek* gegeven (*alfa*betisch - *numeriek*). Strings zijn in wezen constanten, en ze kunnen in een BASIC-programma op dezelfde manier worden gebruikt als getallen, oftewel numerieke constanten. Om de string 'BLABLA' in het computergeheugen op te slaan kunnen we bijvoorbeeld gebruik maken van de toekenningsopdracht:

```
LET A$ = "BLABLA"
```

De stringvariabele A$ functioneert daarbij als een etiket om de geheugenplaats aan te geven waar de tekst "BLABLA" wordt opgeslagen. Een daaropvolgende verwijzing naar A$ impliceert het gebruik van de bijbehorende string, die als *waarde* van A$ kan worden beschouwd.

Het essentiële verschil tussen stringvariabelen en numerieke variabelen is dat stringvariabelen – ook al bevatten ze numerieke gegevens – niet in rekenkundige uitdrukkingen of bewerkingen kunnen worden gebruikt.

Bij de opdrachten:

```
100 LET A$ = "8.99"
110 LET B$ = A$ + 0.5
```

wordt men door de computer dus onverbiddelijk op de vingers getikt met de melding SYNTAX ERROR, in plaats van de wellicht verwachte toekenning van de waarde 9.04 aan B$. In het laatste geval zou men moeten coderen:

```
100 LET  A = 8.99
110 LET  B = A + 0.5
```

De geheugenruimte die een string inneemt kan worden voorgesteld door een vakje aangegeven door de naam van de variabele. De toekenningsopdracht:

```
LET N$ = "AUTOMOBIELBEDRIJF VAN DER PANNE B.V."
```

heeft dan als het ware de inrichting van het volgende vakje in het geheugen van de computer tot gevolg:

In feite wordt echter alleen voor de waarde van de variabele geheugenruimte gebruikt, althans in de voor gegevens beschikbare ruimte. De koppeling tussen het etiket dat we als programmeur gebruiken (N$) en de eigenlijke plaats waar de bijbehorende waarde zich bevindt wordt door de computer elders bijgehouden.

Merk op dat de toegekende string omgeven is door aanhalingstekens. Deze dienen om de string af te bakenen, maar maken van die string zelf geen deel uit. Stringvariabelen kunnen maximaal 255 tekens bevatten. Ze kunnen echter ook leeg zijn, dat wil zeggen helemaal geen tekens bevatten. We spreken dan van een *nulstring* (Engels: null string). Een toekenningsopdracht voor zo'n nulstring zou zijn: LET Z$ = "".

Er is een belangrijk verschil tussen de *maximale* lengte van een string (dus 255 tekens) en de *eigenlijke* lengte. De eigenlijke lengte is het aantal alfanumerieke tekens dat op een gegeven moment aan een stringvariabele is toegekend, en dus ook daadwerkelijk in de bijbehorende geheugenplaats is opgeslagen. Spaties worden daarbij als alfanumerieke tekens gerekend. De omvattende aanhalingstekens, gebruikt bij de toekenning, doen niet mee.

voorbeelden:

N$	AUTOMOBIELBEDRIJF VAN DER PANNE B.V.

eigenlijke lengte: 36 tekens

A$	FORTLAAN 13-17

eigenlijke lengte: 14 tekens
(inclusief spatie en koppelteken)

Het is een goede gewoonte in een programma aan te geven hoe lang een bepaalde stringvariabele in de context van dat programma maximaal kan worden. Dit kan men met behulp van REMARK's in de inleidende module doen, bijvoorbeeld op de volgende manier:

```
140 REM   STRINGVARIABELEN
150 REM    - N$    : NAAM VAN KLANT (20)
160 REM    - A$    : ADRES VAN KLANT (25)
170 REM    - W$    : POSTCODE (7) EN WOONPLAATS (23)
180 REM              MAXIMAAL 30, INCL. SPATIES
190 REM
```

Toetsvragen

1. Hoeveel tekens bevat een nulstring die aan een stringvariabele wordt toegekend?
2. Hoeveel tekens neemt een spatie, die deel uitmaakt van een string, in beslag?

Antwoorden: 1. nul (geen); 2. één.

In tegenstelling tot de straks te bespreken READ- en INPUT-opdrachten wordt de LET ook wel een *directe* toekenningsopdracht genoemd. MSX BASIC accepteert de opdracht ook zonder het sleutelwoord LET. De opdrachten:

```
240 LET N$ = "BLABLA"          en:    240 N$ = "BLABLA"
```

zijn dus identiek. Ter wille van de duidelijkheid gebruiken we in dit boek in het algemeen de LET-vorm.

2.4 READ/DATA-TOEKENNINGSOPDRACHT

DATA-opdrachten lijken 'in de verte' op gegevensbestanden. Ze bevatten namelijk gegevens die tijdens de uitvoering van het programma aan variabelen worden toegekend. Bij de DATA-opdracht zijn de desbetreffende gegevens echter alleen te gebruiken in het programma waarin die DATA-opdracht is opgenomen. Gegevensbestanden leiden een zelfstandiger bestaan. De gegevens worden daarbij los van het programma opgebouwd, en kunnen dus door verschillende programma's worden gebruikt.

Bij iedere DATA-opdracht horen één of meer READ-opdrachten. De READ-opdracht functioneert daarbij als een toekenningsopdracht: één of meer gegevens van de DATA-opdracht worden toegekend aan één of meer variabelen genoemd in de READ.

voorbeeld:

```
10  READ A
20  DATA 15, 76.5, 1892, -999
```

De opdracht READ A heeft tot gevolg dat één van de getallen in de DATA-opdracht aan de variabele A wordt toegekend. Welk getal dat is hangt af van het feit of deze (of een andere) READ-opdracht al eerder is uitgevoerd. Is regel 10 de eerste READ-opdracht die wordt uitgevoerd, dan krijgt A de waarde 15. Hadden we in plaats van READ A de opdracht READ A, B gecodeerd, dan zou aan A de waarde 15 en aan B de waarde 76.5 zijn toegekend. Hetzelfde zouden we bereikt hebben als we na READ A de opdracht READ B hadden laten uitvoeren, bijvoorbeeld als regel 15.

READ/DATA-opdrachten kunnen ook worden gebruikt voor het toekennen van strings aan stringvariabelen.

voorbeeld:

```
220 READ A$, B$, C$
 :
910 DATA ROOD, WIT, BLAUW
```

Omvattende aanhalingstekens bij de string in de DATA-opdracht zijn in MSX BASIC niet nodig, tenzij de string een komma, puntkomma of één of meer voorafgaande spaties bevat. In deze gevallen wordt de string geheel tussen aanhalingstekens opgenomen, zoals bij de directe LET-toekenning. Eventueel navolgende spaties tussen een string en een daaropvolgende komma worden gerekend als deel van de string, en aan de overeenkomstige stringvariabele toegekend. De eigenlijke lengte van de stringvariabele wordt dan mede door die spatie(s) bepaald.

Oppassen geblazen dus bij het gebruik van komma's als scheidingsteken tussen de gegevens in een DATA-opdracht!

voorbeeld:

In het onderstaande programmadeel zijn aanhalingstekens nodig, omdat de komma's deel uitmaken van de strings zelf.

```
220 READ N$
 :
910 DATA "BROWN, JERALD R.", "FINKEL, LEROY P."
```

Probeer de volgende constructie op uw MSX-computer, en let op de manier waarop navolgende spaties worden verwerkt:

```
220 READ  N$, A$
230 PRINT N$; A$
910 DATA  TESTbbb,bbbGEGEVENS
RUN
```

NB: b = blanko positie, oftewel spatie

Als het goed is levert dit als antwoord:

```
TESTbbbGEGEVENS
```

Slechts drie spaties tussen TEST en GEGEVENS dus, omdat voorafgaande spaties niet meedoen en navolgende spaties wel. Vervang regel 910 nu als volgt en draai het programma opnieuw:

```
910 DATA   "TESTbbb", "bbbGEGEVENS"
```

(a) Hoeveel spaties verschijnen nu tussen beide strings?

(a) zes

Koppeling tussen gegeven en variabele

De computer kent een intern wijzermechanisme – in feite een geheugenlokatie waarvan de inhoud het adres van een andere geheugenlokatie aangeeft – dat tijdens elke programma'run' bijhoudt welke elementen van een DATA-opdracht aan een variabele zijn toegekend. Bij het uitvoeren van READ-opdrachten wordt de wijzer (Engels: pointer) bij iedere toekenning zo aangepast dat hij naar het volgende gegeven in de DATA-opdracht verwijst. Geeft de wijzer een string aan terwijl er volgens de READ-opdracht een numerieke waarde nodig is, dan treedt er een foutconditie op en de uitvoering van het programma wordt gestaakt.

voorbeeld:

```
210 READ A
910 DATA BLABLA
```

In dit geval treedt een foutconditie op omdat de READ-opdracht als het ware op zoek is naar een numeriek gegeven om aan A – een numerieke variabele – toe te kennen, terwijl de wijzer een alfanumerieke waarde aangeeft.

Toetsvraag

Bekijk het volgende programma:

```
210 READ  A$, B$
220 PRINT A$; B$
910 DATA  17926, BLABLA
```

(a) Draait het programma zonder foutmeldingen?
(b) Wat wordt toegekend aan A$ en waarom?

(a) Ja.
(b) 17926 (een getal kan als string aan een stringvariabele worden toegekend, maar niet omgekeerd).

2.5 DE INPUT-OPDRACHT

Numerieke of alfanumerieke gegevens kunnen aan een numerieke respectievelijk stringvariabele worden toegekend door middel van een INPUT-opdracht. Een daaraan verwante toekenningsopdracht – de 'LINE INPUT'-opdracht – accepteert slechts één gegeven (en wel alfanumeriek) dat toegekend wordt aan een stringvariabele. De LINE INPUT komt in de volgende paragraaf aan de orde.

Bij het gebruik van INPUT-opdrachten is het van belang dat degene achter de computer precies weet hoe hij zijn gegevens moet invoeren. Mocht uw programma alleen voor eigen gebruik bestemd zijn, vergis u dan niet. Het zal u verbazen hoe snel u zich achter de oren krabt met de vraag: "Hoe zat dat ook weer?". Om de invoer probleemloos te laten verlopen moet u eerst zelf precies weten hoe de INPUT-opdracht in uw BASIC-versie werkt.

Veel ellende kan worden voorkomen door altijd een zogenaamde *promptstring* aan de invoer te laten voorafgaan. Een *prompt* is een 'aanzet' of boodschap die op het scherm verschijnt, met de bedoeling de computergebruiker aan te geven dat er van hem een bepaalde actie wordt verlangd.

voorbeeld:

```
160 INPUT "GEEF UW NAAM OP, EERST VOORNAAM DAN ACHTERNAAM"; N$
```

Een INPUT-opdracht zonder promptstring heeft tot gevolg dat de computer met een vraagteken reageert, en vervolgens een respons van het toetsenbord afwacht. Niets is frustrerender voor een computergebruiker dan zo'n INPUT-vraagteken zonder enige aanduiding van wat nu eigenlijk wordt verwacht. *Neem daarom altijd een promptstring in een INPUT-opdracht op*. Gebruik zonodig PRINT-opdrachten, voorafgaande aan de INPUT-opdracht, om de gebruiker uit te leggen hoe hij zijn gegevens moet invoeren.

Een andere bron van frustratie is de rare manier waarop de computer op verkeerde gegevens kan reageren. Een voorbeeld:

```
360 INPUT "GEEF ARTIKELNUMMER EN HOEVEELHEID OP"; N, H


RUN
GEEF ARTIKELNUMMER EN HOEVEELHEID OP?137
??
```

De gebruiker typte na het verschijnen van de prompt-boodschap en het vraagteken het getal 137, en drukte vervolgens op de RETURN-toets. De computer reageerde met twee vraagtekens om aan te geven dat hij meer gegevens verwachtte. Begrijpelijk - althans voor een geroutineerde computergebruiker - want slechts één gegeven werd ingevoerd terwijl de INPUT-opdracht twee variabelen bevatte. Voor een onervaren computergebruiker zou dit echter minder duidelijk zijn. De bedoeling was dat u beide gegevens, gescheiden door een komma, in één keer invoerde. Dus:

```
RUN
GEEF ARTIKELNUMMER EN HOEVEELHEID OP?137,12
Ok
```

Laten we hetzelfde programma nog eens uitvoeren met drie gegevens:

```
RUN
GEEF ARTIKELNUMMER EN HOEVEELHEID OP?137,12,164
?EXTRA IGNORED
```

De uitvoering van het programma wordt voortgezet, terwijl het getal 164 wordt genegeerd.

Foutcondities en invoerproblemen kunnen ook bij string-gegevens voorkomen. Neem bijvoorbeeld het volgende programmafragment:

```
180 INPUT "GEEF NUMMER EN NAAM VAN KLANT OP"; K, N$
190 PRINT K, N$

RUN
GEEF NUMMER EN NAAM VAN KLANT OP?13726
??
13726
```

In dit geval typte de gebruiker het klantnummer en drukte vervolgens op RETURN. Het nummer werd netjes toegekend aan variabele K. Bij het verschijnen van de twee vraagtekens - om aan te geven dat de computer nog een gegeven verwachtte - drukte de gebruiker de RETURN-toets opnieuw in, zonder eerst iets ingetypt te hebben. De computer accepteerde daarop een lege string als invoer, en kende die toe aan N$.

MSX BASIC geeft de melding "? REDO FROM START" indien een verkeerd gegevenstype wordt ingevoerd, bijvoorbeeld een string in plaats van een getal bij toekenning aan een numerieke variabele.

Bovendien resulteert het indrukken van de RETURN-toets zonder voorafgaande gegevens in het toekennen van een nul aan numerieke variabelen en een lege string aan stringvariabelen.

Voorkomen van incorrecte invoer

Twee programmeertechnieken kunnen fouten op het gebied van verkeerde invoer bij INPUT-opdrachten helpen voorkomen. De eerste daarvan is, per INPUT-opdracht slechts één gegeven op te vragen. Dit maakt gegevensinvoer zeer eenvoudig en bovendien gemakkelijk te controleren, zoals we in het volgende hoofdstuk zullen zien.

voorbeeld:

```
RUN
GEEF NUMMER VAN KLANT OP?137
GEEF NAAM VAN KLANT OP?ABC B.V.
GEEF ARTIKELNUMMER OP? 18625
GEEF BESTELDE HOEVEELHEID OP?106
```

De tweede techniek bestaat uit het toekennen van alle invoergegevens – zowel numeriek als string – aan stringvariabelen. Dit voorkomt problemen bij het toekennen van alfanumerieke gegevens aan numerieke variabelen. Getallen die aan stringvariabelen zijn toegekend kunnen van string naar numerieke vorm worden geconverteerd met behulp van de VAL-functie. Als bijvoorbeeld H$ = 106

(dus niet het getal 106 maar de cijferreeks 1, 0, 6), dan converteert VAL(H$) de cijferreeks naar een getal dat aan een numerieke variabele kan worden toegekend en/of rechtstreeks als een numerieke waarde in een BASIC-expressie kan worden gebruikt. VAL wordt in het volgende hoofdstuk uitgebreid behandeld. In het volgende hoofdstuk komt bovendien aan de orde hoe men op lege strings (dat wil zeggen: geen invoer) kan testen en de computergebruiker met behulp van meldingen kan aangeven hoe hij zijn gegevens op de juiste wijze kan invoeren.

Toetsvraag

(a) Schrijf een INPUT-opdracht die in de volgende RUN resulteert:

```
RUN
GEEF UW HUISADRES OP?
```

(a) `100 INPUT "GEEF UW HUISADRES OP"; A$`
(het regelnummer en de naam van de variabele kunnen anders zijn).

2.6 DE 'LINE INPUT'-OPDRACHT

MSX BASIC kent (evenals de meeste BASIC-versies) naast de INPUT-opdracht een andere invoermogelijkheid: de LINE INPUT. Deze opdracht - op sommige computers ook wel LINPUT of INPUTLINE genoemd - geeft de mogelijkheid op gemakkelijke wijze gebruik te maken van gegevens waarin komma's, aanhalingstekens en voorafgaande spaties (Engels: leading blanks) zijn opgenomen. Daarbij kan per opdrachtsuitvoering slechts één gegeven worden ingelezen. Het gegeven wordt alfanumeriek opgevat. De variabele die in de 'LINE INPUT'-opdracht wordt genoemd moet dan ook van het alfanumerieke type zijn. Omdat slechts één gegeven kan worden ingelezen, mag de opdracht slechts één variabele bevatten. Dit ligt ook aardig in de lijn van het advies dat we in de vorige paragraaf hebben gegeven over het gebruik van slechts één variabele in een INPUT-opdracht.

Zoals bij de INPUT is ook bij de LINE INPUT een promptstring toegestaan. Bij programma-uitvoering verschijnt echter, in tegenstelling tot bij de INPUT, geen vraagteken. Zo'n vraagteken (of bijvoorbeeld een dubbele punt) moet, indien gewenst, expliciet als deel van de promptstring worden opgenomen. Bevat de LINE INPUT geen prompt dan geeft de computer geen enkele indicatie dat hij invoer vanaf het toetsenbord verwacht.

voorbeeld:

```
160 LINE INPUT  "GEEF UW ADRES OP: "; A$


RUN
GEEF UW ADRES OP: ↑
                  cursor
```

Omdat LINE INPUT komma's, aanhalingstekens en voorafgaande spaties zonder meer accepteert, is hij bijzonder geschikt voor het invoeren van regels tekst. Vandaar ook de naam ("LINE INPUT" betekent regelinvoer). De computer herkent het einde van de tekstregel door het indrukken van de RETURN-toets. Per regel kunnen maximaal 255 tekens (de maximale capaciteit van een stringvariabele) worden ingevoerd. In dit boek zal de LINE INPUT regelmatig worden gebruikt.

Toetsvraag

(a) Geef drie redenen om liever de 'LINE INPUT'- in plaats van de 'INPUT'-opdracht te gebruiken.

(a) 1. Met de LINE INPUT kunnen op gemakkelijke wijze (alleen) stringtoekenningen worden gemaakt.
2. LINE INPUT dwingt de programmeur zich bij invoer te beperken tot één gegeven per opdracht, waardoor de foutkans bij het invoeren van gegevens wordt verkleind.
3. Bij LINE INPUT verschijnt geen vraagteken op het scherm.

2.7 HET AANEENSCHAKELEN VAN STRINGS (CONCATENEREN)

Strings kunnen in BASIC aaneengeschakeld worden tot grotere strings. Dit heet *concateneren*. Strings worden geconcateneerd door ze als het ware bij elkaar op te tellen door middel van een plusteken. In dit verband functioneert het plusteken dus niet als rekenkundige operator maar als concatenatie-operator.

voorbeeld:

```
110 LET  N$ = V$ + A$
```

In dit voorbeeld worden de strings die eerder aan V$ en A$ waren toegekend aaneengeschakeld tot één geheel, dat op zijn beurt wordt toegekend aan N$. Gelijksoortige bewerkingen zijn uiteraard mogelijk tussen stringconstanten onderling of tussen stringconstanten en -variabelen.

voorbeelden:

```
120 LET R$ = "KLANT: " + N$

150 LET N$ = V$ + " " + A$
```

Toetsvragen

Welke uitvoer wordt ten gevolge van onderstaande opdrachten geproduceerd?

(a)
```
10  LET  V$ = "PIET"
20  LET  A$ = "HEIN"
30  LET  N$ = V$ + " " + A$
40  PRINT N$

    RUN
```

(b)
```
10  LET  V$ = "8990"
20  LET  A$ = "3806"
30  LET  N$ = V$ + A$
40  PRINT N$

    RUN
```

(a) PIET HEIN
(b) 89903806

2.8 'IF...THEN'-OPDRACHT

De 'IF...THEN' is een belangrijke opdracht. Hij heeft de volgende algemene vorm:

IF voorwaarde THEN opdracht

Onder 'voorwaarde' verstaan we een vergelijking tussen twee grootheden waarbij de uitkomst van de vergelijking waar of onwaar is. Zo'n vergelijking is bijvoorbeeld X < Y. Is de waarde van X inderdaad kleiner dan Y, dan is de vergelijking waar. Anders is hij onwaar.

Is het voorwaardedeel van de IF...THEN waar, dan wordt het opdrachtdeel uitgevoerd. Is het voorwaardedeel onwaar, dan heeft de IF...THEN in feite geen enkel effect. Er wordt dan eenvoudig verdergegaan met de daaropvolgende opdracht.

voorbeeld:

```
140 IF X < Y THEN GOTO 800
```

effect: Als X kleiner is dan Y, dan wordt gesprongen naar regel 800. Is X groter dan of gelijk aan Y, dan wordt de eerstvolgende opdracht na regel 140 uitgevoerd.

noot: Het woord GOTO in bovenstaande opdracht mag worden weggelaten.

Vergelijkingsoperatoren

De eenvoudigste vorm van het voorwaardedeel van de IF...THEN hebben we hierboven al gezien. Het bestond uit twee rekenkundige grootheden (X en Y) gescheiden door een 'kleiner dan' teken (<). Zo'n 'kleiner dan' teken is een zogenaamde *vergelijkingsoperator*, ook wel relationele operator genoemd. Er zijn meer van zulke operatoren. BASIC kent het volgende arsenaal:

<	kleiner dan	
<=	kleiner dan of gelijk aan	(ook wel: =<)
=	gelijk aan	
>=	groter dan of gelijk aan	(ook wel: =>)
>	groter dan	
<>	ongelijk aan	(ook wel: ><)

Verderop in deze paragraaf zullen we zien hoe dankzij een ander type operator meer ingewikkelde voorwaarden geformuleerd kunnen worden.

Het opdrachtdeel

Wat het opdrachtdeel van de IF...THEN betreft zijn er in MSX BASIC verscheidene mogelijkheden:

- IF ... THEN LET ...	waarbij de normale regels voor LET-opdrachten van toepassing zijn (het woord LET kan dus eventueel worden weggelaten)
- IF ... THEN GOSUB ...	voorwaardelijke sprong naar subroutine
- IF ... THEN RETURN	voorwaardelijke terugkeer uit subroutine
- IF ... THEN PRINT	voorwaardelijke uitvoer
- IF ... THEN INPUT ... IF ... THEN READ ...	voorwaardelijke invoer (niet aan te bevelen omdat ze verwarring en complicaties bij het ontluizen in de hand werken)
- IF ... THEN STOP IF ... THEN END	voorwaardelijke beëindiging van het programma

- IF ... THEN IF ... THEN ... voorwaardelijke opdracht die wordt uitgevoerd indien aan een andere voorwaarde is voldaan
- IF ... THEN GOTO ...
IF ... THEN regelnummer } voorwaardelijke sprongopdracht

Niet alle BASIC-versies kennen deze mogelijkheden. Sommige kennen uitsluitend de (oorspronkelijke) eenvoudige vorm:

IF voorwaarde THEN regelnummer

eventueel met het sleutelwoord GOTO voor 'regelnummer'. Deze vorm is in wezen een voorwaardelijke sprongopdracht.

Samengestelde voorwaarden

Behalve enkelvoudige voorwaarden waarbij twee rekenkundige grootheden worden vergeleken, kunnen in een IF...THEN ook meer voorwaarden worden ingebouwd. We spreken dan van een samengestelde voorwaarde. Hierbij wordt gebruik gemaakt van zogenaamde *logische* operatoren: de AND en de OR. De IF...THEN ziet er dan als volgt uit:

IF ... AND ... THEN ...

of:

IF ... OR ... THEN

Kenmerkend voor logische operatoren is dat ze twee grootheden verbinden die de waarden 'waar' of 'onwaar' vertegenwoordigen. Dit in tegenstelling tot de eerder besproken vergelijkingsoperatoren, die grootheden met een numerieke waarde verbinden. De AND is zo gedefinieerd dat de gehele voorwaarde alleen dan waar is indien zowel de voorwaarde links als de voorwaarde rechts van de AND waar is. Bij de OR is de gehele voorwaarde waar als een van beide (of beide) voorwaarden waar is (zijn).

voorbeelden:

(a) IF A > B AND C > D THEN ...

waar indien zowel A>B als C>D, onwaar indien A<=B of C<=D

(b) IF A > B OR C > D THEN ...

waar indien òf A>B òf C>D, onwaar indien zowel A<=B en C<=D

Eventueel kunnen meer dan één AND en/of OR operatoren tussen de IF en de THEN voorkomen, waardoor drie of meer 'logische' vergelijkingen binnen één IF...THEN kunnen worden gemaakt.

Een nuttige toepassing van AND/OR-operaties is het controleren op correcte invoer bij uitvoering van INPUT-opdrachten. Moet een invoergegeven bijvoorbeeld tussen 5 en 20 liggen, dan kan dat als volgt worden gecontroleerd:

```
150 IF F < 5 OR F > 20 THEN PRINT "INVOER INCORRECT"
```

of (andersom):

```
150 IF F >= 5 AND F <= 20 THEN PRINT "INVOER GEACCEPTEERD"
```

Bij het opbouwen van samengestelde voorwaarden is een vergissing gauw gemaakt. De problemen komen veelal voor bij de randgevallen, dat wil zeggen moet een grenswaarde wel of juist niet meedoen bij het bepalen of een voorwaarde al dan niet waar is. Stroomschema's bieden dan vaak uitkomst. Daardoor worden de verschillende mogelijkheden beter zichtbaar en wordt de kans op vergissingen bij het opbouwen van de voorwaarden kleiner. Uitgebreid testen van de andere mogelijke programmatakken is in elk geval geboden!

Toetsvraag

(a) Schrijf twee 'IF...THEN'-opdrachten waarbij in de één een AND en in de ander een OR voorkomt. De eerste opdracht dient te testen of de waarde van een variabele Y groter dan maar ongelijk nul is, en kleiner dan maar ongelijk 1. Wordt aan deze voorwaarden voldaan, dan moet de tekst "TUSSEN NUL EN EEN" worden afgedrukt. De tweede IF...THEN dient onder dezelfde omstandigheden de tekst "NIET TUSSEN NUL EN EEN" af te drukken.

```
(a) IF Y > 0  AND Y < 1  THEN PRINT "TUSSEN NUL EN EEN"
    IF Y <= 0 OR  Y >= 1 THEN PRINT "NIET TUSSEN NUL EN EEN"
```

Nogmaals: het opdrachtdeel van de IF...THEN

Zoals eerder gezegd kent MSX BASIC de mogelijkheid meer opdrachten op één regel te coderen. Wordt dit toegepast in het opdrachtdeel van een IF...THEN, dan wordt de gehele regel na THEN geacht deel uit te maken van de THEN-tak.

voorbeelden:

```
150 IF X < Y THEN PRINT "TE LAAG" : LET T = T + 1 : GOTO 10
```

wordt uitgevoerd indien X < Y

```
160 IF X > Y THEN LET T = T + 1 : LET G = 0 : GOTO 10
```

wordt uitgevoerd indien X > Y

Het gebruik van deze constructie kan de overdraagbaarheid in de weg staan. Daarom kan het verstandiger zijn toch maar een conventionele oplossing te zoeken. Bovendien kunnen zich moeilijkheden voordoen als de opdrachten die in een THEN-tak thuishoren niet alle op één regel passen. Soms wordt de regel dan afgesloten door een sprong naar een ander deel van het programma waar de THEN-activiteiten worden voortgezet, bijvoorbeeld:

```
150 IF X < Y THEN LET X = X + D : LET Y = Y / N : GOTO 200
160 IF X > Y THEN LET X = X - D : LET Y = Y / N : GOTO 10
:
:
200 LET T = T + 1 : PRINT "TE LAAG" : GOTO 10
```

Dit werkt verwarrend. Een betere oplossing zou zijn:

```
150 IF X < Y THEN 200
160 IF X > Y THEN 250
:
200 LET X = X + D
210 LET Y = Y / N
220 LET T = T + 1
230 PRINT "TE LAAG"
240 GOTO  10
250 ...
:
:
```

} eventueel geheel of gedeeltelijk op één regel

Afrondingen

Ten gevolge van interne afrondingen in de computer zelf kunnen onverwachte situaties ontstaan. Een geldwisselprogramma kan bijvoorbeeld 4.9999 stuivers in plaats van 1 kwartje als antwoord geven. Dergelijke fouten zijn veelal terug te voeren tot 'IF...THEN-opdrachten waar – ten gevolge van afronding – een andere tak werd uitgevoerd dan de bedoeling was. Dit speelt vooral een rol bij voorwaarden waarin het gelijkteken voorkomt, bijvoorbeeld:

```
IF X = 1125.75 THEN ...
```

Zo'n test kan beter vervangen worden door:

```
IF X <  1125.75 THEN ...
IF X >  1125.75 THEN ...
...
```

of door:

```
IF ABS(X - 1125.75) <= 0.001 THEN ...
```

waarbij 0.001 een zelf te kiezen tolerantiewaarde en ABS een absolute-waardefunctie is*.

* Het effect van de IF ABS( ) constructie is dat de vergelijking waar is indien X minder dan +0.001 of -0.001 afwijkt van de waarde 1125.75.

Toetsvraag

(a) Waarom kan het testen op gelijkheid (rechtstreeks of gecombineerd met groter of kleiner dan) beter worden vermeden?

(a) Ten gevolge van interne afrondingsfouten kunnen in berekeningen zeer kleine onnauwkeurigheden ontstaan. Een test op gelijkheid zou dan als onwaar kunnen uitvallen terwijl verwacht zou mogen worden dat het resultaat van de test waar is.

2.9 'IF...THEN'-STRINGVERGELIJKINGEN EN ASCII-CODE

Tot nu toe hebben we met de 'IF...THEN'-opdracht alleen numerieke vergelijkingen uitgevoerd. Het vergelijken van strings is echter ook mogelijk. Een voorbeeld:

```
220 LINE INPUT "GEEF UW VOLLEDIGE NAAM: "; N$
230 IF  N$  =  "STOP" THEN 999
```

In regel 230 wordt de stringvariabele N$ vergeleken met de string "STOP". Merk op dat de string tussen aanhalingstekens staat. De uitkomst van de vergelijking (of test) is alleen dan waar als beide waarden (dus de string opgeslagen in N$ en de string "STOP") teken voor teken, en gaande links naar rechts, gelijk zijn. Daarbij worden hoofdletters en kleine letters als verschillende tekens beschouwd. Bovendien moet de lengte van de strings gelijk zijn en moeten eventuele voorafgaande en/of navolgende spaties in beide strings op exact dezelfde wijze voorkomen. Elk verschil – hoe klein ook – leidt tot het resultaat 'onwaar'.

Behalve vergelijkingen tussen stringvariabelen en -constanten kunnen uiteraard ook stringvariabelen onderling worden vergeleken, bijvoorbeeld:

```
310 LINE INPUT "GEEF TITEL OP; " ;T$
320 IF T$ <> D$ THEN PRINT "VERKEERDE TITEL. PROBEER OPNIEUW A.U.B."
```

Tot dusver ging het om het al of niet gelijk zijn van stringwaarden. Het uitvoeren van 'kleiner dan' of 'groter dan' vergelijkingen is moeilijker. Dergelijke bewerkingen zijn onder andere nodig bij het sorteren van gegevens en bij het toevoegen van gegevens aan een alfabetisch geordend bestand. Zoals we hebben gezien worden stringvergelijkingen teken voor teken van links naar rechts uitgevoerd. Daarbij gaat BASIC voor elk teken uit van een bepaalde code. Elk teken dat vanaf het toetsenbord de CPU wordt ingestuurd (en omgekeerd) wordt voorgesteld door één zo'n code. Dit geldt ook voor toegestane toetscombinaties, zoals SHIFT of CONTROL samen met een andere toets. Elk teken komt intern overeen met een

serie elektrische pulsen die op een bepaalde manier zijn gecodeerd. De codes kunnen volgens een codetabel worden voorgesteld door decimale getallen. Een zeer gangbare code, gebruikt door bijna de gehele computerwereld, is ASCII (afkorting van American Standard Code for Information Interchange). Er zijn in ASCII in totaal 256 tekens gedefinieerd, waaronder de hoofdletters en kleine letters van het alfabet, de cijfers, leestekens en andere bijzondere tekens, en ook speciale functietekens zoals het opschuiven van een regel op een printer. Voor onze doeleinden zijn slechts de eerste 128 tekens van belang. Ze komen overeen met de decimale waarden 0 t/m 127. Voor een overzicht hiervan verwijzen we naar appendix B.

Aan de hand van de ASCII-tabel in appendix B kunt u zien dat de cijfers 0 t/m 9 voorgesteld worden door de codegetallen 48 t/m 57. Voor de hoofdletters worden de codegetallen 65 t/m 90 gebruikt. Voor de kleine letters begint de reeks bij 97. De kleine letter die het equivalent is van een bepaalde hoofdletter heeft dus een codegetal dat 32 groter is dan het codegetal van de desbetreffende hoofdletter. Hoofdletter "A", bijvoorbeeld, heeft als codegetal 65. Het codegetal voor de kleine letter "a" is dus 65 + 32 = 97. Van deze eigenschap zullen we later gebruik maken.

Het vergelijkingsmechanisme

Wat gebeurt er nu eigenlijk bij het vergelijken van strings bij een 'IF...THEN'-opdracht? Zoals gezegd 'bekijkt' BASIC de strings teken voor teken. Bij elk teken wordt uitgegaan van het bijbehorende codegetal. Blijkt tussen twee tekens ongelijkheid te bestaan, dan wordt de string met het teken dat de laagste ASCII-waarde heeft aangemerkt als 'kleiner' dan de andere string. BASIC telt dus *niet* de ASCII-codewaarden van de desbetreffende strings op.

Enkele voorbeelden:

A$	B$	
ABC	ABD	A$ is kleiner dan B$
MN!	MNP	A$ " " " B$
123A	123a	A$ " " " B$

Is de ene string korter dan de andere en zijn er verder geen verschillen tussen beide strings, dan is de kortste string 'kleiner' dan de andere. De reden daarvoor is dat de kortste string intern in feite met spaties wordt aangevuld tot de lengte van de grotere. En de spatie (Engels: space) heeft juist een waarde die kleiner is dan alle andere afdrukbare tekens! Volgens dezelfde redenering kunnen we stellen dat een lege string (null string) altijd 'kleiner' is dan een niet-lege string. Met deze wijsheid in het achterhoofd nóg enkele voorbeelden:

A$	B$	
JANSE	JANSEN	A$ is kleiner dan B$
SMIT	SMITH	A$ " " " B$
JOHNSEN	JOHNSON	A$ " " " B$
KELLOG	KELLOGG	A$ " " " B$
EQ 8	EQ-8	A$ " " " B$

Nu de beurt aan u:

	C$	D$	
(a)	JACOB	JACOBS	... is ....
(b)	GERARD	GERALD	... is ....
(c)	JAN-JAAP	JAN JAAP	... is ....
(d)	X12	X-12	... is ....
(e)	Basic	BASIC	... is ....
(f)	95.2	95,2	

(a) C$ is kleiner dan D$ (D$ heeft meer tekens dan C$, verder zijn de strings gelijk)

(b) C$ is groter dan D$ (R heeft een grotere ASCII-waarde dan L)

(c) C$ is groter dan D$ (minteken oftewel koppelteken is groter dan spatie)

(d) C$ is groter dan D$ (cijfer 1 is groter dan minteken)

(e) C$ is groter dan D$ (kleine letters zijn groter dan hoofdletters)

(f) C$ is groter dan D$ (punt is groter dan komma)

De stringfuncties ASC en CHR$

Bij het manipuleren met ASCII-code wordt veel gebruik gemaakt van twee zogenaamde stringfuncties, ASC en CHR$. Bij de ASC-functie wordt tussen de argumenthaakjes een string of stringvariabele opgegeven. De functie levert het ASCII-codegetal van het eerste teken van de desbetreffende string. Dit codegetal is een gewone numerieke waarde en kan op dezelfde wijze als andere getallen in toekenningsopdrachten (LET), PRINT-opdrachten, IF...THEN-vergelijkingen, enz. worden gebruikt.

voorbeelden:

```
LET  X = ASC(A$)
LET  X = ASC("ANTWERPEN")
PRINT ASC(A$)
IF  ASC(N$) = 0 THEN ...
```

Toetsvragen

1. Geef het ASCII-codegetal dat naar aanleiding van onderstaande programmaregels wordt geproduceerd.

(a)
```
10  LET D$ = "DOLLAR"
20  PRINT  ASC(D$)

RUN
.....
```

(b)
```
10  PRINT ASC("NEE")

RUN
.....
```

(c)
```
10  PRINT ASC(" ")

RUN
.....
```

(d)
```
10  LET V$ = "FRANK"
20  LET A$ = "JONES"
30  LET N$ = A$ + ", " + V$
40  PRINT ASC(V$)
50  PRINT ASC(A$)
60  PRINT ASC(N$)

RUN
.....
.....
```

(a) 68 (b) 78 (c) 32 (d) 70 74 74

2. Geef aan welke string aan A$ moet worden toegekend, wil de voorwaarde in de IF...THEN waar zijn.

(a) IF ASC(A$) = 53 THEN 510 .....
(b) IF ASC(A$) <> 48 THEN 810 .....

(a) Eerste teken in A$ moet een 5 zijn.
(b) Eerste teken in A$ mag alles behalve nul zijn.

De functie CHR$ is de tegenhanger van de ASC-functie. Als argument krijgt CHR$ een ASCII-codegetal. De functie levert een ASCII-teken, dat bijvoorbeeld kan worden afgedrukt of als speciaal besturingsteken naar het scherm of randapparatuur kan worden gestuurd. Dat laatste is het geval als de waarde van het codegetal tussen 0 en 31 ligt. Op de TRS-80 kunnen bepaalde codegetallen ook voor grafische tekens worden gebruikt (waarden 129 t/m 191, zie de TRS-80 gebruikershandleiding). De CHR$-functie kan ook in PRINT-opdrachten worden toegepast om het teken dat met een bepaald codegetal overeenkomt af te drukken, bijvoorbeeld:

```
840 PRINT CHR$(72); CHR$(79); CHR$(73); CHR$(33)
```

Toetsvraag

(a) Wat is het resultaat van bovenstaande PRINT-opdracht? (Voer de opdracht eventueel op de computer uit of raadpleeg de ASCII-tabel.)

(a) HOI!

Bijzondere codegetallen

Tenslotte nog enkele bijzondere codegetallen. De meeste micro's kunnen minstens één audiosignaal produceren (een zoemtoon). Daarvoor wordt volgens ASCII het codegetal 7 (BEL) gebruikt, dus:

```
PRINT CHR$(7)
```

Andere akoestische signalen kunnen mogelijk met andere codegetallen worden geproduceerd, maar dit is sterk afhankelijk van het type computer. De 'toeters en bellen' van hoofdstuk 1 kunnen in dit verband dus min of meer letterlijk worden opgevat ...

Een codegetal om te onthouden is 34. Daarmee kunnen aanhalingstekens worden geproduceerd in situaties (vooral aan weerszijden van strings) waar die anders niet zouden verschijnen. Ook andere codegetallen kunnen interessant zijn. Raadpleeg dus uw gebruikershandleiding, vooral wat betreft de ASCII-codes 0 t/m 31. Wellicht geeft dat leuke dingen te zien (en te horen)!

2.10 DE LEN-FUNCTIE

Soms kan het in een programma nodig zijn de lengte van een string te bepalen. Zo'n string kan al dan niet aan een stringvariabele zijn toegekend. De lengte kan worden bepaald door middel van de functie LEN (afkorting van length = lengte), die als argumentwaarde een string of stringvariabele heeft. Omdat de functie een numerieke waarde levert kan hij zonder meer worden gebruikt in numerieke toekenningen, berekeningen, of als vergelijkingswaarde in een IF...THEN. De functie kan uiteraard ook vanuit een PRINT-opdracht worden aangeroepen.

voorbeelden:

1.
```
10 LET S$ = "DIT IS EEN STRING"
20 PRINT LEN(S$)

RUN
17
```

2.
```
100 PRINT LEN("DIT IS OOK EEN STRING")

RUN
21
```

3.
```
10  LINE INPUT "GEEF EEN TEKST OP: "; T$
20  LET N = LEN(T$)
30  PRINT N

RUN
GEEF EEN TEKST OP: 1, 2, 3, 4 KOMT ER NOG WAT VAN?
31
```

4.
```
150 LET  A$ = "JA"
160 IF LEN(A$) = 2 THEN PRINT "GA NAAR VOLGENDE VRAAG"

RUN
GA NAAR VOLGENDE VRAAG
```

5.
```
10  LET M$ = "EERSTE STRING"
20  LET N$ = "TWEEDE STRING"
30  PRINT LEN(M$) + LEN(N$)

RUN
26
```

Toetsvraag

Geef de uitvoer behorende bij de volgende programmaregels:

(a)
```
10 LET C$ = ""
20 PRINT LEN(C$)

RUN
.................
```

(b)
```
10  LET  V$ = "GERARD"
20  LET  A$ = "VAN DEN BOSCH"
30  LET  N$ = A$ + ", " + V$
40  PRINT N$
50  PRINT LEN(N$)

RUN
.............................
.............................
```

(a) 0 (b) VAN DEN BOSCH, GERARD
21

2.11 DE SUBSTRINGFUNCTIES MID$, LEFT$ en RIGHT$

Onder 'substring' verstaan we een deel van een string. Van de string "BLABLA" is bijvoorbeeld "BLAB" of "LAB" een substring. MSX BASIC kent een aantal functies die betrekking hebben op het werken met zulke substrings, waaronder MID$, LEFT$ en RIGHT$. We beperken ons voorlopig tot de eerste - en naar onze mening ook de belangrijkste - van de drie, namelijk MID$.

De MID$-functie maakt het mogelijk willekeurige substrings te selecteren.
Enkele voorbeelden:

1. MID$("BLABLA", 2, 3)
   De functie selecteert 3 tekens vanaf het tweede teken van de string (L), en levert dus de substring "LAB".
2. MID$(T$, 3, 15)
   Aannemende dat aan T$ een string van minstens 17 tekens (= 3 + 15 - 1) is toegekend, selecteert MID$ vanaf de derde stringpositie 15 tekens als substring.
3. MID$(D$, 10)
   In deze aanroep is de lengte van de te selecteren substring (dat wil zeggen de derde waarde binnen de haakjes) weggelaten. De functie levert in dit geval de gehele resterende substring vanaf positie 10.
4. MID$(W$, A, C * D)
   Uit dit voorbeeld blijkt dat de beginpositie en ook de lengte van de substring als een numerieke variabele of expressie opgegeven kan worden. Uiteraard moeten de gebruikte variabelen – net als de stringvariabele – van te voren een waarde hebben gekregen.

Uit deze voorbeelden kunnen we de volgende algemene vorm van de MID$-functie 'destilleren':

```
MID$(s, b, l)
```

waarbij:
- s een stringvariabele of -constante is;
- b de beginpositie is van de te selecteren string;
- l de lengte is van de te selecteren substring.

Zoals uit voorbeeld 3 bleek, mag de derde parameter (l) achterwege blijven. In dat geval wordt de gehele resterende substring vanaf positie b geselecteerd. Merk op dat l niet expliciet de laatste stringpositie aangeeft, maar het aantal tekens waaruit de te vormen substring moet bestaan. Dit geeft soms aanleiding tot misverstanden.

Wijzigen van strings met MID$

Behalve voor het selecteren kan de MID$-functie ook worden gebruikt voor het wijzigen van strings.

voorbeeld:

```
LET MID$(N$, 2, 6) = B$
```

met N$ = "SSSSSSSSSS" en B$ = "XXXXXX"

Ten gevolge van deze opdracht worden 6 tekens van de aan N$ toegekende string vanaf positie 2 vervangen door de aan B$ toegekende string. N$ wordt dus "SXXXXXXSSS".

Bevat B$ minder dan 6 tekens, dan wordt de string in N$ slechts door het kleinere aantal tekens overschreven. De rest van de string blijft onveranderd.

voorbeeld:

B$ = "XXX" in plaats van "XXXXXX" zou met bovenstaande functieaanroep en met N$ wederom "SSSSSSSSSS" de waarde N$ wijzigen in "SXXXSSSSSS".

Heeft B$ meer tekens dan de N$-string vanaf de aangegeven positie lang is, dan wordt de N$-string eenvoudig uitgebreid.

voorbeeld:

```
LET MID$(N$, 7, 6) = B$
```

met N$ = "SSSSSSSSSS" en B$ = "XXXXXX"
levert: N$ = "SSSSSSXXXXXX".

Voorwaarde hierbij is wel dat de maximale lengte van een string (255 tekens) niet wordt overschreden.

Nog een voorbeeld:

```
LET MID$(G$, 5, 5) = MID$(B$, 8, 5)
```

Door deze opdracht worden vanaf positie 5 in G$ 5 tekens vervangen door de substring van 5 tekens die op positie 8 in B$ begint. De string in G$ wordt dus veranderd, terwijl de string in B$ intact blijft.

Tenslotte: MID$ kan niet worden gebruikt om gegevens op te nemen in een nulstring, of in een variabele waaraan in het geheel nog geen waarde is toegekend. De volgende constructie zou dus een foutconditie opleveren omdat de lengte van X$ nul is:

```
210 LET X$ = ""
220 LET MID$(X$, 1, 3) = "MSX"
```

Toetsvragen

Geef de waarde van Z$ na uitvoering van de volgende opdrachten:

(a) `160 LET MID$(K$,8,32) = "AANNEMINGSBEDRIJF ROTHUIZEN B.V."`

K$ vóór uitvoering: KLANT: AAA
K$ na uitvoering: .....

(b)
```
190 LET X$ = "ZANDWEG 13"
200 LET MID$(A$, 8, 10) = X$
```

A$ vóór uitvoering: ADRES: BBB
A$ na uitvoering: .....

(c)
```
300 LET X = 13 : LET Y = 7
310 LET P$ = "1012 UH"
320 LET MID$(W$, X, Y) = P$
```

W$ vóór uitvoering: WOONPLAATS: CCCC DD ROTTERDAM
W$ na uitvoering: .....

(d)
```
400 INPUT "GEEF TELEFOONNUMMER: "; N$
410 LET MID$(T$, 11, 10) = N$
```

T$ vóór uitvoering: TELEFOON: EEE-FFFFFF
respons op INPUT : 354782
T$ na uitvoering: .....

(a) KLANT: AANNEMINGSBEDRIJF ROTHUIZEN B.V.
(b) ADRES: ZANDWEG 13
(c) WOONPLAATS: 1012 UH ROTTERDAM
(d) TELEFOON: 354782FFFF
(Merk op dat het resterende deel van de string zijn oorspronkelijke waarde blijft behouden. Uit dit voorbeeld blijkt dus tevens het nut van goede 'prompts'.)

Nog een selectievoorbeeld:

```
150 LET N$ = "JAN-JAAP VAN HORSSEN"
160 PRINT MID$(N$, 1, 8)
170 PRINT MID$(N$, 10, 11)
180 PRINT N$

RUN
JAN-JAAP
VAN HORSSEN
JAN-JAAP VAN HORSSEN
```

In tegenstelling tot gevallen waarin MID$ als vervangingsfunctie wordt toegepast, blijft hier de waarde van de oorspronkelijke string onveranderd. Er heeft immers geen toekenning plaatsgevonden.

Toetsvraag

Gegeven het volgende programmagedeelte:

```
150 LET N$ = "INGE GEERLING-ENGELBARTS"
160 LET V$ = MID$(N$, 1, 4)
170 LET A$ = MID$(N$, 6, 19)
180 LET M$ = MID$(N$, 15, 10)
190 PRINT "VOORNAAM   : "; V$
200 PRINT "ACHTERNAAM : "; A$
210 PRINT "MEISJESNAAM: "; M$
```

(a) In welke uitvoer resulteert dit?

(b) Welk teken van N$ wordt niet opgenomen in V$, A$ of M$?

(c) Wat is de waarde van N$ na afloop van dit gedeelte?

(a) VOORNAAM : INGE
ACHTERNAAM: GEERLING-ENGELBARTS
MEISJESNAAM: ENGELBARTS

(b) De spatie in positie 5 van N$

(c) INGE GEERLING-ENGELBARTS
(onveranderd)

De functies LEFT$ en RIGHT$

Een deel van de taak van de MID$ – te weten het selecteren van substrings – kan ook worden uitgevoerd door de functies LEFT$ en RIGHT$. De aanroep van deze functies heeft de volgende vorm:

LEFT$(s, n)

en

RIGHT$(s, n)

waarbij s een stringvariabele of -constante is, en n het aantal te selecteren tekens.

voorbeelden:

```
160 PRINT LEFT$(A$, 8)
```

De eerste (dat wil zeggen de linker) 8 tekens van de aan A$ toegekende string worden afgedrukt.

```
170 LET R  = 12
180 LET B$ = RIGHT$(A$, R)
```

De laatste (dat wil zeggen de rechter) 12 tekens van de aan A$ toegekende string wordt aan B$ toegekend.

In tegenstelling tot de MID$-functie kunnen LEFT$ en RIGHT$ uitsluitend voor het selecteren van substrings worden gebruikt. Ze zijn dus niet zo veelzijdig als MID$, en worden in dit boek daarom weinig toegepast. LEFT$ wordt wel gebruikt bij het testen op 'ja/neen' antwoorden naar aanleiding van INPUT-prompts, bijvoorbeeld:

```
240 LINE INPUT "GEBRUIKSAANWIJZING NODIG (J/N)? "; G$
250 IF LEFT$(G$, 1) = "J" THEN 600
```

Het voordeel van zo'n constructie is dat een gebruiker zowel met bijvoorbeeld J, JA, N, NEE, NEEN, enz. kan antwoorden, zonder het goede verloop van het programma te verstoren. De beginletter is immers de enige die telt.

Hoewel RIGHT$ minder wordt toegepast dan MID$ en LEFT$, toch een praktisch voorbeeld:

```
240 LINE INPUT "IN WELK JAAR BENT U GEBOREN? "; J$
250 PRINT "GEBOORTEJAAR: 19"; RIGHT$(J$, 2)
```

Toetsvraag

Geef aan hoe de computer bij bovenstaand programmagedeelte op de volgende antwoorden zou reageren:

(a) Gebruiker antwoordt: 1938
    Uitvoer : .....

(b) Gebruiker antwoordt: '64
    Uitvoer : .....

(c) Gebruiker antwoordt: GEBOREN IN 1958
    Uitvoer : .....

(d) Gebruiker antwoordt: ACHTENVEERTIG
    Uitvoer : .....

(a) GEBOORTEJAAR: 1938
(b) GEBOORTEJAAR: 1964
(c) GEBOORTEJAAR: 1958
(d) GEBOORTEJAAR: 19IG

2.12 DE INSTR-FUNCTIE

Ook handig bij het werken met strings is de functie INSTR (afkorting van instring). De functie dient om de plaats aan te geven van een substring binnen een andere string. De te onderzoeken string wordt in de vorm van een stringconstante of -variabele als eerste

parameter meegegeven, de substring – eveneens in constante- of variabelevorm – als tweede (en laatste) parameter. Als waarde levert INSTR de beginpositie van de substring binnen de te onderzoeken string*. Komt de substring daarin niet voor dan is de functiewaarde nul. Enkele voorbeelden zullen dit verduidelijken.

voorbeeld 1:

```
250 LET X = INSTR ("BLABLA", "LAB")
```

In dit geval is "BLABLA" de te onderzoeken string, en "LAB" de gezochte substring. De functie levert als waarde 2, omdat "LAB" inderdaad in "BLABLA" voorkomt en wel vanaf positie 2. Het coderen van bijvoorbeeld de substring "ABC" in plaats van "LAB" zou als functiewaarde nul leveren. Deze substring komt immers niet in de te onderzoeken string voor.

voorbeeld 2:

```
10  LET  A$ = "AMSTERDAM"
20  LET  B$ = "STER"
30  LET  C$ = "DAM"
40  LET  X  = INSTR (A$, B$)
50  PRINT  X
60  LET  Y  = INSTR (A$, C$)
70  PRINT  Y
```

In dit voorbeeld twee aanroepen van INSTR met stringvariabelen als parameters. Aan u de (toets)vraag: welke uitvoer wordt naar aanleiding hiervan geproduceerd?

Antwoorden: 3 (naar aanleiding van eerste PRINT);
7 (naar aanleiding van tweede PRINT).

voorbeeld 3:

```
250 LET  M$ = "JANFEBMRTAPRMEIJUNJULAUGSEPOKTNOVDEC"
260 LINE INPUT "WELKE MAAND? "; M1$
270 LET  L  = INSTR (M$, M1$)
280 PRINT "POSITIE: "; L
290 GOTO 260

RUN
WELKE MAAND? MEI
POSITIE: 13
WELKE MAAND? AP
POSITIE: 10
WELKE MAAND? OKTOBER
POSITIE: 0
WELKE MAAND? A
POSITIE: 2
```

---

* Merk op dat de functie INSTR in tegenstelling tot MID$, LEFT$ en RIGHT$ geen $-teken achter de naam heeft, ondanks het feit dat de functie betrekking heeft op strings. De reden hiervoor is dat INSTR een numerieke waarde oplevert. Naar analogie van numerieke variabelen krijgt de naam dan geen $-teken. De andere genoemde functies leveren wel een stringwaarde en krijgen daarom – naar analogie van stringvariabelen – wel een $-teken.

Merk op dat zelfs met twee letters (AP) de maand APR gevonden wordt. "OKTOBER" komt in M$ niet voor en levert dus de functiewaarde nul op. Het intypen van A resulteert in de functiewaarde 2, omdat dit de eerste plaats is waar deze letter in M$ voorkomt.

Soortgelijke technieken kunnen ook zonder INSTR worden toegepast om gewenste of ongewenste tekens in een string te zoeken. In het volgende programmagedeelte wordt bijvoorbeeld met behulp van de eerder besproken functies MID$ en LEN het einde van een substring bepaald die afgesloten wordt door een spatie.

```
740 LET N$ = "PIET HEIN"
750 FOR S = 1 TO LEN(N$)
760     IF MID$(N$, S, 1) = " " THEN 780
770 NEXT S
780 PRINT MID$(N$, 1, S-1)
```

Toetsvragen naar aanleiding van bovenstaand programmagedeelte

(a) Wat is de maximale waarde van de FOR-besturingsvariabele S?

(b) Wat is de lengte van de door MID$ geselecteerde substring in regel 760?

(c) In welke positie van de aan N$ toegekende waarde bevindt zich de spatie?

(d) Waarom wordt in regel 780 de uitdrukking S - 1 gebruikt in plaats van S?

(e) Welke uitvoer wordt geproduceerd naar aanleiding van regel 780?

(a) LEN(N$) = 9
(b) één teken
(c) positie 5
(d) S geeft op dat moment niet de positie van het laatste teken van de gezochte substring aan, maar van de spatie
(e) PIET

2.13 'ON...GOTO'-OPDRACHT

We weten dat we met een gewone GOTO een 'sprong' kunnen uitvoeren. Ten gevolge van zo'n sprong wordt de normale volgorde van het uitvoeren van opdrachten doorbroken. Er ontstaat als het ware een andere tak binnen het programma. Anders gezegd: de besturingsstroom van het programma wordt vertakt. Door een GOTO-opdracht op te nemen in een IF-opdracht (al dan niet met het sleutelwoord GOTO) krijgen we een *voorwaardelijke* sprong, dat wil

zeggen de sprong wordt pas daadwerkelijk uitgevoerd als er voldaan is aan de in de IF genoemde voorwaarde.

Zetten we deze gedachtengang door, dan komen we uit bij de 'ON... GOTO'-opdracht. Met deze opdracht kunnen we afhankelijk van een (beperkte) voorwaarde naar verschillende plaatsen in het programma springen. Vanuit de ene 'ON...GOTO'-opdracht ontstaan dan verschillende programmatakken.

De ON...GOTO heeft de volgende algemene vorm:

```
ON a GOTO r1, r2, r3, ..., rn
```

waarbij a een numerieke waarde voorstelt (variabele of expressie), en r1 t/m rn regelnummers voorstellen. De waarde van a moet in principe tussen 1 en n liggen. Heeft a de waarde 1 dan wordt gesprongen naar regel r1, heeft a de waarde 2 dan naar regel r2, enz. Het aantal regelnummers dat kan worden opgegeven is slechts beperkt door de maximale lengte van de opdrachtregel.

Een concreet voorbeeld:

```
ON X GOTO 310, 450, 660, 660, 660, 720, 830, 910
```

Heeft X bij uitvoering van deze opdracht de waarde 1 dan wordt gesprongen naar regel 310. Voor X = 2 wordt gesprongen naar regel 450, voor X = 3, 4 of 5 naar regel 660, enz.

Is de waarde van X nul of groter dan het aantal opgegeven regelnummers, dan wordt de desbetreffende ON...GOTO overgeslagen. De computer geeft een foutmelding en staakt vervolgens de uitvoering van het programma, als X een negatieve waarde heeft (illegal function call).

Een praktische toepassing van de ON...GOTO is het geval dat de gebruiker van een programma verschillende mogelijkheden geboden wordt. Hij kan dan als het ware een keuze maken uit een 'menu'.

voorbeeld:

```
250  REM *** DEMONSTRATIE MENU-PROGRAMMA
260  REM
270          LINE INPUT "GEBRUIKSAANWIJZING NODIG (J/N): "; G$
280          IF  G$ <> "J" THEN 500
290  REM
300  REM     PRESENTEER MENU
310  REM
320          CLS :      REM:  MAAK SCHERM SCHOON
330  REM
340          PRINT "DIT PROGRAMMA BIEDT DE VOLGENDE MOGELIJKHEDEN: "
350          PRINT "A. GEGEVENS WEGSCHRIJVEN NAAR NIEUW BESTAND"
360          PRINT "B. TOEVOEGEN AAN REEDS AANWEZIG BESTAND"
370          PRINT "C. MUTEREN VAN REEDS AANWEZIG BESTAND"
380          PRINT "D. UITLIJSTEN VAN REEDS AANWEZIG BESTAND"
390          PRINT "E. SELECTEREN VAN GEGEVENS UIT BESTAND"
```

```
400  REM
410          LINE INPUT "MAAK UW KEUZE (A-E): "; K$
420          ON  INSTR ("ABCDE", K$) GOTO 500, 600, 700, 800, 900
430              PRINT "*** INVOERFOUT ***"
440              PRINT "U MOET EEN VAN DE LETTERS A T/M E INVOEREN"
450              GOTO 410
460  REM
:
:
```

Merk het gebruik op van de eerder besproken INSTR-functie om de ingevoerde keuzewaarde te converteren naar een (numerieke) selectiewaarde voor de ON...GOTO. Dit kan ook met de ASC-functie gebeuren, bijvoorbeeld:

```
400  REM
410          LINE INPUT "MAAK UW KEUZE (A-E): "; K$
420          LET K = ASC (K$) - 64
430          IF K < 1 OR K > 5 THEN 450
440          ON K GOTO 500, 600, 700, 800, 900
450             PRINT "*** INVOERFOUT ***"
:
:
```

Toetsvragen

(a) Welke waarde krijgt K wanneer de letter C wordt ingevoerd?

(b) Waarom is regel 430 in de alternatieve programmaconstructie opgenomen?

(a) 3. (De ASCII-waarde van de letter C is 67. Deze waarde wordt geleverd door de aanroep ASC (K$) in regel 420.)

(b) Door een illegale keuzewaarde op te geven zou K buiten het bereik 1 t/m 5 kunnen vallen, waardoor de ON...GOTO niet correct uitvoerbaar zou zijn. Regel 430 voorkomt dit.

2.14 FOR/NEXT-CONSTRUCTIES

Bij het herhaald uitvoeren van opdrachten onder besturing van een lusvariabele verdient het aanbeveling gebruik te maken van FOR/NEXT-opdrachten.

De constructie:

```
100  FOR X = 1 TO N
110      PRINT X, X^2
120  NEXT X
```

verdient dus de voorkeur boven de constructie:

```
100  LET X = 1
110  PRINT X, X^2
120  LET X = X + 1
130  IF X <= N THEN 110
```

De FOR/NEXT-oplossing is duidelijk korter (zeker als de test in regel 130 omgedraaid zou worden), ziet er beter uit en is gemakkelijker te lezen.

Probeer bij het gebruik van FOR/NEXT-constructies het voortijdig verlaten van de lus te voorkomen. Zo'n voortijdige beëindiging komt bij wijze van voorbeeld in het volgende programmadeel voor:

```
100  FOR  X = 1 TO N
110       IF  A(X) = B(X)  THEN 200
120  NEXT X
```

Er zijn twee redenen om van dergelijke constructies liever geen gebruik te maken. Ten eerste kan de waarde van de lusvariabele (in bovenstaand voorbeeld X) bij het verlaten van de lus ongedefinieerd raken, dat wil zeggen een waarde krijgen (of hebben) die anders is dan hij logischerwijze bij het verlaten van de lus zou moeten hebben.* Het gebruik van die variabele – in het vertrouwen dat hij de verwachte waarde heeft – kan dan tot hoogst onplezierige verrassingen leiden waarvan de oorzaak vaak moeilijk te achterhalen is. De tweede reden om FOR/NEXT-lussen niet voortijdig te verlaten is dat daardoor de programmastructuur wordt aangetast. Er komt wéér een sprong bij, en het programma wordt dus ook een tikkeltje moeilijker te volgen. Meestal kan het programma toch zo worden geschreven dat de gewenste bewerkingen binnen de lus worden uitgevoerd in plaats van daarbuiten (uitzonderingen daargelaten).

voorbeeld:

slecht:
```
100  FOR  X = 1 TO N
110       IF A(X) = B(X) THEN 130
120  NEXT X
130  LET  S = S + 1
140  GOTO 120
```

noot: Deze constructie – hoewel toegestaan – kan als bijzonder 'smerig' worden aangemerkt omdat behalve het voortijdig verlaten ook nog wordt teruggesprongen in de lus.

---

* Opmerking van de vertaler/bewerker:
Dit hangt samen met gebruik van zogenaamde registers in plaats van 'gewone' geheugenlokaties voor het bijhouden van de waarde van lusvariabelen. Hierbij wordt de 'teller'-waarde niet steeds tussen werkgeheugen en CPU heen en weer getransporteerd, maar op efficiënte wijze in de CPU zelf bijgehouden. Bij beëindiging – vooral voortijdige beëindiging – van de lus is het mogelijk dat de registerinhoud niet naar de desbetreffende geheugenlokatie wordt getransporteerd.

```
goed:  100  FOR  X = 1 TO N
       110       IF A(X) <> B(X)  THEN 130
       120          LET S = S + 1
       130  NEXT X
```

Springen naar een subroutine door middel van een GOSUB is overigens wel acceptabel. Bij uitvoering van de RETURN wordt de uitvoering van de lus immers op de normale wijze voortgezet.

Toetsvraag

(a) Schrijf met behulp van geneste FOR/NEXT-opdrachten een programma dat tot gevolg heeft dat op verschillende regels drie keer het woord "HALLO" wordt afgedrukt, waarbij na elke "HALLO" vier keer het woord "DAG" verschijnt (eveneens op verschillende regels).

```
(a)  10  FOR  X = 1 TO 3
     20       PRINT "HALLO"
     30       FOR  Y = 1 TO 4
     40            PRINT "DAG"
     50       NEXT Y
     60  NEXT X
```

2.15 MULTI-OPDRACHTREGELS

Hoewel de opdracht nuttig kan zijn heeft hij ook zijn slechte kanten. Houd daarom bij het gebruik ervan rekening met het volgende.

1. Multi-opdrachtregels zijn vaak slecht leesbaar, en soms ook moeilijk te begrijpen. Slechte leesbaarheid kan u – of uw collega-programmeur – vooral parten spelen als het programma in een later stadium moet worden veranderd. Het coderen van één opdracht per regel is duidelijker.
2. Wilt u perse multi-opdrachtregels gebruiken, zorg dan dat per regel zo mogelijk één afgeronde procedure of actie wordt aangegeven. Het 'logisch' voorzetten van een multi-opdrachtregel op andere regels werkt verwarrend.
3. Het lokaliseren van fouten in multi-opdrachtregels is betrekkelijk moeilijk.
4. Pas op met 'IF...THEN'-opdrachten in multi-opdrachtregels. Eventuele opdrachten na de IF...THEN die op dezelfde regel staan worden alleen dan uitgevoerd als aan de IF-voorwaarde is voldaan. Ze vormen als het ware een 'blok' dat als geheel (samen met de opdracht direct na THEN) wordt uitgevoerd als de voorwaarde waar is (zie § 2.8).

voorbeeld:

```
200  REM   VERWISSEL INHOUD A EN B INDIEN A < B
210  REM
220        IF  A < B  THEN  C = A : A = B : B = C
230  ...
```

Toelichting. De drie toekenningsopdrachten worden alle uitgevoerd indien A kleiner is dan B. Is A groter dan of gelijk aan B dan worden alle toekenningsopdrachten overgeslagen. De programma-uitvoering wordt voortgezet bij regel 230.

Het toepassen van multi-opdrachtregels is dus niet slechts een kwestie van opdrachten die anders op verschillende regels zouden staan onderbrengen in één fysieke regel. Ook de uitvoering wordt beïnvloed, met alle gevaren van dien.

5. REMARK-opdrachten moeten in multi-opdrachtregels uiteraard altijd als laatste worden opgenomen. Alle tekst na het woord REM – ook uitvoerbare BASIC-opdrachten – wordt immers als commentaar opgevat.

Toetsvragen

(a) Gegeven de constructie:

```
200  IF X = Y  THEN PRINT  "U HEBT GEWONNEN!" : GOTO 10
210  PRINT  "VERKEERD NUMMER" : GOTO 60
```

Welke opdracht(en) wordt/worden uitgevoerd als X ongelijk is aan Y?

(b) Neem aan dat aan de voorwaarde in regel 120 van het onderstaande programmadeel is voldaan. De GOSUB wordt dus uitgevoerd. Bij welke opdracht wordt de uitvoering van het programma voortgezet na terugkeer uit de subroutine?

```
120  IF  X = 2  THEN  GOSUB 510 : GOTO 360
130  PRINT  "X ONGELIJK TWEE"
```

(a) De PRINT-opdracht in regel 210
(b) GOTO 360

2.16 TOETSVRAGEN

1. Bij numerieke vergelijkingen (testen) in 'IF...THEN'-opdrachten kan het beter zijn de test op 'groter dan' of 'kleiner dan' te baseren, in plaats van op gelijkheid. Waarom?

2. Wanneer zijn bij strings in DATA-opdrachten aanhalingstekens nodig?

3. Hoe kan een nulstring (lege string) aan een variabele in een INPUT- of een 'LINE INPUT'-opdracht worden toegekend?

4. Geef het resultaat van het uitvoeren van het volgende programmadeel:

```
10  LET  A$ = "ALFRED"
20  LET  B$ = "CONTRACT"
30  LET  C$ = "32C"
40  PRINT  ASC(A$); ASC(B$); ASC(C$)
```

5. Welke strings moeten vooraf (successievelijk) aan D$ zijn toegekend als de voorwaarden in onderstaande 'IF...THEN'-opdrachten waar zijn?

   (a) IF ASC(D$) < 48 OR ASC(D$) > 57 THEN 660
   (b) IF ASC(D$) > 64 AND ASC(D$) < 91 THEN GOSUB 1520

6. Welke waarde zal de LEN-functie leveren voor een string die uit 15 spaties bestaan?

7. Geef het resultaat van het uitvoeren van het volgende programmadeel:

```
10  LET  M$ = "STAR TREK"
20  LET  N$ = "WARS"
30  LET  G$ = MID$(M$, 1, 5) + N$
40  LET  MID$(M$, 6, 4) = MID$(N$, 1, 4)
50  PRINT  G$
60  PRINT  M$
```

8. Geef een voorbeeld van een gewone numerieke variabele en van een gewone stringvariabele.

9. Geef minstens één reden voor het vermijden van multi-opdrachtregels.

10. Welke opdracht wordt in het onderstaande programmadeel na uitvoering van de GOSUB en de bijbehorende subroutine uitgevoerd?

```
120  IF  X > 10  THEN  GOSUB 810 : GOTO 110
130  PRINT  "GROETJES VAN REGEL 130"
```

11. Als de naam van een variabele uit meer dan twee alfanumerieke tekens bestaat, hoeveel van die tekens zijn dan voor de computer van werkelijke betekenis?

2.17 ANTWOORDEN OP DE TOETSVRAGEN

1. Ten gevolge van afrondingsfouten bij het uitvoeren van berekeningen kunnen zeer kleine afwijkingen ontstaan ten opzichte van het theoretisch verwachte resultaat. Daardoor kan een test op gelijkheid ten onrechte resulteren in 'onwaar'.

2. Aanhalingstekens zijn nodig als de stringwaarde een komma of voorafgaande spaties bevat.

3. Door zondermeer de ENTER- c.q. RETURN-toets in te drukken.

4. 65 67 51

5. (a) Het eerste teken van D$ mag geen cijfer zijn (0 t/m 9).
   (b) Het eerste teken van D$ moet een hoofdletter zijn (A t/m Z).

6. 15 (spaties worden als tekens beschouw).

7. STAR WARS
   STAR WARS

8. Numerieke variabele : A (of een andere letter), eventueel gevolgd door een cijfer. Stringvariabele : A$ (of voorafgaande aan het $-teken een andere letter), eventueel met een cijfer direct vóór het $-teken.

9. Meer dan één opdracht per regel maakt het programma moeilijker te lezen. Fouten zijn moeilijker te vinden. Bij 'IF...THEN'-opdrachten kunnen gemakkelijk vergissingen worden gemaakt. (Eén van deze antwoorden)

10. GOTO 110

11. In MSX BASIC slechts de eerste twee tekens.

3 GEGEVENSINVOER EN -CONTROLE MODULES

3.0 DOELSTELLINGEN

Na het bestuderen van dit hoofdstuk moet u in staat zijn een gegevensinvoermodule samen te stellen. Met de opdrachten die u daarbij gebruikt moeten de ingevoerde gegevens op de volgende punten gecontroleerd kunnen worden:

- juiste lengte
- geen respons (nulstring)
- gegevenstype (numeriek of alfanumeriek)
- verkeerde tekens
- parameterwaarden bij numerieke gegevens.

Bovendien moet u gegevensinvoermodules kunnen schrijven die:

- duidelijke prompts genereren
- gebruik maken van goed ingerichte gegevensvelden
- gegevens kunnen samenvoegen tot één enkel gegevensveld
- gegevens zonodig aanvullen tot de juiste veldlengte
- overbodige spaties uit een gegevensveld verwijderen
- onderdelen van een gegevensveld vervangen
- volledige tekst en uitleg aan de gebruiker geven als hij fouten maakt.

3.1 INLEIDING

In dit hoofdstuk houden we ons bezig met de invoer van gegevens. Dit terrein hebben we in de voorgaande hoofdstukken al enigszins 'voorgeploegd', met name bij de behandeling van de modulaire opbouw van programma's (hoofdstuk 1, §§ 1.5 en 1.9) en van de 'INPUT'- en de 'LINE INPUT'-opdrachten in het vorige hoofdstuk (§§ 2.5 en 2.6). Voor we hiermee verder gaan eerst enkele definities.

Gegevens, gegevensinvoer en gegevensbestanden

Onder *gegevens* verstaat men strikt genomen de invoergrootheden waarop een programma opereert, en die door dat programma tot (zinvolle) informatie worden getransformeerd. De gegevens vormen als het ware de grondstoffen die door een produktieproces – een samenspel tussen de apparatuur en de programmatuur – worden verwerkt tot een eindprodukt: *informatie*. In het kader van dit boek zullen we onder gegevens alle grootheden verstaan die als invoer voor een programma kunnen dienen of als uitvoer van een programma zijn ontstaan, en die in zogenaamde gegevensbestanden op fysieke opslagmedia zoals schijfgeheugens of cassettebanden (kunnen) worden opgeslagen.

Zulke gegevens kunnen bijvoorbeeld bestaan uit:
- adreslijsten, ledenlijsten of rekeningoverzichten
- inventarislijsten van winkel- of magazijnartikelen
- boekhoudkundige of administratieve gegevens
- titels van boeken, geluidsopnamen, tijdschriftartikelen
- aantekeningen voor een publikatie
- resultaten van metingen of experimenten
- resultaten van statistische steekproeven, enz.

Onder *gegevensinvoer* verstaan we het proces van het aanbieden van de te verwerken gegevens aan de computer. De invoer vindt plaats – zoals gezegd – onder besturing van invoeropdrachten zoals INPUT en LINE INPUT.

Gegevens worden op fysieke opslagmedia zoals schijfgeheugens (bijvoorbeeld floppy disks) en cassettes ondergebracht in de vorm van gegevensbestanden. Onder *gegevensbestanden* (Engels: data files) verstaan we een verzameling bij elkaar horende invoergegevens die zo zijn gestructureerd en georganiseerd dat ze door een computer kunnen worden gemanipuleerd. Dat wil zeggen de computer moet (via het programma) een willekeurige gegevensplaats binnen het bestand kunnen benaderen, en het gegeven dat zich daar bevindt kunnen kopiëren naar het werkgeheugen (lezen), kunnen overschrijven met een ander gegeven of verwijderen.

Het belang van goede gegevensinvoer

Om goed bruikbaar te zijn moet een gegevensbestand in de eerste plaats betrouwbaar zijn. Dat wil zeggen: de daarin opgenomen gegevens moeten nauwkeurig zijn (zowel inhoudelijk als wat hun vorm betreft) en ze moeten geldigheid bezitten. We spreken hierbij van *gegevensintegriteit*. De nauwkeurigheid van de programma-uitvoer

– de informatie – staat of valt met de integriteit van de gebruikte gegevens.

Onjuiste of ongeldige gegevens kunnen bij verwerking bovendien tot gevolg hebben dat het programma onderbroken, gestopt of abnormaal beëindigd wordt. Treedt zo'n onverwachte storing op dan kan er met de in bewerking zijnde gegevens van alles gebeuren: half-afgemaakte regeldrukkeruitvoer (bijvoorbeeld rapporten of verslagen), verloren gaan of vernietiging van gegevens, onbruikbaar worden van bestanden door gedeeltelijke verwerking, enz. De gevolgen kunnen rampzalig zijn, hoewel het niet altijd zo'n vaart zal lopen. Dat hangt mede af van de kwaliteit van het programma. Met name de mogelijkheid tot *gegevensverificatie* – het vermogen van het programma om incorrecte gegevens bij voorbaat te signaleren en daarop adequaat te reageren – kan veel ellende voorkomen. En hiermee hebben we u dan hopelijk voldoende gemotiveerd om in dit hoofdstuk even 'pootje te baden' met invoertechnieken, alvorens in het volgende hoofdstuk in het diepe water van het opbouwen en benaderen van bestanden te duiken!

3.2 LENGTE VAN HET GEGEVENSVELD

Om dit aspect van de gegevensinvoer te belichten gaan we uit van een fictief bedrijf dat met behulp van de computer een telefoonlijst wil samenstellen. In de lijst moet van elke medewerker naam, afdeling en telefoonnummer worden vermeld. Deze gegevens moeten naast elkaar als een regel worden afgedrukt.

Een van de eerste beslissingen die we bij het programma-ontwerp moeten nemen – uitgaande van de (zeer realistische) veronderstelling dat het bevoegd bedrijfsgezag het bij deze enigszins vage probleemstelling heeft gelaten – is hoe lang elk gegeven afzonderlijk, en hoe lang de totale regel mag zijn. Met 'lang' bedoelen we in dit verband het aantal tekens waaruit het gegeven of de regel bestaat. Verder gaan we ervan uit dat elke regel een uniforme indeling krijgt, waardoor de overeenkomstige gegevens van de verschillende regels kolomsgewijs onder elkaar komen te staan.

Met het telefoonnummer zullen we de minste moeite hebben. Meestal zal dit bestaan uit een vast aantal tekens, bijvoorbeeld 3. Voor afdelings- en persoonsnamen wordt het moeilijker: beide (in ieder geval persoonsnamen) hebben een variabele lengte. Voor afdelingsnamen zouden we – eventueel in overleg met de opdrachtgever – zonodig afkortingen (liefst met vaste lengte) kunnen invoeren.

Stel dat we tot de volgende regelindeling komen:

1. naam: maximaal 25 tekens
2. twee scheidingsspaties
3. afdelingscode: vaste lengte van 3 tekens
4. twee scheidingsspaties
5. telefoonnummer: vaste lengte van 3 tekens

in totaal dus 35 tekens.

Toetsvraag

Geef een schematisch overzicht van de voorgenomen regelindeling. Gebruik daarbij schuine strepen om de verschillende velden (ook de scheidingsvelden) te markeren, en plaats in elk veld ter identificatie een volgnummer.

```
                    1              2  3   4  5
(a) ........................../../.../../.../
```

Stel dat in de invoermodule de volgende opdrachten voorkomen om de gegevens in 'dialoog' met de gebruiker in te voeren:

```
100  LINE INPUT "NAAM           :  ";N$
110  LINE INPUT "AFDELINGSCODE  :  ";A$
120  LINE INPUT "TELEFOONNUMMER :  ";T$
130  LET R$ = N$ + "  " + A$ + "  " + T$
140  PRINT "REGEL WORDT    :  "
150  PRINT  R$
```

Op het eerste gezicht lijkt dit misschien een gezonde oplossing. De gegevens worden netjes stuk voor stuk ingelezen, en in regel 130 geconcateneerd tot één uitvoerregel, die keurig ter controle wordt afgedrukt. De twee onderstaande RUN's laten zien dat er toch een addertje onder het gras ligt.

(a)
```
RUN
NAAM           :  SMIT, MEJ. A.B.C.
AFDELINGSCODE  :  ADM
TELEFOONNUMMER :  325

REGEL WORDT    :  SMIT, MEJ. A.B.C.  ADM  25
```

(b)
```
RUN
NAAM           :  VAN HIER TOT GUNTER, BARON MR. P.
AFDELINGSCODE  :  DIR
TELEFOONNUMMER :  999

REGEL WORDT    :  VAN HIER TOT GUNTER, BARON MR. P.  DIR  999
```

Toetsvraag

Vul de openstaande plaatsen in om het resultaat van regel 130 in bovenstaand programmadeel weer te geven.

```
(a) R$ = ...../../.../../...
(b) R$ = ...../../.../../...
------------------------
(a) R$ = SMIT, MEJ. A.B.C.  ADM  3/25/.../../...
(b) R$ = VAN HIER TOT GUNTER, BARO/N./ MR/. /P.  DIR  999
```

We zien dus dat deze werkwijze tot verstoringen in de regelindeling leidt als de ingevoerde namen niet precies even lang zijn als van te voren was vastgelegd. Aangezien de lengte van juist een gegeven zoals een persoonsnaam nogal kan variëren is onze methode slecht bruikbaar. In de volgende paragraaf bespreken we een alternatief.

Controle op acceptabele lengte

Het probleem bij de voorgaande programmaconstructie was dat ingevoerde gegevens steeds van dezelfde lengte moesten zijn. Dat hield in dat ze door de gebruiker zonodig tot de juiste lengte moesten worden afgekort of met spaties tot de juiste lengte moesten worden aangevuld. Gebeurde dit niet, dan resulteerde een en ander zonder meer – en in eerste instantie ongemerkt – in een verstoorde lay-out. In dit gedeelte zullen we zien hoe we een programma zo kunnen construeren dat automatisch op de juiste lengte wordt gecontroleerd. Voldoet een gegeven bij invoer niet aan de juiste lengte, dan kan de gebruiker daarop worden geattendeerd. Wat de inhoudelijke kant van het voorbeeld betreft nemen we aan dat de – wellicht enigszins dubieuze – praktijk van het afkorten van persoonsnamen toelaatbaar is.

We demonstreren de lengtecontrole rechtstreeks aan de hand van een programmaconstructie. Het principe is gebaseerd op een aanroep van de LEN-functie als voorwaardedeel van een IF...THEN.

```
100  LINE INPUT "NAAM          : "; N$
102  IF  LEN(N$) > 25  THEN
         PRINT "MAXIMAAL 25 TEKENS. OPNIEUW INVOEREN AUB."  :  GOTO 100

RUN
NAAM          : VAN HIER TOT GUNTER, BARON MR. P.
MAXIMAAL 25 TEKENS. OPNIEUW INVOEREN AUB.
NAAM          :
                ^
```

Toetsvraag

Schrijf soortgelijke opdrachten om de lengte van de andere invoervelden in ons telefoonlijst-voorbeeld te controleren.

```
(a) 110  LINE INPUT "AFDELINGSCODE : "; A$
    112  .....................................
```

```
(b) 120  LINE INPUT "TELEFOONNUMMER : "; T$
    122  ....................................
```

```
(a) 112  IF  LEN(A$) > 3  THEN
             PRINT "MAX. 3 TEKENS. OPNIEUW INVOEREN AUB."  :  GOTO 110

(b) 122  IF  LEN(T$) > 3  THEN
             PRINT "MAX. 3 TEKENS. OPNIEUW INVOEREN AUB."  :  GOTO 120
```

Aanvullen met spaties

Behalve de mogelijkheid dat invoergegevens de toegestane veldlengte overschrijden, bestaat er ook de mogelijkheid dat ze te kort zijn. Ook dat leidt tot onregelmatigheden in de indeling. Om dit te vermijden kunnen we een mechanisme in het programma inbouwen om ingevoerde gegevens zonodig automatisch met spaties tot de juiste lengte aan te vullen. Men noemt dit in het Engels 'padding' (to pad = op- of aanvullen).

Voor de persoonsnamen in ons voorbeeld zouden we daartoe de eerder besproken invoermodule als volgt kunnen uitbreiden:

```
100 LINE INPUT "NAAM"          : "; N$
102 IF   LEN(N$) > 25  THEN
         PRINT "MAX.25 TEKENS. OPNIEUW INVOEREN AUB."  :  GOTO 100
104 IF   LEN(N$) <= 25  THEN LET N$ = N$ + " "         :  GOTO 104
```

Regel 104 wordt hierbij herhaald uitgevoerd, totdat N$ de juiste lengte heeft.

Toetsvraag

(a) Schrijf een gegevensinvoermodule voor het invoeren van een artikelcode van 3 alfanumerieke tekens, gevolgd door een artikelomschrijving van maximaal 20 tekens en een persoonscode van 2 tekens. De persoonscode wordt samengesteld uit de initialen van de gebruiker. Combineer de afzonderlijke gegevens na invoer en controle tot één string van 25 tekens, die aan een enkele variabele wordt toegekend.

(a) zie p.66.

```
(a) 110  REM
    120       LINE INPUT "ARTIKELCODE (3 TEKENS)       : "; C$
    130       IF LEN(C$) <> 3 THEN
                 PRINT "3 TEKENS NODIG. OPNIEUW INVOEREN AUB."  : GOTO 120
    135  REM
    140       LINE INPUT "OMSCHRIJVING (MAX. 20 TEKENS): "; O$
    150       IF LEN(O$) > 20 THEN
                 PRINT "OMSCHRIJVING TE LANG. OPNIEUW AUB."     : GOTO 140
    160       IF LEN(O$) < 20 THEN
                 LET O$ = O$ + " "                              : GOTO 160
    165  REM
    170       LINE INPUT "INITIALEN (2 TEKENS)         : "; N$
    180       IF LEN(N$) <> 2 THEN
                 PRINT "2 TEKENS NODIG. OPNIEUW INVOEREN AUB."  : GOTO 170
    185  REM
    190       LET A$ = C$ + O$ + N$
    195  REM
```

Verwijderen van spaties

We hebben gezien dat het nodig kan zijn strings door het programma met extra spaties aan te vullen om de juiste veldlengte te krijgen. Het omgekeerde kan ook nodig zijn, vooral wanneer gegevens uit bestaande strings worden 'gelicht'.

Stel bijvoorbeeld dat ons fictieve bedrijf haar lijst met telefoongegevens netjes in een computerbestand heeft opgeslagen, en na verloop van tijd het bestand wil gebruiken om de medewerkers uit te nodigen voor een personeelsfeest. Uit het bestand zijn alleen de persoonsnamen nodig. Als eis wordt gesteld dat het desbetreffende programma eerst de voorletters met eventuele titels, aanduidingen zoals 'mevrouw', 'mejuffrouw', 'van de' – of eventuele afkortingen van deze aanduidingen – selecteert. Daarna moet de achternaam aan het eerst geselecteerde gegeven worden geconcateneerd. Het gegeven "BIJL, H. VAN DER", bijvoorbeeld, moet dus worden omgekeerd tot "H. VAN DER BIJL". Zonder spatiereeks tussen voorletters en achternaam uiteraard.

Deze laatste eis brengt ons in moeilijkheden. Het naamveld werd bij het opbouwen van het bestand namelijk juist met spaties tot de juiste lengte aangevuld. Zouden we vanaf de komma na de achternaam de rest van het naamveld selecteren als voornaam – en dat ligt voor de hand – dan krijgen we bij het concateneren van voornaam en achternaam juist wel tussenliggende spaties. Onze "BIJL, H. VAN DER " zou dan worden: "H. VAN DER BIJL".

De volgende programmaconstructie biedt daarvoor een oplossing. Het is gebaseerd op het consequent gescheiden zijn van de twee deelgegevens door een komma en een spatie. Bovendien is ervan uit-

gegaan dat de bestandsregel al is opgeslagen in een programmavariabele, namelijk R$. Hoe de overdracht van een bestandsregel vanaf een floppy of cassette verloopt, laten we nog even buiten beschouwing.

```
2000 REM   ROUTINE VOOR HET OMKEREN VAN DEELGEGEVENS IN NAAMVELD,
2010 REM   EN DAARBIJ VERWIJDEREN VAN OVERBODIGE SPATIES
2020 REM
2030       LET  K   = INSTR(R$, ",")
2040       LET  N1$ = MID$(R$, 1, K-1)
2050       LET  N2$ = MID$(R$, K+2, MID$(R$, K+2, LEN(R$)-K-2)
2060 REM
2070       LET  L2  = LEN (N2$)
2080       FOR  I   =  L2  TO  1  STEP  -1
2090            IF  MID$(N2$, I, 1)  =  " "  THEN  2120
2100                LET L3  =  I
2110                GOTO 2130
2120       NEXT I
2130 REM
2140       LET  N3$  = MID$(N2$, 1, L3)
2150       LET  N$   = N3$  +  " "  +  N1$
2160 REM
```

Toelichting

a. De twee deelgegevens in R$ zijn gescheiden door een komma en een spatie. Het eerste deelgegeven begint in positie 1. Met behulp van de functie INSTR bepalen we eerst de positie K van de komma (regel 2030). We weten dan ook de positie van het laatste teken van het eerste deelgegeven, namelijk K-1. Het eerste deelgegeven (de achternaam, of althans het laatste deel daarvan) is dus de substring van positie 1 t/m positie K-1 van R$. We bepalen deze substring met behulp van de functie MID$ en kennen deze toe aan N1$ (regel 2040).

b. Van het tweede deelgegeven weten we dat het in positie K+2 begint, en dat het in principe kan doorlopen t/m positie 25: de laatste positie van het totale gegevensveld. We selecteren voorlopig deze substring als geheel met MID$, en kennen hem toe aan N2$.

c. Van N2$ moeten nu eventueel achterliggende spatie worden verwijderd. Hiertoe bepalen we eerst de lengte L2 van N2$ (regel 2070) en gebruiken L2 als beginwaarde van een 'aftelproces' waarin de inhoud van N2$ 'van achteren naar voren' wordt bekeken op de aanwezigheid van spaties (regels 2070 t/m 2120). Blijkt voor een gegeven waarde van I geen spatie aanwezig te zijn, dan weten we dat I op dat moment de laatste positie van het tweede deelgegeven aangeeft. We kunnen de FOR/NEXT-lus waarin dit aftelproces plaatsvindt nu in principe verlaten. In verband met het mogelijk ongedefinieerd raken van I bij het (voortijdig) verlaten van de lus, 'redden' we zijn waarde door die toe te kennen aan L3 (regel 2100).

d. Vervolgens verlaten we de lus (regel 2110), bepalen we – alweer met MID$ – de gewenste substring, en kennen deze toe aan N3$ (regel 2140). De gezochte deelgegevens bevinden zich nu in N3$ (voorste deel van de naam) en N1$ (achterste deel van de naam). In regel 2150 worden ze met een tussenspatie tot één string (N$) geconcateneerd.

3.3 CONTROLEREN OP NULSTRINGS

Een van de eigenaardigheden van zowel de INPUT- als de 'LINE INPUT'-opdracht is dat aan stringvariabelen een nulstring wordt toegekend als de gebruiker uitsluitend met het indrukken van de RETURN- of ENTER-toets reageert. Als een programma zo'n respons zonder meer zou aanvullen met spaties, zou dit weliswaar een geldig gegeven opleveren maar we zouden daar waarschijnlijk niet mee uit de voeten kunnen. Het is daarom zaak ingevoerde gegevens op het vóórkomen van nulstrings te controleren.

Er zijn hiervoor twee eenvoudige methoden, die even goed functioneren:

a. IF A$ = "" THEN ...
b. IF LEN(A$) = 0 THEN ...

In ieder geval opnieuw zal de programmeur moeten besluiten hoe hij het per abuis invoeren van een nulstring zal opvangen. Wat hij zeker *niet* mag doen is de invoeropdracht klakkeloos herhalen, bijvoorbeeld:

```
:
170  INPUT "KLANTNUMMER     : "; K$
180  IF  LEN(K$) = 0  THEN  170
:
:

RUN
KLANTNUMMER     :
KLANTNUMMER     :
KLANTNUMMER     :
```

Een gebruiker die hardnekkig volhoudt geen klantnummer op te geven, krijgt hierbij geen enkele aanwijzing van wat er nu eigenlijk fout gaat. Geef daarom altijd een foutmelding – eventueel vergezeld van een audio waarschuwingstoon – zodat de gebruiker weet dat er met zijn respons iets niet in orde is.

voorbeeld:

```
170  INPUT "KLANTNUMMER"     : "; K$
180  IF  LEN(K$)  =  0  THEN  GOSUB 1010
:
:
1010 PRINT "VOORTZETTING PROGRAMMA ALLEEN MOGELIJK MET KLANTNUMMER"
:
:
1100 RETURN
```

Toetsvraag

Schrijf met deze zaken in gedachten een (deel van een) gegevensinvoermodule die onderstaande prompts genereert. Test elk gegeven onmiddellijk na het invoeren op een nulstringwaarde.

(a) KLANTNUMMER :
(b) KLANTNAAM :
(c) ARTIKELNUMMER :
(d) BESTELDE HOEVEELHEID :

```
(a) 210  REM
    220        LINE INPUT "KLANTNUMMER          : "; K$
    230        IF LEN(K$) = 0  THEN
                  PRINT "INVOERFOUT. OPNIEUW INVOEREN AUB"  :  GOTO 220
    235  REM
    240        LINE INPUT "KLANTNAAM            : "; N$
    250        IF LEN(N$) = 0  THEN
                  PRINT "INVOEREN ZOALS AANGEGEVEN AUB"     :  GOTO 240
    255  REM
    260        LINE INPUT "ARTIKELNUMMER        : "; A$
    270        IF LEN(A$) = 0  THEN
                  PRINT "PROGRAMMA KAN ZONDER ARTIKELNR. NIET VERDER : GOTO 260
    275  REM
    280        LINE INPUT "BESTELDE HOEVEELHEID : "; H$
    290        IF LEN(H$) = 0  THEN
                  PRINT "VOER JUISTE HOEVEELHEID IN AUB"    : GOTO 280
    295  REM
```

Merk op:
1. Soortgelijke meldingen en/of andere stringvariabelen zijn uiteraard toegestaan.
2. In plaats van LINE INPUT kan ook INPUT worden gebruikt.

Afhankelijk van het inzicht – of het gebrek daaraan – van de gebruiker kunnen meer gedetailleerde meldingen nodig zijn, ook voor andere problemen dan het per abuis invoeren van nulstrings. De voorbeelden hierboven hebben we bewust voor (onze) duidelijkheid zo eenvoudig mogelijk gehouden. Het laatste voorbeeld zouden we desnoods als volgt kunnen aanpassen:

```
170  LINE INPUT "KLANTNUMMER  : "; K$
180  IF  LEN(K$) = 0  THEN GOSUB 1010  :  GOTO 170
:
1010 PRINT "U HEBT KENNELIJK OP DE RETURNTOETS GEDRUKT ZONDER EEN"
1015 PRINT "GEGEVEN IN TE TYPEN. "
1020 PRINT "ER IS EEN KLANTNUMMER NODIG, BIJVOORBEELD A-121"
1030 RETURN
```

Afbakenen van modules en subroutines

Het gebruik van subroutines voor de opvang van invoerfouten kan problemen met zich meebrengen, met name als het hoofdprogramma niet naar behoren wordt afgesloten met een END- of eventueel STOP-opdracht. De subroutine wordt dan aangezien voor een deel van het hoofdprogramma en als zodanig uitgevoerd. Een en ander komt meestal aan het licht bij de uitvoering van de RETURN-opdracht van de subroutine (aannemende dat deze niet is vergeten). De subroutine is immers niet op de juiste wijze – met een GOSUB – aangeroepen, en 'weet' dus ook niet waarheen hij moet terugkeren. Resultaat: een forse tik op de vingers, oftewel een 'fatal error'.

Toetsvraag

Schrijf een foutverwerkingssubroutine die door middel van een GOSUB in het THEN-deel van een 'IF...THEN'-opdracht wordt aangeroepen. De subroutine moet naar aanleiding van een verkeerde respons op de opdracht:

```
320  LINE INPUT  "ARTIKELOMSCHRIJVING: "; O$
```

aangeven dat aan een beperking van maximaal 20 tekens moet worden voldaan. De aangeboden invoer moet worden afgedrukt.

invoervoorbeeld:

```
ARTIKELOMSCHRIJVING: DUBBELZIJDIGKLEVENDPLAKBAND
------------------------
```

(a) Uw oplossing moet er ongeveer zo uitzien:

```
330  IF  LEN(O$) > 20  THEN  GOSUB 1140  :  GOTO 320
:
1110 STOP
1120 REM   FOUTVERWERKING
1130 REM
1140       PRINT "U VOERDE "; O$; "IN ALS ARTIKELOMSCHRIJVING"
1150       PRINT "VOER OPNIEUW IN AUB, MAAR AFGEKORT TOT MAX."
1160       PRINT "20 TEKENS, INCL. SPATIES EN LEESTEKENS."
1170       RETURN
```

3.4 VERVANGEN VAN DEELGEGEVENS

Ondanks protesten uit feministische kringen is het in de meeste westerse landen nog steeds de gewoonte dat een vrouw bij het trouwen de achternaam van haar man aanneemt. Is de vrouw onder haar meisjesnaam in een computerbestand geregistreerd, waarbij de (meisjes)naam samen met de voorna(a)m(en) in één gegevensveld is opgenomen, dan doet zich na de trouwdatum het probleem voor dat een deel van het naamveld vervangen moet worden – of althans aangepast – dóór een ander gegeven. Eenvoudigheidshalve nemen we aan dat de meisjesnaam daarbij door de nieuwe naam mag worden overschreven.

Om dit probleem op te lossen gaan we uit van een vaste lengte voor zowel de voornaam, voornamen of voorletters enerzijds, als de achternaam anderzijds. We kiezen daarvoor respectievelijk 8 en 12 tekens. Beide deelvelden scheiden we met een spatie, zodat de totale lengte van het gegevensveld 21 is. Verder veronderstellen we dat de invoer van zowel de oorspronkelijke als de nieuwe gegevens vanuit één programma wordt verzorgd.

Het gegevensinvoerdeel zou er als volgt kunnen uitzien:

```
100  REM   INVOEREN VAN OORSPRONKELIJKE GEGEVENS
110  REM
120        LINE INPUT "VOORNAAM  : "; V$
130        IF  LEN(V$) = 0  THEN
               PRINT "U WORDT VERZOCHT VOORNAAM OP TE GEVEN."     :  GOTO 120
140        IF  LEN(V$) > 8  THEN
               PRINT "VOORNAAM TE LANG. MAX. 8 TEKENS AUB."       :  GOTO 120
150        IF  LEN(V$) < 8  THEN
               LET  V$ = V$ + " "                                 :  GOTO 150
160  REM
170        LINE INPUT "ACHTERNAAM: "; A$
180        IF  LEN(A$) = 0  THEN
               PRINT "U WORDT VERZOCHT ACHTERNAAM OP TE GEVEN."   :  GOTO 170
190        IF  LEN(A$) > 12  THEN
               PRINT "ACHTERNAAM TE LANG. MAX. 12 TEKENS AUB."    :  GOTO 170
200        IF  LEN(A$) < 12  THEN
               LET A$ = A$ + " "                                  :  GOTO 200
210  REM
220        LET N$ = V$ + " " + L$
```

Na uitvoering van dit programmadeel staan voor- en achternaam, gescheiden door een spatie, in de variabele N$. Voor het vervangen van de bestaande achternaam door de nieuwe zouden we de volgende constructie kunnen maken:

```
400  REM   VERVANGEN VAN ACHTERNAAM
410  REM
420        LINE INPUT "NIEUWE ACHTERNAAM: "; A1$
430        IF  LEN(A1$) = 0  THEN
               PRINT "NIEUWE ACHTERNAAM OPGEVEN AUB."          :  GOTO 420
440        IF  LEN(A1$) > 12  THEN
               PRINT "ACHTERNAAM TE LANG. MAX. 12 TEKENS."     :  GOTO 420
450        IF  LEN(A1$) < 12  THEN
               LET  A1$ = A1$ + " "                            :  GOTO 450
460  REM
470        LET MID$(N$, 10, 12) = A1$
```

De voornaam, de achternaam of beide (zowel vóór als na de mutatie) kunnen naar wens – afhankelijk van de variabele I, die hiervoor de waarde 1, 2 respectievelijk 3 moet hebben – door middel van het volgende programmadeel worden afgedrukt.

```
600  REM   AFDRUKKEN VAN VOORNAAM (I=1), ACHTERNAAM (I=2) OF BEIDE (I=3)
610  REM
620        IF  I < 1  OR  I >  3  GOSUB 820
625        ON  I  GOTO 640, 670, 700
630  REM
640        PRINT MID$(N$, 1, 8)        : REM *** ALLEEN VOORNAAM ***
650        GOTO 720
660  REM
670        PRINT MID$(N$, 10, 12)      : REM *** ALLEEN ACHTERNAAM ***
680        GOTO 720
690  REM
700        PRINT N$                    : REM *** VOLLEDIGE NAAM ***
710  REM
720        ...
:
800  REM   SUBROUTINE VOOR OPVANGEN VERKEERDE WAARDE VAN I (ON...GOTO)
810  REM
820        ...
```

Toetsvragen

(a) Waartoe dient de opdracht LET A$ = A$ + " " in regel 200?

(b) Wat is het effect van regel 220?

(c) Wat gebeurt er ten gevolge van de aanroep van MID$ in regel 470?

(d) Als "MIES" en "BROUER" zijn ingevoerd als oorspronkelijke voor- en achternaam, welke uitvoer wordt dan ten gevolge van regel 700 geproduceerd?

(a) Vult de resterende (ongebruikte) posities met spaties aan tot de juiste veldlengte. Dezelfde techniek wordt ook in regels 150 en 450 toegepast.

(b) De voor- en achternaam, gescheiden door een spatie, worden in N$ opgeslagen.

(c) 12 tekens van N$ worden vanaf positie 10 vervangen door de aan A1$ toegekende string (dat wil zeggen de nieuwe achternaam).

(d) MIES BROUER
(alle aanvullende spaties worden 'meegenomen' bij het afdrukken van N$).

3.5 GEBRUIK VAN DE VAL-FUNCTIE BIJ INVOERCONTROLE

Met behulp van de functie VAL (afkorting van value = waarde) kan op betrekkelijk eenvoudige wijze worden gecontroleerd of een respons, waarbij een numerieke waarde moet worden ingevoerd, ook daadwerkelijk numeriek is. Voor een bestelhoeveelheid, bijvoorbeeld, verwachten we per definitie een numerieke waarde. Het controlemechanisme berust op het feit dat alfanumerieke invoer in de vorm van een getal door de VAL-functie tot de overeenkomstige numerieke getalwaarde wordt geconverteerd.

voorbeeld:

```
330  LET A$ = "128.95"
340  PRINT VAL(A$)
350  A = VAL(A$)
360  PRINT A

RUN
 128.95
 128.95
```

Aan het getal gaat een tekenpositie vooraf: een spatie als het getal positief is, een minteken als het getal negatief is.

De VAL-functie helpt ons niet altijd uit de brand bij de conversie van strings naar numerieke waarden. Een van de problemen is bijvoorbeeld een vergissing zoals het intypen van de letter O in plaats van het cijfer 0. Ook het toevoegen van een maateenheid aan een getalwaarde (bijvoorbeeld 12 KG) kan bij invoercontrole met behulp van VAL tot problemen leiden.

De waarde die VAL in dergelijke gevallen levert is sterk afhankelijk van de computer waarmee men werkt. Hieronder volgen enkele testen uitgevoerd op een MSX-computer. Probeer ze op uw eigen machine voordat u verdergaat. Mocht u hierbij moeilijkheden ondervinden, dan kunnen de opmerkingen na de testopdrachten u wellicht verder helpen.

```
100  REM   TEST 1, VAL-FUNCTIE
110  REM
120        LET A$ = "ABC"
130        PRINT A$, VAL(A$)
140  REM
150  REM   TEST 2, VAL-FUNCTIE (NULSTRING)
160  REM
170        LET A$ = " "
180        PRINT A$, VAL(A$)
190  REM
200  REM   TEST 3, VAL-FUNCTIE
210  REM
220        LET A$ = "123ABC"
230        PRINT A$, VAL(A$)
240  REM
250  REM   TEST 4, VAL-FUNCTIE
260  REM
270        LET A$ = "ABC123"
280        PRINT A$, VAL(A$)

RUN
ABC          0
             0
123ABC       123
ABC123       0
```

Sommige computers kunnen al bij regel 120 de kluts volledig kwijtraken. In dat geval kunt u de RUN bij regel 170 hervatten met behulp van de commando RUN170 of – als uw computer dit commando niet kent – door regels 100 t/m 150 te verwijderen en een gewone RUN te geven.* Op dezelfde wijze kunt u desnoods ook de rest van het programma doorlopen.

Bespreking van het testprogramma

Deze bespreking is gebaseerd op de MSX-implementatie van de VAL-functie, zoals die in bovenstaande RUN tot uitdrukking komt. Andere methoden om dezelfde effecten te bereiken worden besproken in het gedeelte over ASCII-code.

De eerste conclusie die we kunnen trekken is dat VAL de waarde nul levert als de argumentwaarde zuiver alfabetisch is, of uit een nulstring bestaat. Gemengde alfabetische en numerieke tekens resulteren eveneens in de functiewaarde nul als de alfabetische tekens aan de numerieke tekens voorafgaan, bijvoorbeeld ABC123. Het omkeren van het alfabetische en het numerieke deel, bijvoorbeeld 123ABC, levert de desbetreffende numerieke waarde (in ons voorbeeld dus 123). De MSX-versie van VAL laat alfabetische gegevens die na numerieke gegevens komen dus buiten beschouwing. Dit is plezierig als na de numerieke waarde bijvoorbeeld een maateenheid

* De vorm RUN... is overigens zeer handig bij het 'ontluizen' van programma's.

wordt opgegeven, maar lastig als zo'n gegeven op zuiver numerieke inhoud moet worden gecontroleerd. Voorlopig nemen we aan dat bij gemengde alfabetische en numerieke tekens de laatstgenoemde tekens vooraan staan en de bedoelde getalwaarde voorstellen.

Testen op numerieke invoer

Onder voorbehoud van de zojuist genoemde restrictie kan numerieke invoer met een constructie als de volgende worden gecontroleerd:

```
IF VAL(A$) = 0 THEN PRINT "ALLEEN NUMERIEKE WAARDEN TOEGESTAAN"
```

Omdat sommige computers zich in nulstrings kunnen 'verslikken' is het aan te bevelen deze opdracht onmiddellijk na de test op een nulstring in uw programma op te nemen. Anders loopt u kans op een foutmelding en het stoppen van de RUN.

Toetsvraag

(a) Naar aanleiding van § 3.3, blz.69 werd gevraagd een gegevensinvoermodule te schrijven waarin controle-opdrachten voor nulstrings waren opgenomen. Voeg aan deze module nu opdrachten toe om te controleren of het artikelnummer en de bestelde hoeveelheid als numerieke waarden worden ingevoerd. Test bovendien op een viercijferig artikelnummer.

```
(a) 210  REM
    220       LINE INPUT "KLANTNUMMER          : "; K$
    230       IF LEN(K$) = 0  THEN
                 PRINT "INVOERFOUT. OPNIEUW INVOEREN AUB."          : GOTO 220
    235  REM
    240       LINE INPUT "KLANTNAAM            : "; N$
    250       IF LEN(N$) = 0  THEN
                 PRINT "INVOEREN ZOALS AANGEGEVEN AUB."             : GOTO 240
    255  REM
    260       LINE INPUT "ARTIKELNUMMER        : "; A$
    270       IF LEN(A$) = 0  THEN
                 PRINT "VOORTZETTING ZONDER DIT GEG. ONMOGELIJK"    : GOTO 260
    272       IF VAL(A$) = 0  THEN
                 PRINT "UITSLUITEND NUMERIEKE WAARDEN TOEGESTAAN"   : GOTO 260
    274       IF LEN(A$) <> 4  THEN
                 PRINT "NUMMER MOET UIT 4 CIJFERS BESTAAN"          : GOTO 260
    275  REM
    280       LINE INPUT "BESTELDE HOEVEELHEID : "; H$
    290       IF LEN(H$) = 0  THEN
                 PRINT "JUISTE HOEVEELHEID INVOEREN AUB."           : GOTO 280
    292       IF VAL(H$) = 0  THEN
                 PRINT "UITSLUITEND NUMERIEKE WAARDEN TOEGESTAAN"   : GOTO 280
    295  REM
```

3.6 GEBRUIK VAN DE STR$-FUNCTIE VOOR CONVERSIE NAAR STRINGWAARDEN

De STR$-functie (afkorting van string) is de tegenhanger van VAL. Hij dient voor het converteren van numerieke waarden naar stringwaarden. Dankzij deze functie kunnen bewerkingen met getallen in samenhang met strings gemakkelijker worden uitgevoerd.

Stel bijvoorbeeld dat een artikelnummer, een artikelomschrijving en een voorraadaantal in één lange string zijn opgeslagen. Het voorraadaantal kan op een gegeven moment nodig zijn om een of andere berekening uit te voeren, waarna een nieuw voorraadaantal in de oorspronkelijke string moet worden opgenomen. In dit geval is om te beginnen de getalwaarde van de deelstring 'voorraadaantal' nodig. Daarvoor kunnen we de VAL-functie gebruiken. Na berekening van het nieuwe voorraadaantal kunnen we deze (numerieke) waarde met behulp van de STR$-functie converteren naar een stringwaarde, die op zijn beurt weer in de oorspronkelijke string kan worden opgenomen.

Dus:


```
A$ = A$ + STR$(V)
```

of:

```
V$ = STR$(V)
A$ = A$ + V$
```

waarbij V$ de variabele is waarin het voorraadaantal wordt opgeslagen.

Bij het converteren van een numerieke waarde naar een stringwaarde met STR$ wordt in de resulterende string een spatie opgenomen als het getal positief is. Er komt een minteken vooraan te staan als het getal negatief is. Dit komt onder andere tot uitdrukking bij het toepassen van de LEN-functie op de resulterende string.

voorbeeld:

```
140 LET X  = 847.25
150 LET X$ = STR$(X)
160 LET N  = LEN(X$)
170 PRINT "X  = "; X
180 PRINT "X$ = "; X$
190 PRINT "N  = "; N

RUN
X  = 847.25
X$ = 847.25
N  = 7
```

Met de voorafgaande tekenpositie moet dus uitdrukkelijk rekening worden gehouden. Een negatief getal van zes cijfers, bijvoorbeeld, past niet in een gegevensveld van zes tekens. Probeert men dat toch dan kunnen significante cijfers of het minteken verloren gaan.

Toetsvraag

Uit hoeveel tekens zouden de volgende getallen bestaan als ze met behulp van de STR$-functie naar stringwaarden zouden worden geconverteerd?

(a) 171.83

(b) 2001

(c) -999

(a) 6 ; (b) 4 ; (c) 4

3.7 CONTROLEREN OP ILLEGALE TEKENS

Ingevoerde strings kunnen op het voorkomen van illegale tekens worden onderzocht door collectieve toepassing van de ASC-functie, de MID$-functie, de IF...THEN en de FOR/NEXT. Als eerste stap hierbij wordt met LEN de lengte van de te onderzoeken string bepaald. Deze waarde wordt als eindwaarde van de FOR-besturingsvariabele gebruikt, bijvoorbeeld:

```
350  LINE INPUT  "GEEF CATALOGUSCODE (6 TEKENS): "; C$
360  FOR X = 1  TO  LEN(C$)
         ...
```

Vervolgens wordt met behulp van MID$, waarbij de besturingsvariabele (dus X in bovenstaand voorbeeld) wordt gebruikt om stap voor stap één teken van de string te selecteren. De geselecteerde waarde wordt vergeleken met de ASCII-waarde van het te controleren teken, of eventueel rechtstreeks met het teken zelf.

voortzetting voorbeeld:

```
370  IF  ASC(MID$(C$, X, 1)) = 32  THEN
         PRINT  "OPNIEUW INVOEREN ZONDER SPATIES AUB."          :  GOTO 350
380  NEXT X
```

of:

```
370  IF  MID$(C$, X, 1) = " "  THEN
         PRINT ...
```

In dit voorbeeld gaat het dus om het testen op het vóórkomen van spaties in de ingevoerde string. De eerste methode – waarbij de ASC-functie wordt toegepast – is iets algemener, maar minder duidelijk dan de tweede. Algemener omdat bij samengestelde voorwaarden (AND/OR) gemakkelijker een bepaalde serie tekens als testwaarden kunnen worden gebruikt. Minder duidelijk omdat we – althans als gewone stervelingen – bij de numerieke voorstelling van tekens een codetabel nodig hebben om de desbetreffende tekens te kunnen (terug)lezen.

Toetsvraag 1

Stel dat een gebruiker de volgende RUN maakt van het zojuist besproken programmadeel:

```
RUN
GEEF CATALOGUSCODE (6 TEKENS): Ab-b1314     (b = spatie)
```

(a) Wat is de lengte van de door MID$ geselecteerde string in regel 370?

(b) Welke ASCII-waarde wordt met het getal 32 vergeleken bij eerste uitvoering van de FOR/NEXT-lus?

(c) En bij de tweede uitvoering?

(d) Bij welke iteratie (dat wil zeggen de 'hoeveelste' doorgang) van de FOR/NEXT is de vergelijking in regel 370 waar?

(e) Wat is de theoretische eindwaarde van de besturingsvariabele X? Dat wil zeggen hoeveel keer zou de lus (afgezien van eventueel voortijdige afbreking) maximaal kunnen worden uitgevoerd?

(a) 1
(b) 65 (zijnde de ASCII-waarde van A)
(c) 32 (spatie)
(d) tweede iteratie
(e) LEN(C$) = 8

Toetsvraag 2

(a) Schrijf soortgelijke controle-opdrachten als in bovenstaand voorbeeld (regels 350 t/m 380) om een foutmelding te produceren bij het ontdekken van een illegaal teken. Als *legale* tekens komen hierbij alleen de cijfers 0 t/m 9 en de punt (als decimaalteken) in aanmerking.

(a) zie p.79.

```
(a) 100  LINE INPUT  "GETALWAARDE: "; W$
    110  FOR  X = 1  TO  LEN(W$)
    120       IF  ASC(MID$(W$, X, 1)) >= 48  AND
                  ASC(MID$(W$, X, 1)) <= 57  OR
                  ASC(MID$(W$, X, 1))  = 46  THEN 160
    130       PRINT  "ONGELDIGE INVOER. UITSLUITEND CIJFERS EN"
    140       PRINT  "EVENTUEEL EEN DECIMALE PUNT TOEGESTAAN."
    150       GOTO  100
    160  NEXT X
    ...
```

zie voetnoot

3.8 ALGEMENE BESPREKING VAN GEGEVENSINVOER EN -CONTROLE

In dit hoofdstuk zijn aanbevelingen gedaan, aanwijzingen gegeven en technieken behandeld voor het invoeren en controleren van gegevens. Daarbij ging het overwegend over specifiek technische zaken. In deze paragraaf bespreken we wat algemenere aspecten van de gegevensinvoer en -controle.

Waar controleren we?

Over de vraag op welk punt in de gegevensverwerking invoercontroles het beste kunnen worden uitgevoerd zijn de meningen verdeeld. Er zijn globaal twee benaderingen. De eerste is vooral van belang in een professionele omgeving en gaat ervan uit dat de tijd van degene die gegevens invoert kostbaar is, en dat invoercontroles daarom zo weinig mogelijk tijd in beslag mogen nemen. Controles

---

Opmerking van de vertaler/bewerker:
De efficiëntie (en dus de snelheid) van dit programmadeel kan aanzienlijk worden verbeterd door de ASC/MID$-aanroepen onmiddellijk vóór de IF te plaatsen, en het resultaat aan een hulpvariabele (bijvoorbeeld H) toe te kennen. Deze H wordt dan in plaats van de functie-aanroepen in de IF-opdracht gebruikt. Dus:

```
110  ...
115  LET  H  = ASC(MID$(W$, X, 1))
     ...
120  IF   H >= 48  AND  H <= 57  OR  H = 46  THEN  160
130  ...
```

Dit bespaart per iteratie vier tijdrovende functie-aanroepen.

tijdens het invoeren zelf zijn daarom uit den boze, zo redeneert men. In een later stadium – nadat alle gegevens zijn ingevoerd – controleert een afzonderlijk programma of de invoer acceptabel was. Elke keer dat daarbij een invoerfout wordt ontdekt, wordt de gehele invoer' transactie' voor die serie gegevens geweigerd. De serie wordt in zijn geheel afgedrukt, en aan de hand van de listing en het oorspronkelijke invoerdocument verbeterd.

Deze procedure werkt goed als het aantal fouten betrekkelijk klein is. Daarom is het van belang dat de nodige tijd en moeite wordt geïnvesteerd in het trainen van degenen die met de invoer worden belast.

Zelf geven we de voorkeur aan de tweede benadering: elk gegeven na invoer onmiddellijk controleren. Deze directe controle is overigens ook geschikter voor kleinere administratieve toepassingen, en voor de meer persoonlijk gerichte toepassingen. Wordt bij deze vorm van controle een fout ontdekt, dan wordt de invoerder als het ware direct op de vingers getikt. De fout kan dan meteen worden hersteld. Een van de voordelen hiervan is dat degene die de fout maakt ook verantwoordelijk is voor het verbeteren daarvan. Bij de eerste methode moeten speciale voorzieningen worden getroffen om te achterhalen wie voor de fout verantwoordelijk is. Voor effectieve bedrijfsvoering kan dit een nadeel zijn.

Beeldschermgebruik

Bij de invoer van gegevens via beeldschermen zijn – ook al – twee methoden gangbaar. De eerste gaat uit van een grafische afbeelding van het gedrukte formulier waarop de in te voeren gegevens staan genoteerd. Voor deze methode is veel te zeggen. De computer laat de invoerder als het ware de stippelveldjes invullen op dezelfde manier als bij het invullen van het formulier zou gebeuren. Voorwaarde hierbij is dat het beeldscherm speciale grafische mogelijkheden heeft.

Bij de tweede methode worden één of meer gegevensseries na invoer als geheel op het scherm afgedrukt. De gebruiker wordt dan in de gelegenheid gesteld eventueel verkeerde gegevens opnieuw in te voeren. Ook gegevens die door de mazen van eerste, directe invoer-controle-'netten' zijn geglipt kunnen hierbij aan het licht komen en hersteld worden. Daarbij kan onder andere gedacht worden aan spelfouten, getalverwisselingen en typografische fouten.

Een voorbeeld van zo'n controle achteraf is het volgende:

```
DANK U WEL. UW INVOER WAS:

          KLANTNUMMER    ARTIKELNR    HOEVEELHEID

1:          98213          17892          20
2:          98213          24618          10
3:          98213          81811          100

WILT U NOG IETS WIJZIGEN (J/N): J
NUMMER VAN DE TE WIJZIGEN REGEL :
```

Alvorens zo'n overzicht af te beelden kunnen andere gegevens het beste van het scherm worden verwijderd. Zo'n 'schone lei' kan ook fouten helpen voorkomen als gegevens voor het eerst worden ingetypt. In ieder geval zou het scherm na het invoeren van een gegevenssserie even gewist moeten worden (zie § 2.9, bijzondere codegetallen). Ook bij grafische formulierafbeeldingen is dit aan te bevelen.

Gebruik gezond verstand

Controleprocedures kunnen afhankelijk zijn van persoonlijke voorkeur, maar ook bijvoorbeeld van bedrijfsrichtlijnen. Hoe dan ook, denk als programmeur steeds vooruit en gebruik uw gezond verstand. Houd rekening met de totale toepassingssituatie. In welke vorm en op welk formaat, bijvoorbeeld, worden de gegevens ingevoerd? Zijn er beperkingen – misschien zeer subtiele – die aan de gegevens kunnen worden gesteld, of zijn er tests waaraan u de gegevens kunt onderwerpen om eventuele invoerfouten op te sporen?

Wat het gegevensformaat betreft – dat wil zeggen de grootte en indeling – verdient het aanbeveling consequent te zijn. Schrijft u bijvoorbeeld verschillende programma's waarin persoonsnamen voorkomen, gebruik dan steeds dezelfde totale lengte en – voor zover van toepassing – dezelfde deelveldlengte. De gebruiker 'went' dan aan het formaat, en zal daardoor minder snel fouten maken. Bij opslag van de gegevens in een bestand kunnen bovendien verschillende programma's op gemakkelijke wijze van de gegevens gebruik maken. In een professionele omgeving zal het formaat van veel gegevens – vooral alfanumerieke gegevens zoals adresvelden, artikel- of produktcoderingen enzovoort – bij voorbaat vastliggen. Uiteraard is het dan van belang hiervan op de hoogte te zijn.

Voor numerieke waarden – bijvoorbeeld aantallen, geldbedragen enzovoort – zal misschien enig 'spitwerk' nodig zijn om grenzen of randvoorwaarden waaraan gegevens moeten voldoen te ontdekken. Ook hier is het gebruik van gezond verstand geen overbodige luxe.

Grenzen of randvoorwaarden kunnen vaak aan de hand van bedrijfsrichtlijnen of -procedures, of door gewone ervaringsfeiten worden bepaald. In twijfelgevallen kan de uiteindelijke beslissing altijd in handen van de gebruiker worden gelegd, zoals in het volgende voorbeeld tot uitdrukking komt.

```
400  REM
410        LINE INPUT "VOER PRIJS IN                    : "; P$
420        IF VAL(P$) <= 200.0  THEN 460
425           PRINT "PRIJS MEER DAN GEBRUIKELIJK MAX. VAN FL.200"
430           LINE INPUT "IS PRIJS TOCH CORRECT (J/N): "; A$
440           IF LEFT$(A$, 1) = "N"  THEN
                 PRINT "VOER OPNIEUW IN AUB."  :  GOTO 410
450  REM
460  ...
```

Overzicht

Terwille van het overzicht zetten we de algemene invoercontroleprocedures voor zowel alfanumerieke als numerieke gegevens op een rijtje.

1. Lees alle gegevens alfanumeriek in (dus door toekenning aan stringvariabelen).
2. Laat gegevens uitsluitend invoeren na een verzoek daartoe van de computer door middel van duidelijk gestelde prompts.
3. Laat per invoeropdracht (en dus per prompt) slechts één gegeven invoeren.
4. Bent u van plan verschillende deelgegevens in één string onder te brengen, voer die deelgegevens dan één voor één in met behulp van afzonderlijke stringvariabelen, en voeg ze pas samen nadat alle controles zijn uitgevoerd.
5. Bij de controles dient rekening te worden gehouden met nulstring-respons door een lengtetest van het type:

   ```
   IF  LEN(A$) = 0  THEN  ...
   ```

6. Meld bij het ontdekken van een fout niet alleen het nuchtere feit op zich, maar omschrijf de fout zo volledig mogelijk. Vraag niet slechts om nieuwe invoer.
7. Controleer alfanumerieke gegevens op juiste lengte met behulp van de LEN-functie.
8. Het kan nodig zijn een ingevoerd gegeven met spaties tot de juiste lengte aan te vullen. Dit geldt vooral voor alfanumerieke gegevens.
9. Test numerieke gegevens – liefst ingevoerd in alfanumerieke vorm – uitvoerig in onderstaande volgorde:

a. Nulstring-respons (dus onmiddellijk op RETURN of ENTER drukken zonder iets in te typen).

b. Stringlengte, indien van toepassing.

c. Aanwezigheid van alfabetische of speciale tekens in numerieke gegevens (met behulp van VAL of ASC).

d. Overeenstemming met bedrijfsrichtlijnen en/of bedrijfsmatige testwaarden.

e. Gebruik voor integerwaarden – dat wil zeggen positieve of negatieve gehele getallen, inclusief nul – integervariabelen (in de meeste BASIC-versies een procentteken na de naam van de variabele).

f. Controleer op integerwaarden met behulp van opdrachten als:

```
IF X = INT(X) THEN ...
```

g. Zijn negatieve waarden niet acceptabel, controleer daar dan op.

Een en ander lijkt misschien veel werk met zich mee te brengen. Realiseer u echter dat de betrouwbaarheid van uw programma staat of valt met de betrouwbaarheid van de gegevens. Ga daarom zorgvuldig te werk, en raffel de controles niet zo maar af. Als uw BASIC-versie bijvoorbeeld de mogelijkheid biedt in een IF...THEN een PRINT-opdracht op te nemen, laat u dan niet verleiden tot kortere meldingen door gebrek aan ruimte op de regel. Gebruik een GOSUB om duidelijke en volledige meldingen te genereren.

Verdere tips

1. Het kan nuttig zijn alle foutcontroles en meldingen in subroutines op te nemen. Dit geeft uw programma netheid en duidelijkheid. Verschillende gegevens kunnen mogelijk aan dezelfde controles worden onderworpen als aan bepaalde voorwaarden – zoals lengte van variabelen – is voldaan. Ook dat pleit voor een subroutine-aanpak.

2. Wees op uw hoede voor plaatsen in uw programma waar de waarde van variabelen aanleiding kan geven tot fouten. Het verschil van twee bijna even grote numerieke waarden gebruikt als noemer in een deling, bijvoorbeeld:

   ```
   LET X = A / (B - C)
   ```

   kan leiden tot overschrijding van de getalcapaciteit van de computer en het stoppen van de RUN. In het gegeven voorbeeld dus testen of het verschil tussen B en C nog voldoende groot is!

3. Pas op voor onverwachte resultaten van functies zoals VAL. Zorg dat u de BASIC-versie waarmee u werkt door en door leert kennen door in proefprogramma's de werking van opdrachten en functies onder verschillende omstandigheden uit te pluizen.

4. Het soort fouten dat u krijgt zal grotendeels afhankelijk zijn van uw programmeervaardigheid en het toepassingsgebied waarvoor u programmeert. Wees daarop attent, en leer van uw fouten. Programmeren is voor een belangrijk deel een praktische vaardigheid die alleen te leren is door veel 'doen'. Dit ontwikkelingsproces wordt overigens geremd als u zich blindstaart op steeds dezelfde constructies of technieken.

5. Programma's worden veelal uitsluitend met zinnige gegevens getest. Aan de mogelijkheid dat een gebruiker ook wel eens minder zinnige gegevens kan aandragen wordt eenvoudig voorbijgegaan. Als u denkt dat alles getest te hebben, geef dan uw programma eens in handen van een kind dat geen ervaring heeft met computers. Overleeft uw programma deze vuurproef, dan mag gevoeglijk worden aangenomen dat het wel goed zit met uw programma. Met alle eerbied voor het kind: maak uw programma gek-bestendig!

3.9 TOETSVRAGEN

1. Schrijf een 'IF...THEN'-test die waar is indien:
   (a) een ingevoerd gegeven uit precies 7 tekens bestaat;
   (b) een ingevoerd gegeven niet uit precies 7 tekens bestaat;
   (c) het eerste teken in een ingevoerd gegeven geen cijfer is;
   (d) het eerste teken in een ingevoerd gegeven een cijfer is maar geen nul;
   (e) een ingevoerd gegeven geen nulstring is.

2. Schrijf een opdrachtregel waarin wordt gecontroleerd of een gegeven uit minder dan 12 tekens bestaat en, zo ja, het gegeven met spaties aanvult tot een lengte van precies 12 tekens.

3. Schrijf controle-opdrachten om te voorkomen dat invoergegevens die cijfers bevatten als geldige invoer worden geaccepteerd. Schrijf een bijbehorende subroutine die bij het constateren van cijfers wordt aangeroepen, de gebruiker laat zien wat hij invoerde, en die meldt dat een gegeven zonder cijfers moet worden ingevoerd.

4. Als het goed is staat u nu voldoende stevig in uw schoenen om een module voor de invoer van de meeste soorten gegevens te

schrijven. Aan de hand van de volgende vragen kunt u uw krachten op dit gebied beproeven.

(a) Schrijf opdrachten waardoor de gebruiker wordt gevraagd de volgende gegevens in te voeren. Uiteraard moeten hierbij controle-opdrachten worden geschreven.
1. een alfanumerieke artikelcode van precies 5 tekens;
2. een artikelnaam van maximaal 12 tekens;
3. een bestelhoeveelheid van 3 cijfers, met een maximale waarde van 288 voor iedere bestelling;
4. de stukprijs, bestaande uit maximaal 5 posities, en gebonden aan een maximum van $ 99.99.

(b) Berg de onder (a) genoemde gegevens op in één string (A$) met de volgende indeling:

```
A$ = ...../............/.../.....   (per punt één teken)
       C$       N$       H$  P$
```

Opmerking: neem geen schuine strepen in de string op.

(c) Schrijf opdrachten voor het produceren van een deel van een listing waarin de stukprijs, de bestelhoeveelheid en de code van een artikel op één regel wordt uitgevoerd. Blader daarbij eens terug in dit hoofdstuk, en probeer uw programma te ontluizen alvorens naar onze oplossing te kijken. Er zijn namelijk – zoals zo vaak – verschillende wegen die naar Rome leiden. Onze weg hoeft niet per se de beste te zijn.

3.10 ANTWOORDEN OP DE TOETSVRAGEN

1. (a) `IF LEN(A$) = 7 THEN ...`
   (b) `IF LEN(A$) <> 7 THEN ...`
   (c) `IF ASC(A$) < 48 AND ASC(A$) > 57 THEN ...`
   (d) `IF LEFT$(A$,1) <> "0" THEN ...`
   (e) `IF LEN(A$) <> 0 THEN ...`

   Uiteraard is een andere string dan A$ toegestaan.

2. `120 IF LEN(A$) < 12 THEN LET A$ = A$ + " " : GOTO 120`

   Uiteraard is een andere stringvariabele dan A$ en een ander regelnummer dan 120 mogelijk.

3.
```
310 LINE INPUT "NAAM            : "; N$
320 FOR  X = 1  TO  LEN(N$)
330       IF  ASC(MID$(N$, X, 1)) > 47  AND
              ASC(MID$(N$, X, 1)) < 58  THEN  GOSUB 1100:  GOTO 310
340 NEXT X
:
:
1100 PRINT "U HEBT INGEVOERD     : "; N$
1120 PRINT "VOER OPNIEUW IN ZONDER CIJFERS AUB."
1130 RETURN
```

4.
```
110  REM   OPLOSSING VOOR OPGAVE 4, TOETSVRAGEN HOOFDSTUK 3
114  REM
119        CLEAR 1000   :  REM *** GEHEUGENGRENS INSTELLEN EN VARIABELEN OP
                                   NUL RESP. SPATIE ZETTEN ***
120  REM
130        LINE INPUT "VOER ARTIKELCODE IN (5): "; C$
140        IF  LEN(C$) <> 5  THEN
               PRINT "CODE MOET UIT 5 TEKENS BESTAAN. OPNIEUW AUB."    : GOTO 130
149  REM
150        LINE INPUT "VOER ARTIKELNAAM IN (12): "; N$
160        IF  LEN(N$) > 12  THEN
               PRINT "NAAM TE LANG. AFKORTEN TOT MAX. 12 TEKENS AUB."  : GOTO 150
170        IF  LEN(N$) < 12  THEN
               LET N$ = N$ + " "  : GOTO 170
179  REM
180        LINE INPUT "VOER BESTELHOEVEELHEID IN (MAX. 288): "; H$
190        IF  LEN(H$) > 3  THEN
               PRINT "MAX. 3 CIJFERS TOEGESTAAN. OPNIEUW AUB."         : GOTO 180
200        IF  LEN(H$) < 3  THEN
               LET H$ = H$ + " "  : GOTO 200
210        IF  VAL(H$) = 0  THEN
               PRINT "INVOERFOUT. ALLEEN CIJFERS TOEGESTAAN."          : GOTO 180
220        IF  VAL(H$) > 288  THEN
               PRINT "MAX. HOEVEELHEID IS 288. OPNIEUW AUB."           : GOTO 180
229  REM
230        LINE INPUT "VOER STUKPRIJS IN (MAX. 99.99): "; P$
240        IF  LEN(P$) > 5  THEN
               PRINT "MAX. 5 TEKENS TOEGESTAAN. OPNIEUW AUB."          : GOTO 230
250        IF  LEN(P$) < 5  THEN
               LET P$ = P$ + " "  : GOTO 250
260        IF  VAL(P$) = 0  THEN
               PRINT "ALLEEN CIJFERS EN/OF PUNT. OPNIEUW AUB."         : GOTO 230
270        IF  VAL(P$) > 99.99  THEN
               PRINT "MAX.STUKPRIJS IS 99.99. OPNIEUW AUB."            : GOTO 230
280  REM
290        LET  A$ = C$ + N$ + H$ + P$
300  REM
310        CLS          :  REM ** MAAK SCHERM SCHOON ***
320        PRINT "CODE", TAB(15), "HOEVEELHEID", TAB(30), "STUKPRIJS"
322        PRINT "----", TAB(15), "----------", TAB(30), "---------"
324        PRINT
330        PRINT RIGHT$(A$, 5), TAB(15), MID$(A$, 18, 3), TAB(30), LEFT$(A$, 5)
340  REM
```

4 WERKEN MET SEQUENTIËLE BESTANDEN

4.0 DOELSTELLINGEN

Na het doorwerken van dit hoofdstuk moet u in staat zijn numerieke en/of alfanumerieke gegevens in sequentiële gegevensbestanden op schijfgeheugen (bijvoorbeeld een floppy) of cassettes op te slaan en terug te lezen, gebruikmakend van de volgende BASIC-opdrachten:

- OPEN
- CLOSE
- INPUT#
- PRINT#
- KILL

Voor zover deze opdrachten eerder aan de orde zijn geweest komen in dit hoofdstuk andere mogelijkheden ter sprake.

4.1 INLEIDING

In het kader van dit hoofdstuk zullen we onder een gegevensbestand (Engels: data file) een gestructureerde verzameling alfanumerieke gegevens verstaan, die door een BASIC-programma kan worden verwerkt. Zo'n bestand kan zich op een magneetschijf bevinden – bijvoorbeeld een floppy of op een zogenaamde winchesterschijf – of op cassetteband. In dit hoofdstuk bespreken we uitsluitend bestanden op magneetschijven. De verschillen met cassettebandbestanden zijn echter miniem. Men kan er op ongeveer dezelfde wijze meer werken, en vanuit het programma gezien is er eigenlijk geen enkel principieel verschil. Mocht u alleen over cassette-opslagmogelijkheden beschikken dan is dit hoofdstuk – en de volgende – dus toch actueel.

Gegevensbestanden

Tot nu toe was u waarschijnlijk gewend bij iedere 'run' gegevens braaf met de hand in te toetsen, en wel naar aanleiding van het uitvoeren door de computer van INPUT- of 'LINE INPUT'-opdrachten. Bij grotere hoeveelheden nam u misschien uw toevlucht tot een combinatie van DATA- en READ-opdrachten. Daardoor kon u zich de moeite besparen van het steeds weer intoetsen van dezelfde gegevens. Beide methoden hebben nadelen: met de hand invoeren naar aanleiding van INPUT of LINE INPUT is arbeidsintensief, en bij DATA/READ zitten de gegevens als het ware vastgebakken in het programma. In beide gevallen zijn de gegevens slechts voor het ene programma beschikbaar.

In plaats hiervan kunnen gegevens ook worden ondergebracht in een op zichzelf staand bestand. Ze hoeven daarbij maar één keer ingetoetst te worden. Bovendien kunnen ze dan door verschillende programma's worden benaderd en gebruikt. Een van die programma's zal waarschijnlijk dienen om de gegevens in het bestand op te slaan. Is het bestand eenmaal op schijf of cassetteband vastgelegd, dan kunnen andere programma's daarop worden 'losgelaten'.

Een voorbeeld

Stel dat u de gegevens van uw persoonlijke adresboekje – met de namen, adressen en telefoonnummers van vrienden, kennissen, relaties, enz. – in een gegevensbestand op schijf wilt onderbrengen. U gebruikt daarvoor een speciaal invoerprogramma (hoe u dat kunt schrijven zien we verderop in dit hoofdstuk). Met een ander programma zou u dan aan de hand van de persoonsnaam – die daarbij als 'zoekcriterium' of 'sleutel' functioneert – telefoonnummers kunnen selecteren. Weer een ander programma zou adres- en/of telefoonnummerwijzigingen in het bestand kunnen doorvoeren. Een vierde programma zou adresetiketten, bijvoorbeeld in volgorde van postcode – kunnen produceren. En zo zouden we nog een tijdje kunnen doorgaan. Steeds is er sprake van één bestand dat door verschillende programma's wordt benaderd. Daarbij worden de gegevens overigens niet rechtstreeks op het opslagmedium zelf be- of verwerkt. Dat deel van een bestand dat het programma op een gegeven moment nodig heeft wordt namelijk eerst naar het werkgeheugen (dat wil zeggen het interne geheugen) gekopieerd. Eventueel gewijzigde gegevens kunnen daarna worden teruggeschreven naar het oorspronkelijke bestand, of naar een nieuwe versie daarvan.

Programmabestanden

Al dan niet bewust hebt u waarschijnlijk al enige ervaring met

bestanden op schijfgeheugen opgedaan. Bijvoorbeeld bij het opslaan of laden van een BASIC-programma met behulp van commando's zoals SAVE en LOAD. We spreken hierbij van *programmabestanden* (Engels: program files). Met deze bestanden wordt op een totaal andere manier gewerkt dan met gegevensbestanden. De inhoud van een programmabestand is een BASIC-programma – of een programma in een andere taal – waarop *commando's* (dus geen taalopdrachten) zoals LOAD, RUN, LIST en SAVE kunnen worden losgelaten. De inhoud van een gegevensbestand wordt echter verwerkt door middel van een programma, dus door opdrachten gesteld in programmeertaal. Met die opdrachten kan een gegevensbestand geheel of gedeeltelijk van het (externe) opslagmedium – de floppy bijvoorbeeld – naar het werkgeheugen worden overgebracht, en daar worden verwerkt. Commando's zoals LOAD, RUN, LIST en SAVE kunnen dus niet op gegevensbestanden worden toegepast.

Toetsvraag

(a) Beschrijf globaal hoe de inhoud van een gegevensbestand kan worden benaderd.

(a) Door een BASIC-programma waarin speciale opdrachten voor het werken met bestanden voorkomen.

4.2 GEGEVENSOPSLAG OP SCHIJFGEHEUGEN

Voor MSX-computers wordt meestal de Micro Floppydisk van 3½ Inch gebruikt. Op zo'n schijfje kan, als het zogenaamd 'double density' is, toch meer dan 300 K bytes (300 × 1024 tekens) worden opgeslagen. Daarnaast zijn voor MSX-computers ook 5¼-inch drives te verkrijgen. Deze drives gebruiken de bekende diskette, zoals die bij veel microcomputers (PC's) gebruikt worden. Een voordeel van deze laatste is dat de diskettes 'overdraagbaar' zijn naar systemen die draaien onder MS DOS en die werken met GW BASIC.

Micro Floppies zijn vaak 'enkelzijdig', dat wil zeggen dat slechts één kant van het schijfje in de hoes informatie kan bevatten. Bij 5¼ diskettes is 'double sided' (tweezijdig) het meest gangbaar. Daarnaast kunnen de schijfjes een 'single density' of 'double density' hebben. Dit is de 'dichtheid' waarmee gegevens op de schijfjes worden opgeslagen.

Voorbeeld

Als voorbeeld nemen we een single-density 5¼ inch diskette onder de loep. Figuurlijk uiteraard, maar toch ook min of meer letterlijk. De schijf blijkt in 45 concentrische cirkels te zijn verdeeld (zie figuur 4-1). Die cirkels worden *sporen* genoemd (Engels: tracks). Elk spoor is op zijn beurt onderverdeeld in 10 *sectoren*. Elke sector kan één zogenaamd (fysiek) *record* bevatten, dat wil zeggen 256 tekens oftewel bytes*. Voor gegevensopslag zijn bij deze schijf ongeveer 215 records beschikbaar. De opslagcapaciteit van de gehele schijf is dus circa 215x256 = 55040 bytes. Bij twee of meer schijven ligt de capaciteit per schijf dankzij een geringere 'overhead' (dat wil zeggen ruimte nodig voor systeemgebruik, het zogenaamde disk operating system) hoger, namelijk 83000 bytes.

Andere systemen kunnen uiteraard andere spoor- en/of sectorindelingen, en, ten gevolge daarvan, andere opslagcapaciteiten hebben.

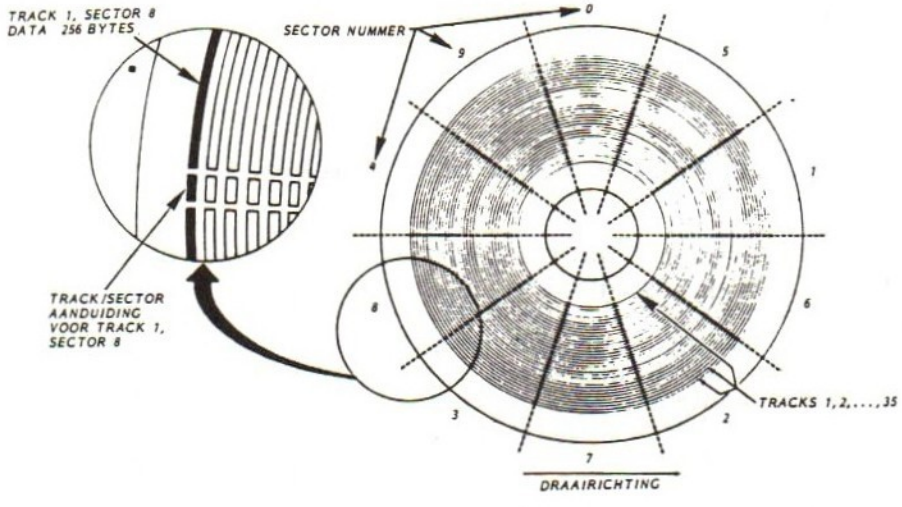


fig.4-1. Een single-density 5¼ inch diskette.

* Op dit begrip komen we straks terug.

Het begrip byte

De 'byte' is een eenheid van geheugenruimte. Dat geldt zowel voor het werkgeheugen van de computer als voor externe geheugenmedia zoals schijven. Bytes bestaan uit kleinere delen – bits – die een binair cijfer (dat wil zeggen 0 of 1) kunnen voorstellen (Engels: bit = binary digit = binair cijfer). Door de bits op een bepaalde wijze te coderen – een bepaald patroon van enen en nullen te kiezen – kan men een byte onder meer een alfanumeriek teken laten voorstellen. Een gangbare code daarvoor is het eerder besproken ASCII, de onvolprezen American Standard Code for Information Interchange.

Voor iedere letter van het alfabet, speciaal teken of cijfer dat in een BASIC-programma als string wordt opgegeven, bijvoorbeeld in een toekenningsopdracht zoals LET B$ = "3", wordt dus één byte geheugenruimte gebruikt. De benodigde geheugenruimte voor het opslaan van strings in gegevensbestanden is in principe dus gelijk aan de lengte van de string. Verder kunnen enkele bytes nodig zijn voor 'huishoudelijke' doeleinden (de zogenaamde overhead). Het precieze aantal hangt af van het computertype en het gebruikte opslagmedium.

Toetsvraag

(a) Hoeveel bytes zijn nodig om de string in onderstaande toekenningsopdracht op een schijf op te slaan?

```
LET N$ = "MOOI WEER VANDAAG!"
```

(a) 18, exclusief overhead. Merk op dat ook de spatie een byte in beslag neemt.

Opslag van getallen

Het berekenen van de opslagruimte voor alfanumerieke gegevens is, zoals we hebben gezien, relatief simpel. Elk teken neemt gewoon één byte in beslag. Bij numerieke waarden die *niet* als string worden opgegeven ligt het moeilijker. Bovendien kan dit sterk variëren van het ene computertype naar het andere. Een en ander hangt af van het soort getal en de precisie daarvan.

Doorgaans kunnen drie verschillende getalsoorten worden onderscheiden, elk met een eigen interne representatiewijze of coderingsvorm. In de eerste plaats zijn er de zogenaamde integer-getallen, dat wil zeggen gehele positieve of negatieve getallen (inclusief nul). Daarvoor kunnen integer-variabelen worden gebruikt, aangegeven door een procentteken achter de naam (bijvoorbeeld G1%). De waarde van deze integer-getallen kan liggen tussen -32768 en 32767. Ze nemen steeds twee bytes in beslag.

Gebroken getallen kunnen met twee verschillende nauwkeurigheden worden vastgelegd. Voor beide nauwkeurigheden bestaat een afzonderlijk gegevenstype: single precision en double precision. Single precision getallen hebben in MSX BASIC een nauwkeurigheid van maximaal zes significante cijfers. 'Significant' wil zeggen 'echt'. Het getal 0.0032, bijvoorbeeld, heeft twee significante cijfers, namelijk 3 en 2. De twee nullen achter het decimaalteken hebben namelijk betrekking op de grootte van het getal, niet op zijn precisie. Voor single precision getallen zijn vier bytes nodig. De bijbehorende variabele wordt aangegeven met een uitroepteken achter de naam, bijvoorbeeld G1!.

Voor double precision getallen geldt bij MSX BASIC een precisie van maximaal 16 significante cijfers. Ze nemen acht bytes in beslag. De bijbehorende variabele wordt aangegeven zonder toevoeging, gewoon G1 dus.

Standaard werkt MSX BASIC met double precision getallen, vandaar dat hiervoor geen extra toevoeging achter de naam van de variabele (zoals G1%, G1! of G1$) nodig is.

Een praktische toepassing

Als illustratie grijpen we even terug naar het persoonlijke adresboekje van de vorige paragraaf. Voor de in het bestand onder te brengen gegevens definiëren we de volgende veldlengten:

gegeven	type	vorm	aantal bytes
naam	alfanumeriek	--	20
straat en huisnummer	alfanumeriek	--	25
woonplaats	alfanumeriek	--	10
postcode	alfanumeriek	XXXXbXX	7
telefoonnummer	alfanumeriek	XXXbXXXXXX of XXXXXbXXXX	10
leeftijd	integer	--	2
geboortedatum	alfanumeriek	XXbXXbXXXX	10

b = spatie (blank)
X = alfanumeriek teken

subtotaal :	84
overhead :	7
totaal :	91

Toetsvragen

(a) Hoeveel fysieke records zouden bij deze toepassing nodig zijn voor de opslag van de gegevens van 150 relaties?

(b) Van hoeveel relaties kunnen we op bovenstaande wijze de gegevens onderbrengen op een schijf met een capaciteit van 55000 bytes?

(a) Aantal bytes = 91×150 = 13650, aantal fysieke records = 13650 / 256 = 53.3, afgerond 54 records.

N.B. Het groeperen van alle 7 gegevens in één lange string zou per relatie een besparing van 6 bytes opleveren, in totaal dus circa 4 records.

(b) 55000 / 91 = 604

Nog enkele begrippen

Iedere groep van zeven gegevens in bovenstaande toepassing noemen we een (logisch) record*. Een handeling waarbij zo'n record als geheel in een gegevensbestand wordt opgenomen noemen we een *transactie*. Gegevens zo groeperen en verwerken maakt het programmeren – en ook de structuur van het programma – aanzienlijk eenvoudiger.

Een gegevensbestand waarbij de gegevens als het ware één lange doorlopende reeks vormen (het ene record na het andere) en in die volgorde moeten worden verwerkt, noemen we een *sequentieel bestand*. We kunnen dat vergelijken met een geluidsband waarop achter elkaar muziekstukken – de 'records' – zijn opgenomen. Om een bepaald stuk muziek te bereiken moet de band vanaf het begin worden afgespoeld. Dat wil zeggen de gegevens moeten in recordvolgorde – oftewel sequentieel – worden benaderd.

Voor een gegevensbestand op magneet- of cassetteband gaat deze vergelijking aardig op. Bij schijfgeheugens kunnen de records van een sequentieel bestand echter in min of meer willekeurige volgorde door elkaar liggen. Ze zouden in principe ook rechtstreeks te benaderen zijn, op dezelfde manier als we de naald van een platenspeler direct boven een bepaald muziekstuk kunnen zetten. We spreken dan van een *direct-toegankelijk bestand* (Engels: 'random access' of

---

* Opmerking van de vertaler/bewerker:
De auteurs spreken van 'dataset' in plaats van 'record'. In aansluiting op de algemene terminologie in de bestandsorganisatie, en om misverstanden te voorkomen ten gevolge van de betekenis die het woord 'dataset' onder meer in IBM-kringen heeft, is in dit boek voor de term 'record' gekozen. Om verwarring met de eerder besproken 'fysieke records' oftewel 'blocks' te voorkomen wordt hieraan de kwalificatie 'logisch' toegevoegd.

'direct access'). De interne organisatie van het bestand moet daar echter wel op gericht zijn. We houden ons voorlopig bij de sequentiële bestanden, die we ons als een lange aaneengesloten gegevensstroom blijven voorstellen. Ook als ze dat op schijfgeheugen – fysiek gezien – niet zijn. In de hoofdstukken 7 en 8 gaan we uitgebreid op de direct-toegankelijke bestandsvorm in.

4.3 KIEZEN TUSSEN SEQUENTIËLE EN DIRECT-TOEGANKELIJKE BESTANDEN

We hebben gezien dat we voor de opslag van een bestand op schijfgeheugen zowel een sequentiële als een direct-toegankelijke vorm kunnen kiezen. Wat zijn nu eigenlijk de voor- en nadelen van deze systemen? Welnu, dankzij hun eenvoudige vorm hebben sequentiële bestanden het voordeel dat ze minder ruimte innemen. Er is immers geen ruimte nodig om de plaats van het ene record ten opzichte van het andere aan te geven. Het ene record komt eenvoudig na het andere. Daartegenover staat het nadeel dat het betrekkelijk moeilijk is een sequentieel bestand te veranderen, oftewel muteren. Ze zijn min of meer ontworpen voor de opslag van gegevens die weinig aan verandering onderhevig zijn. Anders gezegd, de sequentiële vorm is alleen geschikt voor gegevens met een lage mutatiegraad.

Verder is de tijd die nodig is om een gegeven in een bestand te bereiken korter bij direct-toegankelijke bestanden. Om bijvoorbeeld het 450-ste record in een sequentieel bestand te bereiken moet de computer eerst de voorgaande 449 records doorlopen. Bij direct-toegankelijke bestanden kan de computer dat 450-ste record rechtstreeks bereiken.

Toetsvraag

(a) Met welke drie factoren moet men rekening houden bij het kiezen tussen de sequentiële en de direct-toegankelijke opslagvorm?

(a) Efficient gebruik van de beschikbare opslagruimte, verwachte mutatiegraad van het bestand, toegangstijd.

4.4 HET INITIALISEREN VAN SEQUENTIËLE BESTANDEN

Bij het gebruik van schijfgeheugens heeft iedere micro een zogenaamd disk operating system (DOS) nodig. In het Nederlands spreken we van *bedrijfssysteem*. Dit regelt onder andere het gegevens-

transport tussen het interne werkgeheugen van de computer en het 'externe' schijfgeheugen. Zoals gezegd wordt dit systeem - op zich ook een programma, maar dan in kant-en-klare uitvoerbare machine-instructies - op een deel van de schijf gehuisvest. Bij MSX-computers is het MSX-DOS in het systeem ingebouwd.

In een MSX BASIC-programma moet worden aangegeven met hoeveel bestanden (tegelijkertijd) gewerkt gaat worden. Na MAXFILES = 2 kunnen twee verschillende bestanden met de buffernummers 1 en 2 worden geopend.* De grootte van de bestanden hoeft u niet op te geven - bij het opslaan van de gegevens wordt zonodig steeds een block toegevoegd.

De OPEN-opdracht voor sequentiële bestanden

Om een file te gebruiken moet hij vanuit het BASIC-programma worden 'geopend'. Daarvoor bestaat een speciale opdracht: de OPEN. Deze opdracht 'vertelt' de computer welke bestanden het programma nodig heeft en op welke wijze ze worden gebruikt. Hij kan het beste in het initialisatiedeel van het programma worden opgenomen.

In MSX BASIC ziet de OPEN-opdracht voor sequentiële bestanden er in zijn algemeenheid zo uit:

OPEN naam FOR mode A$ bufnum

Hierin is OPEN een vaste term (keyword of sleutelwoord) die letterlijk wordt vermeld. De drie andere onderdelen van de opdracht worden door de programmeur bepaald. Daarbij geldt het volgende.

1. *Naam*. Als eerste onderdeel van de OPEN-opdracht wordt de naam van het te openen bestand aangegeven. Dit kan met een alfanumerieke constante of variabele plaatsvinden.

   voorbeeld:

   ```
   140  OPEN "NAAM1" .........
   ```

   of:

   ```
   144  LINE INPUT "NAAM VAN BESTAND: "; N$
   150  OPEN N$ ...........
   ```

2. *Mode*. Hiermee wordt de gebruikswijze van het bestand aangegeven. De mogelijkheden zijn weergegeven in de volgende tabel.

* De zogenaamde 'default' of verstekwaarde - dat wil zeggen de standaardwaarde die het systeem hanteert als u niets opgeeft - hierbij is 1.

mode	betekenis
INPUT	: Invoermode voor sequentieel bestand. Gegevens worden van schijf gelezen en naar het werkgeheugen overgebracht.
OUTPUT	: Uitvoermode voor sequentieel bestand. Gegevens worden van het werkgeheugen overgebracht naar het schijfgeheugen.
APPEND	: Toevoegen aan einde van een bestaand sequentieel bestand.

De OPEN-opdracht voor een 'random' (= direct toegankelijk) bestand wordt behandeld in hoofdstuk 7.

voorbeeld:

```
140  OPEN "LEDEN" FOR INPUT ...
140  OPEN "ARTI" FOR OUTPUT ...
140  OPEN N$ FOR APPEND ...
```

3. *Bufnum*. Recordtransport tussen werk- en schijfgeheugen vindt blockgewijs plaats via een buffergebied in het werkgeheugen. Dit gebied is 256 bytes groot en kan dus precies één block bevatten. Bij een leesopdracht van schijf worden de gegevens eerst in dit gebied ondergebracht. Van daaruit zijn ze beschikbaar voor verwerking door het programma. Omgekeerd worden de gegevens bij wegschrijven naar schijf eerst in zo'n buffergebied verzameld. Pas als de buffer vol is worden de gegevens daadwerkelijk naar schijf weggeschreven (zie fig.4-2). Voor ieder bestand dat op een gegeven moment open is moet een buffer aanwezig zijn. Zo'n buffer wordt door middel van een getal van 1 t/m 15 als tweede onderdeel van de OPEN-opdracht – dus op de plaats van 'bufnum' in de algemene vorm hierboven – vastgelegd. Let wel: dit getal dient als identificatienummer van de buffer en heeft niets met aantallen buffers te maken. Het aantal wordt opgegeven met MAXFILES = n.

voorbeeld:

```
140  OPEN "NAAM1" FOR APPEND AS 1
```

of:

```
142  INPUT "BUFFERNUMMER: "; B%
150  OPEN "NAAM1" FOR OUTPUT AS B%
```

Merk op dat het buffernummer een geheel (= integer) getal is, en dat daarom een integer-variabele (B%) wordt gebruikt. In plaats van 1 mag ook #1 als buffernummer gebruikt worden.

In MSX BASIC kan eventueel een specificatie van de schijfeenheid, waarop het bestand aanwezig is, aan de naam voorafgaan. Bijvoorbeeld: B:NAAM1. Bovendien kan de naam gevolgd worden door een facultatieve extensie, bijvoorbeeld NAAM1.ABC. De naam zelf mag maximaal uit 8 tekens bestaan, de uitbreiding uit maximaal 3.

Tot zover de vorm van de OPEN. Over het gebruik daarvan ten slotte nog enkele opmerkingen. Een bestand dat voor invoer wordt geopend (INPUT-mode) moet op dat moment uiteraard wel bestaan. Is het bestand op het moment van openen niet aanwezig dan volgt een foutmelding. Verder zal het duidelijk zijn dat een bestand wordt gecreëerd door in OUTPUT-mode te openen met een naam die nog niet eerder is gebruikt, bijvoorbeeld:

```
180  OPEN "KLAD" FOR OUTPUT AS 2
```

Maar let op: bestaat er al een bestand met de aangegeven naam, dan gaat in OUTPUT-mode de gehele inhoud van het bestand op het moment van openen verloren!

Onthouden dus: *Het heropenen van een bestand in OUTPUT-mode resulteert in vernietiging van de inhoud van dat bestand!*

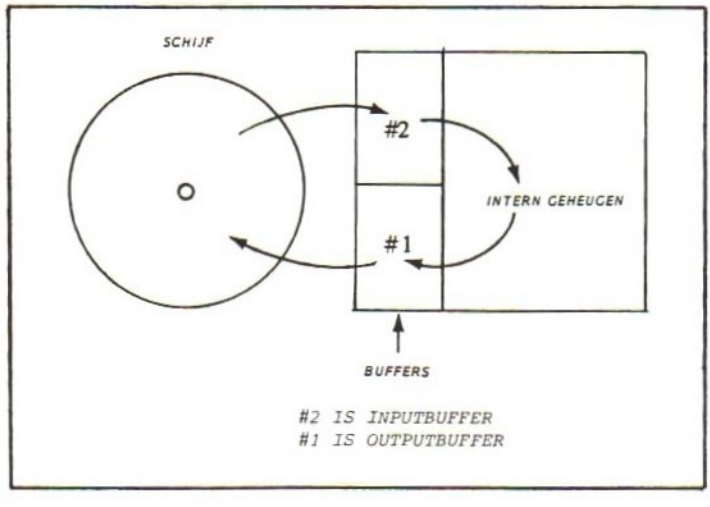


fig.4-2. Gegevenstransport via buffers.

Toetsvragen

(a) Schrijf een opdracht om een sequentieel bestand met de naam TELNUM voor invoer via buffer 6 te openen.

(b) Schrijf opdrachten om een sequentieel bestand voor invoer te openen, waarbij de naam van het bestand via dialoog wordt opgegeven en het buffernummer 2 is.

(c) Open een sequentieel bestand voor invoer, waarbij naam en buffernummer via dialoog worden opgegeven.

(a)
```
110  OPEN "TELNUM" FOR INPUT AS 6
```

(b)
```
140  LINE INPUT "NAAM VAN HET BESTAND: "; N$
150  OPEN N$ FOR OUTPUT AS 2
```

(c)
```
210  LINE INPUT "NAAM VAN HET BESTAND: "; N$
220  INPUT "BUFFERNUMMER: "; B%
230  OPEN N$ FOR INPUT AS B%
```

De CLOSE-opdracht

Ieder bestand dat met een OPEN-opdracht is geopend, dient vóór beëindiging van het programma met een bijbehorende CLOSE-opdracht te worden afgesloten.

De CLOSE heeft de volgende algemene vorm:

CLOSE bufnum

waarbij bufnum uit één of meer buffernummers, onderling gescheiden door komma's, bestaat. Het buffernummer moet uiteraard hetzelfde zijn als het buffernummer waaronder de desbetreffende file werd geopend.

Uitvoering van een CLOSE-opdracht heeft tot gevolg dat eventueel nog in de buffer aanwezige gegevens zonodig worden weggeschreven naar schijf (hierover straks meer) en dat de bufferruimte wordt vrijgegeven. Hetzelfde buffernummer kan daarna voor een ander bestand worden gebruikt.

enkele vcorbeelden:

800 CLOSE 1	Afsluiten en vrijgeven van buffer 1.
820 CLOSE 1, 4, 3	Afsluiten en vrijgeven van buffers 1, 4 en 3
860 CLOSE	Afsluiten en vrijgeven van alle op dat moment geopende files.

Het buffer-probleem

We zagen dat de buffer als tussenstation fungeert tussen werk- en schijfgeheugen. Schrijft u uitvoergegevens weg naar schijf, dan komen die dus eerst in de buffer terecht. Pas als de buffer helemaal vol is (256 bytes) worden de gegevens naar schijf getransporteerd, en het bestand bijgewerkt.

Wat gebeurt er nu als de buffer maar gedeeltelijk gevuld is, terwijl er verder geen gegevens meer zijn om weg te schrijven? U zou kunnen verwachten dat de inhoud van de gedeeltelijk gevulde buffer gewoon naar schijf wordt overgebracht. Maar helaas, de laatste loodjes blijken bijzonder zwaar te wegen. De resterende gegevens worden namelijk pas weggeschreven bij uitvoering van een CLOSE-opdracht voor het desbetreffende buffergebied. De CLOSE 'schoont' dus als het ware de buffer (Engels: to flush = schonen, 'doorspoelen'). Zo'n CLOSE is dus van essentieel belang voor de integriteit – de nauwkeurigheid en betrouwbaarheid – van het bestand.

Over loodjes gesproken: mocht uw programma ten gevolge van een of andere fout het loodje moeten leggen en met een gedeeltelijk gevulde buffer voortijdig afbreken (aborteren), dan kan in direct-mode* een CLOSE worden gegeven. De resterende gegevens worden dan alsnog overgeheveld naar schijf. Zo'n noodsprong pleit echter niet voor de kwaliteit van het programma, en zou in een professionele omgeving onaanvaardbaar zijn. Verderop in dit hoofdstuk zullen we zien hoe dit soort problemen te vermijden zijn.

Toetsvraag

(a) Welke functies heeft een CLOSE-opdracht?

(a) De computer te dwingen zijn bufferinhoud naar schijf te schrijven, en de bufferruimte vrij te geven.

Tenslotte nog enkele opmerkingen. Volgens de MSX gebruikershandleiding wordt een buffer onder normale omstandigheden automatisch geschoond bij programma-beëindiging bij:
- een STOP- of END-opdracht;
- een technische fout van de schijfeenheid (disk error);
- het geven van een RUN-, NEW- of CLEAR-opdracht door de gebruiker;
- toevoegen of verwijderen van een programmaregel.

U doet er echter goed aan hier niet op te vertrouwen. Ook niet als u een ander BASIC-systeem met soortgelijke voorzieningen gebruikt. Laat bewust een geprogrammeerde CLOSE vóór beëindiging van het programma uitvoeren, of geef in noodgevallen een CLOSE in direct-mode.

Verder is het van belang dat u nooit ofte nimmer een schijf van de aandrijfeenheid verwijdert als zich daarop nog een geopend bestand bevindt. Overtuig u ervan dat het bestand netjes met een CLOSE is afgesloten. In twijfelgevallen kan altijd een directe CLOSE worden gegeven.

---

* 'Direct-mode' wil zeggen een opdracht invoeren zonder regelnummer, afgesloten met RETURN. De opdracht wordt dan rechtstreeks uitgevoerd.

Toetsvraag

(a) Wat moet u doen als een programma, dat bezig was gegevens naar een schijf weg te schrijven, voortijdig zonder CLOSE afbreekt?

(a) Het bestand alsnog afsluiten met een CLOSE-opdracht in direct-mode.

4.5 SCHRIJVEN NAAR EEN SEQUENTIEEL BESTAND

In het voorgaande hebt u geleerd met behulp van de OPEN-opdracht een verbinding tot stand te brengen tussen de computer en een bestand op schijfgeheugen. Met de CLOSE-opdracht kon u die verbinding verbreken. In deze paragraaf leert u hoe u gegevens in zo'n schijfbestand kunt opslaan. In MSX BASIC is dit met een speciale vorm van de PRINT-opdracht te doen. Deze is gelijk aan de bekende PRINT, maar dan met een 'hekje' (#) en een nummer na het sleutelwoord PRINT. Het nummer slaat op de buffer.

De PRINT #-opdracht wordt in principe op dezelfde wijze gebruikt als de gewone PRINT, bijvoorbeeld:

```
240 PRINT #1, A, B, C
```

Deze regel heeft tot gevolg dat de waarden, die aan de numerieke variabelen A, B en C zijn toegekend, naar schijf worden weggeschreven via buffer 1. Het buffernummer kan in de PRINT #-opdracht in de vorm van een variabele worden meegegeven, bijvoorbeeld:

```
270 LET X = 3
280 PRINT #X, A, B, C
```

Essentieel hierbij is uiteraard dat aan de variabele vooraf het gewenste buffernummer is toegekend.

Een voorbeeld

Het volgende programma schrijft gegevens naar een bestand op schijf, te beginnen bij het eerste block. Omdat het bestand in O-mode wordt geopend gaan gegevens, die eventueel al op de schijf aanwezig waren, verloren.

```
100  REM * DEMONSTRATIE SCHRIJVEN NAAR SCHIJFBESTAND
110  REM
120        OPEN "DEMO1" FOR OUTPUT AS 1: REM *** OF BIJV. "B:DEMO1" ***
130  REM
200  REM * LEES GEGEVENS EN SCHRIJF WEG
210  REM
220        READ A, B, C
230        IF A = -1 THEN 990:          REM: CHECK OP EINDE GEGEVENS
240        PRINT #1, A; B; C
250        GOTO 220
260  REM
900  REM * GEGEVENS VOOR DEMO
910        DATA 23, 26, 18, 22, 20, 19
920  REM
930  REM * VLAG VOOR EINDE GEGEVENS (AFSLUITWAARDEN)
940        DATA -1, -1, -1
950  REM
980  REM * SLUIT BESTAND AF EN STOP
990        CLOSE 1
999        END
```

Het bestand bevat na afloop:

```
23  26  18
22  20  19
```

Merk op dat in regel 240 puntkomma's zijn gebruikt om de uitvoervariabelen A, B en C te scheiden. In plaats daarvan kunnen ook komma's worden toegepast, bijvoorbeeld:

```
240  PRINT #1, A, B, C
```

Dit heeft tot gevolg dat er tussen de opgeslagen gegevens op schijf meer ruimte ontstaat. De komma functioneert namelijk als een soort tabulator, zoals die ook wel op schrijfmachines voorkomt. Het effect is overigens hetzelfde als bij een gewone PRINT naar het scherm van de computer. Ook daarbij treedt een 'tab'-effect op bij gebruik van komma's.

Advies:
Gebruik bij de PRINT #-opdracht puntkomma's als scheidingsteken tussen uitvoervariabelen. Dit bespaart schijfruimte.

N.B. Direct na het buffernummer mag *geen* puntkomma staan.

Toetsvraag

(a) Schrijf een opdracht die de waarden van de variabelen J, K en L via buffer 3 wegschrijft naar schijf.

(a) `160 PRINT #3, J; K; L`

Wegschrijven van alfanumerieke gegevens

Om gegevens later correct van schijf te kunnen teruglezen is het uiteraard van belang dat ze op de juiste wijze zijn weggeschreven. Bij numerieke gegevens is dat niet zo'n probleem. Gebruik gewoon puntkomma's tussen de uitvoervariabelen en behalve de eerder genoemde ruimtebesparing zult u de gegevens ook zonder problemen kunnen teruglezen.

Bij alfanumerieke gegevens ligt dat moeilijker. Om dat te demonstreren nemen we het volgende voorbeeld onder de loep.

```
220  LINE INPUT "NAAM:           "; N$
230  PRINT #1, N$
240  LINE INPUT "TELEFOONNUMMER: "; T$
250  PRINT #1, T$
260  LINE INPUT "LEEFTIJD:       "; L$
270  PRINT #1, L$
```

In dit programmadeel wordt iedere string met een afzonderlijke PRINT # weggeschreven. In het bestand zelf worden de opeenvolgende strings automatisch met een 'carriage control' teken (ASCII-code 013 decimaal) gescheiden. Bij deze – primitieve – structuur levert het teruglezen geen moeilijkheden op. Het ligt echter voor de hand zulke gegevens te combineren tot één record. Maar dat zou op zijn beurt tot gevolg hebben dat de computer als het ware alle gegevens aan elkaar 'plakt'. Schrijven we dus in bovenstaand voorbeeld in plaats van regels 230, 250 en 270:

```
270  PRINT #1, N$; T$; L$
```

met N$ = "GERARD", T$ = "012-345678" en L$ = "25", dan resulteert dit in het volgende record-'beeld' (Engels: image) op schijf:

GERARD012-34567825

Ruimtebesparend – dat wel – maar het is niet zo eenvoudig om hier de oorspronkelijke deelstrings weer uit te filteren! Dit probleem kunnen we voorkomen door bij het wegschrijven komma's op te nemen (als het ware te forceren) tussen de verschillende gegevens. Dat is eenvoudig te doen door tussen de uitvoervariabelen in de PRINT #-opdracht een komma als alfanumerieke constante te plaatsen. Dus:

```
270  PRINT #1, N$; ","; T$; ","; L$
```

Het bijbehorende recordbeeld wordt dan:

GERARD,012-345678,25

Tenslotte nog een geniepigheidje. Stel dat met een READ-opdracht gegevens vanuit een DATA-opdracht zoals:

```
920  DATA BALLPOINTS, VILTSTIFTEN, POTLODEN
```

worden gelezen en daarna worden weggeschreven. In dat geval kan aan het einde van de laatste string (POTLODEN) onbedoeld een 'carriage control' teken in het recordbeeld terechtkomen. Dit is te voorkomen door bij de laatste string aanhalingstekens te gebruiken, dus "POTLODEN". Terwille van de netheid verdient het dan aanbeveling ook alle andere gegevens in de DATA-opdracht tussen aanhalingstekens te schrijven.

Toetsvraag

(a) Schrijf programma-opdrachten om alle strings in regel 920 (zie boven) in te lezen en vervolgens via buffer 2 in een bestand op te slaan.

(a)
```
240  READ N1$, N2$, N3$
250  PRINT #2, N1$; ","; N2$; ","; N3$
```

Aanhalingstekens forceren

We zagen zojuist dat het zinvol kan zijn bij het wegschrijven naar schijf komma's tussen de verschillende deelstrings van een record te forceren. Daardoor kunnen die deelstrings later zonder problemen worden teruggelezen. Maar wat moeten we doen als de komma zelf deel uitmaakt van de string? Bijvoorbeeld:

```
240  LET N$ = "VAN DEN BOSCH, G.J.M."
```

Bij het wegschrijven van deze string komt alleen de tekst *tussen* de aanhalingstekens in het bestand terecht. De aanhalingstekens zelf hebben namelijk alleen syntactische betekenis (dat wil zeggen hebben alleen betrekking op de vorm van het programma) en worden daarom bij het wegschrijven achterwege gelaten. Ten gevolge hiervan gaan de achternaam en de voorletters als het ware een afzonderlijk bestaan leiden – het worden in feite twee verschillende alfanumerieke constanten. We zullen dus zelf moeten zorgen voor aanhalingstekens. Maar hoe? Aanhalingstekens forceren op dezelfde manier als met komma's – dus met drie opeenvolgende aanhalingstekens – gaat in verband met syntactische problemen niet op. Het BASIC-systeem ziet de vorm """ (dat wil zeggen drie opeenvolgende dubbele aanhalingstekens) namelijk als een nulstring ("") gevolgd

door een los aanhalingsteken. Dit laatste teken zal normaal gesproken niet in de context van de opdracht passen en dus aanleiding geven tot een SYNTAX ERROR.

De oplossing hiervoor is een kunstgreep toe te passen, en wel in de vorm van de CHR$-functie. Het ASCII-codegetal voor het aanhalingsteken is 34. Door CHR$(34) in de PRINT-opdracht op te nemen kunnen we toch een aanhalingsteken forceren. Het voorbeeld wordt dan:

```
240  LET N$ = "VAN DEN BOSCH, G.J.M."
250  PRINT #1, CHR$(34); N$; CHR$(34)
```

Maar we zijn er nog steeds niet. Schrijven we namelijk in één record meer gegevens op deze wijze weg, dan moeten ze onderling ook weer met komma's worden gescheiden. Dus:

```
250  PRINT #1, CHR$(34); N$; CHR$(34); ","; CHR$(34); T$; CHR$(34)
```

Een hele kerstboom – toegegeven – maar wel noodzakelijk!

Voorbeelden

```
150  INPUT "ARTIKELNUMMER: "; N
160  INPUT "PRIJS        : "; P
170  INPUT "HOEVEELHEID  : "; H
210  PRINT #3, N; P; H
```

Merk op: puntkomma's tussen numerieke variabelen.

```
430  LINE INPUT "KLANTNUMMER: "; K$
440  LINE INPUT "DATUM      : "; D$
450  PRINT #3, K$; ","; D$
```

Merk op: geforceerde komma tussen stringvariabelen.

```
600  LINE INPUT "NAAM (EERST ACHTERNAAM): "; N$
610  LINE INPUT "GEBOORTEDATUM          : "; D$
620  PRINT #3, CHR$(34); N$; CHR$(34); ","; D$
```

Merk op: aanhalingstekens geforceerd aan weerszijden van N$ (een variabele die een komma kan bevatten).

```
700  LINE INPUT "SALARISNUMMER:     "; N$
710  INPUT      "TARIEFGROEP  :     "; T
720  INPUT      "BRUTO JAARINKOMEN: "; S
730  PRINT #3, N$; ","; T; S
```

Merk op: tussen gemengde gegevens (numeriek/alfanumeriek) een geforceerde komma na de string (niet altijd nodig, maar wel aan te bevelen).

Toetsvragen

(a) Schrijf met PRINT-opdrachten alle met behulp van onderstaande opdrachten ingevoerde gegevens naar schijf via buffer 3.

```
300  INPUT "TOTAAL AANTAL MONSTERS: "; M
310  INPUT "AANTAL GROENE MONSTERS: "; G
320  PRINT .....

500  LINE INPUT "NAAM      : "; N$
510  LINE INPUT "ADRES     : "; A$
520  LINE INPUT "WOONPLAATS: "; W$
530  PRINT .....

800  LINE INPUT "TITEL, INCL. LEESTEKENS: "; T$
810  LINE INPUT "EERSTE TEKSTREGEL      : "; R1$
820  PRINT .....

900  LINE INPUT "TITEL VAN BOEK           : "; T$
910  LINE INPUT "AUTEUR (EERST ACHTERNAAM): "; A$
920  INPUT      "AANTAL BLADZIJDEN        : "; B
930  PRINT .....
```

(a)
```
320  PRINT #3, M; G
530  PRINT #3, N$; ","; A$; ","; W$
820  PRINT #3, CHR$(34); T$; CHR$(34); ","; CHR$(34); R1$; CHR$(34)
930  PRINT #3, T$; ","; CHR$(34); A$; CHR$(34); ","; B
```

MSX apparatuur. Foto Philips. (Idem blz. 106)

Een voorbeeld: inventarisprogramma (1)

Het gebruik van bestanden eist planning. Punten waarmee rekening moet worden gehouden zijn onder andere:

1. Wat moet wel en wat niet in een record worden opgenomen?
2. Hoe groot zijn de afzonderlijke gegevens die in een record moeten worden opgenomen?
3. Hoe groot moet het record als geheel worden?
4. Hoe groot kan het bestand als geheel worden?
5. Is er speciale aandacht nodig voor het scheiden van deelvelden binnen een record?
6. Nauwkeurigheid en geldigheid van de in het bestand op te nemen gegevens.
7. Afsluiting van het bestand.

Als voorbeeld geven we een programma waarin enkele van deze aspecten verwerkt zijn. Met het programma kan een inventarisbestand voor zakelijk of privé-gebruik worden samengesteld. Het programma bevat onder andere een (kleine) inleidende module en – met het oog op punt 6 – een module voor gegevensverificatie. Voor demonstratiedoeleinden is opzettelijk een fout gemaakt waarop straks wordt ingegaan.

```
100 REM   PROGRAMMA VOOR HET AANMAKEN VAN EEN INVENTARISBESTAND
110 REM
120 REM   VARIABELEN
130 REM    - V$   : OMSCHRIJVING VOORWERP
140 REM    - N    : AANTAL STUKS
150 REM    - W    : WAARDE PER STUK
160 REM    - A$   : HULPVARIABELE T.B.V. DIALOOG
165 REM
170 REM   BESTANDEN
180 REM    - SEQUENTIEEL BESTAND OP FLOPPY
190 REM
```

```
200 REM   INITIALISEREN
205 REM
210       OPEN "INVENT" FOR OUTPUT AS 1: REM *** OF BIJV. "B:INVENT" ***
220 REM
230 REM   GEGEVENSINVOER
235 REM
240       LINE INPUT "OMSCHRIJVING (MAX. 20 TEKENS): "; V$
250       IF LEN(V$) > 20 THEN
             PRINT "AFKORTEN TOT MAX. 20 TEKENS AUB" :          GOTO 240
260       IF LEN(V$) = 0  THEN
             PRINT "OMSCHRIJVING OPGEVEN AUB" :                 GOTO 240
265       IF LEN(V$) < 20 THEN V$ = V$ + " "  :                 GOTO 265
270       INPUT      "AANTAL STUKS                 : "; N
280       IF N <> INT(N) THEN
             PRINT "ALLEEN GEHELE GETALLEN AUB" :               GOTO 270
290       IF N <= 0 THEN
             PRINT "AANTAL OPGEVEN AUB"                         GOTO 270
300       INPUT      "GELDWAARDE PER STUK          : "; W
310       IF W <= 0 THEN 350
315 REM
316 REM   WEGSCHRIJVEN NAAR BESTAND EN HERHALEN
317 REM
320       PRINT #1, V$; ","; N, W
330       GOTO 240
340 REM
341 REM   CHECK OP WAARDE
342 REM
350       LINE INPUT "WAARDE <= 0. OK? (J/N)       : "; A$
360       IF LEFT$(A$,1) = "N" THEN
             PRINT "DAN OPNIEUW INVOEREN AUB" :                 GOTO 300
370       GOTO 320
380 REM
400 REM   BESTAND AFSLUITEN EN STOPPEN
405 REM
410       CLOSE 1
420 REM
999       END
```

Toetsvraag

(a) Dit programma bevat een fout. Welke?

(a) De CLOSE-opdracht in regel 410 – nodig voor het schonen van de buffer – kan nooit worden uitgevoerd. In de volgende paragraaf gaan we hier verder op in.

Afsluiten van het bestand

Zoals gezegd is het afsluiten van het bestand één van de aspecten waarmee tijdens de planning rekening moet worden gehouden. Daarbij moet onder meer worden bepaald hoe de gebruiker te kennen kan geven dat hij klaar is met invoeren. In het bovenstaande voorbeeld zou dit kunnen uitkristalliseren in een van de volgende constructies:

```
238  PRINT "TYPE STOP OF GEEF"
245  IF LEFT$(V$,4) = "STOP" THEN 410
```

of:

```
325  LINE INPUT "NOG MEER GEGEVENS (J/N): "; A$
326  IF LEFT$(V$,1) = "N" THEN 410
```

Tenslotte twee punten waaraan we nog even herinneren. Ten eerste: bij het openen van een sequentieel bestand voor uitvoer (FOR OUTPUT) worden eventueel reeds aanwezige gegevens vernietigd. Toevoegen aan een bestand dat eenmaal bestaat gaat eenvoudig met APPEND. We zullen daar verder op ingaan in hoofdstuk 5. In de tweede plaats: vergeet niet uw programma te documenteren. Vooral een beschrijving van de opbouw van de gebruikte bestanden kan u later goed van pas komen!

Toetsvraag

(a) Waarom is het nodig de computer te informeren dat alle gegevens voor een bestand zijn ingevoerd?

(a) Zodat het bestand met een CLOSE kan worden afgesloten.

4.6 TERUGLEZEN VAN GEGEVENS UIT EEN SEQUENTIEEL BESTAND

Nu we weten hoe gegevens weg te schrijven naar een sequentieel bestand, kunnen we de omgekeerde weg gaan be(w)(h)andelen: gegevens teruglezen uit een reeds aangemaakt bestand. Hierbij is het belangrijk te weten hoe de gegevens oorspronkelijk werden weggeschreven. Het teruglezen kan dan op betrekkelijk eenvoudige wijze plaatsvinden.

Om een bestand te kunnen lezen moet het eerst voor invoer (INPUT) worden geopend. Lezen kan daarna met de INPUT #-opdracht plaatsvinden.

voorbeeld:

```
350  OPEN "BEST1" FOR INPUT AS 2
360  INPUT #2, A, B, C
```

Ten gevolge van regel 360 worden drie gegevens van buffer 2 ingelezen, en toegekend aan de variabelen A, B, C. In principe moet het type van de variabelen (numeriek of string) in de INPUT #-opdracht overeenkomen met het type van de gegevens die vanuit het bestand worden ingelezen. Is het eerstvolgende gegeven in het

bestand bijvoorbeeld numeriek, dan moet de variabele die op dat moment in de INPUT #-opdracht aan de orde is eveneens numeriek zijn. Is dit niet het geval dan kunnen er, inhoudelijk gezien, complicaties optreden.

Technisch gezien maakt het echter weinig uit of een 'binnenkomend' gegeven als numerieke waarde dan wel als string werd weggeschreven. Er ontstaat weliswaar een waarschuwing als men probeert een numerieke waarde uit het bestand toe te kennen aan een stringvariabele in de INPUT #-opdracht, maar de toekenning gaat wel door. In het omgekeerde geval – het bestandsgegeven is een string terwijl in de INPUT #-opdracht een numerieke variabele aan de beurt is – wordt zonder protest de waarde nul toegekend. Het programma stopt in geen van beide gevallen.

De vraag doet zich voor of dat wel gunstig is. Het probleem van een programmastop met een geopend bestand wordt wel omzeild, maar er komt een nieuw probleem voor in de plaats, namelijk ongeldige gegevens. En dat na alle moeite die bij het invoeren (hopelijk ...) is gedaan om uitsluitend correcte gegevens in het bestand op te nemen. Vandaar dat we er eigenlijk toch niet omheen kunnen om precies te weten hoe de gegevens oorspronkelijk werden weggeschreven, dat wil zeggen numeriek of alfanumeriek, en in welke volgorde. Van strings moeten we bovendien de lengte kennen.

We keren nu even terug naar het inventarisprogramma van de vorige paragraaf. Daarbij werden steeds drie gegevens weggeschreven: een alfanumerieke voorwerpomschrijving (V$), gevolgd door een hoeveelheid (N) en een geldwaarde (W). Deze variabelen functioneerden uiteraard als opslagruimte voor de weg te schrijven waarden, dat wil zeggen niet de variabelen zelf maar hun waarden waren van belang. Bij het teruglezen kunnen we dus zonder bezwaar andere variabelen gebruiken als bestemming voor de binnenkomende gegevens. De enige voorwaarde is dat het type van de variabele ('normaal gesproken') moet overeenkomen met het type van het gegeven.

Toetsvraag

(a) Welke van de volgende opdrachten kunnen we gebruiken om gegevens van het inventarisbestand van § 4.5 in te lezen?

```
(1)   270  INPUT #1, A, B, C
(2)   270  INPUT #1, A$, B, C
(3)   270  INPUT #1, A, B$, C$
```

(a) Opdracht 2

Een voorbeeld: inventarisprogramma (2)

Het volgende programma leest gegevens van het bestand geproduceerd door inventarisprogramma 1 (zie § 4.5), en maakt daarvan een eenvoudig overzicht.

```
100 REM   LEES GEGEVENS UIT INVENTARISBESTAND
110 REM
120 REM   VARIABELEN
130 REM    - B$: BESCHRIJVING VOORWERP
140 REM    - H : HOEVEELHEID
150 REM    - G : GELDWAARDE
160 REM
170 REM   BESTANDEN
180 REM    - INVENT, SEQUENTIEEL, FLOPPY
190 REM
200       OPEN "INVENT" FOR INPUT AS #1
210 REM
215 REM   DRUK KOP AF VAN OVERZICHT
220 REM
225       PRINT "OMSCHRIJVING"; TAB(22); "AANTAL"; TAB(30); "WAARDE"
230       PRINT
240 REM
250 REM   GEGEVENS INLEZEN VAN BESTAND EN AFDRUKKEN
255 REM
260       INPUT #1, B$, H, G
270       PRINT B$; TAB(22); H; TAB(30); W
280       GOTO 260
290 REM
300 REM   BESTAND AFSLUITEN EN STOPPEN
305 REM
310       CLOSE
315
999       END
```

De uitvoer zou kunnen zijn:

```
OMSCHRIJVING          AANTAL  WAARDE

STOELEN               4       120.0
TAFEL                 1       250.0
LAMP                  1       69.5
SCHILDERIJ            1       300.0
```

End-of-file

Het bovenstaande programma bevat een onvolkomenheid. De RUN eindigt namelijk met de melding INPUT PAST END IN 260. De strekking hiervan is dat we als het ware proberen voorbij het einde van het bestand te lezen. Het programma wordt daarbij afgebroken. In ons geval is dat geen ramp, omdat de fout pas optreedt als we klaar zijn met het afdrukken van de lijst. Maar stel dat we na het afdrukken van die lijst meer hadden willen doen. Dan zouden we van te voren - dat wil zeggen vóórdat we proberen te ver te lezen - moeten weten dat het einde van het bestand is bereikt.

Gelukkig bestaat daar inderdaad een mogelijkheid voor: de EOF-functie (EOF = end-of-file = einde bestand). Om de werking van die functie duidelijk te maken moeten we eerst even stilstaan bij het begrip 'wijzer' (Engels: pointer), dat in hoofdstuk 2 bij de behandeling van READ/DATA ook al ter sprake kwam. Net als bij deze opdrachten hanteert het besturingssysteem (DOS) een wijzermechanisme om bij te houden welk bestandsgegeven aan de beurt is om te worden ingelezen. Bij het openen wordt de wijzer ingesteld op het begin van het bestand. Hij wijst op dat moment dus het eerste gegeven aan. Bij iedere uitvoering van een INPUT #-opdracht wordt de wijzer evenveel posities vooruitgeschoven als er variabelen zijn in de invoerlijst. Bij de PRINT # gaat dit op dezelfde manier. De wijzer geeft dus steeds een nieuw gegeven aan bij INPUT #, en geeft de plaats aan waar het eerstvolgende gegeven wordt vastgelegd bij PRINT #.

voorbeelden:

PRINT #1, A$	De wijzer wordt één positie opgeschoven.
PRINT #1, N, N$	De wijzer wordt twee posities opgeschoven.
PRINT #1, W, X, Y, Z	De wijzer wordt vier posities opgeschoven.

Verder moeten we weten dat bestanden door het besturingssysteem worden afgesloten met een speciale binaire code: het *end-of-file* teken. Bij iedere toevoeging van gegevens aan een bestand met de PRINT #-opdracht wordt behalve de wijzer ook het end-of-file teken opgeschoven. Dit gebeurt geheel automatisch. Het laatste teken van een file – ook na CLOSE – is dus altijd het end-of-file teken.

We weten inmiddels dat bij het lezen van gegevens uit een bestand met de INPUT-opdracht de wijzer steeds naar het eerstvolgende beschikbare gegeven in het bestand (of beter gezegd, de buffer) 'kijkt'. Door nu de eerdergenoemde end-of-file functie EOF te gebruiken kunnen we constateren of dat teken het end-of-file teken

is. In MSX BASIC zou dat als volgt kunnen:

```
IF EOF(1) THEN CLOSE 1 : GOTO 800
```

Het getal tussen de haakjes – het *functie-argument* – is het nummer van de desbetreffende buffer. Merk op dat er tussen IF en THEN geen vergelijking staat in de normale zin van het woord. De reden hiervoor is dat de EOF-aanroep op zich eigenlijk een logische waarde vertegenwoordigt. Het einde van het bestand is immers wel of niet bereikt (waar/onwaar). Het effect van bovenstaande opdracht is dus dat het bestand gesloten, en er naar regel 800 gesprongen wordt indien het einde van het bestand is bereikt. Is het eerstvolgende teken geen end-of-file, dan wordt de programma-uitvoering bij de eerstvolgende opdracht voortgezet.

Dankzij dit stuk 'gereedschap' kan het inventarisprogramma zo worden aangepast dat voortijdige beëindiging ten gevolge van end-of-file wordt voorkomen. We voegen regel 257 toe en veranderen regel 280:

```
257  IF EOF(1) THEN 310
280  GOTO 257
```

Een andere mogelijkheid zou zijn:

```
257  IF EOF(1) THEN CLOSE 1 : STOP
```

waarbij regel 310 kan vervallen.

Toetsvraag

(a) Wat betekent EOF?

(b) Welke betekenis heeft het argument van de EOF-functie?

(a) end-of-file (teken)
EOF-functie

(b) Geeft het nummer aan van de buffer die op het voorkomen van een EOF-teken wordt onderzocht.

4.7 AANMAKEN EN TERUGLEZEN IN ÉÉN PROGRAMMA

Tot nu toe hebben we uitsluitend programma's bekeken waarbij een bestand werd aangemaakt òf gelezen, maar niet beide. Mits we een bestand, dat voor één van beide gebruiksvormen is geopend, eerst met CLOSE afsluiten, is er echter geen enkel bezwaar tegen het bestand opnieuw voor dezelfde of de andere gebruiksvorm in hetzelfde programma te heropenen.

We illustreren dit aan de hand van een eenvoudig programma voor het registreren van kwaliteitsgegevens bij een produktieproces. De kwaliteit van een produkt wordt daarbij aangegeven door een cijfer van 1 t/m 6. Voor ieder produkt wordt dit cijfer – de kwaliteitsklasse – ingelezen. Het programma maakt daarna een overzicht van het aantal produkten per klasse.

```
100 REM   DEMO AANMAKEN EN LEZEN VAN BESTAND IN EEN PROGRAMMA
110 REM
120 REM   *** KWALITEITSBEWAKING ***
130 REM
140 REM   INVOEREN VAN KWALITEITSKLASSEN EN AFDRUKKEN VAN OVERZICHT
150 REM
160 REM   VARIABELENLIJST
170 REM    - B$     : BESTAND
180 REM    - K%     : KWALITEITSKLASSE
190 REM    - I      : ARRAYINDEX
200 REM    - V(6)   : VERZAMELARRAY VOOR KWALITEITSKLASSEN
210 REM               V(1) = AANTAL PRODUKTEN IN KLASSE 1
220 REM               V(2) = AANTAL PRODUKTEN IN KLASSE 2
230 REM               ENZ.
240 REM   BESTAND
250 REM    - SEQUENTIEEL OP FLOPPY, NAAM DOOR GEBRUIKER IN TE VOEREN
260 REM
270 REM   INITIALISEER ARRAY
280 REM
290       FOR I = 1 TO 6 : LET V(I) = 0: NEXT I
300 REM
310 REM   VRAAG NAAM VAN BESTAND EN OPEN VOOR UITVOER
320 REM
330       LINE INPUT "NAAM VAN BESTAND: "; B$
340       OPEN B$ FOR OUTPUT AS #1
350 REM
360 REM   GEGEVENSINVOER
370 REM
380       PRINT "GEEF UITSLUITEND GEHELE GETALLEN VAN 1 T/M 6 OP, "
390       PRINT "OF 99 ALS U WILT STOPPEN"
400       PRINT ""
410       INPUT "KWALITEITSKLASSE: "; K%
420       IF K% = 99 THEN 490
430       IF K% < 1 OR K% > 6 THEN
             PRINT "GEHEEL GETAL 1 T/M 6 OPGEVEN AUB" :       GOTO 410
440       PRINT #1, K%
450       GOTO 410
460 REM
```

```
470 REM   AFSLUITEN BESTAND EN OPENEN VOOR INVOER
480 REM
490       CLOSE 1
500       OPEN B$ FOR INPUT AS #1
510 REM
520 REM   LEES KWALITEITSKLASSEN EN SOMMEER IN VERZAMELARRAY
530 REM
540       IF EOF(1) THEN 610
550          INPUT #1, I
560          LET V(I) = V(I) + 1
570       GOTO 540
580 REM
590 REM   OVERZICHT MAKEN
600 REM
610       CLS : REM *** SCHERM SCHOONMAKEN ***
620       PRINT "OVERZICHT AANTAL PRODUKTEN PER KWALITEITSKLASSE"
630       PRINT
640       PRINT "KLASSE", "AANTAL"
650       FOR I = 1 TO 6
660           PRINT I, V(I)
670       NEXT I
680 REM
690 REM   AFSLUITEN EN STOPPEN
700 REM
710       CLOSE 1
999       END
```

Toetsvragen

De volgende toetsvragen hebben betrekking op bovenstaand programma.

(a) Via welke opdracht wordt de naam van het gegevensbestand bepaald?

(b) Via welke opdracht worden de ingevoerde kwaliteitsklassen gecontroleerd?

(c) Hoe 'weet' de computer dat alle gegevens zijn ingevoerd?

(d) Waarom zijn binnen één en hetzelfde programma twee "CLOSE 1"-opdrachten opgenomen?

(e) Wat is het doel van regel 540?

(f) Hoeveel verschillende waarden kan de variabele I in regel 580 aannemen?

(a) Opdracht 330.
(b) Opdracht 430.
(c) De gebruiker voert als 'klasse' de waarde 99 in.
(d) Het bestand moet zowel na invoer als na uitvoer worden afgesloten.
(e) Controleren of het einde van het bestand is bereikt.
(f) Zes (1 t/m 6).

Oefenprogramma

We schrijven nu samen een soortgelijk programma als hierboven. Het programma moet een gegevensbestand met de naam DEMO1 maken, en vervolgens de inhoud van dat bestand afdrukken. In een ijverige bui hebben we de klus zelf al grotendeels geklaard. U wordt verzocht de ontbrekende regels (200, 240, 270, 310, 340, 350 en 390) te leveren. Let daarbij vooral op de REMARKs.

```
100 REM   DEMONSTRATIEPROGRAMMA
110 REM
120 REM   VARIABELENLIJST
130 REM    - G1$    : WEG TE SCHRIJVEN GEGEVEN
140 REM    - G2$    : TERUG TE LEZEN GEGEVEN
150 REM    - X      : BESTURINGSVARIABELE FOR/NEXT
160 REM
170 REM   BESTAND : SEQUENTIEEL, FLOPPY, 1 STRING PER RECORD
180 REM             NAAM = DEMO1
190 REM
200       ..................................... : REM *** OPEN BESTAND ***

210 REM   WEGSCHRIJVEN VAN 8 STRINGS NAAR GEGEVENSBESTAND
220       FOR X = 1 TO 8
230           LET G1$ = "TEST" + STR$(X)

240       ..................................... : REM *** SCHRIJF WEG ***

250       NEXT X


260 REM   SLUIT NU HET BESTAND

270       .....................................

280 REM   BEVESTIGING AFDRUKKEN
290       PRINT "BESTAND AFGEMAAKT EN AFGESLOTEN"


300 REM   OPEN BESTAND VOOR INVOER (HOEFT NIET MET ZELFDE BUFFERNUMMER)

310       .....................................


320 REM   LEES GEGEVENS UIT BESTAND EN DRUK AF
330 REM   TEST DAARBIJ OP EINDE BESTAND MET EOF-FUNCTIE

340       .....................................


350       .....................................

360       PRINT G2$
370       GOTO 340

380 REM   SLUIT BESTAND EN STOP

390       .....................................

400       PRINT "BESTAND AFGESLOTEN"


999       END
```

```
(a) 200  OPEN "DEMO1" FOR OUTPUT AS 1
(b) 240      PRINT #1, G1$
(c) 270  CLOSE 1
(d) 310  OPEN "DEMO1" FOR INPUT AS 1
(e) 340  IF EOF(1) THEN 390
(f) 350  INPUT #1, G2$
(g) 390  CLOSE 1
```

(h) Geef nu de uitvoer van het programma.

```
(h) BESTAND AANGEMAAKT EN AFGESLOTEN
    TEST1
    TEST2
    TEST3
    TEST4
    TEST5
    TEST6
    TEST7
    TEST8
    BESTAND AFGESLOTEN
```

Tenslotte

Bij het werken met bestanden lijkt het na het geven van een RUN-opdracht vaak alsof er niets gebeurt. Na lang wachten verschijnt dan eindelijk de verlossende boodschap Ok. op het scherm. De onervaren gebruiker krijgt hierbij al gauw het vermoeden dat er iets mis is. Tussentijdse meldingen kunnen dan paniekreacties voorkomen.

Toetsvraag

(a) Bevat het bovenstaande programma dergelijke tussentijdse meldingen? Zo ja, in welke regel(s)?

(a) Ja, regels 290 en 400.

4.8 VERWIJDEREN VAN BESTANDEN (KILL)

Bestanden kunnen desgewenst van schijf worden verwijderd. Het commando hiervoor in MSX BASIC heet - zeer toepasselijk - KILL.

De vorm van het commando is:

KILL naam (met eventuele uitbreiding, bijv. OPBRE.JAN)

waarbij naam een alfanumerieke constante of variabele is die de naam van het te verwijderen bestand aangeeft, bijvoorbeeld KILL "INVENT". Behalve als systeemcommando (dus buiten een programma om) kan KILL ook als programma-opdracht worden gebruikt.

KILL moet met de nodige voorzichtigheid worden gehanteerd. Eenmaal gegeven is het desbetreffende bestand namelijk voorgoed van het toneel verdwenen, waardoor ettelijke uren of zelfs dagen werk verloren kunnen gaan. Het is daarom aan te raden van belangrijke bestanden steeds een reserve-exemplaar op een andere schijf – een zogenaamde backup – bij te houden. Gezien de verstrekkende gevolgen van een KILL raden we bovendien aan deze opdracht nooit in een programma op te nemen (tenzij het om een onbelangrijk bestand gaat), maar het bestand expliciet door middel van de commandovorm te verwijderen.

KILL wordt overigens nogal eens verward met CLOSE. Het effect van CLOSE is dat de toegekende buffer wordt losgekoppeld van het bestand. In het geval van uitvoer worden bovendien eventueel resterende buffergegevens naar schijf overgebracht. Na een CLOSE bestaat het bestand dus nog steeds. KILL daarentegen verwijdert de naam van het bestand uit de zogenaamde *directory* (een adrestabel oftewel 'inhoudsopgave' van de schijf) en maakt het bestand daardoor onbereikbaar. *Overigens moet een bestand ge-CLOSEd zijn alvorens het ge-KILL-ed wordt.* Gebeurt dit niet dan kunnen andere bestanden op de schijf verminkt worden.

In het boek "Programmeercursus MSX BASIC" vindt u veel informatie over KILL en andere BASIC DOS-opdrachten als COPY, FORMAT, NAME, MERGE en FILES.

4.9 TOETSVRAGEN

In de toetsvragen van deze paragraaf wordt u gevraagd eerst een programma te schrijven waarmee gegevens in een bestand worden opgeslagen. Vervolgens wordt gevraagd een bijbehorend programma te schrijven waarmee de weggeschreven gegevens op scherm of printer zichtbaar worden gemaakt. De bestanden die u hierbij maakt zullen in hoofdstuk 5 worden gebruikt. Het is dan ook aan te raden zowel de programma's als de bestanden te bewaren.

Mocht u niet over de benodigde schijfapparatuur beschikken, dan kunt u desondanks de programma's alvast schrijven. Er zullen namelijk slechts minimale aanpassingen nodig zijn om uw programma voor cassette-apparatuur geschikt te maken. In hoofdstuk 6 gaan we daar nader op in.

Van alle 'aanmaak'programma's wordt - terwille van de uniformiteit tussen uw uitwerkingen en de onze - het inleidende deel gegeven.

1(a) Schrijf een programma om een bestand te maken dat aan de volgende eisen voldoet:

- per record vier gegevens
- eerste twee gegevens zijn strings
- resterende twee gegevens zijn numerieke waarden die als strings worden ingevoerd.

Test bij invoer op nulstrings, en controleer of de laatste twee gegevens inderdaad numeriek zijn. Converteer deze string-waarden eerst naar numerieke waarden, toegekend aan numerieke variabelen, alvorens ze in een bestand op te nemen. Neem minstens vijf records in het bestand op.

```
100 REM   OPLOSSING VRAAG 1A
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$     : GEGEVEN 1, ALFANUMERIEK
140 REM    - B$     : GEGEVEN 2, ALFANUMERIEK
150 REM    - C$/C   : GEGEVEN 3, ALFANUMERIEK
160 REM    - D$/D   : GEGEVEN 4, NUMERIEK
180 REM
190 REM   BESTANDEN
200 REM    - VRAAG1 (SEQUENTIEEL, FLOPPY)
210 REM
```

(b) Schrijf een bijbehorend programma om de inhoud van het bestand op het scherm zichtbaar te maken.

2(a) U bent geheel in de ban van de automatisering geraakt en hebt besloten uw boodschappenlijstje voor de kruidenier voortaan via de computer samen te stellen. Daarvoor gebruikt u een bestand met de naam BOODSCHAP, waarin voor ieder artikel een omschrijving van maximaal 20 tekens en de aan te schaffen hoeveelheid (numerieke waarde) wordt opgenomen. Schrijf het programma, en neem in het bestand minstens zes records op.

```
100 REM   OPLOSSING VRAAG 2A: BOODSCHAPPENLIJST
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$   : OMSCHRIJVING ARTIKEL
140 REM    - H    : HOEVEELHEID / AANTAL
150 REM    - N$   : NAAM BESTAND (DOOR GEBRUIKER OP TE GEVEN)
160 REM    - D$   : VARIABELE T.B.V. DIALOOG
170 REM
```

(b) Schrijf een bijbehorend programma om het lijstje op scherm of printer zichtbaar te maken.

3(a) Schrijf een programma om ten behoeve van een klein bedrijf een eenvoudig klantenbestand bij te houden. Voor iedere klant dient in het bestand een record met de volgende drie gegevens voor te komen:

- klantnummer (precies 5 cijfers)
- klantnaam (maximaal 20 tekens)
- kredietwaardigheid (codegetal 1, 2, 3, 4 of 5)

Neem in het programma invoercontroles op voor nulrespons (dat wil zeggen de gebruiker voert niets in) en voor geldig klantnummer, -naam en kredietwaardigheid. Bouw minstens 12 records op, gebruikmakend van steeds verschillende klantnummers gerangschikt volgens toenemende grootte, bijvoorbeeld 19652, 19653, 19654, enz.

```
100 REM   OPLOSSING VRAAG 3A: AANMAKEN KLANTENBESTAND
110 REM
120 REM   VARIABELENLIJST
130 REM    - B$    : NAAM VAN BESTAND
140 REM    - N$    : KLANTNUMMER
150 REM    - K$    : KLANTNAAM
160 REM    - C$/C  : KREDIETCODE (1, 2, 3, 4 OF 5)
170 REM    - A$    : VARIABELE T.B.V. DIALOOG
180 REM
190 REM   BESTAND : SEQUENTIEEL/FLOPPY
200 REM             NAAM DOOR GEBRUIKTER IN TE VOEREN
210 REM
```

(b) Schrijf een bijbehorend programma om de inhoud van het bestand op het scherm zichtbaar te maken.

4(a) Schrijf een programma om gegevens in een transactiebestand op te nemen. Een transactiebestand bestaat uit gegevens die betrekking hebben op zakelijke transacties zoals bijvoorbeeld in het bankwezen, detailhandel of postorderbedrijf. Ten behoeve van deze vraag nemen we aan dat voor iedere transactie een record van maximaal 14 tekens wordt geproduceerd met de volgende indeling:

rekeningnummer (5 tekens) — bedrag (7 tekens)

transactiecode (2 tekens)

De naam wordt door de gebruiker opgegeven. Maak met het programma twee verschillende bestanden aan met elk zeven records (transacties). Gebruik daarbij de volgende rekeningnummers:

bestand 1	bestand 2
10762	10761
18102	18203
43611	43611
43611	80111
43611	80772
80223	80772
98702	89012

(Let wel: alleen de rekeningnummers zijn gegeven; de transactiecodes en bedragen bepaalt u zelf.)

```
100 REM   OPLOSSING VRAAG 4A
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$   : NAAM VAN BESTAND
140 REM    - H$/H : HULPVARIABELE
150 REM    - R$   : REKENINGNUMMER (5 CIJFERS)
160 REM    - T$   : TRANSACTIECODE
170 REM    - F$   : BEDRAG
180 REM    - I    : BESTURINGSVARIABELE FOR/NEXT
190 REM
200 REM   BESTAND : SEQUENTIEEL/FLOPPY, NAAM DOOR GEBRUIKER TE BEPALEN
210 REM
```

(b) Schrijf een bijbehorend programma om de inhoud van de geproduceerde bestanden op het scherm zichtbaar te maken.

5(a) Schrijf een programma om een bestand op te bouwen met willekeurige namen en adressen (inclusief postcodes). De records dienen de volgende indeling te hebben:

```
/1        20/21          40/41   47/48        67/
```

Denk aan invoercontroles en het aanvullen van deelvelden met spaties. Neem minstens zes records in het bestand op.

```
100 REM   OPLOSSING VRAAG 5A: ADRESBESTAND
110 REM
120 REM   VARIABELENLIJST
130 REM
140 REM    - N$(20)   : NAAM
150 REM    - A$(20)   : (STRAAT-)ADRES, INCL. HUISNUMMER
160 REM    - P$(7)    : JA NATUURLIJK ... DE POSTCODE
170 REM    - W$(20)   : WOONPLAATS
180 REM    - H1$/H1   : HULPVARIABELE T.B.V. TEKENCONTROLE EN DIALOOG
190 REM    - H2$(67)  : HULPVARIABELE T.B.V. SAMENSTELLEN RECORD
200 REM
210 REM   BESTAND     : NAPW, SEQUENTIEEL/FLOPPY
220 REM
```

(b) Schrijf een bijbehorend programma om de inhoud van het bestand op het scherm zichtbaar te maken.

6(a) Schrijf een programma dat een standaardbrief als sequentieel uitvoerbestand produceert. De tekstregels dienen als een gegeven te worden ingevoerd en weggeschreven. Gebruik het programma om drie verschillende brieven met de namen BRIEF1, BRIEF2 en BRIEF3 te maken. Ieder bestand dient minstens 3 tekstregels te bevatten.

```
100 REM   OPLOSSING VRAAG 6A: STANDAARD BRIEF MAKEN
110 REM
120 REM   VARIABELENLIJST
130 REM    - R$    : TEKSTREGEL
140 REM    - B$    : NAAM VAN BESTAND
150 REM
160 REM   BESTAND : BRIEFX, WAARBIJ X = 1, 2, 3 ENZ.
170 REM             SEQUENTIEEL/FLOPPY
180 REM
```

(b) Schrijf een bijbehorend programma om de brieven op scherm of printer zichtbaar te maken.

4.10 ANTWOORDEN OP DE TOETSVRAGEN

1(a)
```
100 REM   OPLOSSING VRAAG 1A
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$     : GEGEVEN 1, ALFANUMERIEK
140 REM    - B$     : GEGEVEN 2, ALFANUMERIEK
150 REM    - C$/C   : GEGEVEN 3, NUMERIEK
160 REM    - D$/D   : GEGEVEN 4, NUMERIEK
170 REM    - X$     : VARIABELE T.B.V. DIALOOG
180 REM
190 REM   BESTANDEN
200 REM    - VRAAG1 (SEQUENTIEEL, FLOPPY)
210 REM
220       OPEN "VRAAG1" FOR OUTPUT AS 1
230 REM
240 REM   GEGEVENSINVOER EN -CONTROLE
250 REM
260       LINE INPUT "GEGEVEN 1 (ALFANUMERIEK): "; A$
270       IF LEN(A$) = 0 THEN
             PRINT "INVOER OPGEVEN AUB"  : GOTO 260
275 REM
280       LINE INPUT "GEGEVEN 2 (ALFANUMERIEK): "; B$
290       IF LEN(B$) = 0 THEN
             PRINT "INVOER OPGEVEN AUB"  : GOTO 280
```

(wordt vervolgd)

```
295 REM
300       LINE INPUT "GEGEVEN 3 (NUMERIEK)    : "; C$
310       IF LEN(C$) = 0 THEN
             PRINT "INVOER OPGEVEN AUB"  : GOTO 300
320       IF VAL(C$) = 0 THEN
             PRINT "ALLEEN GETALLEN OPGEVEN AUB"  : GOTO 300
330       LET C = VAL(C$)
335 REM
340       LINE INPUT "GEGEVEN 4 (NUMERIEK)    : "; D$
350       IF LEN(D$) = 0 THEN
             PRINT "INVOER OPGEVEN AUB"  : GOTO 340
360       IF VAL(D$) = 0 THEN
             PRINT "ALLEEN GETALLEN OPGEVEN AUB"  : GOTO 340
370       LET D = VAL(D$)
380 REM
390 REM   WEGSCHRIJVEN EN EVENTUEEL HERHALEN
400 REM
410       PRINT #1, A$; ","; B$; ","; C; D
420       LINE INPUT "MEER GEGEVENS? (J/N)    : "; X$
430       IF LEFT$(X$,1) = "J" THEN 260
440 REM
450 REM   AFSLUITEN EN STOPPEN
460 REM
470       CLOSE
480       PRINT " " : PRINT "BESTAND AFGESLOTEN"
999       END
```

1(b)

```
100 REM   OPLOSSING VRAAG 1B
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$     : GEGEVEN 1, ALFANUMERIEK
140 REM    - B$     : GEGEVEN 2, ALFANUMERIEK
150 REM    - C$/C   : GEGEVEN 3, NUMERIEK
160 REM    - D$/D   : GEGEVEN 4, NUMERIEK
170 REM
180 REM   BESTANDEN
190 REM    - VRAAG1 (SEQUENTIEEL, FLOPPY)
200 REM
210       OPEN "VRAAG1" FOR INPUT AS 1
220 REM
230 REM   GEGEVENS INLEZEN EN AFDRUKKEN
240 REM
250       IF EOF(1) THEN 320
260          INPUT #1, A$, B$, C, D
270          PRINT     A$, B$, C, D
280       GOTO 250
290 REM
300 REM   AFSLUITEN EN STOPPEN
310 REM
320       CLOSE 1
330       PRINT " " : PRINT "EINDE OVERZICHT, BESTAND GESLOTEN"
999       END
```

2(a)
```
100 REM   OPLOSSING VRAAG 2A: BOODSCHAPPENLIJST
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$  : OMSCHRIJVING ARTIKEL
140 REM    - H   : HOEVEELHEID / AANTAL
150 REM    - N$  : NAAM BESTAND (DOOR GEBRUIKER OP TE GEVEN)
160 REM    - D$  : VARIABELE T.B.V. DIALOOG
170 REM
180 REM   NAAM BESTAND OPVRAGEN EN OPENEN VOOR UITVOER
190 REM
200       LINE INPUT "NAAM VAN BESTAND: "; N$
210       OPEN N$ FOR OUTPUT AS 1
220 REM
230 REM   GEGEVENSINVOER EN -CONTROLE
240 REM
250       PRINT "GEEF 'STOP' ALS U KLAAR BENT"
260       PRINT ""
270       LINE INPUT "ARTIKEL (MAXIMAAL 20 TEKENS OF 'STOP'): "; A$
280       IF A$ = "STOP" THEN 450
290       IF LEN(A$) = 0 THEN
             PRINT "ARTIKELOMSCHRIJVING OF STOP GEVEN AUB"  : GOTO 270
300       IF LEN(A$) > 20 THEN
             PRINT "MAX. 20 TEKENS. OPNIEUW AUB"            : GOTO 270
310 REM
320       INPUT "HOEVEELHEID: "; H
330       IF H >= 1 AND H < 10 THEN 400
340          PRINT "HOEVEELHEID WAS "; H; "OK? (J/N): ";
350          LINE INPUT D$
360          IF LEFT$(D$,1) = "N" THEN 320
370 REM
380 REM   WEGSCHRIJVEN EN HERHALEN
390 REM
400       PRINT #1, A$; ","; H
410       GOTO 270
420 REM
430 REM   AFSLUITEN EN STOPPEN
450       CLOSE 1
460       PRINT " " : PRINT "BESTAND GESLOTEN"
999       END
```

2(b)
```
100 REM   OPLOSSING VRAAG 2B: BOODSCHAPPENLIJST AFDRUKKEN
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$  : OMSCHRIJVING ARTIKEL
140 REM    - H   : HOEVEELHEID / AANTAL
150 REM    - N$  : NAAM BESTAND (DOOR GEBRUIKER OP TE GEVEN)
160 REM
170 REM   NAAM BESTAND OPVRAGEN EN OPENEN VOOR INVOER
180 REM
190       LINE INPUT "NAAM VAN BESTAND: "; N$
200       OPEN N$ FOR INPUT AS 1
210 REM
```

(wordt vervolgd)

```
220 REM   GEGEVENS INLEZEN EN AFDRUKKEN
230 REM
240       PRINT "ARTIKEL", "HOEVEELHEID" : PRINT ""
250       IF EOF(1) THEN 320
260          INPUT #1, A$, H
270          PRINT     A$, H
280       GOTO 250
290 REM
300 REM   AFSLUITEN EN STOPPEN
310 REM
320       CLOSE 1
330       PRINT "" : PRINT "EINDE OVERZICHT, BESTAND GESLOTEN"
999       END
```

3(a)

```
100 REM   OPLOSSING VRAAG 3A: AANMAKEN KLANTENBESTAND
100 REM
120 REM   VARIABELENLIJST
130 REM    - B$    : NAAM VAN BESTAND
140 REM    - N$    : KLANTNUMMER
150 REM    - K$    : KLANTNAAM
160 REM    - C$/C  : KREDIETCODE (1, 2, 3, 4 OF 5)
170 REM    - A$    : VARIABELE T.B.V. DIALOOG
180 REM
190 REM   BESTAND : SEQUENTIEEL/FLOPPY
200 REM             NAAM DOOR GEBRUIKER IN TE VOEREN
210 REM
220 REM   INITIALISATIES
230 REM
240       LINE INPUT "NAAM VAN BESTAND: "; B$
250       OPEN B$ FOR OUTPUT AS 1
260 REM
270 REM   GEGEVENSINVOER EN -CONTROLE
280 REM
290       PRINT "GEEF 'STOP' (ZONDER AANH. TEKENS) OM TE EINDIGEN"
300       PRINT ""
310 REM
320       LINE INPUT "NUMMER VAN KLANT (5 CIJFERS).  : "; N$
330       IF N$ = "STOP" THEN 570
340       IF LEN(N$) = 0 THEN
             PRINT "GETAL OF 'STOP' INVOEREN"           : GOTO 320
350       IF LEN(N$) <> 5 THEN
             PRINT "INVOERFOUT. 5 CIJFERS AUB"          : GOTO 320
360       IF VAL(N$) = 0 THEN
             PRINT "INVOERFOUT. ALLEEN GETALLEN AUB"    : GOTO 320
370 REM
380       LINE INPUT "NAAM VAN KLANT (MAX. 20 TEKENS): "; K$
390       IF LEN(K$) = 0 THEN
             PRINT "NAAM INVOEREN AUB"                  : GOTO 380
400       IF LEN(K$) > 20 THEN
             PRINT "MAXIMAAL 20 TEKENS. OPNIEUW AUB"    : GOTO 380
410 REM
420       LINE INPUT "KREDIETCODE (1, 2, 3, 4 OF 5)  : "; C$
430       IF LEN(C$) <> 1 THEN
             PRINT "MAXIMAAL 1 CIJFER. OPNIEUW AUB"     : GOTO 420
440       IF VAL(C$) = 0 THEN
             PRINT "EEN CIJFER AUB"                     : GOTO 420
450       LET C = VAL(C$)
460 REM
```

```
470 REM   SCHRIJF WEG
480 REM
490       PRINT #1, N$; ","; K$; ","; C
500 REM
510 REM   HERHALEN OF STOPPEN
520 REM
530       CLS : REM *** MAAK SCHERM SCHOON ***
540       LINE INPUT "NOG EEN KLANT (J/N)            : "; A$
550       IF LEFT$(A$) = "J" THEN 320
560 REM
570       CLOSE 1
580       PRINT "" : PRINT "EINDE PROGRAMMA, BESTAND GESLOTEN"
999       END
```

3(b)

```
100 REM   OPLOSSING VRAAG 3B: AFDRUKKEN KLANTENBESTAND
110 REM
120 REM   VARIABELENLIJST
130 REM    - B$   : NAAM VAN BESTAND
140 REM    - N$   : KLANTNUMMER
150 REM    - K$   : KLANTNAAM
160 REM    - C$/C : KREDIETCODE (1, 2, 3, 4 OF 5)
170 REM
180 REM   BESTAND : SEQUENTIEEL/FLOPPY
190 REM             NAAM DOOR GEBRUIKER IN TE VOEREN
200 REM
210 REM   VRAAG BESTANDNAAM EN OPEN
220 REM
230       CLS
240       LINE INPUT "NAAM VAN BESTAND               : "; B$
250       OPEN B$ FOR INPUT AS 1
260 REM
270 REM   LEES RECORD EN DRUK AF
280 REM
290       PRINT "KLANTNUMMER", "KLANTNAAM", "KREDIETCODE"
300       IF EOF(1) THEN 370
310          INPUT #1, N$, K$, C
320          PRINT     N$, K$, C
330       GOTO 300
340 REM
350 REM   AFSLUITEN EN STOPPEN
360 REM
370       CLOSE 1
380       PRINT "" : PRINT "EINDE OVERZICHT, BESTAND GESLOTEN"
999       END
```

4(a)

```
100 REM   OPLOSSING VRAAG 4A: TRANSACTIEBESTAND
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$   : NAAM VAN BESTAND
140 REM    - H$/H : HULPVARIABELE
150 REM    - R$   : REKENINGNUMMER (5 CIJFERS)
160 REM    - T$   : TRANSACTIECODE
170 REM    - F$   : BEDRAG
180 REM    - I    : BESTURINGSVARIABELE FOR/NEXT
190 REM
```

(wordt vervolgd)

```
200 REM   BESTAND : SEQUENTIEEL/FLOPPY, NAAM DOOR GEBRUIKER TE BEPALEN
210 REM
220 REM   INITIALISATIES GEHEUGEN/BESTAND
230 REM
240       CLEAR 500
250       LINE INPUT "NAAM VAN BESTAND          : "; N$
260       OPEN N$ FOR OUTPUT AS 1
270       CLS
280 REM
290 REM   GEGEVENSINVOER EN -CONTROLE
300 REM
310       PRINT "GEEF -1 OM MET INVOER TE STOPPEN"
320 REM
330       LINE INPUT "REKENINGNUMMER (5 CIJFERS)        : "; R$
340       IF R$ = "-1" THEN 660
350       IF VAL(R$) = 0 THEN
             PRINT "INVOER NODIG OF "                          : GOTO 310
360       IF LEN(R$) <> 5 THEN
             PRINT R$; ": NIET ACCEPTABEL. OPNIEUW AUB"        : GOTO 310
370 REM
380 REM
390       LINE INPUT "TRANSACTIECODE (2 CIJFERS): "; T$
400       IF VAL(T$) = 0 THEN
             PRINT "INVOER NODIG"                              : GOTO 390
410       IF LEN(T$) <> 2 THEN
             PRINT T$; ": NIET ACCEPTABEL. OPNIEUW AUB"        : GOTO 390
430 REM
440       PRINT      "BEDRAG (ZONDER VALUTATEKEN, MAXI-"
450       LINE INPUT "                      MAAL 999.99)  : "; F$
460       IF VAL(F$) = 0 THEN
             PRINT "INVOER NODIG"                              : GOTO 440
470       IF VAL(F$) > 999.99 THEN
             PRINT F$; ": NIET ACCEPTABEL. OPNIEUW AUB"        : GOTO 440
480       FOR I = 1 TO LEN(F$)
490           H = ASC(MID$(F$,I,1))
500           IF H >= 48  AND  H <= 57  OR  H = 46 THEN 530
510           PRINT "ONGELDIGE INVOER. ALLEEN CIJFERS EN ";
520           PRINT "DECIMALE PUNT TOEGESTAAN"                  : GOTO 440
530       NEXT I
540       IF LEN(F$) < 6 THEN LET F$ = " " + F$                 : GOTO 540
550 REM
560 REM   STEL RECORD SAMEN EN SCHRIJF WEG
570 REM
580       LET H$ = R$ + T$ + F$
590       PRINT #1, H$
600 REM
610 REM   HERHALEN/EINDIGEN
620 REM
630       CLS
640       GOTO 330
650 REM
660       CLOSE 1
670       PRINT "BESTAND GESLOTEN"
999       END
```

4(b)
```
100 REM   OPLOSSING VRAAG 4B: TRANSACTIEBESTAND UITPRINTEN
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$    : NAAM VAN BESTAND
140 REM    - H$    : HULPVARIABELE VOOR RECORD
150 REM
160 REM   BESTAND : SEQUENTIEEL/FLOPPY, NAAM DOOR GEBRUIKER TE BEPALEN
170 REM
180 REM   INITIALISATIE
190 REM
200       LINE INPUT "NAAM VAN BESTAND                    : "; N$
210       OPEN N$ FOR INPUT AS 1
220 REM
230 REM   LEES GEGEVENS UIT BESTAND EN DRUK AF
240 REM
250       PRINT "REKENINGNR", "CODE", "BEDRAG"
260       IF EOF(1) THEN 330
270          INPUT #1, H$
280          PRINT LEFT$(H$,5), MID$(H$,6,2), RIGHT$(H$,6)
290       GOTO 260
300 REM
310 REM   BESTAND AFSLUITEN EN STOPPEN
320 REM
330       CLOSE 1
340       PRINT "" : PRINT "EINDE OVERZICHT, BESTAND GESLOTEN"
999       END
```

5(a)
```
100 REM   OPLOSSING VRAAG 5A: ADRESBESTAND
110 REM
120 REM   VARIABELENLIJST
130 REM
140 REM    - N$(20)    : NAAM
150 REM    - A$(20)    : (STRAAT-)ADRES, INCL. HUISNUMMER
160 REM    - P$(7)     : JA NATUURLIJK ... DE POSTCODE
170 REM    - WS(20)    : WOONPLAATS
180 REM    - H1$/H1    : HULPVARIABELE T.B.V. TEKENCONTROLE EN DIALOOG
190 REM    - H2$(67)   : HULPVARIABELE T.B.V. SAMENSTELLEN RECORD
200 REM
210 REM   BESTAND      : NAPW, SEQUENTIEEL/FLOPPY
220 REM
230 REM   INITIALISATIES
240 REM
250       CLEAR 1000
260       OPEN "NAPW" FOR OUTPUT AS 1
270 REM
280 REM   GEGEVENSINVOER EN -CONTROLE
290 REM
300       LINE INPUT "NAAM (MAXIMAAL 20 TEKENS)          : "; N$
310       IF LEN(N$) < 20 THEN LET N$ = N$ + " "                : GOTO 310
320       IF LEN(N$) > 20 THEN
             PRINT "MAX. 20 TEKENS. OPNIEUW AUB"                : GOTO 300
```

(wordt vervolgd)

```
330 REM
340       LINE INPUT "STRAAT + HUISNR (MAX. 20 TEKENS)  : "; A$
350       IF LEN(A$) < 20 THEN LET A$ = A$ + " "                : GOTO 350
360       IF LEN(A$) > 20 THEN
             PRINT "MAX. 20 TEKENS. OPNIEUW AUB"                : GOTO 340
370 REM
380       LINE INPUT "POSTCODE (XXXX AA)                : "; P$
390       IF VAL(LEFT$(P$,4)) = 0 THEN
             PRINT "IN POS. 1-4 ALLEEN CIJFERS. OPNIEUW AUB"    : GOTO 380
400       H1 = ASC(MID$(P$,6,1))                : REM *** POS. 6 = LETTER? ***
410       IF H1 < 65  OR H1 > 90 THEN
             PRINT "LETTER IN POS. 6 NODIG. OPNIEUW AUB"        : GOTO 380
420       H1 = ASC(MID$(P$,7,1))                : REM *** POS. 7 = LETTER? ***
430       IF H1 < 65  OR H1 > 90 THEN
             PRINT "LETTER IN POS. 7 NODIG. OPNIEUW AUB"        : GOTO 380
440       H1 = ASC(MID$(P$,5,1))                : REM *** POS. 5 = SPATIE? ***
450       IF H1 <> 32 THEN
             PRINT " SPATIE IN POS. 5 NODIG. OPNIEUW AUB"       : GOTO 380
460 REM
470       LINE INPUT "WOONPLAATS (MAX. 20 TEKENS)       : "; W$
480       IF LEN(W$) < 20 THEN W$ = W$ + " "                    : GOTO 480
490       IF LEN(W$) > 20 THEN
             PRINT "MAXIMAAL 20 TEKENS. OPNIEUW AUB"            : GOTO 470
500 REM
510 REM   RECORD SAMENSTELLEN EN WEGSCHRIJVEN
520 REM
530       H2$ = N$ + A$ + P$ + W$
540       PRINT #1, H2$
550 REM
560 REM   HERHALEN OF STOPPEN
570 REM
580       LINE INPUT "HERHALEN? (J/N)                   : "; H1$
590       IF LEFT$(H1$,1) = "J" THEN 300
600 REM
610       CLOSE 1
620       PRINT "BESTAND GESLOTEN"
999       END
```

5(b)

```
100 REM   OPLOSSING VRAAG 5B: ADRESBESTAND AFDRUKKEN
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$(20)  : NAAM
140 REM    - A$(20)  : STRAAT EN HUISNUMMER
150 REM    - P$(7)   : POSTCODE
160 REM    - W$(20)  : WOONPLAATS
170 REM    - R$(67)  : RECORD
180 REM    - D$      : DIALOOG
190 REM
200 REM   BESTAND    : NAPW, SEQUENTIEEL/FLOPPY
210 REM
220       OPEN "NAPW" FOR INPUT AS 1
230 REM
```

(wordt vervolgd)

```
240 REM   LEES RECORD EN DRUK VELDEN AF
250 REM
260       IF EOF(1) THEN 370
270          INPUT #1, R$
280          PRINT LEFT$(R$,20)
290          PRINT MID$ (R$,21,20)
300          PRINT MID$ (R$,41,7); "  "; MID$(R$, 48, LEN(R$-47))
310          PRINT ""
320          LINE INPUT "GEEF ENTER OF RETURN ALS U VERDER WILT GAAN"; D$
330       GOTO 260
340 REM
350 REM   AFSLUITING
360 REM
370       CLOSE 1
380       PRINT "EINDE OVERZICHT, BESTAND AFGESLOTEN"
999       END
```

6(a)

```
100 REM   OPLOSSING VRAAG 6A: STANDAARD BRIEF MAKEN
110 REM
120 REM   VARIABELENLIJST
130 REM    - R$    : TEKSTREGEL
140 REM    - B$    : NAAM VAN BESTAND
150 REM
160 REM   BESTAND : BRIEFX, WAARBIJ X = 1, 2, 3 ENZ.
170 REM             SEQUENTIEEL/FLOPPY
180 REM
190 REM   INITIALISATIES
200 REM
210       CLEAR 1000
220       LINE INPUT "VOLGNUMMER BESTAND        : "; B$
230       LET B$ = "BRIEF" + B$
240       OPEN B$ FOR OUTPUT AS 1
250 REM
260 REM   TEKST INLEZEN EN WEGSCHRIJVEN
270 REM
280       PRINT "GEEF TEKSTREGEL OF 'STOP' (ZONDER AANH. TEKENS)"
290       LINE INPUT ">"; R$
300       IF R$ = "STOP" THEN 360
310       PRINT #1, R$
320       GOTO 290
330 REM
340 REM   AFSLUITEN EN STOPPEN
350 REM
360       CLOSE 1
370       PRINT "" : PRINT "BESTAND "; B$; "GESLOTEN"
999       END
```

6(b)

```
100 REM   OPLOSSING VRAAG 6B: STANDAARD BRIEF AFDRUKKEN
110 REM
120 REM   VARIABELENLIJST
130 REM    - R$    : TEKSTREGEL
140 REM    - B$    : NAAM VAN BESTAND
150 REM
160 REM   BESTAND : BRIEFX, WAARBIJ X = 1, 2, 3 ENZ.
170 REM             SEQUENTIEEL/FLOPPY
180 REM
190 REM   INITIALISATIES
200 REM
210       LINE INPUT "VOLGNUMMER BESTAND          : "; B$
220       LET B$ = "BRIEF" + B$
230       OPEN B$ FOR INPUT AS 1
240 REM
250 REM   LEES TEKST EN DRUK AF
260 REM
270       IF EOF(1) THEN 350
280          INPUT #1, R$
290          PRINT CHR$(34), R$, CHR$(34) : REM *** 34 = " ***
300       GOTO 270
310 REM
320 REM   AFSLUITEN EN STOPPEN
330 REM
340       CLOSE 1
999       END
```

5 UTILITY-PROGRAMMA'S VOOR SEQUENTIËLE BESTANDEN

5.0 DOELSTELLINGEN

Na het doorwerken van dit hoofdstuk moet u in staat zijn programma's te schrijven waarmee:

- een sequentieel bestand kan worden gekopieerd;
- gegevens aan een reeds aanwezig bestand kunnen worden toegevoegd (uitbreiden);
- gegevens in een reeds aanwezig bestand kunnen worden veranderd (mutatie);
- incidentele gegevens uit een sequentieel bestand kunnen worden opgevraagd, en gegevens kunnen worden veranderd, toegevoegd of verwijderd;
- de inhoud van twee sequentiële bestanden tot één sequentieel bestand kan worden samengevoegd, onder handhaving van de eventuele numerieke of alfabetische volgorde van gegevens;
- gegevens uit meer dan één sequentieel bestand kunnen worden gebruikt of gecombineerd.

Programma's waarmee dergelijke algemene handelingen kunnen worden uitgevoerd heten utility-programma's (dat wil zeggen 'dienst'- of 'hulp'-programma's, meervoud: utilities). Naast deze utilities krijgt u in dit hoofdstuk ook te maken met enkele algemene bestandstechnieken. Van daaruit zult u in staat zijn eigen toepassingen te maken.

We gaan in dit hoofdstuk overigens nog steeds uit van het schijvengeheugen als opslagmedium. Nogmaals: voor cassette-apparatuur is de materie niet wezenlijk verschillend. De sequentiële bestandsorganisatie komt immers bij beide geheugenvormen voor. Mocht u niet over schijfapparatuur (floppy drives) beschikken, dan is dit hoofdstuk dus toch relevant.

Zoals eerder bij de toetsvragen aan het einde van het vorige hoofdstuk opgemerkt, steunt dit hoofdstuk in belangrijke mate op de programma's en bestanden die daar ter sprake kwamen.

5.1 KOPIËREN VAN EEN SEQUENTIEEL BESTAND

In deze paragraaf bespreken we een utility programma waarmee sequentiële bestanden kunnen worden gekopieerd. MSX BASIC beschikt over een COPY-opdracht voor het kopiëren van bestanden. Kijk in uw handleiding. U kunt dan op commandoniveau bestanden van de ene schijfeenheid naar de andere kopiëren.

Behalve een kopieerprogramma op commandoniveau is ook een BASIC-versie interessant. Zo'n programma kan namelijk eenvoudig in andere programma's worden ingebouwd zodat ook op programma-niveau ('dynamisch') kan worden gekopieerd. Ook kunnen bestaande gegevensbestanden hiermee worden uitgebreid. Bovendien illustreert het goed hoe de diverse opdrachten als OPEN, INPUT # en PRINT # en de EOF-functie gebruikt moeten worden.

De algemene aanpak voor het schrijven van zo'n kopieerprogramma is als volgt.
1. Open het te kopiëren bestand voor invoer.
2. Open een tweede bestand voor uitvoer.
3. Test het invoerbestand op EOF-conditie; indien einde bestand, ga dan verder bij stap 7.
4. Lees het eerste c.q. eerstvolgende record van het invoerbestand.
5. Schrijf dit record naar het uitvoerbestand.
6. Herhaal stappen 3 t/m 5 totdat bij stap 3 de EOF-conditie optreedt.
7. Sluit beide bestanden.

Opdracht

(a) Neem aan dat u een bestand wilt kopiëren waarvan het aantal records onbekend is. Elk record bevat twee strings van 25 tekens, gevolgd door twee numerieke waarden. Een programma is hieronder grotendeels gegeven. Geef de ontbrekende regels (nummers 240, 250, 320, 330, 340 en 420), daarbij uitgaande van de gegeven REMARKs.

```
100 REM   KOPIEERPROGRAMMA
110 REM
120 REM   VARIABELENLIJST
130 :
140 :
150 :
160 :
170 REM   BESTANDEN
180 :
190 :
200 REM   INITIALISEREN VAN BESTAND
205       MAXFILES = 2
210 REM
220       LINE INPUT "NAAM INVOERBESTAND  : "; F1$
230       LINE INPUT "NAAM UITVOERBESTAND : "; F2$
```

```
240       ..............................  : REM *** OPEN INVOERBESTAND   ***
250       ..............................  : REM *** OPEN UITVOERBESTAND  ***
260 REM
300 REM   LEES RECORD EN SCHRIJF WEG
310 REM
320       ..............................  : REM *** TEST OP EOF          ***
330       ..............................  : REM *** LEES RECORD VAN IN-
                                              VOERBESTAND              ***
340       ..............................  : REM *** SCHRIJF RECORD NAAR
                                              UITVOERBESTAND           ***
350       GOTO 320
360 REM
400 REM   AFSLUITEN EN STOPPEN
410 REM
420       ..............................  : REM *** SLUIT BESTANDEN      ***
999 END
```

(a)
```
240       OPEN F1$ FOR INPUT AS 1
250       OPEN F2$ FOR OUTPUT AS 2
320       IF EOF(1) THEN 420
330       INPUT #1, A$, B$, A, B
340       PRINT #2, A$; ","; B$; ","; A; B
420       CLOSE 1, 2
```

Opmerking: het forceren van komma's (regel 340) is essentieel voor een exacte kopie.

Toetsvraag

(a) Welke regel(s) in bovenstaand programma komen overeen met de stappen van de eerder genoemde algemene aanpak:
1. Openen van het te kopiëren bestand.
2. Openen van het uitvoerbestand.
3. Testen van het invoerbestand op EOF; indien einde bestand, verder gaan bij stap 7.
4. Lezen van het eerste c.q. eerstvolgende record.
5. Schrijven van dit record naar het uitvoerbestand.
6. Herhalen van stappen 3 t/m 5 tot EOF.
7. Sluiten van beide bestanden.

(b) Wat verschijnt bij het RUNnen van het programma op het scherm?

(a) 1. 240
2. 250
3. 320
4. 330
5. 340
6. 350
7. 420

(b)
```
RUN

NAAM INVOERBESTAND  :
NAAM UITVOERBESTAND :

Ok
```

Zoals uit de laatste toetsvraag blijkt, verschijnt bij het RUNnen van het programma weinig uitvoer op het scherm. Tijdens de RUN kan de computer echter een behoorlijke activiteit ontplooien, waarbij het enige tijd kan duren voor de verlossende Ok boodschap verschijnt. En ook dan weten we niet met absolute zekerheid of alles naar wens is gegaan. Om aan dit bezwaar enigszins tegemoet te komen kunnen we het programma op de volgende punten modificeren.

Ten eerste kunnen we na het afsluiten van de bestanden een boodschap, die aangeeft dat het kopiëren voltooid is, laten afdrukken. Bijvoorbeeld:

```
420       PRINT "KOPIEREN KLAAR"
```

Een tweede modificatie is het uitvoerbestand onmiddellijk na het kopiëren voor invoer te openen, en enkele regels ter verificatie in te lezen en op het scherm af te drukken. Bijvoorbeeld:

```
430       OPEN F2$ FOR INPUT AS 1
440       PRINT : PRINT "GEKOPIEERDE FILE BEGINT MET:"
450       FOR X = 1 TO 3
460           INPUT #1, A$, B$, A, B
470           PRINT     A$, B$, A, B
480       NEXT X
490       CLOSE 1
```

U hebt nu een volledige kopieer-utility tot uw beschikking. Door eenvoudig de INPUT #- en PRINT #-opdrachten in regels 330 en 340 aan te passen aan de gewenste record-vorm, kunt u het programma gebruiken om een willekeurig sequentieel bestand te dupliceren.

5.2 UITBREIDEN VAN EEN SEQUENTIEEL BESTAND

We kunnen met APPEND heel eenvoudig gegevens aan een bestaand sequentieel bestand toevoegen. Als we het programma op p. 106/107 als voorbeeld nemen en daarin alleen veranderen

```
100 REM   PROGRAMMA VOOR HET UITBREIDEN VAN EEN INVENTARISBESTAND
```

èn

```
210       OPEN "INVENT" FOR APPEND AS 1
```

Dan hebben we een programma voor het toevoegen van gegevens aan het INVENT-bestand. MSX BASIC zorgt verder voor alles!

Toch willen we aan de hand van een programma zonder APPEND laten zien hoe dit uitbreiden in zijn werk gaat. We gaan als het

ware iets doen, wat met APPEND het BASIC systeem ook moet doen.

De methode om een bestand zonder gebruik te maken van APPEND uit te breiden is:
1. Kopieer de reeds aanwezige gegevens naar een tijdelijk bestand.
2. Voeg de nieuwe gegevens toe aan het einde van het tijdelijke bestand.
3. Kopieer het tijdelijke naar het oorspronkelijke bestand.

Laten we een eenvoudige toepassing bekijken. Neem aan dat u de computer gebruikt om een boodschappenlijst te maken voor een wekelijks bezoek aan de supermarkt (zie toetsvraag 2 aan het einde van hoofdstuk 2). Bij tijd en wijle merkt u dat de kritische voorraad van een of ander levensmiddel nadert, en u klimt dan ijverig in de toetsen om (al weer) een artikel aan uw geautomatiseerde boodschappenlijstje toe te voegen. Iedere toevoeging vormt een record, bestaande uit een string van maximaal 20 tekens voor omschrijving van het artikel en één numerieke waarde voor de hoeveelheid of het aantal.

Typisch een toepassing dus, waarbij een reeds aanwezig bestand met nieuwe gegevens wordt uitgebreid. De algemene procedure hiervoor is:
1. Open het invoer- (= oorspronkelijke) bestand.
2. Open het uitvoer- (= tijdelijke) bestand.
3. Test het invoerbestand op EOF. Indien einde bestand, ga dan naar stap 7.
4. Lees het eerste c.q. eerstvolgende record.
5. Schrijf dit record naar het tijdelijke bestand.
6. Herhaal de stappen 3 t/m 5 tot EOF voor het invoerbestand.
7. Voer de nieuwe gegevens in en controleer ze. Denk daarbij aan de mogelijkheid voor de gebruiker om te kennen te geven dat hij klaar is met invoeren.
8. Schrijf het nieuwe record naar het tijdelijke bestand.
9. Herhaal de stappen 7 en 8 tot alle nieuwe gegevens zijn ingevoerd.
10. Sluit beide bestanden.
11. Open het tijdelijke bestand voor invoer.
12. Open het oorspronkelijke bestand voor uitvoer (waarbij de inhoud verloren gaat).
13. Test op EOF tijdelijk (= invoer-) bestand. Indien geen EOF, lees eerste c.q. eerstvolgende record en schrijf dit naar het oorspronkelijke (= uitvoer-) bestand.
14. Sluit beide bestanden.

De ontwikkeling van het programma

Bovenstaande procedure vertalen we nu in een concreet programma. Allereerst het inleidende deel:

```
1000 REM   BOODSCHAPPENLIJST
1010 REM
1020 REM   ************************************************************************
1030 REM   INLEIDEND MODULE
1040 REM   ************************************************************************
1050 REM
1060 REM   VARIABELENLIJST
1070 REM    - A$    : OMSCHRIJVING ARTIKEL
1080 REM    - H     : HOEVEELHEID
1090 REM    - N$    : NAAM VAN OORSPRONKELIJK BESTAND
1100 REM    - D$    : VARIABELE T.B.V. DIALOOG
1110 REM
1020 REM   BESTANDEN
1130 REM
1140 REM   1. NAAM IN TE VOEREN DOOR GEBRUIKER (N$)
1150 REM   2. TIJDELIJK BESTAND "HULP"
1160 REM
1170 REM   BEIDE BESTANDEN OP FLOPPY, SEQUENTIEEL GEORGANISEERD
1180 REM
```

Opdracht

Maak het volgende initialisatiesegment in overeenstemming met de gegeven REMARKs af.

```
1190 REM   BESTANDSINITIALISATIES
1200 REM
1205       MAXFILES = 2
1210       LINE INPUT "NAAM INVOERBESTAND"  : "; N$

1220       ............................... : REM *** OPEN INVOERBESTAND ***

1230       ............................... : REM *** OPEN UITVOERBESTAND ***

1240 REM
```

(a)
```
1220       OPEN N$ FOR INPUT AS 1
1230       OPEN "HULP" FOR OUTPUT AS 2
```

N.B. Een logische keuze van buffernummers en volgorde van openen dragen bij tot betere leesbaarheid van het programma.

Het volgende programmasegment kopieert het oorspronkelijke bestand naar het hulpbestand. Omdat de kopieerbewerking verschillende malen nodig is, kunnen we dit deel het beste in routine-vorm schrijven en het aan het einde van het programma opnemen.

```
2000 REM   ************************************************************************
2010 REM   KOPIEERROUTINE BESTAND 1 --> BESTAND 2
2020 REM   ************************************************************************
2030 REM
2040       IF EOF(1) THEN RETURN
2050          INPUT #1, A$, H
2060          PRINT #2, A$, ","; H
2070       GOTO 2040
2080 REM
```

De eerste aanroep kan aansluitend op de bestandsinitialisatie in de regels 1190 t/m 1240 plaatsvinden, bijvoorbeeld:

```
1235 REM
1236        GOSUB 2040    : REM *** OORSPRONKELIJK DEEL BESTAND KOPIEREN ***
1237 REM
```

Toetsvragen

(a) Werkt de routine als het invoerbestand leeg is?

(b) Op welke plaats staat de recordwijzer (pointer) van het uitvoerbestand na afloop van RETURN, aannemende dat het kopiëren gelukt is en de routine direct na de bestandsinitialisaties wordt aangeroepen (regel 1250)?

(a) Ja, mits het bestand op de schijf bekend is. Het bestand kan leeg of gevuld zijn.

(b) Op de eerste plaats na het laatste gegeven in HULP, dat wil zeggen hij geeft de plaats aan waar het eerstvolgende gegeven moet worden opgeslagen.

We kunnen nu vervolgens het gegevensinvoerdeel samenstellen.

Opdracht

(a) Maak het volgende programmasegment in overeenstemming met de gegeven REMARKs af.

```
1250 REM   ************************************************************************
1260 REM   GEGEVENSINVOER EN -CONTROLE
1270 REM   ************************************************************************
1280 REM
1290       PRINT "GEEF 'STOP' OM TE EINDIGEN"
1300       PRINT ""
1310 REM
1320       LINE INPUT "ARTIKEL (MAXIMAAL 20 TEKENS): "; A$

1330       ..................................... : REM *** TEST OP STOP          ***

1340       IF LEN(A$) = 0 THEN PRINT "GEEF ARTIKEL OMSCHRIJVING OF STOP AUB" :
                              GOTO 1320

1350       ..................................... : REM *** TEST OP MAX. LENGTE   ***

1360 REM
1370       INPUT     "HOEVEELHEID                    : "; H
1380       IF  H >= 1   AND H < 10  THEN 1450
1390           PRINT "HOEVEELHEID WAS "; H; "AKKOORD? (J/N): "; D$
1400           LINE INPUT D$
1410           IF LEFT$(D$,1) = "N" THEN 1370
1420 REM
1430 REM   SCHRIJF NAAR HULPBESTAND EN HERHAAL
1440 REM
```

```
1450        ..................................  : REM *** NIEUW GG. ->          ***

1460        GOTO 1320
1470 REM
1480 REM    AFSLUITEN
1490 REM

1500        ..................................  : REM *** SLUIT BESTANDEN       ***

1510 REM
```

```
(a) 1330        IF A$ = "STOP" THEN 1500
    1350        IF LEN(A$) > 20 THEN
                   PRINT "MAX. 20 TEKENS. OPNIEUW AUB"       : GOTO 1320
    1450        PRINT #2, A$; ","; H
    1500        CLOSE 1, 2
```

Het grootste deel van de klus is nu geklaard. De oude en nieuwe gegevens zijn namelijk samen ondergebracht in het tijdelijke hulpbestand, dat nu kan worden teruggekopieerd naar het oorspronkelijke bestand. Hiervoor kunnen we de eerder geschreven kopieerroutine gebruiken.

Opdracht

(a) Maak het volgende programmasegment in overeenstemming met de gegeven REMARKs af.

```
1520 REM    BESTANDEN HER-OPENEN
1530 REM

1540        ..................................  : REM *** OPEN INVOERBESTAND    ***

1550        ..................................  : REM *** OPEN UITVOERBESTAND   ***

1560 REM
1570 REM    KOPIEER TIJDELIJK --> OORSPRONKELIJK BESTAND EN SLUIT AF
1580 REM

1590        ..................................  : REM *** ROEP KOPIEERMODULE    ***

1600 REM
1610        CLOSE 1, 2

1620        ..................................  : REM *** BEVESTIG AFSLUITEN    ***

1630 REM
1640        END
1650 REM
```

```
(a) 1540        OPEN "HULP" FOR INPUT AS 1
    1550        OPEN N$ FOR OUTPUT AS 2
    1590        GOSUB 2040
    1620        PRINT "KOPIEREN KLAAR"
```

Toetsvraag

(a) Wat gebeurt er met de inhoud van het door N$ aangegeven bestand bij uitvoering van regel 1550?

(a) De inhoud gaat bij het openen in OUTPUT-mode verloren (maar wordt later vervangen door de inhoud van HULP).

Overzicht volledig programma

Voor het overzicht van het geheel drukken we het volledige programma af. Eventueel kan nog een KILL-opdracht worden toegevoegd om het tijdelijke hulpbestand – dat onnodig schijfruimte in beslag neemt – te verwijderen. Bijvoorbeeld:

```
1625        KILL "HULP"
```

Verder is het verstandig eerst een backup exemplaar (dat wil zeggen een kopie) van het oorspronkelijke bestand te maken alvorens het programma te draaien. Hiervoor kan gebruik worden gemaakt van het in § 5.1 ontwikkelde utility-programma.

```
1000 REM   BOODSCHAPPENLIJST
1010 REM
1020 REM   ************************************************************************
1030 REM   INLEIDEND MODULE
1040 REM   ************************************************************************
1050 REM
1060 REM   VARIABELENLIJST
1070 REM    - A$    : OMSCHRIJVING ARTIKEL
1080 REM    - H     : HOEVEELHEID
1090 REM    - N$    : NAAM VAN OORSPRONKELIJK BESTAND
1100 REM    - D$    ; VARIABELE T.B.V. DIALOOG
1110 REM
1120 REM   BESTANDEN
1130 REM
1140 REM   1. NAAM IN TE VOEREN DOOR GEBRUIKER (N$)
1150 REM   2. TIJDELIJK BESTAND "HULP"
1160 REM
1170 REM   BEIDE BESTANDEN OP FLOPPY, SEQUENTIEEL GEORGANISEERD
1180 REM
1190 REM   BESTANDSINITIALISATIES
1200 REM
1205        MAXFILES = 2
1210        LINE INPUT "NAAM INVOERBESTAND       : "; N$
1220        OPEN N$ FOR INPUT AS 1
1230        OPEN "HULP" FOR OUTPUT AS 2
1235 REM
1236        GOSUB 2040
1237 REM
1240 REM
1250 REM   ************************************************************************
1260 REM   GEGEVENSINVOER EN -CONTROLE / VERWERKING
1270 REM   ************************************************************************
1280 REM
1290        PRINT "GEEF 'STOP' OM TE EINDIGEN"
1300        PRINT
1310 REM
```

```
1320        LINE INPUT "ARTIKEL (MAXIMAAL 20 TEKENS): "; A$
1330        IF A$ = "STOP" THEN 1500
1340        IF LEN(A$) = 0 THEN PRINT "GEEF ARTIKEL OMSCHRIJVING OF STOP AUB" :
                                GOTO 1320
1350        IF LEN(A$) > 20 THEN
               PRINT "MAX. 20 TEKENS. OPNIEUW AUB"       : GOTO 1320
1355                                          REM *** TEST OP MAX. LENGTE ***
1360 REM
1370        INPUT      "HOEVEELHEID                    : "; H
1380        IF  H >= 1  AND H < 10  THEN 1450
1390            PRINT "HOEVEELHEID WAS "; H; "AKKOORD? (J/N): ";
1400            LINE INPUT D$
1410            IF LEFT$(D$,1) = "N" THEN 1370
1420 REM
1430 REM    SCHRIJF NAAR HULPBESTAND EN HERHAAL
1440 REM
1450        PRINT #2, A$, ","; H                : REM *** NIEUW GG. -> HULP    ***
1460        GOTO 1320
1470 REM
1480 REM    AFSLUITEN
1490 REM

1500        CLOSE 1, 2                          : REM *** SLUIT BESTANDEN      ***
1510 REM
1520 REM    BESTANDEN HEROPENEN
1530 REM
1540        OPEN "HULP" FOR INPUT AS 1 : REM *** OPEN INVOERBEST.     ***
1550        OPEN N$ FOR OUTPUT AS 2    : REM *** OPEN UITVOERBEST.    ***
1560 REM
1570 REM    KOPIEER TIJDELIJK --> OORSPRONKELIJK BESTAND EN SLUIT AF
1580 REM
1590        GOSUB 2040
1600 REM
1610        CLOSE 1, 2
1620        PRINT "KOPIEREN KLAAR"              : REM *** BEVESTIG AFSLUITEN  ***
1630 REM
1640        END
1650 REM
2000 REM    ************************************************************************
2010 REM    KOPIEERROUTINE BESTAND 1 --> BESTAND 2
2020 REM    ************************************************************************
2030 REM
2040        IF EOF(1) THEN RETURN
2050           INPUT #1, A$, H
2060           PRINT #2, A$, ","; H
2070        GOTO 2040
2080 REM
9999        END
```

Toetsvraag

(a) Bepaal welke regel(s) in bovenstaand programma overeenkomen met de stappen van de eerder genoemde procedure.

1. Open invoerbestand.
2. Open tijdelijk uitvoerbestand.
3. Test invoerbestand op EOF; indien einde bestand, ga naar stap 7.
4. Lees eerste c.q. eerstvolgende record.
5. Schrijf record naar tijdelijk bestand.

6. Herhaal stappen 3 t/m 5 tot EOF voor invoerbestand.
7. Voer in en controleer nieuwe gegevens.
8. Schrijf nieuw record naar tijdelijk bestand.
9. Herhaal stappen 7 en 8 tot alle gegevens zijn ingevoerd.
10. Sluit beide bestanden.
11. Open tijdelijk bestand voor invoer.
12. Open oorspronkelijk bestand voor uitvoer.
13. Indien geen EOF voor invoerbestand, lees record en schrijf naar uitvoerbestand.
14. Sluit beide bestanden.

(a)
1. 1220
2. 1230
3. 1236 / 2040 t/m 2070
4. 2050
5. 2060
6. 2070
7. 1290 t/m 1410
8. 1450
9. 1460
10. 1500
11. 1540
12. 1550
13. 1590 / 2040 t/m 2070
14. 1610

Alternatieve procedure

Ook bij het uitbreiden van bestanden zijn er verschillende wegen die naar Rome leiden. De volgende procedure is gebaseerd op het gebruik van arrays. De toepasbaarheid hangt af van de beschikbare hoeveelheid geheugen.

1. Open het uit te breiden bestand voor invoer.
2. Kopieer de gehele inhoud van het bestand naar één of meer arrays.
3. Sluit het invoerbestand.
4. Heropen het bestand, ditmaal voor uitvoer.
5. Schrijf de inhoud van de array(s) terug naar het bestand.
6. Voeg de nieuwe gegevens toe aan het einde van het bestand.
7. Sluit het bestand.

De procedure is alleen geschikt voor kleine bestanden, waarbij bovendien eventuele deelvelden van een record samen als één string te manipuleren zijn. Doorgaans zal de eerste procedure aan te bevelen zijn.

5.3 MUTEREN VAN EEN SEQUENTIEEL BESTAND

We hebben gezien dat een bestand, dat voor invoer is geopend, uitsluitend kan worden gelezen. Omgekeerd kan een bestand, dat voor uitvoer is geopend, uitsluitend worden beschreven. Eventueel

eerder aanwezige gegevens gaan daarbij verloren. Eenmaal in een bestand opgenomen kan een gegeven dus niet zonder meer worden gewijzigd. Ook hier is dus een omweg nodig. Die komt erop neer dat de oorspronkelijke gegevens record voor record worden ingelezen. Aan de hand van een in te voeren kenmerk – de zogenaamde zoeksleutel – wordt voor ieder record bepaald of hij het te veranderen gegeven bevat. Zo ja, dan wordt het record aangepast en naar een tijdelijk uitvoerbestand geschreven. Zo neen, dan wordt het record rechtstreeks naar het uitvoerbestand geschreven. Nadat alle records op deze wijze zijn verwerkt, wordt het tijdelijke bestand – dat nu alle gegevens in de gewenste vorm bevat – teruggekopieerd naar het oorspronkelijke bestand.

We lichten dit toe aan de hand van een concreet geval: een programma voor het veranderen van de kredietcode in een klantenbestand van een klein bedrijf (zie toetsvraag 3, einde hoofdstuk 4). Elk record bevat drie deelvelden:
1. Een 5-cijferig klantnummer (precies 5 cijfers).
2. Naam van de klant (maximaal 20 tekens).
3. Kredietcode van de klant (cijfer 1, 2, 3, 4 of 5).

Doel van het programma is de kredietcode van een of meer klanten te veranderen op basis van een door de gebruiker in te voeren klantnummer.

Een voorbeeld-RUN (waarbij de cursieve tekst de gebruikersrespons voorstelt) is:

```
RUN

NAAM INVOERBESTAND      : KCODE1
GEEF 'STOP' (ZONDER AANH.TEKENS) ALS U KLAAR BENT
KLANTNUMMER             : 13762
KLANTNR. 13762 KOMT NIET VOOR IN BESTAND. CONTROLEER EN VOER REGEL OPNIEUW IN

KLANTNUMMER             : 13763

GARAGE V.D.PANNE, KREDIETCODE: 3
NIEUWE KREDIETCODE      : 2
NOG EEN KLANT? (J/N)    : J

KLANTNUMMER             : 11127

BOUWBEDRIJF ROTHUIZEN, KREDIETCODE: 1
NIEUWE KREDIETCODE        2
NOG EEN KLANT? (J/N)    : N

EINDE PROGRAMMA

Ok
```

De volgende procedure geeft aan hoe we een en ander kunnen bereiken.

1. Open het invoerbestand.
2. Open een tijdelijk uitvoerbestand.
3. Voer het kenmerk in waarmee het aan te passen record kan worden vastgesteld (in ons geval het nummer van de klant wiens kredietcode moet worden veranderd). Denk daarbij aan invoercontrole en aan de mogelijkheid om te stoppen.
4. Test op EOF voor het invoerbestand. Indien einde bestand:
   a. geef melding dat het record niet gevonden is;
   b. sluit beide bestanden (om de pointers terug te zetten naar het begin van het bestand);
   c. herhaal de stappen 1 t/m 4 totdat de gebruiker te kennen geeft dat hij wil stoppen.
5. Lees een volledig record van het invoerbestand.
6. Test of het in stap 3 ingevoerde kenmerk gelijk is aan de inhoud van het overeenkomstige record.
   Zo ja:
   a. druk het gevonden record af, en vraag de gebruiker het nieuwe gegeven (in ons geval de kredietcode) in te voeren;
   b. controleer het nieuwe gegeven op geldigheid;
   c. stel het nieuwe record samen en schrijf dit naar het tijdelijke uitvoerbestand;
   d. schrijf de resterende records rechtstreeks van invoer- naar uitvoerbestand.
   Zo neen:
   e. schrijf record naar uitvoerbestand, en herhaal vanaf stap 4.
7. Open beide bestanden opnieuw, maar ditmaal het oorspronkelijke bestand voor uitvoer, en het tijdelijke voor invoer (de inhoud van het oorspronkelijke bestand gaat daarbij verloren).
8. Kopieer de inhoud van het tijdelijke bestand – dat nu de nieuwe versie is – naar het oorspronkelijke bestand.
9. Sluit bij EOF van het tijdelijke bestand beide bestanden.
10. Stel de gebruiker in de gelegenheid de gehele procedure te herhalen.

Net als in de vorige paragraaf, ontwikkelen we het programma stap voor stap. Allereerst de inleidende module.

Opdracht

Maak het volgende programmasegment in overeenstemming met de gegeven REMARKs af.

(a)

```
1000 REM   WIJZIGEN KREDIETCODE
1010 REM
1020 REM   ************************************************************************
1030 REM   INLEIDEND MODULE
1040 REM   ************************************************************************
1050 REM
1060 REM   VARIABELENLIJST
1070 REM    - B$      : NAAM VAN BESTAND
1080 REM    - N1$     : KLANTNUMMER IN BESTAND
1090 REM    - N2$     : OPGEGEVEN KLANTNUMMER
1100 REM    - K$      : KLANTNAAM
1110 REM    - C1      : KREDIETCODE IN BESTAND
1120 REM    - C2$/C2  : OPGEGEVEN KREDIETCODE
1130 REM    - A$      : DIALOOG
1140 REM
1150 REM   BESTANDEN
1160 REM
1170 REM   1. OORSPRONKELIJK BESTAND OP FLOPPY, NAAM IN TE VOEREN DOOR
1180 REM      GEBRUIKER (B$)
1190 REM   2. TIJDELIJK HULPBESTAND "HULP"
1200 REM
1210 REM   BEIDE BESTANDEN SEQUENTIEEL GEORGANISEERD
1220 REM
1230 REM   INITIALISATIES
1240 REM
1250       CLS: MAXFILES = 2
1260       LINE INPUT "NAAM INVOERBESTAND       : "; B$
1270       OPEN B$ FOR INPUT AS 1
1280       OPEN "HULP" FOR OUTPUT AS 2
1290 REM
1300 REM   ************************************************************************
1310 REM   GEGEVENSINVOER EN -CONTROLE
1320 REM   ************************************************************************
1330 REM
1340       PRINT "GEEF 'STOP' (ZONDER AANH. TEKENS) ALS U KLAAR BENT"
1350       PRINT
1360       LINE INPUT "KLANTNUMMER              : "; N2$
1370       IF N2$ = "STOP" THEN 2100

1380       ..............................  : REM *** MELDING INDIEN NULSTRING ***

1390       ..............................  : REM *** TEST OP 5 CIJFERS        ***

1400       ..............................  : REM *** TEST OP NUMERIEKE WAARDE ***

1410 REM
```

```
(a) 1380       IF LEN(N2$) = 0 THEN
                  PRINT "GEEF KLANTNUMMER OF 'STOP'"       : GOTO 1360
    1390       IF LEN(N2$) <> 5 THEN
                  PRINT "INVOERFOUT. 5 CIJFERS AUB"        : GOTO 1360
    1400       IF VAL(N2$) = 0 THEN
                  PRINT "INVOERFOUT. ALLEEN GETALLEN AUB"  : GOTO 1360
```

Nu wordt het interessanter. Het programma moet nu in het invoerbestand het record van de gewenste klant opsporen. Aan de hand van de volgende vragen kunt u zelf bedenken hoe dat in zijn werk gaat.

(a) Wat zou het programma moeten doen als tijdens het doorzoeken van het gegevensbestand de EOF-conditie optreedt zonder dat het gewenste record gevonden is?

(b) Wat moet er met de bestanden gebeuren voordat het programma het record van een volgende klant kan gaan zoeken?

(c) Maak het volgende programmasegment in overeenstemming met de gegeven REMARKs af (regels 2050 en 2070).

```
2000 REM   ************************************************************
2010 REM   ZOEKROUTINE
2020 REM   ************************************************************
2030 REM
2040       IF EOF(1) THEN 2090
2050       ........................... : REM *** LEES VAN BESTAND NAAR
                                             N1$, K$ en C1          ***
2060          IF N2$ = N1$ THEN 2150

2070       ........................... : REM *** KLANTNUMMER KLOPT NIET,
                                             SCHRIJF NAAR HULP      ***
2080            GOTO 2040
2090       PRINT "***FOUT***KLANTNR. "; N2$; "KOMT NIET VOOR IN BESTAND.
                 CONTROLEER EN VOER OPNIEUW IN AUB"
2100       CLOSE 1, 2                  : REM *** ZET WIJZERS TERUG NAAR
                                                  BEGIN  ***
2110       GOTO 1270
2120       REM
```

(d) Stel dat u regel 2100 verwijdert en het programma vervolgens RUNt, eerst met een incorrect klantnummer en dan met een correct klantnummer. Wat gebeurt er?

(a) Melden dat het opgegeven klantnummer niet in het bestand voorkomt (zie de demonstratie-RUN aan het begin van deze paragraaf, en regel 2090 hierboven).

(b) De bestanden moeten gesloten en opnieuw geopend worden om de wijzers weer aan het begin van de bestanden te positioneren (zie regel 2100).

(c)
```
2050       INPUT #1, N1$, K$, C1
2070       PRINT #2, N1$; ","; K$; ","; C1
```

(d) In beide gevallen ontstaat een EOF-conditie. Via de regels 2040 en 2090 wordt een foutmelding afgedrukt.

Wordt het gewenste record gevonden, dan laten we de desbetreffende naam en de daarbij gevonden kredietcode op het scherm verschijnen (regel 2160), zodat de gebruiker kan controleren of de kredietcode voor de juiste klant wordt aangepast.

```
2130 REM   KLANTNUMMER GEVONDEN, GA VERDER MET INVOER
2140 REM
2150       PRINT
2160       PRINT K$, ", KREDIETCODE: "; C1
2170       LINE INPUT "NIEUWE KREDIETCODE: "; C2$
2180       IF LEN(C2$) <> 1 THEN
              PRINT "1 CIJFER AUB"           :     GOTO 2170
2190       IF VAL(C2$) < 1  OR  VAL(C2$) > 5  THEN
              PRINT "CIJFER VAN 1 T/M 5 AUB" :     GOTO 2170
2200       LET C2 = VAL(C2$)
2210 REM
2220 REM   SCHRIJF NIEUW RECORD NAAR HULPBESTAND
2230 REM
2240       PRINT #2, N1$; ","; K$; ","; C2
2250 REM
```

In regel 2240 wordt de nieuwe kredietcode, samen met de klantnaam en het klantnummer die daarbij behoren, naar het hulpbestand weggeschreven. Daarmee zijn we op het punt aangeland dat er routines ontwikkeld zijn voor het doorzoeken van het oorspronkelijke bestand, en voor het plaatsen van oude en nieuwe gegevens in het bestand HULP. Aan de hand van de volgende vragen en opdrachten maken we het programma af.

Vraag

(a) Welke actie moet in dit stadium, gezien de stand van de recordwijzer in bestand 1, worden ondernomen?

(a) De resterende gegevens in het (oorspronkelijke) invoerbestand (buffer 1) moeten naar het hulpbestand (buffer 2) worden overgebracht.

Opdracht

(a) Maak het volgende programmasegment in overeenstemming met de gegeven REMARKs af (regels 2280, 2290 en 2300).

```
2260 REM   SCHRIJF RESTERENDE GEGEVENS NAAR HULPBESTAND
2270 REM
2280       ..............................  : REM *** INDIEN EOF GA NAAR CLOSE ***

2290       ..............................  : REM *** LEES VAN #1              ***

2300       ..............................  : REM *** SCHRIJF NAAR #2          ***

2310       GOTO 2280
2320 REM
2330 REM   SLUIT BESTANDEN
2340 REM
2350       CLOSE 1, 2
2400 REM
```

```
-------------------------
(a) 2280        IF EOF(1) THEN 2350
    2290        INPUT #1, N1$, K$, C1
    2300        PRINT #2, N1$; ","; K$; ","; C1
```

Opdracht

(a) Het volgende programmasegment kopieert de inhoud van het hulpbestand naar het oorspronkelijke bestand. Maak het segment in overeenstemming met de gegeven REMARKs af.

```
3000 REM    INITIALISEER BESTANDEN EN KOPIEER HULPBESTAND TERUG
3010 REM
3020        ............................. : REM *** OPEN INVOERBESTAND       ***
3030        ............................. : REM *** OPEN UITVOERBESTAND      ***
3040 REM
3050        IF EOF(1) THEN 3120
3060        ..................... : LEES RECORD VAN INVOERBESTAND            ***
3070        ..................... : REM *** SCHRIJF RECORD NAAR UITVOERBEST. ***
3080        GOTO 3050
3090 REM
3100 REM    SLUIT BESTANDEN EN BEPAAL OF HERHALING NODIG IS
3110 REM
3120        ..................... : REM *** SLUIT BESTANDEN 1 EN 2           ***
3130        CLS
3140        LINE INPUT "NOG EEN KLANT? (J/N): "; A$
3150        IF LEFT$(A$,1) = "J" THEN 1270
3160        PRINT "EINDE PROGRAMMA"
3170 REM
-------------------------
(a) 3020        OPEN "HULP" FOR INPUT AS 1
    3030        OPEN B$ FOR OUTPUT AS 2
    3060        INPUT #1, N1$, K$, C1
    3070        PRINT #2, N1$; ","; K$; ","; C1
    3120        CLOSE 1, 2
```

Draait u dit programma met grote bestanden, dan kan iedere mutatie een aanzienlijke hoeveelheid computertijd vergen. Door de gegevens in het oorspronkelijke bestand en de mutaties in volgorde van klantnummer te ordenen, vervalt de noodzaak om resterende bestandsgegevens steeds naar het hulpbestand te kopiëren. Regels 2260 t/m 3120 hoeven in dat geval slechts na de laatste mutatie uitgevoerd te worden. Dit is in het onderstaande overzicht niet verwerkt.

Overzicht volledig programma

```
1000 REM   WIJZIGEN KREDIETCODE
1010
1020 REM   ************************************************************************
1030 REM   INLEIDEND MODULE
1040 REM   ************************************************************************
1050 REM
1060 REM   VARIABELENLIJST
1070 REM    - B$      : NAAM VAN BESTAND
1080 REM    - N1$     : KLANTNUMMER IN BESTAND
1090 REM    - N2$     : OPGEGEVEN KLANTNUMMER
1100 REM    - K$      : KLANTNAAM
1110 REM    - C1      : KREDIETCODE IN BESTAND
1120 REM    - C2$/C2  : OPGEGEVEN KREDIETCODE
1130 REM    - A$      : DIALOOG
1140 REM
1150 REM   BESTANDEN
1160 REM
1170 REM   1. OORSPRONKELIJK BESTAND OP FLOPPY, NAAM IN TE VOEREN DOOR
1180 REM      GEBRUIKER (B$)
1190 REM   2. TIJDELIJK HULPBESTAND "HULP"
1200 REM
1210 REM   BEIDE BESTANDEN SEQUENTIEEL GEORGANISEERD
1220 REM
1230 REM   INITIALISATIES
1240 REM
1250       CLS   : MAXFILES = 2
1260       LINE INPUT "NAAM INVOERBESTAND       : "; B$
1270       OPEN B$ FOR INPUT AS 1
1280       OPEN "HULP" FOR OUTPUT AS 2
1290 REM
1300 REM   ************************************************************************
1310 REM   GEGEVENSINVOER EN -CONTROLE
1320 REM   ************************************************************************
1330 REM
1340       PRINT "GEEF 'STOP' (ZONDER AANH. TEKENS) ALS U KLAAR BENT"
1350       PRINT ""
1360       LINE INPUT "KLANTNUMMER              : "; N2$
1370       IF N2$ = "STOP" THEN 2100
1380       IF LEN(N2$) = 0 THEN
              PRINT "GEEF KLANTNUMMER OF 'STOP'"       : GOTO 1360
1390       IF LEN(N2$) <> 5 THEN
              PRINT "INVOERFOUT. 5 CIJFERS AUB"        : GOTO 1360
1400       IF VAL(N2$) = 0 THEN
              PRINT "INVOERFOUT. ALLEEN GETALLEN AUB"  : GOTO 1360
1410 REM
2000 REM   ************************************************************************
2010 REM   ZOEKROUTINE
2020 REM   ************************************************************************
2030 REM
2040       IF EOF(1) THEN 2090
2050          INPUT #1, N1$, K$, C1  : REM *** LEES VAN BESTAND NAAR
                                                  N1$, K$ en C1        ***
2060          IF N2$ = N1$ THEN 2150
2070          PRINT #2, N1$; ","; K$; ","; C1  : REM *** KLANTNUMMER KLOPT
                                                    NIET. SCHRIJF NAAR HULP ***
2080             GOTO 2040
2090       PRINT "***FOUT***KLANTNR. "; N2$; "KOMT NIET VOOR IN BESTAND.
              CONTROLEER EN VOER OPNIEUW IN AUB"
2100       CLOSE 1, 2                : REM *** ZET WIJZERS TERUG NAAR BEGIN  ***
2110       GOTO 1270
2120 REM
```

```
2130 REM   KLANTNUMMER GEVONDEN, GA VERDER MET INVOER
2140 REM
2150       PRINT ""
2160       PRINT K$, ", KREDIETCODE: "; C1
2170       LINE INPUT "NIEUWE KREDIETCODE: "; C2$
2180       IF LEN(C2$) <> 1 THEN
              PRINT "1 CIJFER AUB"                    : GOTO 2170
2190       IF VAL(C2$) < 1  OR  VAL(C2$) > 5 THEN
              PRINT "CIJFER VAN 1 T/M 5 AUB"          : GOTO 2170
2200       LET C2 = VAL(C2$)
2210 REM
2220 REM   SCHRIJF NIEUW RECORD NAAR HULPBESTAND
2230 REM
2240       PRINT #2, N1$; ","; K$; ","; C2
2250 REM
2260 REM   SCHRIJF RESTERENDE GEGEVENS NAAR HULPBESTAND
2270 REM
2280       IF EOF(1) THEN 2350          : REM *** INDIEN EOF GA NAAR CLOSE ***
2290          INPUT #1, N1$, K$, C1          : REM *** LEES VAN #1          ***
2300          PRINT #2, N1$; ","; K$; ","; C1 : REM *** SCHRIJF NAAR #2      ***
2310       GOTO 2280
2320 REM
2330 REM   SLUIT BESTANDEN
2340 REM
2350       CLOSE 1, 2
2400 REM
3000 REM   INITIALISEER BESTANDEN EN KOPIEER HULPBESTAND TERUG
3010 REM
3020       OPEN "HULP" FOR INPUT AS 1
3030       OPEN B$ FOR OUTPUT AS 2
3040 REM
3050       IF EOF(1) THEN 3120
3060          INPUT #1, N1$, K$, C1    : REM *** LEES RECORD VAN INVOERBEST. ***
3070          PRINT #2, N1$; ","; K$; ","; C1 : REM *** SCHRIJF RECORD NAAR
                                                    UITVOERBESTAND       ***
3080       GOTO 3050
3090 REM
3100 REM   SLUIT BESTANDEN EN BEPAAL OF HERHALING NODIG IS
3110 REM
3120       CLOSE 1, 2                  : REM *** SLUIT BESTANDEN 1 en 2      ***
3130       CLS
3140       LINE INPUT "NOG EEN KLANT? (J/N): "; A$
3150       IF LEFT$(A$,1) = "J" THEN 1270
3160       PRINT "EINDE PROGRAMMA"
3170 REM
9999       END
```

Toetsvragen

(a) Bepaal welke regel(s) in bovenstaand programma overeenkomen met de stappen van de eerder genoemde procedure (zie p.143).

(b) Modificeer het programma zodanig dat in plaats van de kredietcode de naam van de klant wordt veranderd (bedrijven – vooral dubieuze – hebben nogal eens de neiging om van naam te veranderen!). Er zijn slechts vijf wijzigingen nodig.

(a)
1. 1270
2. 1280
3. 1340 t/m 1400
4. 2040
   a. 2090
   b. 2100
   c. 2110
5. 2050
6. 2060
   a. 2160, 2170
   b. 2180, 2190
   c. 2200, 2240
   d. 2280 t/m 2310
   e. 2070, 2080
7. 3020, 3030
8. 3050 t/m 3080
9. 3050, 3120
10. 3140, 3150

(b)
```
2170       LINE INPUT "NIEUWE NAAM : "; K$
2180       IF LEN(K$) = 0  THEN
              PRINT "VOER NIEUWE NAAM IN AUB"            : GOTO 2170
2190       IF LEN(K$) > 20 THEN
              PRINT "MAXIMAAL 20 TEKENS. OPNIEUW AUB"    : GOTO 2170
2200 REM   *** DEZE REGEL VERWIJDERD ***
2240       PRINT #2, N1$; ","; K$; ","; C1
```

N.B. Dankzij het feit dat volledige records werden gelezen en (al dan niet gemodificeerd) weggeschreven, konden de wijzigingen tot een minimum beperkt blijven. Iets om te onthouden!

5.4 EDITTEN VAN EEN SEQUENTIEEL BESTAND

Onder 'editten' (een in de computerwereld vrij gangbare verbastering van het Engelse werkwoord 'to edit', dat 'redigeren' betekent) verstaan we het stapsgewijs zichtbaar maken van een bestand op het scherm, met de mogelijkheid delen van het bestand te wijzigen of te verwijderen, of nieuwe gegevens toe te voegen.

In deze paragraaf bespreken we een utility-programma om gegevensbestanden te editten. We gaan daarbij uit van dezelfde toepassing als in de vorige paragraaf, namelijk het klantenbestand met kredietcodes. De recordindeling daarbij – om nog even te recapituleren – was als volgt:

1. klantnummer van precies 5 cijfers
2. klantnaam van maximaal 20 tekens
3. kredietcode, opgeslagen als een numerieke waarde van 1, 2, 3, 4 of 5.

Eerste versie: veldgewijs afdrukken van record

We nemen een eerste aanzet met het onderstaande programma, waarbij we elk record veld voor veld op het scherm toveren door steeds de RETURN-toets in te drukken. Nadat alle velden van

een record op het scherm verschenen zijn, en we dus op het punt staan een nieuw record te bekijken, laten we het programma door middel van een CLS-opdracht het scherm schoonmaken.

Het programma ziet er als volgt uit:

```
1000 REM   EDIT-PROGRAMMA VOOR KLANTENBESTAND (VERSIE 1)
1010 REM
1020 REM   VARIABELENLIJST
1030 REM    - N$    : KLANTNUMMER
1040 REM    - K$    : NAAM VAN KLANT
1050 REM    - C     : KREDIETCODE
1060 REM    - B$    : NAAM VAN BESTAND
1070 REM    - A$    : DIALOOG
1080 REM
1090 REM   BESTANDEN
1100 REM    - SEQUENTIEEL/FLOPPY (1X), NAAM DOOR GEBRUIKER TE BEPALEN
1110 REM
1120 REM
1130       LINE INPUT "NAAM BESTAND: "; B$
1140       OPEN B$ FOR INPUT AS 1
1150 REM
1160 REM   LEES RECORD EN DRUK VELDEN AF OP SCHERM
1170 REM
1180       IF EOF(1) THEN 1340
1190       CLS
1200          PRINT "GEEF RETURN OM DOOR TE GAAN"
1210          INPUT #1, N$, K$, C
1220          PRINT N$
1230          LINE INPUT A$
1240 REM
1250          PRINT K$
1260          LINE INPUT A$
1270 REM
1280          PRINT C
1290          LINE INPUT A$
1300       GOTO 1180
1310 REM
1320 REM   AFSLUITING
1330 REM
1340       CLOSE 1
1350       PRINT "EINDE PROGRAMMA"
1360 REM
9999       END
```

Toetsvragen

(a) Wat wordt aan A$ toegekend in de regels 1230, 1260 en 1290?

(b) Wat is de bedoeling van de regels 1230, 1260 en 1290?

(c) Wanneer wordt in bovenstaand programma het scherm schoongemaakt?

(a) Niets, A$ functioneert als een 'dummy' (dummy = schijn, loos) variabele.

(b) De gegevens op het scherm vast te houden tot RETURN gegeven wordt.

(c) Vóór het afdrukken van de gegevens van een nieuw record.

Tweede versie: recordvelden veranderen ('change')

We passen het programma nu zo aan, dat de gebruiker de verschillende velden van een record kan veranderen. Daarbij gebruiken we een tijdelijk hulpbestand (HULP). Voorlopig beperken we ons tot het veranderen van het klantnummer.

```
1000 REM   EDIT-PROGRAMMA VOOR KLANTENBESTAND (VERSIE 2)
1010 REM
1020 REM   VARIABELENLIJST
1030 REM    - N$     : KLANTNUMMER
1040 REM    - K$     : NAAM VAN KLANT
1050 REM    - C/C$   : KREDIETCODE
1060 REM    - B$     : NAAM VAN BESTAND
1070 REM    - A$     : DIALOOG
1080 REM
1090 REM   BESTANDEN
1100 REM   1. INVOERBESTAND OP FLOPPY, NAAM IN TE VOEREN DOOR GEBRUIKER (B$)
1110 REM   2. TIJDELIJK HULPBESTAND "HULP"
1120 REM
1130 REM   BEIDE BESTANDEN SEQUENTIEEL GEORGANISEERD
1140 REM
1145       MAXFILES = 2
1150       LINE INPUT "NAAM BESTAND: "; B$
1160       OPEN B$ FOR INPUT AS 1
1170       OPEN "HULP" FOR OUTPUT AS 2
1180 REM
1190 REM   LEES RECORD EN DRUK VELDEN AF OP SCHERM
1200 REM
1210       CLS
1220       IF EOF(1) THEN 1470
1230          PRINT "GEEF 'CHANGE' OF 'C' (ZONDER AANH. TEKENS) "
1240          PRINT "OM TE VERANDEREN"
1250          PRINT "GEEF RETURN/ENTER OM VERDER TE GAAN"
1260          INPUT #1, N$, K$, C
1270          PRINT N$
1280          LINE INPUT A$
1290          IF LEFT$(A$,1) = "C" THEN GOSUB 2040
1300 REM


2000 REM   ************************************************************************
2010 REM   ROUTINE VOOR VERANDEREN KLANTNUMMER
2020 REM   ************************************************************************
2030 REM
2040       LINE INPUT "NIEUW KLANTNUMMER (5 CIJFERS): "; N$
2050       IF LEN(N$) = 0  THEN
              PRINT "GEEF KLANTNUMMER AUB"                              : GOTO 2040
2060       IF LEN(N$) <> 5 THEN
              PRINT "INVOERFOUT. OPNIEUW MET PRECIES 5 CIJFERS AUB" : GOTO 2040
2070       IF VAL(N$) = 0 THEN
              PRINT "INVOERFOUT. ALLEEN CIJFERS AUB"                    : GOTO 2040
2080       RETURN
2090 REM
```

Let op de belangrijkste wijzigingen ten opzichte van de vorige versie: het uitvoerbestand (regel 1170), de test op een 'CHANGE'-opdracht in regel 1290, en de routine voor het invoeren van, en het controleren op, een nieuw klantnummer. De invoercontroles zijn dezelfde als destijds bij het allereerste programma voor het veranderen van de kredietcode. Merk overigens op dat de routine niet zelf het nieuwe klantnummer naar HULP wegschrijft. Er is bewust gekozen voor een 'centrale' PRINT-opdracht om een heel record in één keer weg te schrijven.

Aan u de eer om de volgende routine – bedoeld om de naam van een klant te veranderen – af te maken (regels 3040 t/m 3060). Ga daarbij uit van de gegeven REMARKs.

```
1310          PRINT K$
1320          LINE INPUT A$
1330          IF LEFT$(A$,1) = "C" THEN GOSUB 3040
1340 REM


3000 REM   ************************************************************************
3010 REM   ROUTINE VOOR VERANDEREN VAN KLANTNAAM
3020 REM   ************************************************************************
3030 REM
3040       ............................. : REM *** NAAM INVOEREN              ***

3050       ............................. : REM *** TEST OP NULSTRING          ***

3060       ............................. : REM *** TEST OP LENGTE             ***

3070       RETURN
3080 REM
-------------------------
3040       LINE INPUT "NIEUWE NAAM (MAX. 20 TEKENS): "; K$
3050       IF LEN(K$) = 0 THEN
              PRINT "NIEUWE NAAM INVOEREN AUB"                  : GOTO 3040
3060       IF LEN(K$) > 20 THEN
              PRINT "INVOERFOUT. MAX. 20 TEKENS. OPNIEUW AUB"   : GOTO 3040
```

Nu een routine om de kredietcode te veranderen, en opdrachten om het al dan niet gewijzigde record weg te schrijven naar het hulpbestand. Maak daartoe het volgende programmasegment af, in overeenstemming met de gegeven REMARKs (regels 1410, 4040, 4050, 4060 en 4070).

```
1350          PRINT C
1360          LINE INPUT A$
1370          IF LEFT$(A$,1) = "C" THEN GOSUB 4040
1380 REM
1390 REM      SCHRIJF AL DAN NIET GEWIJZIGDE RECORDS NAAR HULPBESTAND
1400 REM
1410          ............................. : REM *** SCHRIJF NAAR HULP      ***
1420 REM
1430       GOTO 1220
1440 REM
```

```
4000 REM   ************************************************************************
4010 REM   ROUTINE VOOR VERANDEREN KREDIETCODE
4020 REM   ************************************************************************
4030 REM
4040       ........................... : REM *** KREDIETCODE INVOEREN        ***

4050       ........................... : REM *** TEST OP LENGTE              ***

4060       ........................... : REM *** TEST OP WAARDE 1-5          ***

4070       ........................... : REM *** CONVERTEER NAAR GETALWAARDE***

4080       RETURN
4090 REM
```

```
1410          PRINT #2, N$; ","; K$; ","; C

4040       LINE INPUT "NIEUWE KREDIETCODE (1-5): "; C$
4050       IF LEN(C$) <> 1 THEN
              PRINT "SLECHTS 1 CIJFER TOEGESTAAN. OPNIEUW AUB"       : GOTO 4040
4060       IF VAL(C$) < 1  THEN
              PRINT "ALLEEN CIJFER 1 T/M 5 TOEGESTAAN. OPNIEUW AUB" : GOTO 4040
4070       LET C = VAL(C$)
```

Rest nog het kopiëren van het hulpbestand naar het oorspronkelijke bestand. U wordt weer gevraagd volgens het bekende recept de ontbrekende regels te leveren (regels 1490, 1500, 1540 t/m 1560, 1610).

```
1450 REM   SLUIT BESTANDEN EN INITIALISEER OPNIEUW
1460 REM
1470       CLOSE 1, 2
1480 REM
1490       ........................... : REM *** OPEN INVOERBESTAND        ***

1500       ........................... : REM *** OPEN UITVOERBESTAND       ***

1510 REM
1520 REM   KOPIEER TERUG
1530 REM
1540       ........................... : REM *** BIJ EOF NAAR 1610         ***

1550          ........................ : REM *** LEES RECORD               ***

1560          ........................ : REM *** SCHRIJF WEG               ***

1570       GOTO 1540
1580 REM
1590 REM   AFSLUITING
1600 REM
1610       ........................... : REM *** SLUIT BESTANDEN           ***

1620       PRINT "EINDE PROGRAMMA"
1630 REM
```

```
(a) 1490        OPEN "HULP" FOR INPUT AS 1
    1500        OPEN B$ FOR OUTPUT AS 2

(b) 1540        IF EOF(1) THEN 1610
    1550           INPUT #1, N$, K$, C
    1560           PRINT #2, N$; ","; K$; ","; C

(c) 1610        CLOSE 1, 2
```

Derde versie: records verwijderen ('delete') en toevoegen ('insert')

In deze versie bouwen we, naast de mogelijkheid om recordgegevens te veranderen, ook de mogelijkheid in om records geheel te verwijderen en toe te voegen. Eerst het verwijderen. Daarbij lezen we het record wel in van het oorspronkelijke invoerbestand, maar schrijven het niet naar HULP. Het record verdwijnt dus. We beginnen met de gebruiker te informeren dat deze mogelijkheid bestaat:

```
1245        PRINT "GEEF 'DELETE' OF 'D' (ZONDER AANH. TEKENS) "
1246        PRINT "OM EEN RECORD TE VERWIJDEREN"
```

Opdracht

(a) Schrijf zelf de opdracht (direct na regel 1290) om op het invoeren van DELETE of D te testen; bij invoer hiervan dient gesprongen te worden naar regel 1210 (waarom?).

(a) 1295 `IF LEFT$(A$,1) = "D" THEN 1210`

Door de sprong naar 1210 wordt het wegschrijven van het record omzeild, en gaat het programma verder met een nieuw record.

Vervolgens het toevoegen van volledige records aan het bestand. Hiervoor kunnen we dankbaar gebruik maken van de eerder geschreven routines (regels 2000 t/m 2090, 3000 t/m 3080 en 4000 t/m 4090). Het record wordt toegevoegd na het lokaliseren van het record dat eraan vooraf moet gaan. Op grond van deze informatie en de gegeven REMARKs kunt u het volgende programmasegment afmaken (regels 1296, 1446 en 1447).

```
1247        PRINT "GEEF 'INSERT' OF 'I' OM NA DIT RECORD ";
1248        PRINT "EEN NIEUW TOE TE VOEGEN"

1296        ..............................  : REM *** TEST OP I. INDIEN AANWEZIG,
                                                  GA NAAR 1441      ***
                                      (op regel 1441 komen we straks terug)
```

```
1443 REM   STEL NIEUWE RECORDS SAMEN EN VOEG TOE AAN HULP
1444 REM
1445       GOSUB 2040                    : REM *** VOER KLANTNUMMER IN      ***
1446       ............................. : REM *** VOER NAAM KLANT IN       ***
1447       ............................. : REM *** VOER KREDIETCODE IN      ***
1448       GOTO 1410
1449 REM
-------------------------
1296       IF LEFT$(A$,1) = "I" THEN 1444
1445       GOSUB 3040
1446       GOSUB 4040
```

Zoals beloofd komen we nu terug op regel 1441. Deze is nodig om de het huidige record – dat dus aan het toe te voegen record vooraf moet gaan – naar HULP over te brengen. Dus:

```
1441       PRINT #2, N$; ","; K$; ","; C
```

We knopen de eindjes nu aan elkaar. Het resulterende programma geeft ons de mogelijkheid recordvelden te veranderen, records te verwijderen of toe te voegen, of simpelweg de inhoud van het bestand op het scherm af te drukken.

```
1000 REM   EDIT-PROGRAMMA VOOR KLANTENBESTAND (VERSIE 3)
1010 REM
1020 REM   VARIABELENLIJST
1030 REM    - N$     : KLANTNUMMER
1040 REM    - K$     : NAAM VAN KLANT
1050 REM    - C/C$   : KREDIETCODE
1060 REM    - B$     : NAAM VAN BESTAND
1070 REM    - A$     : DIALOOG
1080 REM
1090 REM   BESTANDEN
1100 REM   1. INVOERBESTAND OP FLOPPY, NAAM IN TE VOEREN DOOR GEBRUIKER (B$)
1110 REM   2. TIJDELIJK HULPBESTAND "HULP"
1120 REM
1130 REM   BEIDE BESTANDEN SEQUENTIEEL GEORGANISEERD
1140 REM
1145       MAXFILES = 2
1150       LINE INPUT "NAAM BESTAND: "; B$
1160       OPEN B$ FOR INPUT AS #1
1170       OPEN "HULP" FOR OUTPUT AS #2
1180 REM
```

```
1190 REM   LEES RECORD EN DRUK VELDEN AF OP SCHERM
1200 REM
1210       CLS
1220       IF EOF(1) THEN 1470
1230          PRINT "GEEF 'CHANGE' OF 'C' (ZONDER AANH. TEKENS) ";
1240          PRINT "OM TE VERANDEREN"
1245          PRINT "GEEF 'DELETE' OF 'D' (ZONDER AANH. TEKENS) ";
1246          PRINT "OM EEN RECORD TE VERWIJDEREN
1247          PRINT "GEEF 'INSERT' OF 'I' (ZONDER AANH. TEKENS) ";
1248          PRINT "OM NA DIT RECORD EEN NIEUW TOE TE VOEGEN"
1250          PRINT "GEEF RETURN/ENTER OM VERDER TE GAAN"
1260          INPUT #1, N$, K$, C
1270          PRINT N$
1280          LINE INPUT A$
1290          IF LEFT$(A$,1) = "C" THEN GOSUB 2040
1295          IF LEFT$(A$,1) = "D" THEN 1210
1296          IF LEFT$(A$,1) = "I" THEN 1445
1300 REM
1310          PRINT K$
1320          LINE INPUT A$
1330          IF LEFT$(A$,1) = "C" THEN GOSUB 3040
1340 REM
1350          PRINT C
1360          LINE INPUT A$
1370          IF LEFT$(A$,1) = "C" THEN GOSUB 4040
1380 REM
1390 REM      SCHRIJF AL DAN NIET GEWIJZIGDE RECORDS NAAR HULPBESTAND
1400 REM
1410          PRINT #2, N$; ","; K$; ","; C
1420 REM
1430       GOTO 1220
1440 REM
1441       PRINT #2, N$; ","; K$; ","; C
1442 REM
1443 REM   STEL NIEUWE RECORDS SAMEN EN VOEG TOE AAN HULP
1444 REM
1445       GOSUB 2040                    : REM *** VOER KLANTNUMMER IN      ***
1446       GOSUB 3040                    : REM *** VOER NAAM KLANT IN       ***
1447       GOSUB 4040                    : REM *** VOER KREDIETCODE IN      ***
1448       GOTO 1410
1449 REM
1450 REM   SLUIT BESTANDEN EN INITIALISEER OPNIEUW
1460 REM
1470       CLOSE 1, 2
1480 REM
1490       OPEN "HULP" FOR INPUT AS 1
1500       OPEN B$ FOR OUTPUT AS 2
1510 REM
1520 REM   KOPIEER TERUG
1530 REM
1540       IF EOF(1) THEN 1610           : REM *** BIJ EOF NAAR 1610        ***
1550          INPUT #1, N$, K$, C        : REM *** LEES RECORD              ***
1560          PRINT #2, N$; ","; K$; ","; C : REM *** SCHRIJF WEG           ***
1570       GOTO 1540
1580 REM
```

(wordt vervolgd)

```
1590 REM   AFSLUITING
1600 REM
1610       CLOSE 1, 2                    : REM *** SLUIT BESTANDEN          ***
1620       PRINT "EINDE PROGRAMMA"
1625       END
1630 REM
2000 REM   ************************************************************************
2010 REM   ROUTINE VOOR VERANDEREN KLANTNUMMER
2020 REM   ************************************************************************
2030 REM
2040       LINE INPUT "NIEUW KLANTNUMMER (5 CIJFERS): "; N$
2050       IF LEN(N$) = 0 THEN
              PRINT "GEEF KLANTNUMMER AUB"                                  : GOTO 2040
2060       IF LEN(N$) <> 5 THEN
              PRINT "INVOERFOUT. OPNIEUW MET PRECIES 5 CIJFERS AUB"  : GOTO 2040
2070       IF VAL(N$) = 0 THEN
              PRINT "INVOERFOUT. ALLEEN CIJFERS AUB"                        : GOTO 2040
2080       RETURN
2090 REM
3000 REM   ************************************************************************
3010 REM   ROUTINE VOOR VERANDEREN VAN KLANTNAAM
3020 REM   ************************************************************************
3030 REM
3040       LINE INPUT "NIEUWE NAAM (MAX. 20 TEKENS):  "; K$
3050       IF LEN(K$) = 0 THEN
              PRINT "NIEUWE NAAM INVOEREN AUB"                              : GOTO 3040
3060       IF LEN(K$) > 20 THEN
              PRINT "INVOERFOUT. MAX. 20 TEKENS. OPNIEUW AUB"               : GOTO 3040
3070       RETURN
3080 REM
4000 REM   ************************************************************************
4010 REM   ROUTINE VOOR VERANDEREN KREDIETCODE
4020 REM   ************************************************************************
4030 REM
4040       LINE INPUT "NIEUWE KREDIETCODE (1-5)     : "; C$
4050       IF LEN(C$) <> 1 THEN
              PRINT "SLECHTS 1 CIJFER TOEGESTAAN. OPNIEUW AUB"      : GOTO 4040
4060       IF VAL(C$) < 1  OR  VAL(C$) > 5  THEN
              PRINT "ALLEEN CIJFER 1 T/M 5 TOEGESTAAN. OPNIEUW AUB" : GOTO 4040
4070       LET C = VAL(C$)
4080       RETURN
4090 REM
9999       END
```

Algemene procedure

De algemene procedure voor het veranderen, verwijderen of toevoegen van records is als volgt.

1. Open invoerbestand.
2. Open een tijdelijk bestand voor uitvoer.
3. Check op EOF voor het invoerbestand; indien einde bestand, ga naar stap 11.
4. Druk een overzicht af van de mogelijkheden van het programma (de zogenaamde opties, dat wil zeggen 'C', 'D', 'I' of 'niets veranderen').

5. Lees het eerste c.q. het eerstvolgende record van het invoerbestand en druk het eerste veld daarvan af.
6. Stel de gebruiker in de gelegenheid één van de programmaopties (zie stap 4) te kiezen.
7. Indien optie 'C' gekozen:
   a. Laat gebruiker het vervangende gegeven invoeren, en controleer dat gegeven op geldigheid.
   b. Druk het volgende veld van het record af (indien aanwezig).
   c. Stel gebruiker in de gelegenheid ook dit veld te veranderen (optie C).
   d. Indien gebruiker wil veranderen, voer vervangend gegeven in en test op geldigheid.
   e. Herhaal 7b t/m d tot alle velden van record verwerkt zijn.
   f. Schrijf al dan niet gewijzigd record naar hulpbestand.
   g. Ga naar stap 3.
8. Indien optie 'I' gekozen:
   a. Schrijf huidige record naar hulpbestand.
   b. Laat gebruiker nieuw record invoeren, en controleer op geldigheid.
   c. Schrijf nieuw record naar hulpbestand.
   d. Ga naar stap 3.
      N.B. Eventueel kan men in plaats hiervan opnieuw de mogelijkheid geven om een record in te voegen.
9. Indien optie 'D' gekozen:
   ga naar stap 3 (het huidige record wordt dan niet naar het hulpbestand geschreven, en wordt in effect dus verwijderd).
10. Indien geen C-, D- of I-optie gekozen, maar RETURN, ga terug naar stap 7b en herhaal.
11. Sluit beide bestanden.
12. Open oorspronkelijk bestand voor uitvoer, en hulpbestand voor invoer.
13. Test op EOF voor tijdelijk bestand; indien einde bestand, sluit beide bestanden.
14. Lees records van tijdelijk bestand en schrijf naar oorspronkelijk bestand.
15. Herhaal de stappen 13 en 14 tot EOF optreedt en de bestanden gesloten worden.

Toetsvraag

(a) Bepaal welke regel(s) in versie 3 van het edit-programma overeenkomen met de stappen in bovenstaande procedure.

(a) 1. 1160
2. 1170
3. 1220
4. 1230 t/m 1250
5. 1260, 1270
6. 1280
7. a. 1290, 2040 t/m 2090
b. 1310
c. 1320
d. 3040 t/m 3080
e. 1350 t/m 1370
4040 t/m 4090
f. 1410
g. 1430
8. a. 1296
b. 1445 t/m 1447
c. 1448/1410
d. 1430
9. 1295/1210
10. 1430
11. 1470
12. 1500, 1490
13. 1540, 1610
14. 1550, 1560
15. 1570

5.5 SAMENVOEGEN VAN SEQUENTIËLE BESTANDEN (MERGE)

In administratieve computerbestanden worden gegevens veelal in alfabetische of numerieke volgorde opgeborgen. Het bestand is dan volgens een zogenaamde *sleutel* geordend of gesorteerd. Zo'n sleutel kan bijvoorbeeld een klantnummer, klantnaam of een artikelcode zijn. Vaak is het nodig de inhoud van twee bestanden, waarin reeds een of andere ordening aanwezig is, te combineren of samen te voegen (Engels: merge) tot een derde bestand met dezelfde ordening. In deze paragraaf gaan we hier nader op in. Niet slechts wegens de praktische kant van de zaak, maar ook om u kennis te laten maken met bestandtechnieken die in breder verband toe te passen zijn.

De algemene procedure om twee bestanden samen te voegen is als volgt.

1. Open beide samen te voegen bestanden (nummers 1 en 2) voor invoer, en check beide op EOF om zeker te zijn dat er in ieder geval gegevens zijn. Indien EOF, ga naar stap 11.
2. Open een derde bestand (dat de inhoud van bestanden 1 en 2 moet gaan bevatten) voor uitvoer (bestandnummer 3).
3. Test bestand 1 op EOF; indien einde bestand ga naar stap 10.
4. Lees eerste record van bestand 1.
5. Test bestand 2 op EOF; indien einde bestand ga naar stap 10.
6. Lees eerste record van bestand 2.

7. Bepaal welk van beide records (van bestand 1 of 2) overgebracht moet worden naar bestand 3.
8. Schrijf het gekozen record naar bestand 3. Hiervoor zijn twee verschillende programmadelen nodig: één voor een record van bestand 1, en één voor een record van bestand 2.
9. Test op EOF en lees een volgend record van dat bestand waarvan in stap 8 een record naar bestand 3 werd geschreven. Ook hier zijn twee verschillende routines nodig:
   a. test op EOF en lees een volgend record van bestand 1, of
   b. test op EOF en lees een volgend record van bestand 2.
10. Er zijn weer twee verschillende programmadelen nodig om de resterende gegevens in bestand 1 of 2 naar bestand 3 over te brengen:
    a. indien EOF het eerst voor bestand 1 optreedt, kopieer dan de resterende gegevens in bestand 2 naar bestand 3;
    b. indien EOF het eerst voor bestand 2 optreedt, kopieer dan de resterende gegevens in bestand 1 naar bestand 3.
11. Sluit alle bestanden.
12. Voer facultatief een routine uit om de inhoud van bestand 3 geheel of gedeeltelijk ter controle af te drukken.

Een voorbeeld

Als voorbeeld gebruiken we twee transactiebestanden zoals besproken in toetsvraag 4 aan het einde van het vorige hoofdstuk. Elk record van zo'n bestand bestaat uit een string van 14 tekens, als volgt ingedeeld:

```
1 - - - 5 / 6 7 / 8 - - - - - 14

rekeningnr.          bedrag
 (5 tekens)        (7 tekens)

        transactiecode
          (2 tekens)
```

We nemen aan dat de records in beide bestanden in volgorde van geopende rekeningnummers geordend zijn. De bedoeling is een derde bestand te maken dat de gegevens van de eerste twee bestanden zo samenvoegt dat de klantnummervolgorde wordt gehandhaafd. Daarbij gaan we ervan uit dat hetzelfde rekeningnummer in één of meer records van één of beide bestanden kan vóórkomen. In het geval dat twee records van verschillende bestanden inderdaad hetzelfde rekeningnummer hebben, nemen we aan dat het record van bestand 1 het eerst naar bestand 3 moet worden gekopieerd (zie fig. 5-1).

bestand 1	bestand 2
10762	10761
18102	18203
43611	43611
43611	80111
43611	80772
80223	80772
98702	89012

bestand 3

```
10761
10762
18102
18203
43611   (van bestand 1)
43611   (idem)
43611   (idem)
43611   (van bestand 2)
80111
80223
80772
80772
89012
98702
```

fig.5-1. Voorbeeld van volgorde waarin records worden samengevoegd. Alleen de rekeningnummers zijn aangegeven.

We noemen het programma MERGE, naar het Engelse woord voor samenvoegen. Het eerste deel ziet er als volgt uit:

```
100 REM   PROGRAMMA MERGE VOOR SAMENVOEGEN BESTANDEN
110 REM
120 REM   VARIABELENLIJST
130 REM    - N1$, N2$, N3$ : BESTANDNAMEN
140 REM    - R1$, R2$      : RECORDS VAN BESTANDEN 1 EN 2
150 REM    - R1, R2        : REKENINGNUMMERS
160 REM    - A$            : DIALOOG
165 REM
170 REM   BESTANDEN
180 REM    - TWEE INVOERBESTANDEN EN EEN UITVOERBESTAND, NAMEN DOOR GEBRUIKER
190 REM      IN TE VOEREN, ALLE BESTANDEN SEQUENTIEEL OP FLOPPY
200 REM
210 REM   INITIALISATIE BESTANDEN
215 REM
220       LINE INPUT "BESTAND 1 (INVOER) : "; N1$
230       LINE INPUT "BESTAND 2 (INVOER) : "; N2$
240       LINE INPUT "BESTAND 3 (UITVOER): "; N3$
250 REM
255       MAXFILES = 3
260       OPEN N1$ FOR INPUT AS 1
270       OPEN N2$ FOR INPUT AS 2
280       OPEN N3$ FOR OUTPUT AS 3
290 REM
```

Vervolgens lezen we het eerste record van bestand 1. Merk op dat (in overeenstemming met de algemene procedure) eerst een EOF-check wordt uitgevoerd voor het geval dat bestand 1 leeg is (regel 320). In dat geval wordt de inhoud van bestand 2 toch naar bestand 3 overgebracht, hoewel er eigenlijk niet meer van samenvoegen sprake is. Zijn beide bestanden leeg, dan valt er helemaal weinig te doen. Het programma produceert dan via regel 900 een foutmelding.

```
300 REM   LEES BESTAND 1
310 REM
315       IF  EOF(1)  AND  EOF(2)  THEN 900
320       IF  EOF(1)  THEN  INPUT #2, R2$ : GOTO 610
330       INPUT #1, R1$
```

De volgende regel converteert het eerste veld van een record van bestand 1 – het rekeningnummer – naar een numerieke waarde:

```
340       LET R1 = VAL(LEFT$(R1$,5))
```

U wordt verzocht nu zelf een soortgelijk segment te schrijven voor het lezen van een record van bestand 2 en het converteren van het rekeningnummer naar een numerieke waarde.

```
(a) 360 REM   LEES BESTAND 2
    365 REM
    370       ..............................

    380       ..............................

    390       ..............................

    400 REM
```

```
(a) 370       IF EOF(2) THEN 670
    380       INPUT #2, R2$
    390       LET R2 = VAL(LEFT$(R2$,5))
```

Vervolgens moet worden bepaald welk record – van bestand 1 of bestand 2 – het eerst naar bestand 3 moet worden gekopieerd:

```
410 REM   WELK RECORD NAAR BESTAND 3?
415 REM
420       IF R1 = R2 THEN 470
430       IF R1 < R2 THEN 470
440       GOTO 540
450 REM
```

Tot nu toe ziet het programma er als volgt uit:

```
100 REM   PROGRAMMA MERGE VOOR SAMENVOEGEN BESTANDEN
110 REM
120 REM   VARIABELENLIJST
130 REM    - N1$, N2$, N3$ : BESTANDNAMEN
140 REM    - R1$, R2$      : RECORDS VAN BESTANDEN 1 EN 2
150 REM    - R1, R2        : REKENINGNUMMERS
160 REM    - A$            : DIALOOG
165 REM
170 REM   BESTANDEN
180 REM    - TWEE INVOERBESTANDEN EN EEN UITVOERBESTAND, NAMEN DOOR GEBRUIKER
190 REM      IN TE VOEREN, ALLE BESTANDEN SEQUENTIEEL OP FLOPPY
200 REM
210 REM   INITIALISEER BESTANDEN
215 REM
220       LINE INPUT "BESTAND 1 (INVOER) : "; N1$
230       LINE INPUT "BESTAND 2 (INVOER) : "; N2$
240       LINE INPUT "BESTAND 3 (UITVOER): "; N3$
250 REM
255       MAXFILES = 3
260       OPEN N1$ FOR INPUT AS 1
270       OPEN N2$ FOR INPUT AS 2
280       OPEN N3$ FOR OUTPUT AS 3
290 REM
300 REM   LEES BESTAND 1
310 REM
315       IF EOF(1)  AND  EOF(2)  THEN 900
320       IF EOF(1)  THEN INPUT #2, R2$          : GOTO 610
330       INPUT #1, R1$
340       LET R1 = VAL(LEFT$(R1$,5))
350 REM
360 REM   LEES BESTAND 2
365 REM
370       IF EOF(2) THEN 670
380       INPUT #2, R2$
390       LET R2 = VAL(LEFT$(R2$,5))
400 REM
410 REM   WELK RECORD NAAR BESTAND 3?
415 REM
420       IF R1 = R2 THEN 470
430       IF R1 < R2 THEN 470
440       GOTO 540
450 REM
```

Toetsvragen

(a) Wat zou het eerstvolgende deel van het programma moeten doen?

(b) Het programma test in regel 420 op gelijkheid. In regel 430 op het kleiner zijn van rekeningnummer R1 dan rekeningnummer R2. Als beide tests 'onwaar' zijn, hoe is dan de relatie tussen R1 en R2?

(c) Wat moet er gebeuren in het programmadeel dat bij regel 540 begint, en waarnaar vanuit regel 440 wordt gesprongen?

(a) De inhoud van R1$ naar bestand 3 kopiëren.
(b) R1 is groter dan R2.
(c) De inhoud van R2$ moet worden gekopieerd naar bestand 3.

De opdracht voor het kopiëren van een record van bestand 1 naar bestand 3 (met REMARK) is:

```
460 REM   KOPIEER RECORD VAN 1 NAAR 2
465 REM
470       PRINT #3, R1$
```

Toetsvraag

(a) Na uitvoering van dit segment moet het programma een volgend record van bestand 1 lezen. U zou in de verleiding kunnen komen om terug te springen naar regel 315 en van daaruit verder te gaan. Waarom kan dit niet?

(a) Het programmadeel vanaf regel 315 leest inderdaad van bestand 1, maar gaat direct daarna door met het lezen vanaf bestand 2. Het record dat al in R2$ was opgeslagen wordt daarbij overschreven, zonder naar bestand 3 te zijn overgebracht.

Gezien het in de toetsvraag aangestipte probleem, schrijven we een afzonderlijk segment om het volgende record van bestand 1 te lezen:

```
480       IF EOF(1) THEN 610
490       INPUT #1, R1$
500       LET R1 = VAL(LEFT$(R1$,5))
510       GOTO 420
520 REM
```

Toetsvraag

(a) Bij het einde van bestand 1 wordt gesprongen naar regel 610. Wat moet daar gebeuren?

(a) Aangezien alle records van bestand 1 naar bestand 3 zijn gekopieerd, moeten nu de resterende records van bestand 2 worden gekopieerd ('ge-dumped') naar bestand 3. Daar komen we straks op terug.

In aansluiting op het bovenstaande segment is (ook) een programmadeel nodig om het huidige record van bestand 2 naar bestand 3 te kopiëren en een nieuw record in te lezen. Dat ziet er als volgt uit:

```
530 REM   KOPIEER RECORD VAN 2 NAAR 3
535 REM
540       PRINT #3, R2$
550       IF EOF(2) THEN 670
560       INPUT #2, R2$
570       LET R2 = VAL(LEFT$(R2$,5))
580       GOTO 420
590 REM
```

We zijn nu al een eind op weg. Als volgende stap schrijven we twee gelijksoortige segmenten om de resterende records van bestand 2 naar 3, en van bestand 1 naar 3 over te brengen. De kop van het eerste segment is:

```
600 REM   DUMP RESTERENDE REGELS VAN 2 NAAR 3
605 REM
610       ...
```

Toetsvraag

(a) Naar het hier begonnen programmasegment wordt vanaf regel 480 gesprongen, als zojuist een record van bestand 1 naar bestand 3 is overgebracht. Aanleiding tot de sprong was het feit dat voor bestand 1 een EOF-conditie optrad, dat wil zeggen alle gegevens van dat bestand zijn gekopieerd naar bestand 3. Gezien het feit dat een record aan R2$ is toegekend, wat moet er in regel 610 gebeuren?

(a) De inhoud van R2$ moet worden gekopieerd naar bestand 3.

De rest is betrekkelijk eenvoudig. We testen bestand 2 op EOF en 'dumpen' eventueel resterende records naar bestand 3.

```
610       PRINT #3, R2$
620       IF EOF(2) THEN 730
630       INPUT #2, R2$
640       GOTO 610
650 REM
```

Schrijf nu een overeenkomstig segment om records van bestand 1 in bestand 3 te dumpen. Bij EOF voor bestand 1 dient eveneens naar regel 730 te worden gesprongen.

(a)
```
660 REM   DUMP RESTERENDE RECORDS VAN 1 NAAR 3
665 REM
670       ..............................

680       ..............................

690       ..............................

700       ..............................

710 REM
```

(b) Wat moet er bij regel 730 gebeuren?

(a)
```
670       PRINT #3, R1$
680       IF EOF(1) THEN 730
690       INPUT #1, R1$
700       GOTO 670
```

(b) Alle bestanden moeten worden gesloten; alle gegevens zijn immers gekopieerd en samengevoegd.

Na het sluiten van de bestanden stellen we de gebruiker in de gelegenheid de inhoud van het resulterende bestand te controleren. Alle activiteit in het programma vond namelijk tot nu toe plaats tussen het werk- en het schijfgeheugen, zonder dat de gebruiker daar op het scherm iets van merkte. Mits het bestand niet al te groot is, kan zo'n controle dus nuttig zijn. Een en ander is in het volgende overzicht van MERGE verwerkt.

```
100 REM   PROGRAMMA MERGE VOOR SAMENVOEGEN BESTANDEN
110 REM
120 REM   VARIABELENLIJST
130 REM    - N1$, N2$, N3$ : BESTANDNAMEN
140 REM    - R1$, R2$      : RECORDS VAN BESTANDEN 1 EN 2
150 REM    - R1, R2        : REKENINGNUMMERS
160 REM    - A$            : DIALOOG
165 REM
170 REM   BESTANDEN
180 REM    - TWEE INVOERBESTANDEN EN EEN UITVOERBESTAND, NAMEN DOOR GEBRUIKER
190 REM      IN TE VOEREN, ALLE BESTANDEN SEQUENTIEEL OP FLOPPY
200 REM
210 REM   INITIALISEER BESTANDEN
215 REM
220       LINE INPUT "BESTAND 1 (INVOER) : "; N1$
230       LINE INPUT "BESTAND 2 (INVOER) : "; N2$
240       LINE INPUT "BESTAND 3 (UITVOER): "; N3$
250 REM
255       MAXFILES = 3
260       OPEN N1$ FOR INPUT AS 1
270       OPEN N2$ FOR INPUT AS 2
280       OPEN N3$ FOR OUTPUT AS 3
290 REM
300 REM   LEES BESTAND 1
310 REM
315       IF EOF(1)  AND  EOF(2)  THEN 900
320       IF EOF(1)  THEN INPUT #2, R2$          : GOTO 610
330       INPUT #1, R1$
340       LET R1 = VAL(LEFT$(R1$,5))
350 REM
360 REM   LEES BESTAND 2
365 REM
370       IF EOF(2) THEN 670
380       INPUT #2, R2$
390       LET R2 = VAL(LEFT$(R2$,5))
400 REM
410 REM   WELK RECORD NAAR BESTAND 3?
415 REM
420       IF R1 = R2 THEN 470
430       IF R1 < R2 THEN 470
440       GOTO 540
450 REM
```

(wordt vervolgd)

```
460 REM   KOPIEER RECORD VAN 1 NAAR 3
465 REM
470       PRINT #3, R1$
480       IF EOF(1) THEN 610
490       INPUT #1, R1$
500       LET R1 = VAL(LEFT$(R1$,5))
510       GOTO 420
520 REM
530 REM   KOPIEER RECORD VAN 2 NAAR 3
535 REM
540       PRINT #3, R2$
550       IF EOF(2) THEN 670
560       INPUT #2, R2$
570       LET R2 = VAL(LEFT$(R2$,5))
580       GOTO 420
590 REM
600 REM   DUMP RESTERENDE REGELS VAN 2 NAAR 3
605 REM
610       PRINT #3, R2$
620       IF EOF(2) THEN 730
630       INPUT #2, R2$
640       GOTO 610
650 REM
660 REM   DUMP RESTERENDE RECORDS VAN 1 NAAR 3
665 REM
670       PRINT #3, R1$
680       IF EOF(1) THEN 730
690       INPUT #1, R1$
700       GOTO 670
710 REM
720 REM   SLUIT BESTANDEN / EVENTUEEL CONTROLEREN
725 REM
730       CLOSE 1, 2, 3
740       PRINT "SAMENVOEGEN KLAAR"
750       PRINT "" : PRINT ""
760       LINE INPUT "WILT U MERGE BESTAND ZIEN? (J/N): "; A$
770       IF LEFT$(A$,1) = "N" THEN 870
780 REM
790 REM   INHOUD MERGE BESTAND AFDRUKKEN
795 REM
800       OPEN N3$ FOR INPUT AS 1
810       IF EOF(1) THEN 860
820          INPUT #1, R1$
830          PRINT     R1$
840       GOTO 810
850 REM
855 REM   AFSLUITING
857 REM
860       CLOSE 1
870       STOP
880 REM
890 REM   EOF FOUTMELDING
895 REM
900       PRINT "BEIDE BESTANDEN BIJ BEGIN MERGE LEEG"
910 REM
999       END
```

Toetsvraag

(a) Bepaal welke regels in bovenstaand programma overeenkomen met de stappen van de op p.160/161 genoemde procedure.

(a) 1. 260, 270
2. 280
3. 320
4. 330
5. 370
6. 380
7. 420, 430
8. 470, 540
9. a. 480, 490
   b. 560, 570
10. a. 610 t/m 640
    b. 670 t/m 700
11. 730
12. 760 t/m 860

5.6 PROGRAMMA VOOR STANDAARDBRIEVEN

Als laatste behandelen we een utility-programma om standaardbrieven te produceren. Daarbij komen enkele nieuwe bestandstechnieken ter sprake, en passeren enkele reeds behandelde technieke nogmaals de revue.

We nemen aan dat u vraag 6 van de toetsvragen aan het einde van het vorige hoofdstuk hebt gemaakt, en dus over drie standaardbrieven (opgeslagen in de bestanden BRIEF1, BRIEF2 en BRIEF3) beschikt. Bij het afdrukken van deze brieven zou het geen buitensporige luxe zijn een programma te hebben dat, aan de hand van een afzonderlijk adresbestand, naam en adres van de ontvanger boven aan de brief afdrukt, en de aanhef verzorgt.

Voor het adresbestand doen we alweer een beroep op een produkt van de toetsvragen aan het einde van hoofdstuk 4: het NAPW-bestand van vraag 5. De recordindeling daarbij was:

/1 naam 20/	21 adres 40/	41 postcode 47/	48 woonplaats 67/

De aanhef van de brief luidt:

Geachte heer/mevrouw .....

waarbij de stippellijnen de in het NAPW-record gevonden naam is.

Het programma drukt op het scherm de verschillende namen uit het NAPW-bestand een voor een af, en stelt bij elke naam de vraag welke standaardbrief (1, 2 of 3) naar die persoon moet worden verstuurd. Er worden tegelijkertijd twee van de in totaal vier bestanden gebruikt. Aan apparatuur zijn in principe twee schijfeenheden, een printer en een scherm nodig.

De algemene procedure is als volgt.

1. Open het adresbestand voor invoer.
2. Check op EOF voor het adresbestand; indien einde bestand, sluit alle bestanden en beëindig programma.
3. Lees eerste c.q. eerstvolgende adresrecord en druk naam op scherm af.
4. Geef gebruiker de mogelijkheid een brief te selecteren voor verzending aan deze persoon, of deze persoon over te slaan. Indien persoon wordt overgeslagen, ga naar stap 2.
5. Open het bestand waarin de geselecteerde brief is opgeslagen voor invoer.
6. Druk gegevens van geadresseerde af boven aan brief (printer).
7. Druk op printer aanhef af (inclusief naam geadresseerde).
8. Test briefbestand op EOF; bij einde bestand:
   a. sluit briefbestand;
   b. herhaal vanaf stap 2.
9. Lees eerste c.q. eerstvolgende briefrecord (dat wil zeggen een tekstregel) en druk af op de printer.
10. Herhaal vanaf stap 8.

Het eerste deel van het programma ziet er als volgt uit:

```
100 REM   PROGRAMMA VOOR MAKEN VAN STANDAARDBRIEF
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$    : NAAM, ADRES ETC.
140 REM    - R$    : TEKSTREGEL BRIEF
150 REM    - B$    : NAAM BRIEFBESTAND
160 REM    - D$    : DIALOOG
165 REM
170 REM   BESTANDEN (WAARVAN 2 TEGELIJKERTIJD OPEN)
180 REM   1. NAPW (INVOER)
190 REM   2. BRIEF1, BRIEF2, BRIEF3 (UITVOER)
185 REM   ALLE BESTANDEN: SEQUENTIEEL/FLOPPY
190 REM
200       CLEAR 1000
210 REM
215       MAXFILES = 2
220       OPEN "NAPW" FOR INPUT AS 1
230 REM
240 REM   TWEEDE BESTAND WORDT T.Z.T. GESELECTEERD EN GEOPEND
250 REM
260 REM   LEES NAAM, ADRES ETC.
265 REM
270       IF EOF(1) THEN 620
280       INPUT #1, N$
290 REM
```

Het programma kent in regel 280 de inhoud van het eerste NAPW-record toe aan variabele N$. Merk op dat de EOF-check vóór het lezen van het eerste record plaatsvindt.

Nu is de beurt aan u. Laat het programma de gevonden naam (posities 1 t/m 20 van het NAPW-record) op het scherm afdrukken, en de gebruiker vragen een brief te selecteren of de desbetreffende persoon over te slaan. Vul daartoe de ontbrekende regels in het volgende programmasegment in:

```
300 REM   GEEF NAAM EN SELECTEER BRIEF
305 REM
310       CLS

320       ..................... : REM *** DRUK NAAM AF EN SLA REGEL OVER     ***

330       ..................... : REM *** SELECTEER BRIEF                    ***

335       ..................... : REM *** TERUG NAAR 270 INDIEN GEEN BRIEF
                                                                    NODIG ***
340       ..................... : REM *** TERUG NAAR 330 INDIEN SELECTIE
                                                                  ONGELDIG ***
350 REM
```

```
(a) 320       PRINT LEFT$(N$,20) : PRINT ""
    330       LINE INPUT "WELKE BRIEF (1, 2, 3 OF RETURN): "; D$
    335       IF LEN(D$) = 0 THEN 270
    340       IF VAL(D$) < 1 OR VAL(D$) > 3 THEN
                 PRINT "INVOERFOUT. ALLEEN 1, 2, 3 OF RETURN"     : GOTO 330
```

Bestudeer vervolgens het volgende programmadeel voor het samenstellen van de naam van het briefbestand.

```
360 REM   INITIALISEER BRIEFBESTAND
365 REM
370       LET B$ = "BRIEF" + D$
380       OPEN B$ FOR INPUT AS 2
390 REM
```

Toetsvraag

(a) Stel dat de gebruiker naar aanleiding van regel 330 de waarde 2 invoert. Welk bestand wordt dan geopend?

(a) BRIEF2, dankzij stringconcatenatie in regel 370.

Geef vervolgens de ontbrekende regels in onderstaand programmasegment om naam, adres, etc. van de geadresseerde aan het begin van de brief op de printer af te drukken. Gebruik daarbij de LPRINT-opdracht.

```
400 REM   DRUK ADRESGEGEVENS AF OP BRIEF
405 REM
410       ..................... : REM *** 3 BLANKO REGELS MBV CHR$(10)      ***
420       ..................... : REM *** DRUK AF NAAM                      ***
430       ..................... : REM *** IDEM STRAAT EN HUISNR.            ***
440       ..................... : REM *** IDEM POSTCODE EN WOONPLAATS       ***
450 REM
```

(a)

```
410       FOR I = 1 TO 3 : LPRINT CHR$(10) : NEXT I   : REM *** 10 = LINE FEED ***
420       LPRINT LEFT$(N$,20)
430       LPRINT MID$(N$,21,20)
440       LPRINT MID$(N$,41,7); "  "; MID$(N$,48,20)
```

Het volgende deel verzorgt de aanhef:

```
460 REM   AANHEF AFDRUKKEN
465 REM
470       LPRINT CHR$(10) : LPRINT CHR$(10)
480       LPRINT "GEACHTE HEER/MEVROUW "; LEFT$(N$,20)
485       LPRINT : LPRINT
490 REM
```

Nu het afdrukken van de inhoud van de brief.*

```
500 REM   DRUK EIGENLIJKE BRIEF AF
510 REM
520       IF EOF(2) THEN 580
530          LINE INPUT #2, R$
540          LPRINT R$
550       GOTO 520
560 REM
570 REM   SLUIT BESTAND EN HERHAAL VOOR VOLGEND ADRES
575 REM
580       CLOSE 2
590       GOTO 270
600 REM
```

Toetsvragen

(a) Is de LINE INPUT in regel 530 essentieel, of kan met een gewone INPUT worden volstaan?

(b) Zonder te 'spieken': wat gebeurt er vanaf regel 270 (de opdracht waarnaar vanuit regel 590 wordt gesprongen)?

(a) LINE INPUT maakt het mogelijk komma's en aanhalingstekens in de tekst van de brief op te nemen. Bij INPUT is dat niet (zonder meer) mogelijk.

---

* Diepgravers mogen aannemen dat alle zogenaamde line-feeds en carriage returns in de tekstregels zijn opgenomen.

(b) Na een EOF-check op het invoerbestand wordt het volgende record van het NAPW-bestand gelezen en verwerkt.

Tenslotte de afsluiting van het programma:

```
610 REM   AFSLUITING ADRESBESTAND
615 REM
620       PRINT "EINDE PROGRAMMA"
630 REM
999       END
```

Overzicht

```
100 REM   PROGRAMMA VOOR MAKEN VAN STANDAARDBRIEF
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$    : NAAM, ADRES ETC.
140 REM    - R$    : TEKSTREGEL BRIEF
150 REM    - B$    : NAAM BRIEFBESTAND
160 REM    - D$    : DIALOOG
165 REM
170 REM   BESTANDEN (WAARVAN 2 TEGELIJKERTIJD OPEN)
180 REM   1. NAPW (INVOER)
190 REM   2. BRIEF1, BRIEF2, BRIEF3 (UITVOER)
185 REM   ALLE BESTANDEN: SEQUENTIEEL/FLOPPY
190 REM
200       CLEAR 1000
210 REM
215       MAXFILES = 2
220       OPEN "NAPW" FOR INPUT AS 1
230 REM
240 REM   TWEEDE BESTAND WORDT T.Z.T. GESELECTEERD EN GEOPEND
250 REM
260 REM   LEES NAAM, ADRES ETC.
265 REM
270       IF EOF(1) THEN 620
280       INPUT #1, N$
290 REM
300 REM   GEEF NAAM EN SELECTEER BRIEF
305 REM
310       CLS
320       PRINT LEFT$(N$,20) : PRINT ""
330       LINE INPUT "WELKE BRIEF (1, 2, 3 OF RETURN): "; D$
335       IF LEN(D$) = 0 THEN 270
340       IF VAL(D$) < 1 OR VAL(D$) > 3 THEN
             PRINT "INVOERFOUT. ALLEEN 1, 2, 3 OF RETURN"     : GOTO 330
350 REM
360 REM   INITIALISEER BRIEFBESTAND
365 REM
370       LET B$ = "BRIEF" + D$
380       OPEN B$ FOR INPUT AS 2
390 REM
400 REM   DRUK ADRESGEGEVENS AF OP BRIEF
405 REM
410       FOR I = 1 TO 3 : LPRINT CHR$(10) : NEXT I    : REM *** 10 = LINE FEED ***
420       LPRINT LEFT$(N$,20)
430       LPRINT MID$(N$,21,20)
440       LPRINT MID$(N$,41,7); "  "; MID$(N$,48,20)
450 REM
```

```
460 REM   AANHEF AFDRUKKEN
465 REM
470       LPRINT CHR$(10) : LPRINT CHR$(10)
480       LPRINT "GEACHTE HEER/MEVROUW "; LEFT$(N$,20)
485       LPRINT : LPRINT
490 REM
500 REM   DRUK EIGENLIJKE BRIEF AF
510 REM
520       IF EOF(2) THEN 580
530          LINE INPUT #2, R$
540          LPRINT R$
550       GOTO 520
560 REM
570 REM   SLUIT BESTAND EN HERHAAL VOOR VOLGEND ADRES
575 REM
580       CLOSE 2
590       GOTO 270
600 REM
610 REM   AFSLUITING ADRESBESTAND
615 REM
620       PRINT "EINDE PROGRAMMA"
630 REM
999       END
```

5.7 PROBLEMEN EN TIPS BIJ HET WERKEN MET SEQUENTIËLE BESTANDEN

Een gewaarschuwd mens telt voor twee. Met deze volkswijsheid in het achterhoofd laten we enkele fouten zien die bij het werken met sequentiële bestanden veel worden gemaakt. Daaraan koppelen we meteen enkele tips.

1. De meest voorkomende fout is dat verzuimd wordt de plaats van wijzers (pointers) bij te houden. In MSX BASIC, waarbij de computer bij het openen onderscheid maakt tussen in- en uitvoerbestanden, geven invoerbestanden meestal de grootste problemen. Het is daarom verstandig bij ieder gebruik van de 'INPUT #'-opdracht in een programma uzelf af te vragen welk effect dat heeft op de recordwijzer.

   Toetsvraag

   (a) Hoe kunt u de recordwijzer zo terugstellen dat hij naar het het begin van het bestand wijst?

   (a) Sluit het bestand met een CLOSE-opdracht. Bij heropenen geeft de wijzer het begin van het bestand aan.

2. Bij het sequentieel doorzoeken van een gegevensbestand voor een bepaald record of gegeven worden ook vaak fouten gemaakt. Stel u hebt een gegevensbestand van persoonsnamen alfabetisch geordend op achternaam. U geeft de naam op waarnaar gezocht moet worden. De computer vindt de naam en u geeft vervolgens een tweede naam op. Neemt u verder geen actie, dan begint het zoeken naar de tweede naam na het record waarin de eerste naam voorkwam. Is de tweede naam alfabetisch 'lager' dan de eerste, dan zal de computer hem niet kunnen vinden. De oplossing is uiteraard na iedere zoekactie de recordwijzer terug te stellen door middel van een CLOSE/OPEN, zoals in punt 1 reeds werd genoemd.

3. Problemen komen ook voor bij het gebruik van 'INPUT #'-opdrachten in een programmalus. Vaak zal zo'n opdracht niet in een lus kunnen worden opgenomen zonder dat de eerste keer een reeds ingelezen record wordt overschreven. Het kan daarom nodig zijn twee afzonderlijke 'INPUT #'-opdrachten – elk in aparte delen van het programma – te gebruiken (zie de toetsvraag naar aanleiding van het segment 450 t/m 470, programma MERGE).

4. Het kopiëren van de inhoud van arrays naar een gegevensbestand geeft ook wel eens aanleiding tot problemen. De volgende voorbeelden laten zien hoe die omzeild kunnen worden.

   (a) Inhoud van een één-dimensionaal array naar een gegevensbestand wegschrijven:

P(1)	1761
(2)	18
(3)	1942
(4)	24
(5)	8209
(6)	2

Fout :
```
PRINT #1, P
```

Goed :
```
FOR I = 1 TO 6
    PRINT #1, P(I)
NEXT I
```

   Opmerkingen:
   1. De FOR/NEXT-constructie is ook als een multi-opdracht te schrijven.
   2. Een puntkomma na de PRINT-opdracht kan mogelijk enige schrijfruimte besparen.
   3. Met deze methode maakt het niet uit of de gehele array of slechts een deel ervan wordt weggeschreven.

   (b) Inhoud meer-dimensionale array wegschrijven:

C	(1,1)	(1,2)	(1,3)
(1,1)	A	C	P
(2,1)	N	M	S
(3,1)	G	H	T
(4,1)	B	D	E

Fout :
```
PRINT #1, C
```

Goed :
```
FOR I = 1 TO 4
    FOR J = 1 TO 3
        PRINT #1, C(I,J)
    NEXT J
NEXT I
```

Opmerkingen:

1. In principe maakt het hierbij niet uit of de gehele array of slechts een deel daarvan wordt weggeschreven. Problemen kunnen wel ontstaan door vergissingen tussen rij- en kolomindices.
2. Een puntkomma na de PRINT-opdracht kan uiteraard ook hier mogelijk enige schijfruimte besparen.

5. Bij toepassingen waar in de loop van de tijd gegevens aan een bestand worden toegevoegd, met inachtname van een bepaalde volgcode, kan het nuttig zijn de eerst beschikbare volgcode als een afzonderlijk record aan het begin van het bestand op te nemen. In een klantenbestand zou in het eerste record bijvoorbeeld het eerste vrije klantnummer kunnen staan. In tegenstelling tot de overige records bevat dit beginrecord geen verdere gegevens.

   Bij het maken van nieuwe klantenrecords wordt dan de volgende procedure gevolgd:

   1. Lees het eerste bestandsgegeven (dus het eerste vrije klantnummer). Noem dat N.
   2. Ken N toe aan de eerste, nieuw in te voeren klant.
   3. Verhoog N met 1 (of eventueel 2, 5 of 10 om ruimte te laten voor tussenvoegingen. Noem deze waarde N1. De waarde N1 is dan het eerstvolgende beschikbare klantnummer.
   4. Open een tijdelijk bestand en schrijf hierin als eerste record de waarde van N1. Het record bevat verder geen andere gegevens.
   5. Kopieer de rest van het oorspronkelijke bestand (dat wil zeggen de bestaande, 'echte' klantenrecords) naar het tijdelijke bestand.
   6. Stel het record van de nieuwe klant samen, en schrijf ook dit naar het tijdelijke bestand. Het nieuwe record volgt dus na het laatste oorspronkelijke record.

7. Kopieer de inhoud van het tijdelijke bestand terug naar het oorspronkelijke bestand.
8. Herhaal vanaf stap 1 voor iedere nieuwe klant.

Deze veel voorkomende techniek maakt dus gebruik van een afzonderlijk gebied aan het begin van het bestand. De inhoud van dit gebied bevat een of meer gegevens die een programma nodig kan hebben bij het opereren op het bestand. De gegevens horen bij het bestand, en behoeven dankzij deze techniek niet los daarvan te worden bewaard. Uiteraard is het van groot belang dat er nauwkeurige documentatie is over opbouw en indeling van zo'n bestand.

5.8 TOETSVRAGEN

1. Schrijf een programma om het adresbestand NAPW (toetsvraag 5 aan het einde van hoofdstuk 4) te kopiëren.

2. a. Schrijf een programma dat een sequentieel bestand creëert waarvan de inhoud uit de namen van computertijdschriften bestaat. Gebruik het programma om de bij b. genoemde bestanden te maken, waarbij de titels in oplopende alfabetische volgorde worden opgenomen.

   b. Schrijf een programma om twee, in oplopende alfabetische volgorde geordende bestanden, samen te voegen. Het resulterende bestand dient op zijn beurt ook in alfabetische volgorde geordend te zijn. De oorspronkelijke bestanden bevatten de volgende gegevens:

bestand 1	bestand 2
BYTE Magazine	Creative Computing
Computer	DATAMATION
Databus	De microcomputer
MicroMix	ON Computing
Recreational Computing	Personal Computing

3. Schrijf een programma waarmee u uit te voeren huishoudklusjes in een bestand kunt opnemen. Het bestand moet met nieuwe gegevens kunnen worden uitgebreid, en bestaande gegevens (records) moeten kunnen worden verwijderd.

5.9 ANTWOORDEN OP DE TOETSVRAGEN

1.
```
100 REM   TOETSVRAAG 1, HOOFDSTUK 5
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$(20)    : NAAM
140 REM    - A$(20)    : ADRES (STRAAT + HUISNR.)
150 REM    - P$(7)     : POSTCODE
160 REM    - W$(20)    : WOONPLAATS
170 REM    - R$        : GEHELE RECORD
180 REM    - B$        : BESTANDSNAAM
190 REM
200 REM   BESTANDEN
210 REM   1. NAPW                    FLOPPY/SEQUENTIEEL
220 REM   2. GEBRUIKER-BEPAALD       IDEM
230 REM
240 REM   INITIALISATIES
245 REM
247       MAXFILES = 2
250       LINE INPUT "NAAM KOPIE-BESTAND: "; B$
260       OPEN "NAPW" FOR INPUT AS 1
270       OPEN B$ FOR OUTPUT AS 2
280 REM
290 REM   KOPIEER BESTAND
295 REM
300       IF EOF(1) THEN 350
310          INPUT #1, R$
320          PRINT #2, R$
330       GOTO 300
340 REM
350       CLOSE
360 REM
999       END
```

2. a.
```
100 REM   TOETSVRAAG 2A, HOOFDSTUK 5
110 REM
120 REM   VARIABELENLIJST
130 REM    - B$    : BESTANDNAAM
140 REM    - T$    : TIJDSCHRIFTTITEL
150 REM    - D$    : DIALOOG
160 REM
170 REM   BESTAND : NAAM DOOR GEBRUIKER TE BEPALEN,
180 REM             FLOPPY/SEQUENTIEEL
190 REM
200 REM   INITIALISATIES
205 REM
210       CLEAR 500
220       LINE INPUT "NAAM BESTAND: "; B$
230       OPEN B$ FOR OUTPUT AS 1
240 REM
250 REM   GEGEVENSINVOER
255 REM
260       PRINT "GEEF 'STOP' (ZONDER AANH. TEKENS) ALS U KLAAR BENT"
270       LINE INPUT "TITEL TIJDSCHRIFT: "; T$
280       IF T$ = "STOP" THEN 360
290       IF LEN(T$) = 0 THEN
             PRINT "GEEF TIJDSCHRIFTTITEL AUB"     : GOTO 260
300       IF LEN(T$) < 40 THEN
             LET T$ = T$ + " "                     : GOTO 300
```

```
305 REM
310 REM   WEGSCHRIJVEN EN HERHALEN
315 REM
320       PRINT #1, T$
330       CLS
340       GOTO 260
345 REM
350 REM   AFSLUITING
355 REM
360       CLOSE
370       PRINT "BESTAND GESLOTEN"
380 REM
999       END
```

2. b.

```
100 REM   TOETSVRAAG 2B; HOOFDSTUK 5
110 REM
120 REM   VARIABELENLIJST
130 REM    - N1$, N2$, N3$   : BESTANDNAMEN
140 REM    - R1$, R2$        : RECORDS VAN BESTAND 1 RESP. 2
150 REM    - D$              : DIALOOG
160 REM
170 REM   BESTANDEN
180 REM   1. TWEE SEQ. INVOERBESTANDEN (N1$, N2$)
190 REM   2. EEN  SEQ. UITVOERBESTAND  (N3$)
200 REM
210 REM   INITIALISATIES
220 REM
220       CLEAR 500
230       LINE INPUT "NAAM BESTAND 1: "; N1$
240       LINE INPUT "NAAM BESTAND 2: "; N2$
250       LINE INPUT "NAAM BESTAND 3: "; N3$
260 REM
265       MAXFILES = 3
270       OPEN N1$ FOR INPUT AS 1
280       OPEN N2$ FOR INPUT AS 2
290       OPEN N3$ FOR OUTPUT AS 3
295 REM
300       PRINT "SAMENVOEGEN BEGONNEN"
310 REM
320 REM   LEES RECORD BESTAND 1
325 REM
330       IF EOF(1) THEN 590
340       INPUT #1, R1$
350 REM
360 REM   LEES RECORD BESTAND 2
365 REM
370       IF EOF(2) THEN 640
380       INPUT #2, R2$
390 REM
400 REM   BEPAAL WEG TE SCHRIJVEN RECORDS
410 REM
420       IF R1$ = R2$ THEN 470
430       IF R1$ < R2$ THEN 470
440       GOTO 530
450 REM
```

(wordt vervolgd)

```
460 REM   SCHRIJF RECORD VAN BESTAND 1 NAAR 3 EN LEES NIEUW RECORD
465 REM
470       PRINT #3, R1$
480       IF EOF(1) THEN 590
490       INPUT #1, R1$
500       GOTO 410
510 REM
520 REM   IDEM,  BESTAND 2 NAAR 3
525 REM
530       PRINT #3, R2$
540       IF EOF(2) THEN 640
550       INPUT #2, R2$
560       GOTO 410
570 REM
580 REM   DUMP RESTANT 2 IN 3
585 REM
590       PRINT #3, R2$
600       IF EOF(2) THEN 700
605       INPUT #2, R2$
610       GOTO 590
620 REM
630 REM   DUMP RESTANT 1 IN 3
635 REM
640       PRINT #3, R1$
650       IF EOF(1) THEN 700
660       INPUT #1, R1$
670       GOTO 640
680 REM
690 REM   SLUIT BESTANDEN EN GEEF CONTROLE-OPTIE
695 REM
700       CLOSE 1, 2, 3
710       CLS
720       PRINT "SAMENVOEGEN KLAAR"
730       PRINT "" : PRINT ""
740       LINE INPUT "WILT U RESULTEREND BESTAND ZIEN? (J/N): "; D$
750       IF LEFT$(D$,1) = "N" THEN 850
760 REM
770 REM   DRUK BESTAND 3 AF
775 REM
780       OPEN N3$ FOR INPUT AS 1
790       IF EOF(1) THEN 840
800          INPUT #1, R1$
810          PRINT     R1$
820       GOTO 790
825 REM
830 REM   AFSLUITING
835 REM
840       CLOSE
850 REM
999       END
```

3.
```
100 REM   TOETSVRAAG 3, HOOFDSTUK 5
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$        : NAAM VAN BESTAND
140 REM    - D1$,D2$   : DIALOOG
150 REM    - K$        : KLUS
155 REM
160 REM   INITIALISATIES
165 REM
170       LINE INPUT "NAAM BESTAND: "; N$
175       MAXFILES = 2
180       OPEN N$ FOR INPUT AS 1
190       OPEN "HULP" FOR OUTPUT AS 2
200 REM
210       LINE INPUT "WILT U TOEVOEGEN OF VERWIJDEREN (T/V): "; D1$
220       IF D1$ <> "T" AND D1$ <> "V" THEN
             PRINT "INVOERFOUT. OPNIEUW AUB"     : GOTO 210
230       IF D1$ = "T" THEN 390
240 REM
250 REM   VERWIJDEREN
255 REM
260       PRINT "GEEF NA VERSCHIJNEN VAN BESTANDREGEL OP SCHERM "
270       PRINT "EEN 'V' (ZONDER AANH.TEKENS) OM TE VERWIJDEREN"
280 REM
290       IF EOF(1) THEN 500
300       INPUT #1, K$
310       PRINT     K$
320       LINE INPUT "GEEF 'V' OF RETURN: "; D2$
330       IF D2$ = "V" THEN 290
335       IF LEN(D2$) <> THEN PRINT "INVOERFOUT"     : GOTO 320
340 REM
350       PRINT #2, K$
360       GOTO 290
370 REM
380 REM   TOEVOEGEN
385 REM
390       IF EOF(1) THEN 440
400          INPUT #1, K$
410          PRINT #2, K$
420       GOTO 390
430 REM
440       LINE INPUT "GEEF NIEUWE KLUS OP OF 'STOP': "; K$
450       IF K$ = 'STOP' THEN 500
460       PRINT #2, K$
470       GOTO 440
480 REM
490 REM   KOPIEER HULPBESTAND TERUG
495 REM
500       CLOSE 1, 2
510 REM
520       OPEN "HULP" FOR INPUT AS 1
530       OPEN N$ FOR OUTPUT AS 2
540 REM
550       IF EOF(1) THEN 600
560          INPUT #1, K$
570          PRINT #2, K$
580       GOTO 550
590 REM
600       CLOSE
610 REM
999       END
```

6 GEGEVENSBESTANDEN OP CASSETTEBAND

6.0 DOELSTELLINGEN

De eerder besproken technieken voor sequentiële schijfbestanden leert u in dit hoofdstuk toepassen op cassettebestanden.

6.1 INLEIDING

Als u het voorgaande hoofdstuk met succes hebt doorgewerkt dan bent u nu in staat met gegevenbestanden op schijfgeheugen te werken. In dit hoofdstuk breiden we die kennis zo uit, dat u ook met bestanden op cassettes kunt omgaan.

Het gebruik van cassettes heeft nadelen. Cassettes zijn storinggevoelig en, in vergelijking met schijfgeheugens, tergend langzaam. In verband met dat laatste raden we aan zo mogelijk met een set van twee recorders te werken. Bent u van plan serieus en op grote schaal met gegevensbestanden te werken, dan verdient schijfapparatuur overigens zeker de voorkeur.

Ondanks deze reserves en de bekende gruwelverhalen over het per ongeluk wissen of verminken van cassettebanden, valt in het algemeen toch wel redelijk met dit medium te werken. Voorwaarde daartoe is een systematische en nauwkeurige werkwijze. Enkele potentiële problemen kunnen bovendien bij voorbaat worden voorkomen door gebruik te maken van speciale computer- in plaats van audiocassettes. Deze hebben geen aanloopstrook, zodat gegevens bij het beschrijven van het begin van de band niet verloren kunnen gaan.

Bedieningsfouten kunnen voor een belangrijk deel worden uitgesloten door instructies op het scherm af te drukken, zodat de gebruiker weet wanneer het apparaat moet worden gestart, wanneer de band moet worden teruggespoeld, enzovoort. Enkele voorbeelden:

```
LAAD ONBESCHREVEN CASSETTE
DRUK OP RECORD/PLAY
GEEF RETURN OM VERDER TE GAAN
STOP CASSETTE
SPOEL CASSETTE TERUG NAAR BEGIN
DRUK OP PLAY
```

Tegenwoordig komt gelukkig steeds meer apparatuur en programmatuur beschikbaar waarbij de fysieke besturing software-matig (vanuit het gebruikersprogramma) kan worden geregeld. Bedieningsfouten kunnen hierdoor gemakkelijker worden uitgesloten. We gaan ervan uit, dat u over zo'n datarecorder beschikt.

6.2 BESCHRIJVEN EN LEZEN VAN CASSETTEBANDEN

De BASIC-opdrachten voor het beschrijven en lezen van cassettebanden zien er in grote lijnen ongeveer hetzelfde uit als voor schijfgeheugens. Op sommige punten zijn ze zelfs eenvoudiger. Een voorbeeld van een leesopdracht is:

```
180  INPUT #1, A, B$, C$
```

en van een schrijfopdracht:

```
200  PRINT #1, A; B$; ","; C$
```

Bij het schrijven moeten we ook komma's tussen strings forceren, zoals bij schijfbestanden.

De bestanden worden op cassetteband geregistreerd onder de naam "CAS:naam". Geheugenbuffers worden op dezelfde wijze geopend en gesloten als bij schijfbestanden. Het openen van het bestand INVENT op cassetteband gaat bijvoorbeeld als volgt:

```
OPEN "CAS:INVENT" FOR OUTPUT AS 1
```

Ogenschijnlijk wordt ons programmeerleven er alleen maar eenvoudiger op. Er zijn echter wel de nodige regels waarmee rekening moet worden gehouden.

N.B. Denk eraan dat bij een INPUT #-opdracht de variabelen door een , gescheiden worden (INPUT #1,A,B,C) en bij een PRINT #-opdracht door een ; (PRINT #1,A;B;C).

1. Records worden op cassetteband onderling gescheiden door een zogenaamd inter-record gap, dat wil zeggen een stukje onbeschreven band. Gebruikt u drie verschillende 'PRINT #'-opdrachten waarbij steeds één variabele wordt weggeschreven, dan ontstaat op de band driemaal zo'n stukje loze ruimte. Dit pleit voor het wegschrijven van meer gegevens per record, zodat minder ruimte wordt verspild.

   voorbeeld:

   De constructie:

   ```
   PRINT #1, A
   PRINT #1, B
   PRINT #1, C
   ```

   vergt driemaal een inter-record gap.

   Bij de constructie

   ```
   PRINT #1, A; B; C
   ```

   ontstaat slechts eenmaal een inter-record gap.

2. Als de computer bij het uitvoeren van een INPUT-opdracht zijn gegevens zoekt voorbij de plaats waar ze op de cassetteband zijn vastgelegd, kan hij als het ware geblokkeerd raken. Ook de BREAK-toets heeft dan geen effect. Het kan dan nodig zijn het gehele systeem te herstarten, waarbij programma en de in het geheugen aanwezige gegevens verloren gaan. Een paardemiddel dus. Het verdient daarom aanbeveling cassettes voor of na gebruik steeds naar het begin terug te spoelen.

3. Programmeer één der functietoetsen voor het commando "motor" om met deze toets tijdens de uitvoering van een programma de moter van de datarecorder aan te kunnen zetten. Terugspoelen naar het begin van een cassetteband lukt anders niet zonder programmaonderbreking.

Een voorbeeld: inventarisprogramma

Als voorbeeld gebruiken we het in hoofdstuk 4 besproken inventarisprogramma (§ 4.5). Dit programma was gebaseerd op schijfbestand en zag er als volgt uit:

```
100 REM   PROGRAMMA VOOR HET AANMAKEN VAN EEN INVENTARISBESTAND
110 REM
120 REM   VARIABELEN
130 REM    - V$   : OMSCHRIJVING VOORWERP
140 REM    - N    : AANTAL STUKS
150 REM    - W    : WAARDE PER STUK
160 REM    - A$   : HULPVARIABELE T.B.V. DIALOOG
165 REM
170 REM   BESTANDEN
180 REM    - SEQUENTIEEL BESTAND OP FLOPPY
190 REM
200 REM   INITIALISEREN
205 REM
210       OPEN "INVENT" FOR OUTPUT AS 1
220 REM
230 REM   GEGEVENSINVOER
235 REM
240       LINE INPUT "OMSCHRIJVING (MAX. 20 TEKENS): "; V$
250       IF LEN(V$) > 20 THEN
             PRINT "AFKORTEN TOT MAX. 20 TEKENS AUB" :          GOTO 240
260       IF LEN(V$) = 0  THEN
             PRINT "OMSCHRIJVING OPGEVEN AUB" :                 GOTO 240
265       IF LEN(V$) < 20 THEN V$ = V$ + " "  :                GOTO 265
270       INPUT      "AANTAL STUKS                 : "; N
280       IF N <> INT(N) THEN
             PRINT "ALLEEN GEHELE GETALLEN AUB" :               GOTO 270
290       IF N <= 0 THEN
             PRINT "AANTAL OPGEVEN AUB"                         GOTO 270
300       INPUT      "GELDWAARDE PER STUK          : "; W
310       IF W <= 0 THEN 350
315 REM
316 REM   WEGSCHRIJVEN NAAR BESTAND EN HERHALEN
317 REM
320       PRINT #1, V$; ","; N; W
325       LINE INPUT "NOG MEER GEGEVENS (J/N): "; A$
326       IF LEFT$(A$,1) = "N" THEN 410
330       GOTO 240
340 REM
341 REM   CHECK OP WAARDE
342 REM
350       LINE INPUT "WAARDE <= 0. OK? (J/N)       : "; A$
360       IF LEFT$(A$,1) = "N" THEN
             PRINT "DAN OPNIEUW INVOEREN AUB" :                 GOTO 300
370       GOTO 320
380 REM
400 REM   BESTAND AFSLUITEN EN STOPPEN
405 REM
410       CLOSE 1
420 REM
999       END
```

Vraag

(a) Geef aan welke aanpassingen nodig zijn om dit programma geschikt te maken voor een cassette- in plaats van een schijfbestand.

(a) Regel 180, commentaar aanpassen.
210 OPEN "CAS:INVENT" FOR OUTPUT AS 1

Behalve deze aanpassingen zou een goed opgezet programma bovendien de volgende aanwijzingen bevatten:

```
200 REM   INITIALISEREN
205 REM
207       PRINT "PLAATS CASSETTE IN RECORDER"
208       PRINT "DRUK RECORD/PLAY IN"

430       PRINT "STOP RECORDER EN SPOEL TERUG"
440 REM
```

Opdracht

Draai het resulterende programma, en maak daarbij een cassettebestand met enkele inventarisgegevens.

We schrijven vervolgens een programma om het cassettebestand, dat door bovenstaand programma wordt geproduceerd, te lezen en daarvan een overzicht te maken. U wordt gevraagd de ontbrekende regels te leveren.

(a)

```
100 REM   OVERZICHT MAKEN VAN INVENTARISBESTAND
110 REM
120 REM   VARIABELEN
130 REM    - V$  : OMSCHRIJVING VOORWERP
140 REM    - N   : AANTAL STUKS
150 REM    - W   : WAARDE PER STUK
155 REM    - X$  : DIALOOG
160 REM
170 REM   BESTANDEN
180 REM    - INVENT  : SEQUENTIEEL (CASSETTE)
190 REM
200 REM   INITIALISEREN
205 REM
207       ..............................  : REM *** GEEF AANWIJZING DAT GEBRUI-
                                                KER RECORDER MOET STARTEN ***
208       ..............................  : REM *** LAAT GEBRUIKER ENTER GEVEN
                                                OM PROGRAMMA VOORT TE
                                                ZETTEN                    ***
210       OPEN "CAS:INVENT" FOR INPUT AS 1
220 REM
230 REM
240 REM   DRUK KOP AF VAN OVERZICHT
245 REM
250       PRINT "OMSCHRIJVING"; TAB(22); "AANTAL"; TAB(30); "WAARDE"
255       PRINT
260 REM
```

```
(a) 207       PRINT "LAAD CASSETTE EN SPOEL ZONODIG TERUG"
    208       INPUT "GEEF RETURN VOOR VOORTZETTING PROGRAMMA"; X$
```

Nu nog het volgende stukje:

```
(a)   270 REM   GEGEVENS INLEZEN VAN BESTAND EN AFDRUKKEN
      275 REM
      280       ..............................  : REM *** LEES RECORD ***

      290       ..............................  : REM *** DRUK REGEL AF ***

      300       ..............................  : REM *** SPRING TERUG VOOR
                                                          VOLGENDE RECORD ***
      310 REM
--------------------
(a)   280       INPUT #1, V$, N, V
      290       PRINT     V$, TAB(22), N, V
      300       GOTO 280
```

Het volledige programma ziet er als volgt uit:

```
100 REM   OVERZICHT MAKEN VAN INVENTARISBESTAND
110 REM
120 REM   VARIABELEN
130 REM    - V$   : OMSCHRIJVING VOORWERP
140 REM    - N    : AANTAL STUKS
150 REM    - W    : WAARDE PER STUK
155 REM    - X$   : DIALOOG
160 REM
170 REM   BESTANDEN
180 REM    - INVENT  : SEQUENTIEEL (CASSETTE)
190 REM
200 REM   INITIALISEREN
205 REM
207       PRINT "LAAD CASSETTE EN SPOEL ZONODIG TERUG"
208       INPUT "GEEF RETURN VOOR VOORTZETTING PROGRAMMA "; X$
210       OPEN "CAS: INVENT" FOR INPUT AS 1
220 REM
230 REM
240 REM   DRUK KOP AF VAN OVERZICHT
245 REM
250       PRINT "OMSCHRIJVING"; TAB(22); "AANTAL"; TAB(30); "WAARDE"
255       PRINT
260 REM
270 REM   GEGEVENS INLEZEN VAN BESTAND EN AFDRUKKEN
275 REM
280       INPUT #1, V$, N, W
290       PRINT     V$, TAB(22), N, W
300       GOTO 280
310 REM
999       END
```

End-of-file (EOF) bij cassettes

Uitvoering van het bovenstaande programma zou een schoonheidsfout aan het licht brengen. Er is namelijk geen enkele controle op het bereiken van het einde van het bestand, de zogenaamde end-of-file toestand. Het programma zou dan ook eindigen met de melding "INPUT PAST END".

Ook bij het lezen van cassettes kan de EOF-functie, die we eerder bij schijfbestanden zijn tegengekomen, gebruikt worden. We kunnen daarom voor regel 300 opnemen:

```
300       IF NOT EOF(1) THEN 280
```

We kunnen ook zelf het einde van het bestand aangeven. Daarvoor kunnen we een bijzondere gegevenswaarde kiezen die buiten het normale 'waardenbereik' van het desbetreffende gegeven valt, bijvoorbeeld STOP, *, 999 of -1. Door met een IF...THEN op de aanwezigheid van zo'n gegeven te controleren kunnen we eenvoudig constateren of het einde van het bestand is bereikt. Vergeet echter niet deze *afsluitwaarde* te documenteren.

Toetsvragen

(a) Pas het programma van p.185 nogmaals aan, en wel zodanig dat sterretjes (*) en nullen als afsluitwaarden aan het einde van het bestand worden opgenomen.

(b) Hoeveel afsluitwaarden zijn nodig? Waarom?

(c) Pas het bovenstaande programma voor het maken van een overzicht van het inventarisbestand zo aan, dat er op de in (a) bedoelde afsluitwaarden wordt getest. Indien afsluitwaarden gevonden worden dient een sprong uitgevoerd te worden naar de volgende opdrachten:

```
400       PRINT "TAAK BEEINDIGD"
999       END
```

(a) `410 PRINT #1, "*"; 0; 0`
(Verander het regelnummer van de CLOSE-opdracht (410) in 415.)

(b) Drie, te weten één sterretje (voor het alfanumerieke gegeven V$) en twee nullen (voor de numerieke gegevens N en W). Gebruikt u alleen een sterretje dan ontstaat bij uitvoering van de INPUT-opdracht in regel 270 een invoerfout (INPUT ERROR).

(c) `285 IF V$ = "*" THEN 400`

Wissen van cassetteband

Om oude gegevens van cassetteband te verwijderen verdient het aanbeveling gebruik te maken van een speciaal apparaat, een zogenaamde *bulk eraser*. De ingebouwde wiskop van de gewone recorder voldoet misschien uitstekend voor audio-toepassingen, maar voor computergebruik is hij niet betrouwbaar. Overschrijft u een oud bestand met nieuwe gegevens, dan kunnen daardoor problemen ontstaan. Dit pleit tevens voor het opslaan van slechts één bestand per cassette. Afgezien van het voordeel dat u de band niet hoeft af te zoeken voor een bepaald bestand, kunt u bovendien de gehele band in één keer wissen als het bestand niet meer nodig is.

Een of twee cassettes?

Programmeren met cassettebanden wordt eenvoudiger als u de beschikking hebt over twee recorders. De utility-programma's van hoofdstuk 5 laten zich dan gemakkelijk aanpassen. Met slechts één recorder gaat het moeilijker. Als 'tijdelijk bestand' zult u dan gebruik moeten maken van een array in het interne geheugen van de computer. U simuleert dus als het ware het tijdelijke bestand in het interne geheugen. Daarbij loopt u het risico dat grotere bestanden wegens tekort aan geheugenruimte niet verwerkbaar zijn. Het interne geheugen is immers betrekkelijk klein. Het is zaak hierop te letten, en grotere bestanden zoveel mogelijk te vermijden of eventueel in kleinere op te splitsen.

6.3 KOPIËREN MET ÉÉN CASSETTE

Als voorbeeld van het werken met slechts één cassetterecorder behandelen we in deze paragraaf het kopiëren van een cassettebestand. We beginnen met een programma om het te kopiëren bestand aan te maken. De specificaties zijn als volgt.

1. De gegevens zijn van statistische aard (bijvoorbeeld leeftijd, snelheid waarmee gereden is, enz.) en bestaan uit numerieke waarden van 1 t/m 100. Controleer deze gegevens bij invoer.
2. De ingevoerde waarden worden stuk voor stuk met één 'PRINT #'-opdracht naar het bestand weggeschreven (één variabele).
3. Minimaal 20 waarden, maximaal 400.
4. Gebruik -999 als afsluitwaarde.

Vraag

(a) Schrijf overeenkomstig bovengenoemde specificaties een programma voor het aanmaken van een bestand. Voer het programma uit.

(a)
```
100 REM   AANMAKEN BESTAND
110 REM
120 REM   BESTAND: STAT1 (CASSETTE)
130 REM
140 REM   INITIALISATIE
150 REM
160       OPEN "CAS:STAT1" FOR OUTPUT AS 1
170 REM
200 REM   GEGEVENS INVOEREN EN WEGSCHRIJVEN
205 REM
210       INPUT "GEEF WAARDE OF -999: "; W
220       IF W = -999 THEN 299
230       IF W < 1  OR  W > 100 THEN
             PRINT "INVOERFOUT. OPNIEUW AUB" : GOTO 210
```

```
240       PRINT #1, W
250       GOTO 210
298 REM
299       PRINT #1, -999
300       CLOSE: END
```

Het kopiëren

Vervolgens schrijven we een kopieerprogramma gebaseerd op het gebruik van één recorder. Hierbij kan de array worden opgevat als een tijdelijk bestand. De grootte van de array (die we A zullen noemen) hangt af van het aantal waarden. Aangezien er een maximum van 400 waarden te verwachten is kunnen we de array dus declareren met DIM A(400). We kunnen ook uitgaan van een op te geven maximum (X) en een variabele arraydeclaratie gebruiken: DIM A(X).

Verder is het van belang te weten hoe de gegevens naar het oorspronkelijke bestand werden weggeschreven. De opdracht daarvoor was:

```
PRINT #1, W
```

Met deze zaken in gedachten kunnen we gaan programmeren:

```
100 REM   KOPIEREN
110 REM
120 REM   VARIABELENLIJST
130 REM    - W   : WAARDE UIT BESTAND
140 REM    - T   : TELLER
150 REM    - A   : ARRAY
160 REM    - X   : ARRAY DIMENSIE (VARIABEL)
165 REM    - R$  : RESPONS
170 REM
180 REM   BESTANDEN
190 REM    - STAT1  : OORSPR. CASSETTEBESTAND
200 REM    - STAT1A : KOPIE
210 REM      AFSLUITWAARDE -999 (NUMERIEK)
220 REM
230 REM   INITIALISATIES
235 REM
240       CLS
250       PRINT "PLAATS STAT1 BANDJE IN RECORDER"
260       PRINT "SPOEL TERUG EN DRUK OP PLAY"
270       INPUT "GEEF RETURN OM VERDER TE GAAN "; R$
275       OPEN "CAS:STAT1" FOR INPUT AS 1
280 REM
290       INPUT "MAX. AANTAL WAARDEN "; X
300       DIM A(X)
310       LET T = 0
320 REM
```

Tot zover het inleidende deel. We kunnen nu de waarden van het bestand inlezen en opslaan in array A. Het volgnummer van de ingelezen waarden houden we bij met de teller T. Maak het volgende programmasegment zelf af.

```
(a)   330 REM   ARRAY VULLEN MET BESTANDINHOUD
      335 REM
      340       ..............................  : REM *** LEES WAARDE UIT BESTAND ***
      350       ..............................  : REM *** AFSLUITWAARDE? GOTO 410 ***
      360       ..............................  : REM *** HOOG TELLER OP ***
      370       ..............................  : REM *** WAARDE --> ARRAY ***
      380       GOTO 340
      390 REM
--------------------------
(a)   340       INPUT #1, W
      350       IF W = -999 THEN 410
      360       LET T = T + 1
      370       LET A(T) = W
```

Wellicht hebt u in regel 340 in plaats van W het array-element in de INPUT-opdracht opgenomen, dus: INPUT #1, A(T). Dit is mogelijk (hoewel het eerste gegeven zich dan in A(0) in plaats van A(1) zou bevinden). Zoals we bij volgende toepassingen zullen zien is het echter beter een invoerwaarde niet rechtstreeks aan een array-element toe te kennen.

Toetsvraag

(a) Indien bij uitvoering van regel 350 een afsluitwaarde wordt geconstateerd en er dus naar regel 410 wordt gesprongen, welke handelingen moeten daar dan worden uitgevoerd?

(a) Wisselen van cassettebanden, en kopiëren van array naar de nieuwe band.

In het volgende deel programmeren we de in (a) bedoelde handelingen. Geef zelf de ontbrekende regel (490) en beantwoord vraag (b).

```
(a)   400 REM   INITIALISEER KOPIE
      405 REM
      410       CLS
      415       CLOSE
      420       PRINT "SPOEL STAT1 BANDJE TERUG"
      430       PRINT "PLAATS NIEUW BANDJE IN RECORDER"
      440       PRINT "DRUK OP RECORD/PLAY"
      450       INPUT "GEEF RETURN OM VERDER TE GAAN "; R$
      455       OPEN "CAS:STAT1A" FOR OUTPUT AS 1
      460 REM
      470 REM   KOPIEER ARRAY NAAR CASSETTE
      475 REM
      480       FOR I = 1 TO T
      490       ..........................  : REM *** SCHRIJF WAARDE NAAR CASSETTE ***
      500       NEXT I
      510 REM
```

(b) Waarvoor is de teller T nodig?

(a) `490 PRINT #1, A(I)`

(b) Bij het kopiëren van de array naar de cassette moet bekend zijn hoeveel array-elementen geldige gegevens bevatten.

We sluiten het programma met het schrijven van de afsluitwaarde.

```
520 REM   AFSLUITWAARDE PLAATSEN
525 REM
530       PRINT #1, -999
540       PRINT "EINDE PROGRAMMA"
550 REM
560       CLOSE
999       END
```

Het volledige programma voor het kopiëren van een cassettebestand ziet er als volgt uit:

```
100 REM   KOPIEREN
110 REM
120 REM   VARIABELENLIJST
130 REM    - W   : WAARDE UIT BESTAND
140 REM    - T   : TELLER
150 REM    - A   : ARRAY
160 REM    - X   : ARRAY DIMENSIE (VARIABEL)
165 REM    - R$  : RESPONS
170 REM
180 REM   BESTANDEN
190 REM    - STAT1  : OORSPR. CASSETTEBESTAND
200 REM    - STAT1A : KOPIE
210 REM      AFSLUITWAARDE -999 (NUMERIEK)
220 REM
230 REM   INITIALISATIES
235 REM
240       CLS
250       PRINT "PLAATS STAT1 BANDJE IN RECORDER"
260       PRINT "SPOEL TERUG EN DRUK OP PLAY"
270       INPUT "GEEF RETURN OM VERDER TE GAAN "; R$
275       OPEN "CAS:STAT1" FOR INPUT AS 1
280 REM
290       INPUT "MAX. AANTAL WAARDEN "; X
300       DIM A(X)
310       LET T = 0
320 REM
330 REM   ARRAY VULLEN MET BESTANDINHOUD
335 REM
340       INPUT #1, W
350       IF W = -999 THEN 410
360       LET T = T + 1
370       LET A(T) = W
380       GOTO 340
390 REM
400 REM   INITIALISEER KOPIE
405 REM
410       CLS
415       CLOSE
420       PRINT "SPOEL STAT1 BANDJE TERUG"
430       PRINT "PLAATS NIEUW BANDJE IN RECORDER"
440       PRINT "DRUK OP RECORD/PLAY"
450       INPUT "GEEF RETURN OM VERDER TE GAAN "; R$
455       OPEN "CAS:STAT1A" FOR OUTPUT AS 1
460       REM
```

```
470 REM   KOPIEER ARRAY NAAR CASSETTE
475 REM
480       FOR I = 1 TO T
490           PRINT #1, A(I)
500       NEXT I
510 REM
520 REM   AFSLUITWAARDE PLAATSEN
525 REM
530       PRINT #1, -999
540       PRINT "EINDE PROGRAMMA"
550 REM
560       CLOSE
999       END
```

6.4 UITBREIDEN VAN EEN CASSETTEBESTAND

Voor het uitbreiden van een cassettebestand kunnen we dankbaar gebruik maken van een flink deel van het kopieerprogramma van de vorige paragraaf. Althans als we (nog steeds) uitgaan van het gebruik van één recorder. Ook in dit geval moet de inhoud van de cassetteband eerst in een array worden opgeslagen en vervolgens – maar dan zonder afsluitwaarde – naar een nieuw bandje worden geschreven. (Eventueel kan ook het oorspronkelijke bandje worden gebruikt, hoewel we dan geen backup-exemplaar hebben als er iets mis mocht gaan). We kunnen dus de regels 100 (met aangepaste titel) t/m 510 zonder meer van het bovenstaande kopieerprogramma overnemen.

Toetsvragen

(a) Wat moet er aansluitend op regel 510 gebeuren?

(b) Maak in overeenstemming daarmee het volgende programmasegment af.

```
520 REM   INVOER NIEUWE GEGEVENS (-999 = STOP)
530 REM   EN SCHRIJF NAAR BESTAND
535 REM
540       ..... : REM *** WAARDE INLEZEN --> W ***

550       ..... : REM *** WEGSCHRIJVEN --> BESTAND ***

560       ..... : REM *** W <> - 999? GOTO 540 ***

570 REM
999       END
```

(a) Invoeren van aanvullende gegevens en toevoegen aan bestand.

(b)
```
540       INPUT "GEGEVEN: "; W
550       PRINT #1, W
560       IF W <> -999 THEN 540
```

N.B.: Voor het gemak hebben we hier afgezien van invoercontrole.

Array-dimensionering

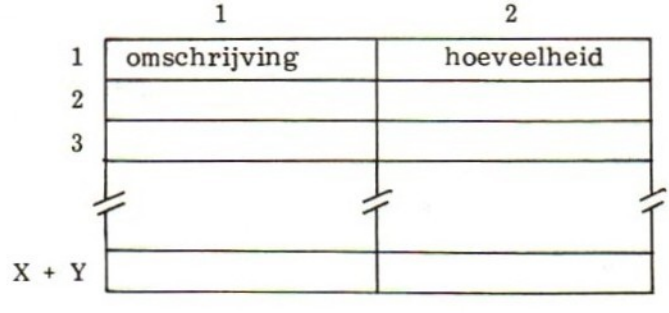
Bij het gebruik van een array als tijdelijk 'bestand' is het uiteraard van belang het aantal gegevens in dat bestand te kennen. De array kan dan in overeenstemming daarmee worden gedimensioneerd. In plaats van een schatting (zoals we tot nu toe hebben gedaan) kunnen we ook het aantal bijhouden en als eerste gegeven in het bestand opnemen.

De procedure voor het uitbreiden van een cassettebestand wordt dan:

1. Lees het eerste gegeven (zijnde het aantal) in, en ken dit toe aan bijvoorbeeld X.
2. Bepaal via dialoog hoeveel records moeten worden toegevoegd (Y).
3. Dimensioneer de array met X + Y.
4. Kopieer het bestand naar de array.
5. Lees de toe te voegen gegevens in (opslaan in array, aantal bijhouden).
6. Schrijf totaal aantal gegevens (oud + nieuw) als eerste gegeven naar de cassette.
7. Kopieer de gehele array naar cassette.

Voorbeeldprogramma: boodschappenlijst (2)

Als voorbeeld van de zojuist genoemde procedure passen we het boodschappenprogramma van hoofdstuk 5, § 5.2 aan voor cassettebestanden. Het eerste gegeven in het bestand geeft daarbij het aantal records in het bestand aan. De overige records bevatten een artikelomschrijving en de hoeveelheid die van dat artikel moet worden aangeschaft. We gebruiken een twee-dimensionale array (zie figuur 6-1).



figuur 6-1 Array-indeling.

In verband daarmee werken we (in afwijking van § 5.2) met een alfanumerieke variabele voor de hoeveelheid.

```
100 REM   UITBREIDEN VAN CASSETTE BOODSCHAPPENBESTAND
110 REM
120 REM   VARIABELENLIJST
130 REM    - X   : TELLER (AANTAL RECORDS)
140 REM    - Y   : TELLER (AAN TOEGEVOEGDE RECORDS)
150 REM    - A$  : OMSCHRIJVING ARTIKEL
160 REM    - H$  : HOEVEELHEID
170 REM    - R$  : ARRAY VOOR TIJDELIJKE OPSLAG RECORDS
180 REM    - D$  : DIALOOG
190 REM
200 REM   BESTANDEN
210 REM    - BOODSCHAP (CASSETTE)
220 REM
230 REM   INITIALISATIES
235 REM
240       PRINT "PLAATS BOODSCHAPCASSETTE IN RECORDER"
250       PRINT "DRUK OP PLAY EN GEEF DIRECT DAARNA"
260       INPUT "RETURN OM VERDER TE GAAN"; D$
265       OPEN "CAS:BOODSCH" FOR INPUT AS 1
270       INPUT #1, X
280       INPUT "AANTAL TOEVOEGINGEN: "; Y
290       DIM R$(X+Y, 2)
300 REM
```

Toetsvragen

(a) Wat is het doel van regel 270?

(b) Verklaar regel 290.

(c) Maak het volgende programmasegment af:

```
310 REM   KOPIEER BESTAND NAAR ARRAY
315 REM
320       FOR I = 1 TO X
330           ..... : REM *** LEES RECORD VAN CASSETTE ***
340       NEXT I
350 REM
```

(d) Bestudeer de FOR/NEXT-lus in het volgende programmasegment en geef de opdrachten voor de regels 430 en 440. Regel 370 bepaalt hoeveel nieuwe records worden toegevoegd. Opdracht 380 houdt het eerstvolgende vrije element in array R$ bij.

```
360 REM   GEGEVENSINVOER
365 REM
370       FOR I = 1 TO Y
380           LET X = X + 1
390           LINE INPUT "ARTIKELOMSCHRIJVING: "; A$
400           LINE INPUT "HOEVEELHEID:         "; H$
410 REM
420 REM       VUL ARRAY
425 REM
430           ..... : REM *** PLAATS A$ IN ARRAY ***

440           ..... : REM *** PLAATS H$ IN ARRAY ***

450       NEXT I
455       CLOSE
460 REM
```

(a) Inlezen van eerste gegeven in bestand (= aantal records).

(b) Dimensioneert array R$ als X+Y (= aantal reeds aanwezige plus aantal toe te voegen records) bij 2, op te vatten als X+Y rijen van 2 kolommen.

(c)
```
330       INPUT #1, R$(I,1), R$(I,2)
```

(d)
```
430       LET R$(X,1) = A$
440       LET R$(X,2) = H$
```

Tenslotte kopiëren we de inhoud van de array naar cassette. Maak zelf de ontbrekende regels in het volgende segment af.

(a)
```
470 REM   AANWIJZINGEN RECORDER
475 REM
480       CLS
490       PRINT "PLAATS NIEUW BANDJE OF EVENTUEEL"
500       PRINT "OORSPRONKELIJK BANDJE (TERUGSPOELEN!) IN RECORDER"
510       INPUT "GEEF RETURN OM VERDER TE GAAN"; D$
515       OPEN "CAS:BOODSCH" FOR OUTPUT AS 1
520 REM
530 REM   SCHRIJF AANTAL RECORDS NAAR CASSETTE
535 REM
540       ..............................

550 REM
560 REM   SCHRIJF ARRAYINHOUD NAAR CASSETTE
565 REM
570       FOR I = 1 TO X

580           ..............................

590       NEXT I
600 REM
610       PRINT "EINDE PROGRAMMA"
620 REM
```

(a)
```
540       PRINT #1, X
580       PRINT #1, R$(I,1); ","; R$(I,2)
```

Het volledige programma:

```
100 REM   UITBREIDEN VAN CASSETTE BOODSCHAPPENBESTAND
110 REM
120 REM   VARIABELENLIJST
130 REM    - X   : TELLER (AANTAL RECORDS)
140 REM    - Y   : TELLER (AAN TOEGEVOEGDE RECORDS)
150 REM    - A$  : OMSCHRIJVING ARTIKEL
160 REM    - H$  : HOEVEELHEID
170 REM    - R$  : ARRAY VOOR TIJDELIJKE OPSLAG RECORDS
180 REM    - D$  : DIALOOG
190 REM
200 REM   BESTANDEN
210 REM    - BOODSCHAP (CASSETTE)
220 REM
```

```
230 REM   INITIALISATIES
235 REM
240       PRINT "PLAATS BOODSCHAPCASSETTE IN RECORDER"
250       PRINT "DRUK OP PLAY EN GEEF DIRECT DAARNA"
260       INPUT "RETURN OM VERDER TE GAAN"; D$
265       OPEN "CAS:BOODSCH" FOR INPUT AS 1
270       INPUT #1, X
280       INPUT "AANTAL TOEVOEGINGEN: "; Y
290       DIM R$(X+Y, 2)
300 REM
310 REM   KOPIEER BESTAND NAAR ARRAY
315 REM
320       FOR I = 1 TO X
330           INPUT #1, R$(I,1), R$(I,2)
340       NEXT I
350 REM
360 REM GEGEVENSINVOER
365 REM
370       FOR I = 1 TO Y
380           LET X = X + 1
390           LINE INPUT "ARTIKELOMSCHRIJVING: "; A$
400           LINE INPUT "HOEVEELHEID        : "; H$
410 REM
420 REM       VUL ARRAY
425 REM
430           LET R$(X,1) = A$
440           LET R$(X,2) = H$
450       NEXT I
455       CLOSE
460 REM
470 REM   AANWIJZINGEN RECORDER
475 REM
480       CLS
490       PRINT "PLAATS NIEUW BANDJE OF EVENTUEEL"
500       PRINT "OORSPRONKELIJK BANDJE (TERUGSPOELEN!) IN RECORDER"
510       INPUT "GEEF RETURN OM VERDER TE GAAN"; D$
515       OPEN "CAS:BOODSCH" FOR OUTPUT AS 1
520 REM
530 REM   SCHRIJF AANTAL RECORDS NAAR CASSETTE
535 REM
540       PRINT #1, X
550 REM
560 REM   SCHRIJF ARRAYINHOUD NAAR CASSETTE
565 REM
570       FOR I = 1 TO X
580           PRINT #1, RS(I,1); ","; R$(I,2)
590       NEXT I
600 REM
610       PRINT "EINDE PROGRAMMA"
620 REM
630       CLOSE
640 REM
999       END
```

U beschikt nu over een redelijke basiskennis voor het werken met cassettebestanden. Programma's in voorgaande hoofdstukken die op schijfbestanden waren gebaseerd kunt u nu zelf aanpassen aan het gebruik van cassettebestanden. De clou is steeds een of meer arrays te laten functioneren als tijdelijk bestand. De grootte van het interne geheugen is daarbij in wezen de enige beperkende factor.

6.5 TOETSVRAGEN

1(a) Schrijf een programma om records in een cassettebestand op te nemen. Ieder record bevat twee alfanumerieke gegevens, gevolgd door twee numerieke. Laat na invoer van ieder record de vraag "MEER GEGEVENS?" op het scherm verschijnen en neem een afsluitwaarde op als laatste record.

```
100 REM   OPGAVE 1A, HOOFDSTUK 6
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$, B$       : GEGEVEN 1 REXP. 2 (ALFANUMERIEK)
140 REM    - C$/C, D$/D   : GEGEVEN 3 RESP. 4 (NUMERIEK)
145 REM    - R$           : RESPONS
150 REM
```

(b) Schrijf een begeleidend programma voor vraag 1(a) om de inhoud van het cassettebestand te laten zien. Test daarbij op het bereiken van het bestandeinde.

```
100 REM   OPGAVE 1B, HOOFDSTUK 6
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$, B$   : GEGEVEN 1 RESP. 2 (ALFANUMERIEK)
140 REM    - C, D     : GEGEVEN 3 RESP. 4 (NUMERIEK)
150 REM
```

2(a) Schrijf een boodschappenprogramma waarmee de volgende gegevens in een cassettebestand kunnen worden opgenomen:
- artikelomschrijving (maximaal 20 tekens)
- hoeveelheid

Bij de gegevensinvoercontrole dient nagegaan te worden of de ingevoerde hoeveelheid tussen 1 en 10 ligt. Valt de hoeveelheid buiten dit bereik, dan moet de gebruiker een andere hoeveelheid kunnen invoeren.

```
100 REM   OPGAVE 2A, HOOFDSTUK 6
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$   : ARTIKELOMSCHRIJVING (MAX. 20 TEKENS)
140 REM    - H    : HOEVEELHEID/AANTAL
150 REM    - D$   : DIALOOG
160 REM
```

(b) Schrijf een begeleidend programma voor vraag 2(a) waarmee de inhoud van het boodschappenbestand zichtbaar kan worden gemaakt.

```
100 REM   OPGAVE 2B, HOOFDSTUK 6
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$   : ARTIKELOMSCHRIJVING
140 REM    - H    : HOEVEELHEID
150 REM
```

3(a) Schrijf een programma om op cassette een adresbestand op te nemen zoals aangegeven in onderstaande variabelenlijst. Voeg daarbij de verschillende gegevens samen tot één record per persoon.

```
100 REM   OPGAVE 3A, HOOFDSTUK 6
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$(20)  : NAAM
140 REM    - A$(20)  : ADRES (STRAAT + HUISNR.)
150 REM    - P$(7)   : POSTCODE
160 REM    - W$(20)  : WOONPLAATS
170 REM    - R$(67)  : GEHELE RECORD
180 REM    - D$      : DIALOOG
190 REM
```

(b) Schrijf een begeleidend programma bij vraag 3(a) om de records stuk voor stuk op het scherm op te roepen. Houd tijdens dit proces het aantal records bij, en druk na het laatste record het totaal af.

```
100 REM   OPGAVE 3B, HOOFDSTUK 6
110 REM
120 REM   VARIABELENLIJST
130 REM    - R$  : EEN VOLLEDIG RECORD (67 TEKENS)
140 REM    - T   : TELLER VOOR AANTAL RECORDS
150 REM
```

(c) Schrijf een begeleidend programma bij vraag 3(a) om een kopie van het adresbestand op een afzonderlijke cassette te maken. Denk aan de bedieningsinstructies voor de recorder.

```
100 REM   OPGAVE 3C, HOOFDSTUK 6
110 REM
120 REM   VARIABELENLIJST
130 REM    - R$  : EEN VOLLEDIG RECORD (67 TEKENS)
140 REM    - N   : AANTAL RECORDS IN BESTAND
145 REM    - A$  : ARRAY (1..N)
150 REM    - D$  : DIALOOG
155 REM
```

6.6 ANTWOORDEN OP DE TOETSVRAGEN

1(a)
```
100 REM   OPGAVE 1A, HOOFDSTUK 6
110 REM
120 REM   VARIABELENLIJST
130 REM   - A$, B$        : GEGEVEN 1 RESP. 2 (ALFANUMERIEK)
140 REM   - C$/C, D$/D    : GEGEVEN 3 RESP. 4 (NUMERIEK)
145 REM   - R$            : RESPONS
150 REM
160 REM   GEGEVENSINVOER EN -CONTROLE
165 REM
167       OPEN "CAS:OPG1" FOR OUTPUT AS 1
170       LINE INPUT "GEGEVEN 1 (ALFANUM.): "; A$
180       IF LEN(A$) = 0 THEN
             PRINT "INVOER AUB"                    : GOTO 170
190       LINE INPUT "GEGEVEN 2 (ALFANUM.): "; B$
200       IF LEN(B$) = 0 THEN
             PRINT "INVOER AUB"                    : GOTO 190
210       LINE INPUT "GEGEVEN 3 (NUMERIEK): "; C$
220       IF LEN(C$) = 0 THEN
             PRINT "INVOER AUB"                    : GOTO 210
230       IF VAL(C$) = 0 THEN
             PRINT "UITSLUITEND GETALLEN AUB"      : GOTO 210
240       LET C = VAL(C$)
250       LINE INPUT "GEGEVEN 4 (NUMERIEK): "; D$
260       IF LEN(D$) = 0 THEN
             PRINT "INVOER AUB"                    : GOTO 250
270       IF VAL(D$) = 0 THEN
             PRINT "UITSLUITEND GETALLEN AUB"      : GOTO 250
280       LET D = VAL(D$)
290 REM
300       PRINT #1, A$; ","; B$; ","; C; D
310 REM
320       LINE INPUT "MEER GEGEVENS? (J/N): "; R$
330       IF LEFT$(R$,1) = "J" THEN 170
340 REM
350 REM   AFSLUITRECORD WEGSCHRIJVEN EN STOPPEN
355 REM
360       PRINT #1, -999; 0; 0; 0
370       CLOSE
380  REM
999       END
```

(b)
```
100 REM   OPGAVE 1B, HOOFDSTUK 6
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$, B$   : GEGEVEN 1 RESP. 2 (ALFANUMERIEK)
140 REM    - C, D     : GEGEVEN 3 RESP. 4 (NUMERIEK)
150 REM
160 REM   INLEZEN VAN BESTAND EN AFDRUKKEN
170 REM
175       OPEN "CAS:OPG1" FOR INPUT AS 1
180       INPUT #1, S$, B$, C, D
190          IF A$ = -999 THEN STOP
200           PRINT A$, B$, C, D
210       GOTO 180
220 REM
230       CLOSE
240 REM
999       END
```

2(a)
```
100 REM   OPGAVE 2A, HOOFDSTUK 6
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$   : ARTIKELOMSCHRIJVING (MAX. 20 TEKENS)
140 REM    - H    : HOEVEELHEID/AANTAL
150 REM    - D$   : DIALOOG
160 REM
170 REM   GEGEVENSINVOER EN -CONTROLE
180 REM
190       PRINT "GEEF -999 OM TE STOPPEN"
200       PRINT
205       OPEN "CAS:OPG2" FOR OUTPUT AS 1
210       LINE INPUT "ARTIKELOMSCHRIJVING: "; A$
220       IF A$ = "-999" THEN 350
230       IF LEN(A$) = 0 THEN
             PRINT "INVOER AUB"                                  : GOTO 210
240       IF LEN(A$) > 20 THEN
             PRINT "MAX. 20 TEKENS TOEGESTAAN. OPNIEUW AUB" : GOTO 210
250       INPUT "HOEVEELHEID:              "; H
260       IF H >= 1  AND  H <= 10 THEN 320
270       PRINT "INGEVOERD:                "; H
280       LINE INPUT "AKKOORD? (J/N)     : "; D$
290       IF LEFT$(D$,1) = "N" THEN 250
300 REM
310 REM   SCHRIJF NAAR BESTAND EN HERHAAL
315 REM
320       PRINT #1, A$; ","; H
330       GOTO 210
340 REM
345 REM   AFSLUITRECORD SCHRIJVEN EN STOPPEN
346 REM
350       PRINT #1, -999; 0
360       CLOSE
370 REM
999       END
```

(b)
```
100 REM   OPGAVE 2B, HOOFDSTUK 6
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$   : ARTIKELOMSCHRIJVING
140 REM    - H    : HOEVEELHEID
150 REM
160 REM
170 REM   RECORD INLEZEN EN AFDRUKKEN
180 REM
185       OPEN "CAS:OPG2" FOR INPUT AS 1
190       PRINT "ARTIKEL"; TAB(24); "HOEVEELHEID"
200       INPUT #1, A$, H
210       IF A$ = "-999" THEN 250
220       PRINT A$; TAB(24); H
230       GOTO 200
240 REM
250       PRINT: PRINT "EINDE BESTAND"
260 REM
270       CLOSE
280 REM
999       END
```

3(a)
```
100 REM   OPGAVE 3A, HOOFDSTUK 6
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$(20)  : NAAM
140 REM    - AS(20)  : ADRES (STRAAT + HUISNR.)
150 REM    - P$(7)   : POSTCODE
160 REM    - W$(20)  : WOONPLAATS
170 REM    - R$(67)  : GEHELE RECORD
180 REM    - D$      : DIALOOG
190 REM
200 REM
210 REM   INITIALISEREN
215 REM
220       CLEAR 1000
230 REM
240 REM   GEGEVENSINVOER EN -CONTROLE
250 REM
260       OPEN "CAS:OPG3" FOR OUTPUT AS 1
300       LINE INPUT "NAAM (MAXIMAAL 20 TEKENS)          : "; N$
310       IF LEN(N$) < 20 THEN LET N$ = N$ + " "                  : GOTO 310
320       IF LEN(N$) > 20 THEN
             PRINT "MAXIMAAL 20 TEKENS. OPNIEUW AUB"              : GOTO 300
330 REM
340       LINE INPUT "STRAAT + HUISNR (MAX. 20 TEKENS)  : "; A$
350       IF LEN(A$) < 20 THEN LET A$ = A$ + " "                  : GOTO 350
360       IF LEN(A$) > 20 THEN
             PRINT "MAXIMAAL 20 TEKENS. OPNIEUW AUB"              : GOTO 340
370 REM
380       LINE INPUT "POSTCODE (XXXX XX)                 : "; P$
390       IF VAL(LEFT$(P$,4)) = 0 THEN
             PRINT "IN POS. 1-4 ALLEEN CIJFERS. OPNIEUW AUB"      : GOTO 380
400       H1 = ASC(MID$(P$,6,1))                  : REM *** POS. 6 = LETTER? ***
410       IF H1 < 65  OR  H1 > 90 THEN
             PRINT "LETTER IN POS. 6 NODIG. OPNIEUW AUB"          : GOTO 380
420       H1 = ASC(MID$(P$,7,1))                  : REM *** POS. 7 = LETTER? ***
430       IF H1 < 65  OR H1 > 90 THEN
             PRINT "LETTER IN POS. 7 NODIG. OPNIEUW AUB"          : GOTO 380
440       H1 = ASC(MID$(P$,5,1))                  : REM *** POS. 5 = SPATIE? ***
450       IF H1 <> 32 THEN
             PRINT "SPATIE IN POS. 5 NODIG. OPNIEUW AUB"          : GOTO 380
460 REM
470       LINE INPUT "WOONPLAATS (MAX. (20 TEKENS)       : "; W$
480       IF LEN(W$) < 20 THEN W$ = W$ + " "                      : GOTO 480
490       IF LEN(W$) > 20 THEN
             PRINT "MAXIMAAL 20 TEKENS. OPNIEUW AUB"              : GOTO 470
495 REM
500       LET R$ = N$ + A$ + P$ + W$
510 REM
520       PRINT #1, R$
530 REM
540       LINE INPUT "MEER INVOER? (J/N): "; D$
550       IF LEFT$(D$,1) = "J" THEN 300
560 REM
570       PRINT #1, -999
580 REM
590       CLOSE
999       END
```

3(b)
```
100 REM   OPGAVE 3B, HOOFDSTUK 6
110 REM
120 REM   VARIABELENLIJST
130 REM   - R$    : EEN VOLLEDIG RECORD (67 TEKENS)
140 REM   - T     : TELLER VOOR AANTAL RECORDS
150 REM
160 REM   INITIALISEREN
165 REM
```

```
      170       CLEAR 500
      175       OPEN "CAS:OPG3" FOR INPUT AS 1
      180       LET T = 0
      190 REM
      200 REM   LEES/TEL RECORD EN DRUK AF
      205 REM
      210       INPUT #1, R$
      220       LET T = T + 1
      230       IF R$ = "-999" THEN 270
      240       PRINT T; " "; R$
      250       GOTO 210
      255 REM
      260       PRINT
      270       PRINT "TOTAAL AANTAL RECORDS (INCL. AFSLUITRECORDS): "; T
      280 REM
      290       CLOSE
      300 REM
      999       END
```

3(c)

```
      100 REM   OPGAVE 3C, HOOFDSTUK 6
      110 REM
      120 REM   VARIABELENLIJST
      130 REM   - R$    : EEN VOLLEDIG RECORD (57 TEKENS)
      140 REM   - N     : AANTAL RECORDS IN BESTAND
      145 REM   - A$    : ARRAY (1..N)
      150 REM   - D$    : DIALOOG
      155 REM
      160 REM   INITIALISATIES
      165 REM
      170       CLEAR 500
      175       OPEN "CAS:OPG3" FOR INPUT AS 1
      180       INPUT "HOEVEEL RECORD IN BESTAND: "; N
      190       DIM A$(N)
      200 REM
      210 REM   RECORDS INLEZEN IN ARRAY
      215 REM
      220       FOR I = 1 TO N
      230           INPUT #1, R$
      240           LET A$(I) = R$
      250       NEXT I
      255       CLOSE
      260 REM
      270       PRINT "STOP RECORDER, SPOEL TERUG EN VERWIJDER CASSETTE"
      280       PRINT "PLAATS NIEUWE CASSETTE IN RECORDER, SPOEL TERUG"
      290       PRINT "DRUK OP PLAY/RECORD"
      300       LINE INPUT "GEEF RETURN ALS U KLAAR BENT"; D$
      305       OPEN "CAS:OPG3" FOR OUTPUT AS 1
      310 REM
      320 REM   SCHRIJF AANTAL ALS EERSTE RECORD IN BESTAND
      325 REM
      330       PRINT #1, N
      340 REM
      350 REM   KOPIEER ARRAY NAAR CASSETTE
      355 REM
      360       FOR I = 1 TO N
      370           LET R$ = A$(I)
      380           PRINT #1, R$
      390       NEXT I
      400 REM
      410       PRINT "KOPIE KLAAR"
      420 REM
      430       CLOSE
      440 REM
      999       END
```

7 DIRECT-TOEGANKELIJKE GEGEVENSBESTANDEN

7.0 DOELSTELLINGEN

Na bestudering van dit hoofdstuk moet u in staat zijn gebruik te maken van de volgende programma-opdrachten en functies om direct-toegankelijke gegevensbestanden te creëren, verifiëren, kopiëren en muteren:

- FIELD-opdracht
- LSET-opdracht
- RSET-opdracht
- PUT-opdracht
- GET-opdracht
- LOF-functie
- MKS$-functie
- MKI$-functie
- MKD$-functie
- CVS-functie
- CVI-functie
- CVD-functie

U moet bovendien in staat zijn sequentiële gegevensbestanden om te zetten in direct-toegankelijke.

7.1 INLEIDING

In deze laatste hoofdstukken gaan we in op het gebruik van direct-toegankelijke gegevensbestanden, een bestandsvorm die alleen op schijfapparatuur mogelijk is.

Bij direct toegankelijke (Random Access) bestanden is alles anders. Niet alleen gebruiken we dit type bestanden voor *andere doeleinden* dan sequentieel toegankelijke bestanden, maar de

bufferorganisatie is anders; de *opdrachten* om records uit de buffer te halen en erin te stoppen zijn anders; de wijze waarop de *gegevens op diskette* staan is anders, kortom, zoals gezegd, alles is anders.

7.2 WAT IS EEN DIRECT-TOEGANKELIJK BESTAND?

Een direct-toegankelijk gegevensbestand (Engels: 'direct access' of 'random access' data file) is een bestand waarbij de records rechtstreeks te benaderen zijn. Daartoe worden ze genummerd. De records van een bestand hebben een standaardlengte van 256 bytes. De mogelijkheid bestaat om bij het aanmaken van het bestand een andere lengte te kiezen. Daarover straks meer.

Dankzij de rechtstreekse benaderbaarheid maakt het voor het lokaliseren van een record weinig uit waar dit zich in het bestand bevindt. Alle records kunnen in principe even snel bereikt worden, zonder eerst andere records te moeten doorlopen. Bovendien kunnen records afzonderlijk worden gemuteerd. Met direct-toegankelijke bestanden kan daarom in het algemeen sneller en efficiënter worden gewerkt dan met sequentiële bestanden. Dit komt vooral tot uitdrukking als de mutatiegraad (dat wil zeggen de frequentie waarmee records moeten worden gewijzigd) hoog is, en/of de gegevens in willekeurige volgorde verwerkt moeten worden.

Het belangrijkste nadeel van direct-toegankelijke bestanden – althans in de hier beschreven vorm – is de vaste recordlengte. We bedoelen hier de lengte van het *fysieke* record. In uw programma werkt u (overigens net als bij sequentiële bestanden) met *logische* records. De lengte hiervan wordt bepaald door de inhoud van uw uitvoeropdrachten. Schrijft u met één PRINT een of meer gegevens met een gezamenlijke lengte van 100 bytes naar schijf, dan wordt (uitgaande van de standaard-recordlengte van 256 bytes) ruim 60% van de beschikbare ruimte niet benut.*

Bij grotere toepassingen kan het voordelig zijn zowel de sequentiële als de direct-toegankelijke bestandsvorm te gebruiken.

* Om ruimteverspilling te voorkomen of te beperken kunt u werken met zogenaamde *sub-records*. Dit is echter een nogal ingewikkelde techniek die u beter kunt uitstellen tot u meer vertrouwd bent met direct-toegankelijke bestanden.

Toetsvragen

(a) Wat is de optimale (logische) recordlengte bij een computersysteem dat met fysieke records van 256 bytes werkt?

(b) Welke voordelen hebben direct-toegankelijke bestanden ten opzichte van sequentiële?

(a) 256 bytes

(b) 1. Snelle bereikbaarheid van alle records, onafhankelijk van van hun plaats in het bestand.
2. De inhoud van een record is gemakkelijk te veranderen (muteren).

7.3 INITIALISEREN EN AFSLUITEN

Voor het initialiseren (instellen) van een direct-toegankelijk bestand wordt dezelfde opdracht gebruikt als bij de sequentiële (schijf-) bestanden, namelijk de OPEN-opdracht. De daarbij toegekende bufferruimte dient zowel voor de in- als uitvoer. "FOR INPUT" en "FOR OUTPUT" moeten worden weggelaten.

Een voorbeeld:

```
120       OPEN "NAAM1" AS 3
```

We gaan ervan uit dat u met één drive werkt. Werkt u met twee drives dan kunt u op drive B een bestand openen met

```
OPEN "B:NAAM1" AS 3
```

Dit geldt ook voor sequentiële bestanden. In dit hoofdstuk gaan we uit van één drive, namelijk drive A. De OPEN-opdracht is dan gewoon

```
OPEN "NAAM1" AS 3   of   OPEN "A:NAAM1" AS 3
```

Met 3 geeft u het nummer van de buffer aan; NAAM1 is de naam van het direct-toegankelijke bestand. Uiteraard kunnen in plaats van deze alfanumerieke of numerieke constanten ook variabelen voor de naam en het buffernummer gebruikt worden.

Bij MSX BASIC is nog een gegeven mogelijk, namelijk de recordlengte. Zoals eerder gezegd is de standaardlengte 256 bytes. Wilt u liever met een andere lengte werken, bijvoorbeeld met 100 bytes, dan is dat eenvoudig te realiseren door deze waarde als constante of variabele na het buffernummer in de OPEN-opdracht op te nemen, dus:

```
120       OPEN "NAAM1" AS 3 LEN = 100
```

Ook het afsluiten gebeurt op dezelfde wijze als bij sequentiële bestanden, namelijk met de CLOSE-opdracht. Een CLOSE zonder meer, bijvoorbeeld

```
860       CLOSE
```

sluit alle openstaande bestanden – ook sequentiële – af. Om individuele bestanden te sluiten schrijven we na CLOSE het desbetreffende buffernummer.

voorbeeld:

```
860     CLOSE 3
```

Toetsvragen

Wat is het effect van de volgende opdrachten?

(a) `310       OPEN "ADRES FOR INPUT AS 1`

(b) `310       OPEN "HULP" FOR OUTPUT AS 2`

(c) `310       OPEN "KLANT" AS 1`

(a) Een sequentieel schijfbestand met buffernummer 1 en naam ADRES wordt voor invoer geopend.
(b) Een sequentieel schijfbestand met buffernummer 2 en naam HULP wordt voor uitvoer geopend. Eventueel eerder aanwezige gegevens in HULP worden vernietigd.
(c) Een direct-toegankelijk bestand met buffernummer 1 en naam KLANT wordt voor in- en/of uitvoer geopend.

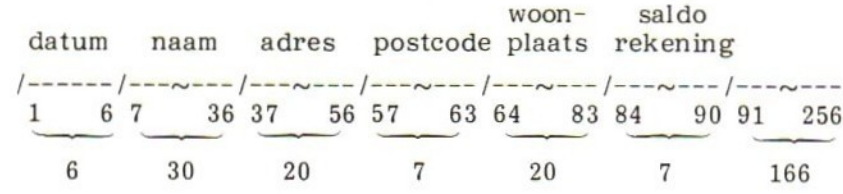
7.4 BUFFERVELDEN

Alvorens een record naar een direct-toegankelijk bestand te kunnen schrijven, dient dat record door de programmeur in de buffer te worden opgebouwd. Daartoe wordt de buffer (lees: record) met behulp van een zogenaamde *FIELD-opdracht* in velden onderverdeeld. De velden komen overeen met de gekozen recordindeling, en kunnen uitsluitend alfanumerieke gegevens* bevatten. Ze functioneren daarbij als afzonderlijke *buffervariabelen*, waaraan met behulp van een speciale opdracht (alfanumerieke) toekenningen kunnen worden gedaan. Is de buffer op deze wijze eenmaal gevuld, dan kan de inhoud daarvan – dat wil zeggen het record – met een andere opdracht naar schijf worden geschreven. Daarover straks meer. Voorlopig nemen we aan de hand van een voorbeeld eerst de zogenaamde FIELD-opdracht onder de loep.

---

* Of getallen in een bijzondere, interne representatie. Hierop komen we later terug.

Stel dat we voor een direct-toegankelijk klantenbestand de volgende recordindeling hebben gekozen:



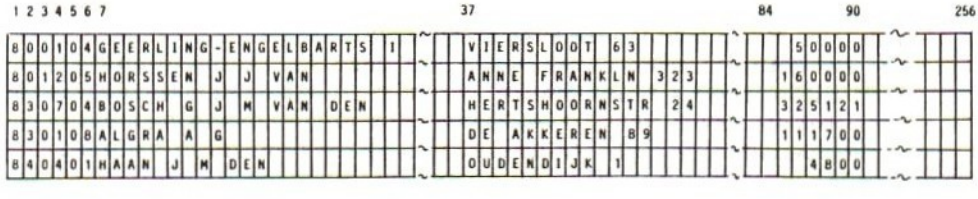
In overeenstemming met deze recordindeling kiezen we als buffervariabelen DATUM$, NAAM$, ADRES$, PCODE$, WOONPL$ en SALDO$. Merk op dat deze namen voldoen aan de naamgevingconventie voor alfanumerieke oftewel stringvariabelen. Ten opzichte van alfanumerieke variabelen is wel een belangrijk verschil: de bijbehorende geheugenlokatie bevindt zich namelijk niet in het 'gewone' gegevensgebied van het interne geheugen, maar in het bufferdeel. Deze buffer moet nu zo worden ingericht dat de eerste zes bytes gereserveerd worden voor DATUM$, de volgende 30 bytes voor NAAM$, enz.

Dit is te realiseren met de volgende FIELD-opdracht:

```
130       FIELD 3, 6 AS DATUM$, 30 AS NAAM$, 20 AS ADRES$, 7 AS PCODE$,
                   20 AS WOONPL$, 7 AS SALDO$
```

Hierin stelt 3 het buffernummer voor. FIELD en AS zijn sleutelwoorden (keywords).

Een bestand van vijf records, op deze wijze opgebouwd, zou als volgt kunnen worden voorgesteld:



Toetsvraag

Beantwoord aan de hand van het bovenstaande schema de volgende vragen.

(a) Wat is de naam van de persoon in record 2?

(b) Wat is zijn/haar adres?

(c) Welk saldo heeft deze persoon?

(d) Op welke datum vond de desbetreffende transactie plaats?

(a) HORSSEN J J VAN
(b) ANNE FRANKLN 323
(c) 160000
(d) 801205

Naamkeuze van buffervariabelen

We zagen reeds dat voor buffervariabelen dezelfde naamconventie wordt gebruikt als voor gewone alfanumerieke variabelen. Voor de duidelijkheid kozen we langere namen dan tot nu toe, maar essentieel is dit niet. We hadden bijvoorbeeld even goed de namen D$, N$, A$, P$, W$ en S$ kunnen kiezen.

Om duidelijk tot uitdrukking te brengen dat een naam voor een buffervariabele wordt gebruikt, kunnen we daaraan de letter B of F toevoegen (Engels: field variable = buffervariabele), bijvoorbeeld DB$ of DF$. Aangezien bij langere namen vaak slechts de eerste twee tekens significant zijn, verdient het aanbeveling de B of de F vooraan te zetten, bijvoorbeeld BDATUM$ of FDATUM$. De namen DATUMB$ en DATUMF$ zouden anders door de computer als dezelfde variabelen worden opgevat.

Bij een consequent doorvoeren van dergelijke conventies vervalt de noodzaak om buffervariabelen afzonderlijk in het programma te documenteren.

Hoewel sommige stringoperaties zowel met buffer- als op gewone variabelen kunnen worden uitgevoerd, kan dezelfde naam niet binnen één programma voor zowel het ene als het andere type variabele worden gebruikt. Een constructie zoals:

```
120     FIELD  1, 25 AS B$, 25 AS CF$
130     LET   B$ = X$
```

zou dus resulteren in een foutmelding, omdat B$ in regel 120 als een buffervariabele is gedefinieerd en in regel 130 kennelijk als alfanumerieke variabele dienst moet doen.

Toetsvragen

(a) Schrijf een FIELD-opdracht voor een direct-toegankelijk bestand met buffernummer 2 en de volgende recordindeling:
1. een (alfanumeriek) catalogusnummer van 7 tekens (C$);
2. een artikelbeschrijving van maximaal 60 tekens (B$);
3. de prijs van het artikel als alfanumeriek gegeven van maximaal 6 tekens (P$).

(b) Welke van de volgende FIELD-opdrachten zijn illegaal? Waarom?

```
1. 240    FIELD 2,  5 AS NF$, 10 AS PF$,
                    4 AS CF$,  4 AS BF$
2. 540    FIELD 1, 85 AS AF$, 85 AS BF$,
                   85 AS CF$, 85 AS DF$
3. 320    FIELD 2, 25 AS GF$,  4 AS PF,
                   16 AS NF$, 10 AS CF
```

(a) `540 FIELD 2, 7 AS C$, 60 AS B$, 6 AS P$`
(eventueel B of F in de variabelenaam)

(b) 2 (teveel tekens gedefinieerd voor een buffer van maximaal 256 tekens)
3 (2 variabelen zonder $-teken)

7.5 EENVOUDIGE LEES- EN SCHRIJFOPERATIES MET STRINGS

In deze paragraaf behandelen we de opdrachten nodig voor

1. het toekennen van waarden aan buffervariabelen (LSET, RSET);
2. het schrijven van een record naar schijf (PUT);
3. het overbrengen van een record van schijf naar buffer (GET).

We beperken ons hierbij (nog steeds) tot het lezen en schrijven van alfanumerieke gegevens. Voor numerieke gegevens zijn extra bewerkingen nodig. Deze zullen later worden besproken.

De algemene procedure voor het gebruik van direct-toegankelijke bestanden met alfanumerieke gegevens is:

1. Open bestand(en).
2. Richt buffer(s) in met FIELD-opdracht.
3. Voer gegevens in (toekennen aan 'gewone' alfanumerieke variabelen).
4. Ken bij de uitvoer de alfanumerieke waarden toe aan buffervariabelen met behulp van LSET en/of RSET.

5. Schrijf de buffer naar schijf met PUT (bij uitvoer), of breng een record van schijf naar buffer met GET (bij invoer).
6. Voer de gewenste bewerkingen uit.
7. Sluit alle bestanden met CLOSE.

Schematisch kan een en ander door fig.7-1 worden voorgesteld.

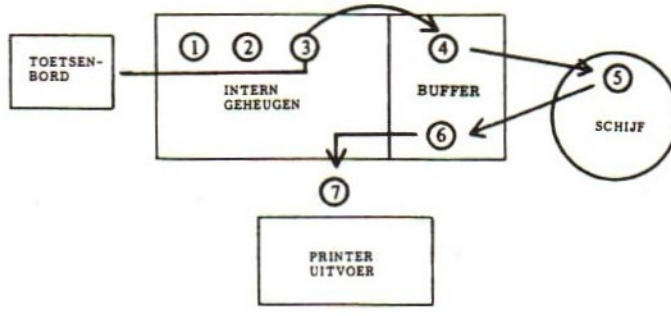


fig.7-1 Gang van zaken bij het gebruik van direct-toegankelijke bestanden met alfanumerieke variabelen. De cijfers komen overeen met de stappen van de eerder behandelde procedure.

Voorbeeldprogramma 1

We ontwikkelen nu volgens de aangegeven procedure een voorbeeldprogramma. Doel van het programma is een direct-toegankelijk voorraadbestand van reserve-onderdelen op te bouwen. Voorlopig nemen we in dit bestand uitsluitend een alfanumeriek codenummer en een omschrijving van het onderdeel op. Later zullen we de aantallen toevoegen. De inleidende module is hieronder gedeeltelijk gegeven. U wordt gevraagd zelf de ontbrekende regels (nrs. 220 en 230) te leveren.

(a)
```
100 REM   ONDERDELENVOORRAAD
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$ = ONDERDEELNUMMER (6)
140 REM    - D$ = OMSCHRIJVING (20)
150 REM
160 REM   BESTAND
170 REM
180 REM    - VOORRAAD (DIRECT TOEGANKELIJK)
190 REM
200 REM   INITIALISATIE VAN BESTAND
210 REM
220       ..............................  : REM *** OPEN BESTAND ***
230       ..............................  : REM *** DEFINIEER VELDEN ***
240 REM
```

```
(a)  220       OPEN "VOORRAAD" AS 1
     230       FIELD 1, 6 AS NF$, 20 AS DF$
```

Wat de gegevensinvoer betreft laten we de controles gemakshalve achterwege. We kunnen ons programma dan met de volgende opdrachten voortzetten:

```
250 REM   GEGEVENSINVOER
260 REM
270       LINE INPUT "NUMMER:(XXXXXX): "; N$
280       LINE INPUT "OMSCHRIJVING   : "; D$
290 REM
```

In de regels 270 en 280 hebben we gewone alfanumerieke variabelen gebruikt. De inhoud van die variabelen moet nu met LSET- en/of RSET-opdrachten aan de eerder gedefinieerde buffervariabelen worden toegekend. Alvorens dit uit te werken eerst iets naders over de werking van deze opdrachten.

Zowel de LSET- als de RSET-opdracht hebben ten doel alfanumerieke waarden aan buffervariabelen toe te kennen. De vorm van beide opdrachten is identiek, en lijkt op de 'gewone' alfanumerieke toekenning met LET:

LSET buffervariabele = stringwaarde

en:

RSET buffervariabele = stringwaarde

waarbij de stringwaarde in de vorm van een alfanumerieke constante of variabele weergegeven kan worden.

Het verschil tussen beide opdrachten komt pas tot uitdrukking als de lengte van de toe te kennen stringwaarde kleiner is dan de beschikbare ruimte in de buffervariabele. Bij LSET wordt de stringwaarde dan links aangesloten (Left SET), en bij RSET rechts (Right SET). De overblijvende ruimte wordt gevuld met spaties.

voorbeeld

Bij de opdracht

LSET DF$ = "SCHROEFBOUT M6"

zou de desbetreffende string als volgt in de buffervariabele DF$ terechtkomen:

S	C	H	R	O	E	F	B	O	U	T		M	6					

Bij de opdracht

```
RSET DF$ = "SCHROEFBOUT M6"
```

staat de string rechts aangesloten in het bufferveld:

Is de lengte van de toe te kennen stringwaarde groter dan de lengte van het desbetreffende bufferveld, dan wordt de string bij de TRS-80 eenvoudig afgekapt, en wel 'rechts' bij LSET en 'links' bij RSET. Bij de Exidy wordt de string in beide gevallen rechts afgekapt.

Tot zover de werking van LSET en RSET. We kunnen nu de buffertoekenningen in ons voorbeeldprogramma maken. Lever zelf de desbetreffende opdrachten in het volgende programmadeel.

```
(a)  300 REM   RECORD OPBOUWEN IN BUFFER
     305 REM
     310       ..............................: REM *** TOEKENNING N$ ***

     320       ..............................: REM *** TOEKENNING D$ ***

     330 REM
-------------------------
(a)  310       LSET NF$ = N$
     320       LSET DF$ = D$
```

Op dit punt staan de gegevens klaar om naar schijf te worden overgebracht. De PUT-opdracht, waarmee dit kan worden gerealiseerd, heeft de vorm:

```
PUT buffernummer, recordnummer
```

waarbij 'recordnummer' het volgnummer van het record op schijf aangeeft. Dankzij dit nummer kan het record naderhand bij het lezen worden teruggevonden.

voorbeeld

```
PUT 1, R
```

Deze opdracht heeft tot gevolg dat de inhoud van buffer 1 als recordnummer R naar schijf wordt geschreven.

Eventueel kan het recordnummer, en de daaraan voorafgaande komma, achterwege worden gelaten. In dat geval komt het record direct na het laatst geschreven record.

Met deze kennis gewapend kunt u het volgende programmadeel afmaken.

```
(a)   340 REM   SCHRIJF NAAR BESTAND
      345 REM
      350       ..............................: REM *** ZONDER RECORDNUMMER ***
      360 REM
-------------------------
(a)   350       PUT 1
```

Het programma is nu bijna klaar. Uitvoering op dit moment zou ons in staat stellen de gegevens voor één record in te voeren, het record in de buffer samen te stellen en te schrijven naar schijf. Met het volgende (en laatste) programmafragment zorgen we voor het herhalen van deze acties, of voor het beëindigen van het programma als we klaar zijn met het opbouwen van het bestand.

```
370 REM   MEER GEGEVENS?
375 REM
380       LINE INPUT "MEER INVOER? (J/N): "; R$
390       IF LEFT$(R$,1) = "J" THEN 270
400 REM
410 REM   SLUIT BESTANDEN EN STOP
415 REM
420       CLOSE 1
999       END
```

Het volledige programma ziet er als volgt uit:

```
100 REM   ONDERDELENVOORRAAD
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$ = ONDERDEELNUMMER (6)
140 REM    - D$ = OMSCHRIJVING (20)
150 REM
160 REM   BESTAND
170 REM
180 REM    - VOORRAAD (DIRECT TOEGANKELIJK)
190 REM
200 REM   INITIALISATIE VAN BESTAND
210 REM
220       OPEN "VOORRAAD" AS 1
230       FIELD 1, 6 AS NF$, 20 AS DF$
240 REM
250 REM   GEGEVENSINVOER
260 REM
270       LINE INPUT "NUMMER (XXXXXX): "; N$
280       LINE INPUT "OMSCHRIJVING   : "; D$
290 REM
```

```
300 REM   RECORD OPBOUWEN IN BUFFER
305 REM
310       LSET NF$ = N$
320       LSET DF$ = D$
330 REM
340 REM   SCHRIJF NAAR BESTAND
345 REM
350       PUT 1
360 REM
370 REM   MEER GEGEVENS?
375 REM
380       LINE INPUT "MEER INVOER? (J/N): "; R$
390       IF LEFT$(R$,1) = "J" THEN 270
400 REM
410 REM   SLUIT BESTANDEN EN STOP
415 REM
420       CLOSE 1
999       END
```

Opmerkingen

Voordat we verder gaan, enkele algemene opmerkingen.

1. De inrichting van een buffer – en dus ook de interne recordstructuur – kan worden veranderd door opnieuw een FIELD-opdracht uit te voeren (Engels: to refield). Dit maakt de programmastructuur echter minder doorzichtig, en kan daarom beter worden vermeden. Wilt u toch 'refielden', zorg dan met behulp van REMARKs voor een goede documentatie.
2. Geeft u bij een PUT-opdracht niet het eerstvolgende, maar een groter recordnummer aan, dan wordt die PUT toch uitgevoerd. De tussenliggende records zijn dan leeg, maar nemen toch schijfruimte in beslag.

   voorbeeld

   Is het hoogst gebruikte recordnummer 25 als de opdracht

   ```
   PUT 1, 226
   ```

   wordt uitgevoerd, dan worden 200 lege records tussen record 25 en record 226 geplaatst.
3. Het 'aan de beurt' zijnde record wordt, net als bij sequentiële bestanden, bijgehouden door een recordwijzer (Engels: record pointer). Na iedere PUT-opdracht geeft de recordwijzer het nummer van het eerstvolgende record aan. Dit record is het zogenaamde *current* record.

Voorbeeldprogramma 2

Stel dat we met het programma van voorbeeld 1 het volgende direct-toegankelijke bestand hebben gemaakt:

onderdeelnummer	omschrijving
112131	PRINTBORD
341232	5 INCH FLOPPY
341233	8 INCH FLOPPY
871256	RS232 INTERFACE
983476	5 INCH DISC DRIVE

In dit voorbeeld ontwikkelen we een programma om de inhoud van dit bestand zichtbaar te maken. In een later stadium kan dit programma eventueel in het eerste worden ingebouwd.

Vooraf enkele opmerkingen over de GET-opdracht die, zoals eerder vermeld, gebruikt wordt om records van schijf naar buffer over te brengen (dat wil zeggen: kopiëren).

1. De GET-opdracht heeft dezelfde vorm als zijn tegenhanger, de PUT. De opdracht:

```
270       GET 2
```

   heeft dus tot gevolg dat het 'aan de beurt' zijnde record (current record) wordt ingelezen in buffer 2. De recordwijzer wordt daarna opgehoogd met 1, en geeft dan de positie aan van het eerstvolgende record.

2. Eventueel kan expliciet worden gespecificeerd welk record ingelezen moet worden, bijvoorbeeld:

```
270       GET 2, R
```

   Hierbij wordt de inhoud van recordnummer R naar buffer 2 gekopieerd, en de recordwijzer met 1 verhoogd.

3. De inhoud van buffervariabelen kan rechtstreeks in PRINT-opdrachten en dergelijke worden gebruikt. Het is hierbij niet nodig de inhoud van deze variabelen eerst aan gewone stringvariabelen toe te kennen.

We beschikken nu over voldoende kennis om een begin te maken met ons voorbeeldprogramma. Het eerste deel staat hieronder. Aan u weer de vraag om de ontbrekende opdrachten te leveren.

(a)
```
100 REM   UITLEZEN VOORRAADBESTAND
110 REM
120 REM   VARIABELENLIJST
130 REM    - NF$ = ONDERDEELNUMMER (6)
140 REM    - DF$ = OMSCHRIJVING (20)
150 REM
160 REM   BESTANDEN
170 REM    - VOORRAAD (DIRECT TOEGANKELIJK)
180 REM
```

```
190 REM   INITIALISATIE VAN BESTAND
195 REM
200       ..............................: REM *** OPEN BESTAND ***

210       ..............................: REM *** BUFFER INRICHTEN ***

220 REM
230       PRINT "NUMMER", TAB(15), "OMSCHRIJVING"
240 REM
```

```
(a)  220       OPEN "VOORRAAD" AS 1
     230       FIELD 1, 6 AS NF$, 20 AS DF$
```

Nu het inlezen van de records en het afdrukken van het overzicht.

```
250 REM   RECORDS INLEZEN EN AFDRUKKEN
255 REM
260       IF EOF(1) THEN 320
270          GET 1
280          PRINT NF$, TAB(15), DF$
290       GOTO 260
300 REM
310 REM   AFSLUITEN
315 REM
320       CLOSE 1
330 REM
```

De EOF-opdracht in regel 260 zorgt voor de foutmelding "Bad file mode", omdat deze opdracht slechts is toegestaan bij sequentiële bestanden.

Een manier om het einde van een direct-toegankelijk bestand te bepalen is met behulp van de functie LOF (length of file). Zoals de naam doet vermoeden levert deze functie de lengte van het bestand in bytes. De vorm van de functie is

LOF(bufnum)

waarbij 'bufnum' het nummer van het desbetreffende bestand is. De aanroep LOF(1) levert dus als functiewaarde de lengte van het aan buffer 1 gekoppelde bestand. Delen we door de recordlengte, dan is het nummer van het laatste record bekend.

Toepassing hiervan in het bovenstaande voorbeeldprogramma met een FOR/NEXT-constructie geeft het volgende alternatief voor de regels 250 t/m 330:

```
250 REM   RECORDS INLEZEN EN AFDRUKKEN
255 REM
260       FOR X = 1 TO LOF(1)/256
270           GET 1
280           PRINT NF$, TAB(15), DF$
290       NEXT X
300 REM
310 REM   AFSLUITEN
315 REM
320       CLOSE 1
330 REM
```

In regel 270 wordt de gehele inhoud van het 'current' record naar de buffer gekopieerd. De daarop volgende regel (280) zorgt voor het afdrukken van de beide buffervelden op het scherm. Hierbij is voorafgaande toekenning aan 'gewone' stringvariabelen overbodig.

Een overzicht van het geheel staat in § 7.6.

Voorbeeldprogramma 3

Behalve bij het lezen kan de LOF-functie ook worden toegepast bij het uitbreiden van een direct-toegankelijk bestand. Een voorbeeld hiervan is:

```
240       LET R = LOF(3)/256 + 1
250       PUT 3, R
```

In regel 240 wordt door middel van de uitdrukking LOF(3)/256 het nummer bepaald van het laatste record. Dit nummer wordt verhoogd met 1 en toegekend aan R. Deze variabele geeft dus het nummer aan van het eerstvolgend te schrijven record, en kan daarom als record-nummer worden gebruikt in de PUT-opdracht (regel 250).

We passen de LOF-functie toe in het volgende voorbeeldprogramma. Het programma voegt nieuwe records toe aan een bestaand direct-toegankelijk bestand met telefoonnummers van klanten. Ook hier worden invoercontroles gemakshalve weggelaten. U wordt gevraagd de ontbrekende regels te leveren.

(a)

```
100 REM   TOEVOEGEN AAN DIRECT TOEGANKELIJK BESTAND
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$ = NUMMER VAN KLANT (5)
140 REM    - K$ = NAAM VAN KLANT (20)
150 REM    - T$ = TELEFOONNUMMER (10)
160 REM    - R  = RECORDTELLER
170 REM
180 REM   BESTAND
190 REM    - TELNUM (DIRECT-TOEGANKELIJK)
200 REM
210 REM   INITIALISATIE BESTAND
215 REM
220       OPEN "TELNUM" AS 1 LEN = 35
230       FIELD 1, 5 AS NF$, 20 AS KF$, 10 AS TF$
240 REM
250 REM   BEPAAL PLAATS VAN LAATSTE RECORD
255 REM
260       LET R = LOF(1)/35 + 1
270 REM
280 REM   GEGEVENSINVOER
285 REM
290       LINE INPUT "KLANTNUMMER     : "; N$
300 REM
310       LINE INPUT "NAAM VAN KLANT  : "; K$
320 REM
330       LINE INPUT "TELEFOONNUMMER  : "; T$
340 REM
350 REM
360 REM   BRENG GEGEVENS NAAR BUFFER
365 REM
370       ..............................

380       ..............................

390       ..............................

400 REM
410 REM   KOPIEER BUFFER NAAR SCHIJF
415 REM
420       ..............................

430 REM
440 REM   EVENTUEEL HERHALEN
445 REM
450       LINE INPUT "MEER RECORDS? (J/N): "; D$
460       IF LEFT$(R$,1) = "J" THEN 290
470 REM
480 REM   AFSLUITEN
485 REM
490       ..............................

500 REM
999       END
```

(a)

```
370       LSET NF$ = N$          420       PUT 1, R
380       LSET KF$ = K$
390       LSET TF$ = T$          490       CLOSE 1
```

Schrijf vervolgens een module om de inhoud van het gehele bestand – inclusief toegevoegde records – te lezen en op het scherm zichtbaar te maken. De module moet aan het einde van het voorgaande programma worden opgenomen, en begint met regelnummer 510.

```
(a)  510 REM   LEZEN/SCHRIJVEN VAN HET GEHELE BESTAND
     515 REM
     :
     :

-------------------------

(a)  520       OPEN "TELNUM" AS 1 LEN = 35
     530       FIELD 1, 5 AS NF$, 20 AS KF$, 10 AS TF$
     540 REM
     550       FOR X = 1 TO LOF(1)/35
     560           GET 1
     570           PRINT NF$, KF$, TF$
     580       NEXT X
     590 REM
     600       CLOSE 1
     610 REM
```

7.6 NUMERIEKE GEGEVENS

Tot nu toe hebben we bij direct-toegankelijke bestanden uitsluitend met alfanumerieke gegevens gewerkt. In principe is deze gegevensvorm de enige die bij direct-toegankelijke bestanden mogelijk is. Langs een omweg kunnen echter ook numerieke gegevens (getalwaarden) worden opgeslagen en gelezen. Daartoe moeten ze eerst naar overeenkomstige alfanumerieke waarden worden geconverteerd.

Er bestaan hiervoor drie functies:

1. MKS$, bestemd voor het converteren van een enkele-precisiewaarde naar een string van vier bytes (make single precision).
2. MKI$, bestemd voor het converteren van een integerwaarde (dat wil zeggen een geheel getal) naar een string van twee bytes (make integer).
3. MKD$, bestemd voor het converteren van een dubbele-precisiewaarde naar een string van acht bytes (make double precision).

Alle drie functies worden op dezelfde wijze gebruikt.
Een voorbeeld:

```
340       LSET DF$ = MKS$(476.23)
```

of:

```
330       LET X = 476.23
340       LSET DF$ = MKS$(X)
```

De getalwaarde 476.23 (enkele precisie) wordt hier geconverteerd naar een string van vier bytes.

De lengte van de resulterende string heeft uiteraard consequenties voor de inrichting van de buffer. In bovenstaand voorbeeld zou voor het bufferveld DF$ een lengte van vier bytes moeten worden gedefinieerd, dus met "4 AS DF$" als onderdeel van de FIELD-opdracht.

Overigens: de geconverteerde waarde laat zich niet zonder meer als tekst op het scherm of op de printer afdrukken. De MK-functies leveren namelijk een interne getalrepresentatie die anders is dan een alfanumerieke representatie volgens (bijvoorbeeld) ASCII.

Het (terug) converteren van getallen in alfanumerieke vorm naar hun oorspronkelijke numerieke vorm vindt ook plaats met behulp van speciale functies. Deze zijn:

1. CVS, bestemd voor het converteren van een string van vier bytes naar enkele-precisie-vorm (convert to single precision).
2. CVI, bestemd voor het converteren van een string van twee bytes naar integervorm (convert to integer).
3. CVD, bestemd voor het converteren van een string van acht bytes naar dubbele-precisie-vorm (convert to double precision).

Deze functies zijn dus tegenhangers van de functies MKS$, MKI$ en MKD$, dat wil zeggen CVS doet precies het omgekeerde van MKS$, CVI het omgekeerde van MKI$ en CVD het omgekeerde van MKD$.

voorbeelden

```
460       PRINT CVS(DF$)

310       LET D = CVS(DF$)
```

De 'MK'-functies worden in het algemeen als voorbereiding op het wegschrijven van een record naar schijf toegepast, de 'CV'-functies daarentegen als voorbereiding op het gebruik in het programma van ingelezen getallen. Daarbij is het van groot belang voor hetzelfde gegeven steeds de overeenkomstige functies toe te passen. Voor een getal dat u met MKS$ naar een string hebt geconverteerd gebruikt u dus CVS, enzovoort. Doet u dit niet dan is er een grote kans op een foutmelding of op onjuiste resultaten.

Aanpassing voorbeeldprogramma 1

We beschikken nu over voldoende kennis om het voorraadprogramma van § 7.5 zo uit te breiden dat ook de hoeveelheden van de voorradige onderdelen in het bestand kunnen worden opgenomen. Ter her-

innering nog even een overzicht van het desbetreffende programma (voorbeeldprogramma 1, p.214/215):

```
100 REM   ONDERDELENVOORRAAD
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$ = ONDERDEELNUMMER (6)
140 REM    - D$ = OMSCHRIJVING (20)
150 REM
160 REM   BESTAND
170 REM
180 REM    - VOORRAAD (DIRECT TOEGANKELIJK)
190 REM
200 REM   INITIALISATIE VAN BESTAND
210 REM
220       OPEN "VOORRAAD" AS 1
230       FIELD 1, 6 AS NF$, 20 AS DF$
240 REM
250 REM   GEGEVENSINVOER
260 REM
270       LINE INPUT "NUMMER (XXXXXX): "; N$
280       LINE INPUT "OMSCHRIJVING   : "; D$
290 REM
300 REM   RECORD OPBOUWEN IN BUFFER
305 REM
310       LSET NF$ = N$
320       LSET DF$ = D$
330 REM
340 REM   SCHRIJF NAAR BESTAND
345 REM
350       PUT 1
360 REM
370 REM   MEER GEGEVENS?
375 REM
380       LINE INPUT "MEER INVOER? (J/N): "; R$
390       IF LEFT$(R$,1) = "J" THEN 270
400 REM
410 REM   SLUIT BESTANDEN EN STOP
415 REM
420       CLOSE 1
999       END
```

We voegen hieraan eerst een documentatieregel toe:

```
145 REM    - H  = HOEVEELHEID (ENKELE PRECISIE)
```

Aan u de taak de FIELD-opdracht in regel 230 aan te passen:

(a) 230 .....

(a) 230 FIELD 1, 6 AS NF$, 20 AS DF$, 4 AS HF$

We kunnen nu een regel toevoegen om de hoeveelheid in te lezen. Ook daarbij laten we voor het gemak invoercontrole achterwege.

```
285       INPUT "HOEVEELHEID     : "; H
```

Geef nu zelf de opdracht om de waarde van H in het desbetreffende bufferveld op te nemen.

(a) 325 .....

(a) 325 LSET HF$ = MKS$(H)

Toetsvraag

(a) Welke verandering moet worden aangebracht in de PUT-opdracht?

(a) Geen. De bufferinhoud wordt in zijn geheel weggeschreven, inclusief het veld voor H.

Aanpassing voorbeeldprogramma 2

We passen nu ook het tweede voorbeeldprogramma aan. Dit zag er tot nu toe als volgt uit:

```
100 REM   UITLEZEN VOORRAADBESTAND
110 REM
120 REM   VARIABELENLIJST
130 REM    - NF$ = ONDERDEELNUMMER (6)
140 REM    - DF$ = OMSCHRIJVING (20)
150 REM
160 REM   BESTANDEN
170 REM    - VOORRAAD (DIRECT-TOEGANKELIJK)
180 REM
190 REM   INITIALISATIE VAN BESTAND
195 REM
200       OPEN "VOORRAAD" AS 1
210       FIELD 1, 6 AS NF$, 20 AS DF$
220 REM
230       PRINT "NUMMER", TAB(15), "OMSCHRIJVING"
240 REM
250 REM   RECORDS INLEZEN EN AFDRUKKEN
255 REM
260       FOR X = 1 TO LOF(1)/256
270           GET 1
280           PRINT NF$, TAB(15), DF$
290       NEXT X
300 REM
310 REM   AFSLUITEN
315 REM
320       CLOSE 1
330 REM
999       END
```

We voegen eerst de volgende regel toe:

```
145 REM    - HF$ = HOEVEELHEID (ENKELE PRECISIE)
```

en veranderen regel 230 in:

```
230       PRINT "NUMMER", TAB(15), "OMSCHRIJVING", TAB(40), "HOEVEELHEID"
```

Aan u de eer om de FIELD-opdracht (regel 210) en de PRINT-opdracht (regel 280) aan te passen:

```
(a)  210 ..............................
     280 ..............................
------------------------------
```

```
(a)  210       FIELD 1, 6 AS NF$, 20 AS DF$, 4 AS HF$

     280       PRINT NF$, DF$, CVS(HF$)
```

N.B. Eventueel kan regel 280 in de volgende twee regels worden gesplitst:

```
280       LET H = CVS(HF$)
285       PRINT NF$, DF$, H
```

7.7 UTILITY-PROGRAMMA'S

In de voorgaande paragrafen zijn de principes van het werken met direct-toegankelijke bestanden behandeld. We ontwikkelen nu twee utility-programma's, één voor het kopiëren, en één voor het muteren van direct-toegankelijke bestanden. Daarmee kan enerzijds uw inzicht in deze bestandsvorm worden verdiept, en anderzijds een voorbeeld worden gegeven dat als basis voor eigen toepassingen kan dienen.

Utility-programma 1: kopiëren

Dit programma kopieert gegevens record voor record van het ene direct-toegankelijke bestand naar het andere. De gegevens kunnen uit tekst en/of getallen bestaan.

De procedure is als volgt:

1. Open het te kopiëren bestand (bestand 1).
2. Open het bestand waarnaar moet worden gekopieerd (bestand 2).
3. Leg met behulp van een FIELD-opdracht de bufferstructuur van bestand 1 vast.
4. Leg met behulp van een tweede FIELD-opdracht de bufferstructuur van bestand 2 vast. Gebruik daarbij andere variabelenamen, maar verander niet de gegevensstructuur.

5. Maak een FOR/NEXT-lus, met LOF als eindwaarde, waarin achtereenvolgens:
   a. een record van bestand 1 wordt ingelezen;
   b. de waarden in de buffervariabelen van bestand 1 worden toegekend aan de overeenkomstige buffervariabelen van bestand 2;
   c. de inhoud van buffer 2 naar bestand 2 wordt geschreven.
6. Sluit beide bestanden.

De inleidende module – met willekeurige variabelen – ziet er als volgt uit:

```
100 REM   UTILITY-PROGRAMMA VOOR KOPIEREN
110 REM
120 REM   VARIABELENLIJST
130 REM    - GF$ = FF$ (20)
140 REM    - SF$ = RF$ (8)
145 REM    - QF$ = PF$ (4)
150 REM    - MF$ = NF$ (30)
160 REM
170 REM   BESTANDEN
180 REM    - BEST1 = TE KOPIEREN DIRECT-TOEGANKELIJK BESTAND
190 REM    - BEST2 = KOPIE BEST1
200 REM
```

De relatie tussen de variabelen van de verschillende buffers is in de regels 130 t/m 150 weergegeven door overeenkomstige variabelen in dezelfde regel te noteren. Daarbij hoort de eerste buffervariabele (bijvoorbeeld GF$) bij bestand 1, de tweede bij bestand 2 (bijvoorbeeld FF$). Gezien de identieke bufferstructuur hebben overeenkomstige variabelen uiteraard steeds dezelfde lengte (bijvoorbeeld 20 in het geval van GF$ en FF$).

Opdracht

(a) Maak aan de hand van de gegeven REMARKs het volgende initialisatiesegment af.

```
210 REM   BESTANDSINITIALISATIE
215 REM
217       MAXFILES = 2
220       .....: REM *** OPEN BESTAND 1 ***
230       .....: REM *** OPEN BESTAND 2 ***
240       .....: REM *** DEFINIEER STRUCTUUR VAN BUFFER 1 ***
250       .....: REM *** DEFINIEER STRUCTUUR VAN BUFFER 2 ***
260 REM
```

```
(a)  220       OPEN "BEST1" AS 1
     230       OPEN "BEST2" AS 2
     240       FIELD 1, 20 AS GF$, 8 AS SF$, 4 AS QF$, 30 AS MF$
     250       FIELD 2, 20 AS FF$, 8 AS RF$, 4 AS PF$, 30 AS NF$
```

In het volgende –en laatste – programmadeel worden gegevens van bestand 1 ingelezen, aan buffer 2 toegekend en vervolgens met een PUT-opdracht naar bestand 2 geschreven. Aan u weer de vraag om het segment af te maken.

```
(a)  270 REM   KOPIEER BESTAND
     275 REM
     280       FOR R = 1 TO LOF(1)/256

     290           .....: REM *** LEES RECORD VAN BESTAND 1 ***

     300           .....: REM *** BRENG GEGEVENS NAAR BUFFER 2 ***

     310           .....

     320           .....

     330           .....

     340           .....: REM *** SCHRIJF NAAR BESTAND 2 ***

     350       NEXT R
     360       CLOSE 1, 2
     370       PRINT "KOPIE KLAAR"
     380 REM
-------------------------
(a)  290           GET 1
     300           LSET FF$ = GF$
     310           LSET RF$ = SF$
     320           LSET PF$ = QF$
     330           LSET NF$ = MF$
     340           PUT 2
```

Het volledige programma ziet er als volgt uit:

```
100 REM   UTILITY-PROGRAMMA VOOR KOPIEREN
110 REM
120 REM   VARIABELENLIJST
130 REM    - GF$ = FF$ (20)
140 REM    - SF$ = RF$ (8)
150 REM    - QF$ = PF$ (4)
150 REM    - MF$ = NF$ (30)
160 REM
170 REM   BESTANDEN
180 REM    - BEST1 = TE KOPIEREN DIRECT-TOEGANKELIJK BESTAND
190 REM    - BEST2 = KOPIE BEST1
200 REM
```

```
210 REM   BESTANDSINITIALISATIE
215 REM
217       MAXFILES = 2
220       OPEN "BEST1" AS 1
230       OPEN "BEST2" AS 2
240       FIELD 1, 20 AS GF$, 8 AS SF$, 4 AS QF$, 30 AS MF$
250       FIELD 2, 20 AS FF$, 8 AS RF$, 4 AS PF$, 30 AS NF$
260 REM
270 REM   KOPIEER BESTAND
275 REM
280       FOR R = 1 TO LOF(1)/256
290           GET 1
300           LSET FF$ = GF$
310           LSET RF$ = SF$
320           LSET PF$ = QF$
330           LSET NF$ = MF$
340           PUT 2
350       NEXT R
355 REM
360       CLOSE 1, 2
370       PRINT "KOPIE KLAAR"
380 REM
999       END
```

Toetsvraag

(a) Geef de regelnummers van de programma-opdrachten die met de op p.224/225 genoemde procedurestappen overeenkomen.

(a)
1. 220
2. 230
3. 240
4. 250
5. 280/350
   a. 290
   b. 300 t/m 330
   c. 340
6. 360

Utility-programma 2: muteren

Het tweede utility-programma geeft de mogelijkheid gegevens in een direct-toegankelijk bestand aan de hand van een zogenaamd *menu* te veranderen (muteren). Met 'menu' bedoelen we een aantal keuze-mogelijkheden – zogenaamde *opties* – waaruit door het intoetsen van een getal kan worden geselecteerd.

We gaan uit van het eerder behandelde voorraadbestand. Gemaks-halve nemen we de eerste versie daarvan, zoals geproduceerd door voorbeeldprogramma 1 van § 7.5. De records bestaan daarbij uit slechts twee gegevens: een onderdeelnummer (6 tekens) en een omschrijving (maximaal 20 tekens).

De bedoeling van het programma is de records van het bestand stuk voor stuk op het scherm zichtbaar te maken, en de gebruiker de volgende opties te bieden:

1. Veranderen van zowel onderdeelnummer als omschrijving.
2. Veranderen van onderdeelnummer.
3. Veranderen van omschrijving.
4. Verwijderen van dit record.
5. Geen wijziging van dit record.

De procedure is als volgt:

1. Open het bestand.
2. Leg met behulp van een FIELD-opdracht de bufferstructuur vast.
3. Kies een variabele (bijvoorbeeld R) om het recordnummer bij te houden, en geef deze variabele een beginwaarde van nul.
4. Verhoog de waarde van R met 1.
5. Vergelijk de waarde van R met het totaal aantal records, en bepaal of het laatste record ingelezen is. Zo ja, ga verder bij stap 19.
6. Lees het record met nummer R.
7. Maak het scherm 'schoon'.
8. Druk de inhoud van de recordvelden af.
9. Druk de 'menukaart' – dat wil zeggen de lijst van opties met bijbehorende keuzecijfers – op het scherm af.
10. Laat de gebruiker een optie selecteren.
11. Controleer of de aangegeven optie geldig is.
12. Schrijf subroutine 1 (ten behoeve van het vervangen van het onderdeelnummer):
    a. Voer nieuw onderdeelnummer in.
    b. Controleer op geldigheid.
    c. Ken onderdeelnummer toe aan overeenkomstige buffervariabele.
    d. RETURN.
13. Schrijf subroutine 2 (ten behoeve van het vervangen van de omschrijving):
    a. Voer nieuwe omschrijving in.
    b. Controleer op geldigheid.
    c. Ken omschrijving toe aan overeenkomstige buffervariabele.
    d. RETURN.
14. Bij keuze van optie 1:
    a. Ga naar subroutine 1.
    b. Ga naar subroutine 2.
    c. Schrijf met PUT het record naar zijn oorspronkelijke plaats in het bestand (dus met recordnummer R) terug.
    d. Ga terug naar stap 4.

15. Bij keuze van optie 2: voer stappen 14a, c en d uit.
16. Bij keuze van optie 3: voer stappen 14b, c en d uit.
17. Bij keuze van optie 4:
    a. Vul alle buffervariabelen met lege strings.
    b. Voer stappen 14c en d uit.
18. Bij keuze van optie 5: ga terug naar stap 4.
19. Druk af "EINDE BESTAND" en sluit bestand.

Het volledige programma – gemakshalve weer afgezien van invoercontroles – ziet er als volgt uit:

```
100 REM   UTILITY-PROGRAMMA VOOR MUTEREN
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$, NF$ = ONDERDEELNUMMER (6)
140 REM    - D$, DF$ = OMSCHRIJVING (20)
150 REM    - R$      = RESPONSVARIABELE
160 REM    - R       = RECORDNUMMER
170 REM
180 REM   BESTAND
190 REM    - VOORRAAD (DIRECT-TOEGANKELIJK)
200 REM
210 REM   INITIALISATIES
215 REM
220       OPEN "VOORRAAD" AS 1
230       FIELD 1, 6 AS NF$, 20 AS DF$
240 REM
250       LET R = 0
260       LET R = R + 1
270       IF R = LOF(1)/256 + 1 THEN 480
280       GET 1, R
290 REM
300       CLS
310       PRINT NF$, DF$
320       PRINT "KEUZEMOGELIJKHEDEN: "
330       PRINT "1 = ALLE VELDEN VERANDEREN"
340       PRINT "2 = ALLEEN NUMMER VERANDEREN"
350       PRINT "3 = ALLEEN OMSCHRIJVING VERANDEREN"
360       PRINT "4 = RECORD GEHEEL VERWIJDEREN"
370       PRINT "5 = NIET WIJZIGEN"
380 REM
390       LINE INPUT "GEEF KEUZEGETAL (1-5): "; R$
400       IF VAL(R$) < 1 OR VAL(R$) > 5 THEN
             PRINT "GETAL TUSSEN 1 EN 5 AUB"          : GOTO 390
410       IF VAL(R$) = 1 THEN
             GOSUB 520 :GOSUB 570 :PUT 1, R           : GOTO 260
420       IF VAL(R$) = 2 THEN
             GOSUB 520 :PUT 1, R                      : GOTO 260
430       IF VAL(R$) = 3 THEN
             GOSUB 570 :PUT 1, R                      : GOTO 260
440       IF VAL(R$) = 4 THEN
             LSET NF$ = "" :LSET DF$ = "" :PUT 1, R   : GOTO 260
450       IF VAL(R$) = 5 THEN 260
460       PRINT "GEHEEL GETAL TUSSEN 1 EN 5 AUB"     : GOTO 390
470 REM
```

```
480       PRINT "EINDE BESTAND"
490       CLOSE 1
500       GOTO 999
510 REM
512 REM   SUBROUTINE 1
514 REM
520       LINE INPUT "NIEUW ONDERDEELNR:      "; N$
530 REM   *** EVENTUELE INVOERCONTROLES HIER TUSSENVOEGEN ***
540       LSET NF$ = N$
550       RETURN
560 REM
562 REM   SUBROUTINE 2
564 REM
570       LINE INPUT "NIEUWE OMSCHRIJVING:    "; D$
580 REM   *** EVENTUELE INVOERCONTROLES HIER TUSSENVOEGEN ***
590       LSET DF$ = D$
600       RETURN
610 REM
999       END
```

Toetsvraag

(a) Geef de regelnummers van de programma-opdrachten die met de op p.228/229 genoemde procedurestappen overeenkomen.

(a)
1. 220
2. 230
3. 250
4. 260
5. 270
6. 280
7. 300
8. 310
9. 320 t/m 370
10. 390
11. 400
12. a. 520
    b. 530
    c. 540
    d. 550
13. a. 570
    b. 580
    c. 590
    d. 600
14. a. 410
    b. 410
    c. 410
    d. 410
15. 420
16. 430
17. a. 440
    b. 440
18. 450
19. 480

7.8 CONVERTEREN VAN SEQUENTIEEL NAAR DIRECT-TOEGANKELIJK

Tot slot behandelen we nog een utility-programma voor het omzetten van sequentiële bestanden in direct-toegankelijke.

We gaan daarbij uit van een eenvoudige administratieve toepassing,

waarbij records met de volgende indeling in een sequentieel bestand zijn ondergebracht:

- klantnummer (5 alfanumerieke tekens)
- telefoonnummer van klant (10 alfanumerieke tekens)
- kredietcode (enkele-precisie-waarde van 1 t/m 10)
- uitstaand saldo (enkele-precisie-waarde).

De procedure voor het omzetten van zo'n bestand in een direct-toegankelijk bestand met dezelfde recordindeling is:

1. Open het séquentiële bestand (bestand 1) voor invoer.
2. Open het direct-toegankelijke bestand (bestand 2).
3. Leg met behulp van een FIELD-opdracht de bufferstructuur van bestand 2 vast.
4. Controleer bestand 1 op EOF. Indien einde bestand, ga naar stap 9.
5. Lees een record van bestand 1.
6. Ken de ingelezen waarden toe aan de overeenkomstige buffer-variabelen van bestand 2 (denk daarbij aan de conversie van numerieke waarden met behulp van de MK-functies).
7. Schrijf het record naar bestand 2.
8. Ga terug naar stap 4.
9. Sluit bestanden.

Het eerste deel van het programma is hieronder weergegeven.
U wordt gevraagd de ontbrekende opdrachten te geven.

```
(a) 100 REM   KOPIEREN SEQUENTIEEL NAAR DIRECT-TOEGANKELIJK
    110 REM
    120 REM   VARIABELENLIJST
    130 REM    - N$ = KLANTNUMMER (5)
    140 REM    - T$ = TELEFOONNUMMER (10)
    150 REM    - K  = KREDIETCODE
    160 REM    - S  = UITSTAAND SALDO
    170 REM
    180 REM   BESTANDEN
    190 REM    - KLANT1 (SEQUENTIEEL)
    200 REM    - KLANT2 (DIRECT-TOEGANKELIJK)
    210 REM
    220 REM   INITIALISATIES
    225 REM
    227       MAXFILES = 2
    230       .........................: REM *** OPEN SEQUENTIEEL BESTAND ***

    240       .........................: REM *** OPEN D.T. BESTAND ***

    250       .........................: REM *** DEFINIEER BUFFERSTRUCTUUR ***

    260 REM
```

```
(a) 230       OPEN "KLANT1" FOR INPUT AS 1
    240       OPEN "KLANT2" AS 2
    250       FIELD 2, 5 AS NF$, 10 AS TF$, 4 AS KF$, 4 AS SF$
```

In het volgende deel wordt gecontroleerd op 'einde bestand'. Bij het optreden van EOF moet worden gesprongen naar regel 420. Anders wordt het eerstvolgende record van het sequentiële bestand gelezen. Aan u wederom de vraag om de desbetreffende opdrachten te leveren.

```
(a)   270 REM   LEES SEQUENTIEEL BESTAND
      275 REM
      280       .....................: REM *** TEST OP EOF ***

      290       .....................: REM *** LEES RECORD ***

      300 REM
-----------------------
(a) 280       IF EOF(1) THEN 420
    290       INPUT #1, N$, T$, K, S
```

Met behulp van een GOTO maken we straks een lus waardoor het bovenstaande programmadeel herhaaldelijk wordt uitgevoerd totdat het einde van het bestand wordt bereikt. Ondertussen moet de inhoud van het ingelezen record worden overgebracht naar buffer 2. De eerste opdracht daarvoor is hieronder gegeven. Aan u de taak de andere toekenningsopdrachten te geven.

```
(a) 310 REM   KOPIEER GEGEVENS NAAR BUFFER
    315 REM
    320       LSET NF$ = N$

    330       .....................

    340       .....................

    350       .....................

    360 REM
-------------------------
(a) 330       LSET TF$ = T$
    340       LSET KF$ = MKS$(K)
    350       LSET SF$ = MKS$(S)
```

In het laatste deel van het programma
- wordt de bufferinhoud naar het direct-toegankelijke bestand geschreven;
- wordt met cen sprongopdracht naar regel 280 de lus gesloten;
- worden beide bestanden gesloten.

U wordt gevraagd de opdracht voor de eerstgenoemde actie te geven.

```
(a) 370 REM   SCHRIJF NAAR DIRECT-TOEGANKELIJK BESTAND
    375 REM
    380       .....................: REM *** SCHRIJF-OPDRACHT ***

    390       GOTO 280
    400 REM
    410 REM   AFSLUITING
    415 REM
    420       CLOSE 1, 2
    430       PRINT "KOPIE KLAAR"
    440 REM
    999       END
-------------------------
(a) 380       PUT 2
```

Merk op dat we geen gebruik hebben gemaakt van de mogelijkheid het recordnummer expliciet in de PUT-opdracht aan te geven. Het bijhouden van de recordwijzer hebben we aan de computer overgelaten.

Het volledige programma ziet er als volgt uit:

```
100 REM   KOPIEREN SEQUENTIEEL NAAR DIRECT-TOEGANKELIJK BESTAND
110 REM
120 REM   VARIABELENLIJST
130 REM    - NS = KLANTNUMMER (5)
140 REM    - T$ = TELEFOONNUMMER (10)
150 REM    - K  = KREDIETCODE
160 REM    - S  = UITSTAAND SALDO
170 REM
180 REM   BESTANDEN
190 REM    - KLANT1 (SEQUENTIEEL)
200 REM    - KLANT2 (DIRECT-TOEGANKELIJK)
210 REM
220 REM   INITIALISATIES
225 REM
227       MAXFILES = 2
230       OPEN "KLANT1" FOR INPUT AS 1
240       OPEN "KLANT2" AS 2
250       FIELD 2, 5 AS NF$, 10 AS TF$, 4 AS KF$, 4 AS SF$
260 REM
270 REM   LEES SEQUENTIEEL BESTAND
275 REM
280       IF EOF(1) THEN 420
290       INPUT #1, N$, T$, K, S
300 REM
310 REM   KOPIEER GEGEVENS NAAR BUFFER
315 REM
320       LSET NF$ = N$
330       LSET TF$ = T$
340       LSET KF$ = MKS$(K)
350       LSET SF$ = MKS$(S)
360 REM
370 REM   SCHRIJF NAAR DIRECT-TOEGANKELIJK BESTAND
375 REM
380       PUT 2
390       GOTO 280
400 REM
410 REM   AFSLUITING
415 REM
420       CLOSE 1, 2
430       PRINT "KOPIE KLAAR"
440 REM
999       END
```

Toetsvraag

(a) Geef de regelnummers van de programma-opdrachten die met de op p.231 genoemde procedurestappen overeenkomen.

(a) 1. 230
2. 240
3. 250
4. 280
5. 290
6. 320 t/m 350
7. 380
8. 280 t/m 390
9. 420

7.9 VERANDEREN VAN RECORDVOLGORDE

De recordteller in GET- en PUT-opdrachten biedt de mogelijkheid op eenvoudige wijze records van de ene naar de andere plaats binnen een bestand te dirigeren. Daartoe wordt de gehele buffer als één gegeven gedefinieerd (bij de TRS-80 dus een buffervariabele van 256 tekens, bij de Exidy van 128 tekens). Met een GET wordt de buffer vanuit de gewenste recordpositie gevuld. Een PUT, met daarin de nieuwe recordpositie als tellerwaarde, zorgt vervolgens voor het overbrengen van het record naar de gewenste plaats.

Schematisch kunnen we dit als volgt weergeven:

```
FIELD 1, 256 AS F$
GET 1,  oude plaats
PUT 1,  nieuwe plaats
```

Komt bijvoorbeeld record 43 bij een bestandsmutatie vrij, dan kan het laatste record als volgt naar deze positie verhuizen:

```
420       FIELD 1, 256 AS F$
430       GET 1, LOF(1)/256
440       PUT 1, 43
```

7.10 AFDRUKKEN

Met het volgende programma kan de inhoud van een direct-toegankelijk bestand onafhankelijk van de oorspronkelijke bufferindeling record voor record op het computerscherm of een printer worden afgedrukt.

```
10        LINE INPUT "NAAM VAN FILE "; A$
20        OPEN A$ AS 1
30        FIELD 1, 256 AS B$
40        FOR R = 1 TO LOF(1)/256
50            GET 1, R
60            PRINT B$        : REM *** PRINTER: LPRINT I.P.V. PRINT ***
70        NEXT R
80        END
```

Om het afdruktempo bij uitvoer op een scherm in de hand te houden kan de volgende opdracht worden toegevoegd:

```
65        LINE INPUT ""; C$
```

Het indrukken van een willekeurige toets leidt dan tot het afdrukken van de volgende regel.

7.11 TOETSVRAGEN

1. Schrijf een programma om voor een willekeurig bedrijf een direct-toegankelijk voorraadbestand te maken. Voor ieder artikel wordt één record gebruikt. De recordindeling en de gegevensvolgorde binnen het record is hieronder aangegeven. De getallen tussen haakjes verwijzen naar de maximale stringlengte. Voor getalwaarden kunnen gemakshalve enkele-precisie-grootheden worden gebruikt, ook als integer-grootheden meer voor de hand zouden liggen.

   N$ = artikel (4)
   D$ = omschrijving van produkt (20)
   L$ = leverancier (20)
   K = kritische voorraad (voorraadniveau waarbij bijbesteld moet worden om tijdige levering te waarborgen)
   H1 = te bestellen hoeveelheid
   H2 = momentele voorraad
   P1 = inkoopprijs per verpakkingseenheid
   P2 = verkoopprijs per verpakkingseenheid

   De inleidende module is:

```
100 REM   OPGAVE 1, HOOFDSTUK 7
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$, NB$ = ARTIKELNUMMER (4)
140 REM    - X$, XB$ = OMSCHRIJVING ARTIKEL (20)
150 REM    - L$, LB$ = LEVERANCIER (20)
160 REM    - K , KB$ = KRITISCHE VOORRAAD
170 REM    - B , BB$ = BESTELHOEVEELHEID
180 REM    - M , MB$ = MOMENTELE VOORRAAD
190 REM    - I , IB$ = INKOOPPRIJS
200 REM    - V , VB$ = VERKOOPPRIJS
210 REM    - R$      = RESPONSVARIABELE
220 REM
230 REM   BESTAND
240 REM    - ARTBES (DIRECT-TOEGANKELIJK)
250 REM
```

2. Maak met behulp van het programma van opgave 1 een direct-toegankelijk bestand van 20 records. Verzin daarbij uw eigen gegevens. Van dit bestand zal in hoofdstuk 8 gebruik worden gemaakt.

3. In § 7.8 werd het omzetten van een sequentieel in een direct-toegankelijk bestand behandeld. Schrijf nu een programma dat het direct-toegankelijke bestand kopieert. Het resulterende bestand dient eveneens direct-toegankelijk te zijn.
   De inleidende module is:

```
100 REM   OPGAVE 3, HOOFDSTUK 7
110 REM
120 REM   KOPIEREN DIRECT-TOEGANKELIJK BESTAND
130 REM
140 REM   VARIABELENLIJST
150 REM    - NB$, NC$ = KLANTNUMMER (5)
160 REM    - TB$, TC$ = TELEFOONNUMMER (10)
170 REM    - KB$, KF$ = KREDIETCODE
180 REM    - SB$, SF$ = UITSTAAND SALDO
190 REM
200 REM   BESTANDEN
210 REM    - KLANT1 = ORIGINEEL (DIRECT-TOEGANKELIJK)
220 REM    - KLANT2 = KOPIE     (DIRECT-TOEGANKELIJK)
230 REM
```

4. Schrijf aansluitend op het programma van opgave 3 een programma om ter controle beurtelings een record van het oorspronkelijke bestand en zijn kopie zichtbaar te maken. Na het verschijnen van record 1 van het ene bestand moet record 1 van het andere bestand verschijnen, daarna record 2 van beide bestanden, enz., totdat het einde van de bestanden wordt bereikt.

7.12 ANTWOORDEN OP DE TOETSVRAGEN

1.
```
100 REM   OPGAVE 1, HOOFDSTUK 7
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$, NB$ = ARTIKELNUMMER (4)
140 REM    - X$, XB$ = OMSCHRIJVING ARTIKEL (20)
150 REM    - L$, LB$ = LEVERANCIER (20)
160 REM    - K , KB$ = KRITISCHE VOORRAAD
170 REM    - B , BB$ = BESTELHOEVEELHEID
180 REM    - M , MB$ = MOMENTELE VOORRAAD
190 REM    - I , IB$ = INKOOPPRIJS
200 REM    - V , VB$ = VERKOOPPRIJS
210 REM    - R$      = RESPONSVARIABELE
220 REM
230 REM   BESTAND
240 REM    - ARTBES (DIRECT-TOEGANKELIJK)
250 REM
260 REM   INITIALISATIES
265 REM
270       OPEN "ARTBES" AS 1
280       FIELD 1, 4 AS NB$, 20 AS XB$, 20 AS LB$, 4 AS KB$, 4 AS BB$,
                   4 AS MB$, 4 AS IB$, 4 AS VB$
290 REM
300 REM   GEGEVENSINVOERCONTROLE GEMAKSHALVE WEGGELATEN
305 REM
310       LINE INPUT "ARTIKELNUMMER (4 CIJFERS)     : "; N$
320 REM
330       LINE INPUT "OMSCHRIJVING (MAX. 20 TEKENS) : "; X$
340 REM
350       LINE INPUT "LEVERANCIER (MAX. 20 TEKENS)  : "; L$
360 REM
370       INPUT      "KRITISCHE VOORRAAD            : "; K
380 REM
390       INPUT      "BESTELHOEVEELHEID             : "; B
400 REM
410       INPUT      "MOMENTELE VOORRAAD            : "; M
420 REM
430       INPUT      "INKOOPPRIJS                   : "; I
440 REM
450       INPUT      "VERKOOPPRIJS                  : "; V
460 REM
470 REM   BRENG WAARDEN NAAR BUFFER
480 REM
500       LSET NB$ = N$
510       LSET XB$ = X$
520       LSET LB$ = L$
530       LSET KB$ = MKS$(K)
540       LSET BB$ = MKS$(B)
550       LSET MB$ = MKS$(M)
560       LSET IB$ = MKS$(I)
570       LSET VB$ = MKS$(V)
580 REM
590       PUT 1
600 REM
610       LINE INPUT "MEER GEGEVENS? (J/N)          : "; R$
620       IF LEFT$(R$,1) = "J" THEN CLS  : GOTO 310

630 REM
640 REM   AFSLUITING
645 REM
650       CLOSE
660       PRINT "BESTAND GESLOTEN"
```

2. --

3.
```
100 REM   OPGAVE 3, HOOFDSTUK 7
110 REM
120 REM   KOPIEREN DIRECT-TOEGANKELIJK BESTAND
130 REM
140 REM   VARIABELENLIJST
150 REM    - NB$, NC$ = KLANTNUMMER (5)
160 REM    - TB$, TC$ = TELEFOONNUMMER (10)
170 REM    - KB$, KF$ = KREDIETCODE
180 REM    - SB$, SF$ = UITSTAAND SALDO
190 REM
200 REM   BESTANDEN
210 REM    - KLANT1 = ORIGINEEL (DIRECT-TOEGANKELIJK)
220 REM    - KLANT2 = KOPIE     (DIRECT-TOEGANKELIJK)
230 REM
240 REM   INITIALISATIES
245 REM
247       MAXFILES = 2
250       OPEN "KLANT1" AS 1
260       OPEN "KLANT2" AS 2
270       FIELD 1, 5 AS NB$, 10 AS TB$, 4 AS KB$, 4 AS SB$
280       FIELD 2, 5 AS NC$, 10 AS TC$, 4 AS KF$, 4 AS SF$
290 REM
300 REM   KOPIEER RECORDS
310 REM
320       FOR X = 1 TO LOF(1)/256
330           GET 1
340           LSET NC$ = NB$
350           LSET TC$ = TB$
360           LSET KF$ = KB$
370           LSET SF$ = SB$
380           PUT 2
390       NEXT X
400 REM
410       CLOSE 1, 2
420       PRINT "BESTANDEN GESLOTEN. KOPIE KLAAR"
999       END
```

4.
```
100 REM   OPGAVE 4, HOOFDSTUK 7
110 REM
120 REM   AFDRUKKEN DIRECT-TOEGANKELIJK BESTAND
130 REM
140 REM   VARIABELENLIJST
150 REM    - NB$, NC$ = KLANTNUMMER (5)
160 REM    - TB$, TC$ = TELEFOONNUMMER (10)
170 REM    - KB$, KF$ = KREDIETCODE
180 REM    - SB$, SF$ = UITSTAAND SALDO
190 REM
200 REM   BESTANDEN
210 REM    - KLANT1 = ORIGINEEL (DIRECT-TOEGANKELIJK)
220 REM    - KLANT2 = KOPIE     (DIRECT-TOEGANKELIJK)
230 REM
240 REM   INITIALISATIES
245 REM
247       MAXFILES = 2
250       OPEN "KLANT1" AS 1
260       OPEN "KLANT2" AS 2
270       FIELD 1, 5 AS NB$, 10 AS TB$, 4 AS KB$, 4 AS SB$
280       FIELD 2, 5 AS NC$, 10 AS TC$, 4 AS KF$, 4 AS SF$
290 REM
```

```
300 REM   AFDRUKKEN OP SCHERM
310 REM
320       FOR X = 1 TO LOF(1)/256
330           PRINT "BESTAND 1", "BESTAND 2"
340           GET 1 : GET 2
350           PRINT NB$, NC$
360           PRINT TB$, TC$
370           PRINT CVS(KB$), CVS(KF$)
380           PRINT CVS(SB$), CVS(SF$)
390           LINE INPUT "GEEF ENTER VOOR VOLGEND RECORD "; R$
400       NEXT X
410 REM
420       'CLOSE
430       PRINT: PRINT "EINDE GEGEVENS. BESTANDEN GESLOTEN"
999       END
```

8 TOEPASSINGEN DIRECT-TOEGANKELIJKE BESTANDEN

8.0 DOELSTELLINGEN

In dit hoofdstuk komen min of meer geavanceerde technieken ter sprake voor het werken met direct-toegankelijke bestanden. Na bestudering ervan moet u de behandelde technieken zelfstandig kunnen toepassen in eigen programma's. Daarbij gaat het vooral om het gebruik van sequentiële 'pointer'-bestanden als index voor direct-toegankelijke bestanden. De desbetreffende technieken worden behandeld aan de hand van een tweetal toepassingen.

8.1 VOORRAADBEHEER

Als eerste behandelen we een programma voor voorraadbeheer, zoals dat bijvoorbeeld in een magazijn van een winkelbedrijf zou kunnen plaatsvinden. Daarbij wordt behalve een direct-toegankelijk ook een sequentieel bestand gebruikt. Het sequentiële bestand functioneert daarbij als 'wijzer' oftewel 'pointer' (Engels: to point = (aan)wijzen) naar de records van het direct-toegankelijke bestand. Hoewel we uitgaan van voorraadbeheer gelden de bestandstechnieken uiteraard evengoed voor andere toepassingen, bijvoorbeeld op het gebied van verzendlijsten (mailing lists) of het beheren van klantenkredieten.

In deze toepassing worden de eigenlijke voorraadgegevens opgeslagen in één of meer direct-toegankelijke bestanden. Voor ieder artikel of produkt in zo'n bestand wordt een afzonderlijk record vastgelegd, en wel met de volgende indeling:

P$ = produktnummer (4)
B$ = beschrijving (20)
L$ = leverancier (20)
K = kritische voorraad
H = bestelhoeveelheid
M = momentele voorraad
C = inkoopprijs/kostprijs
V = verkoopprijs per verpakkingseenheid

Stel nu dat de in- en verkoopprijs van een produkt moet worden veranderd. In dat geval dient eerst het desbetreffende record te worden opgespoord. Bij grotere bestanden zal dit gemiddeld relatief veel tijd vergen. Het is dan efficiënter de nummers van de records in het direct-toegankelijk bestand samen met een identificatiegegeven – in dit geval het produktnummer – in een afzonderlijk (sequentieel) bestand op te slaan. Door eerst in zo'n betrekkelijk eenvoudig bestand het produktnummer op te zoeken, kan de plaats van het direct-toegankelijke record sneller worden gevonden.

We nemen hiertoe voorlopig de volgende stappen:

1. Voer het produktnummer en de nieuwe prijsgegevens in.
2. Zoek in het sequentiële bestand het ingevoerde produktnummer, en bepaal het nummer (oftewel *adres*) van het overeenkomstige record in het direct-toegankelijke bestand.
3. Lees het in (2) bedoelde record.
4. Verander de inhoud van de desbetreffende velden (C en V).

We 'vertalen' deze eerste stappen nu in een concreet programmadeel:

```
100 REM   TOEPASSING 1: VOORRAADBEHEER
102 REM
104 REM   MET DIT PROGRAMMA KAN HET VELD VOOR IN- EN VERKOOPPRIJS
106 REM   VAN EEN WILLEKEURIG RECORD IN EEN DIRECT-TOEGANKELIJK
108 REM   WORDEN VERANDERD
110 REM
120 REM   VARIABELENLIJST
130 REM    - R$  = TE WIJZIGEN RECORD (= PRODUKTNUMMER SEQ. BESTAND)
140 REM    - P1$ = PRODUKTNUMMER (4) (IN SEQ. BESTAND)
150 REM    - R   = RECORDNUMMER      (IN SEQ. BESTAND)
160 REM    - P$  = PRODUKTNUMMER (4)
170 REM    - B$  = BESCHRIJVING (20)
180 REM    - L$  = LEVERANCIER  (20)
190 REM    - K   = KRITISCHE VOORRAAD
200 REM    - H   = BESTELHOEVEELHEID
210 REM    - M   = MOMENTELE VOORRAAD
220 REM    - C   = INKOOPPRIJS
230 REM    - V   = VERKOOPPRIJS PER VERPAKKINGSEENHEID
240 REM
```

```
250 REM   BESTANDEN
260 REM    - POINT     = SEQ. "POINTER" BESTAND
270 REM    - VOORRAAD  = DIRECT TOEG. VOORRAADBESTAND
280 REM
290 REM   INITIALISATIE VAN BESTANDEN
300 REM
310       OPEN "POINT" FOR INPUT AS 1
320       OPEN "VOORRAAD" AS 2
330       FIELD 2, 4 AS PF$, 20 AS BF$, 20 AS LF$, 4 AS KF$, 4 AS HF$, 4 AS MF$,
                4 AS CF$, 4 AS VF$
340 REM
350 REM   INVOER WIJZIGEN (ZONDER CONTROLES)
355 REM
360       CLS
370       LINE INPUT "PRODUKTNUMMER       : "; R$
380 REM
390       INPUT      "NIEUWE INKOOPPRIJS  : "; C
400 REM
410       INPUT      "NIEUWE VERKOOPPRIJS : "; V
420 REM
```

Vervolgens doorzoeken we het sequentiële bestand om de plaats van het te muteren record in het direct-toegankelijke bestand te bepalen. Hoewel controles bij de invoer van de nieuwe gegevens gemakshalve achterwege zijn gelaten, is het wel van belang rekening te houden met de invoer van niet-bestaande recordnummers. Daarvoor dienen de opdrachten 680 t/m 710 in het volgende fragment. U wordt gevraagd zelf de ontbrekende opdrachten te leveren.

(a)

```
430 REM   DOORZOEK POINTER-BESTAND
440 REM
450       .........................: REM *** BIJ EOF NAAR 690 ***

460       .........................: REM *** LEES SEQ. RECORD ***

470       .........................: REM *** VERGELIJK INGEVOERD NR. MET NR.
                                          IN BESTAND ***
480 REM




670 REM
680 REM   OPVANG NIET-VOORKOMEND RECORDNUMMER
685 REM
690       PRINT "OPGEGEVEN PRODUKTNUMMER KOMT NIET IN BESTAND VOOR"
700       PRINT "CONTROLEER EN VOER OPNIEUW IN"
710       GOTO 610
720 REM
```

(b) Welke variabele bevat het nummer van het gezochte record in het direct-toegankelijke bestand?

```
(a)  450       IF EOF(1) THEN 690
     460       INPUT #1, P1$, R
     470       IF P1$ <> R$ THEN 450
```

(b) R

Alvorens de inhoud van het direct-toegankelijke record naar de buffer te kopiëren controleren we met LOF of het gevonden recordnummer inderdaad in het direct-toegankelijke bestand bestaat. Zo niet, dan genereren we een foutmelding.

```
(a)  490 REM   LEES RECORD VAN DIRECT-TOEGANKELIJK BESTAND
     495 REM
     500       .....: REM *** NAAR 730 ALS RECORD NIET
                                BESTAAT
     510       .....: REM *** LEES RECORD ***

     520 REM

     730       PRINT "FOUT RECORDNUMMER IN POINTER BESTAND"
     740       GOTO 610
     750 REM
-------------------------
(a)  500       IF R > LOF(2)/256 THEN 730


     510       GET 2, R
```

We converteren de prijsgegevens vervolgens naar stringvariabelen, en plaatsen ze in de buffer. Aan u weer de taak het segment af te maken.

```
(a)  530 REM   VERANDER DE NODIGE BUFFERVELDEN
     535 REM
     540       .....: REM *** KOPIEER INKOOPPRIJS NAAR
                                BUFFER ***
     550       .....: REM *** KOPIEER VERKOOPPRIJS NAAR
                                BUFFER ***
     560 REM
     570 REM   SCHRIJF BUFFER NAAR BESTAND
     575 REM
     580       .....: REM *** BUFFER → D.T. BESTAND ***

     590 REM
-----------------------
(a)  540       LSET CF$ = MKS$(C)
     550       LSET VF$ = MKS$(V)

     580       PUT 2, R
```

Merk op dat in de PUT-opdracht (regel 580) het recordnummer R gespecificeerd is. Zonder deze specificatie zou het eerstvolgende recordnummer worden overschreven.

Het resterende deel van het programma ziet er als volgt uit:

```
600 REM   HERHALEN OF AFSLUITEN
605 REM
610       LINE INPUT "MEER GEGEVENS? (J/N): "; R$
620       IF LEFT$(R$,1) <> "J" THEN 650
630       CLOSE 1: OPEN "POINT" FOR INPUT AS 1: GOTO 360
640 REM
750       CLOSE 1, 2
660       STOP
670       REM
 :                                zie boven voor
 :                                regels 680 t/m 750
750
999       END
```

Toetsvraag

(a) Waarom moet het sequentiële bestand gesloten en opnieuw geopend worden (regel 630)?

(a) De interne recordwijzer van het sequentiële bestand moet opnieuw op het begin van het bestand worden gepositioneerd.

Tot zover het wijzigen van de prijsgegevens. Het programma zou kunnen worden uitgebreid door de mogelijkheid te bieden ook andere recordvelden te wijzigen, bijvoorbeeld leverancier of kritische voorraad. Ook de mogelijkheid om records toe te voegen of te verwijderen zou zinvol zijn. We laten dit aan de geïnteresseerde lezer over!

8.2 PERSOONLIJKE FINANCIËLE ADMINISTRATIE

Het tweede programma zou een onderdeel kunnen vormen van een uitgebreid software-pakket voor een persoonlijk financieel administratiesysteem. De bedoeling van deze toepassing is het verwerken van een zogenaamd *transactiebestand* te laten zien, en te demonstreren hoe boekingsnummers kunnen worden gebruikt om direct-toegankelijke bestanden en records aan te wijzen.

De eerste stap is beslissen welke financiële posten u met de computer wilt bijhouden. Leg vervolgens voor ieder van deze posten een afzonderlijk boekingsnummer (of kostensoort) vast. Ga daarbij uit van verschillende boekingsnummers voor belastbare en niet-belastbare posten. Daardoor kunnen de gegevens ook worden gebruikt voor het invullen van het belastingbiljet.

Voor deze toepassing hebben we de volgende boekingsnummers vastgelegd:

1001 belastbare inkomsten uit arbeid
1002 belastbare inkomsten uit rente
1003 belastbare inkomsten uit effecten
1004 ander belastbaar inkomen
1005 niet-belastbaar inkomen
1006 diverse niet-belastbare bedragen
2001 levensmiddelen
2002 huishoudelijke artikelen anders dan levensmiddelen
2003 hypotheek
2004 gas, water en elektriciteit
2005 gemeentelijke belastingen
2006 telefoon
2007 opstalverzekering
2008 onroerendgoedbelasting
2009 meubilair
2010 vaste auto-onkosten
2011 benzine en smeermiddelen
2012 reparatiekosten auto
2013 parkeren/parkeerboetes
2014 autoverzekering
2015 kleding vader
2016 kleding moeder
2017 kleding jongen(s)
2018 kleding meisje(s)
2019 onderhoud kleding/stomerij
2020 contributies/toegangsbewijzen sport
2021 sportartikelen
2022 tijdschriften en boeken
2023 bioscoop/toneel/concert/opera
2024 alcoholische dranken
2025 restaurant
2026 vakantie-uitgaven
2027 porti
2028 schoolgeld
3001 boekhouder
3002 levensverzekering
3003 ziektekostenverzekering
3004 tandartskostenverzekering
3005 zelf te dragen medische onkosten
3006 medicijnen
3007 cursussen en opleidingen
3008 overige studiekosten
3009 BTW
3010 contributies
3011 inleg spaarrekeningen
3012 investeringen

Voor ieder boekingsnummer wordt een afzonderlijk record bijgehouden.

Het boekingsnummer heeft een belangrijk doel. Het is namelijk zo gekozen dat het eerste cijfer het nummer aangeeft van het (direct-toegankelijke) bestand waarin de bijbehorende gegevens zijn ondergebracht. Al deze bestanden hebben de naam BUDGET gevolgd door het desbetreffende cijfer. Uit de lijst van boekingsnummers blijkt dus dat er in deze toepassing sprake is van drie bestanden.

Toetsvraag

(a) Welk bestand bevat de gegevens over cursussen en opleiding?

(a) BUDGET3

De laatste drie cijfers van het boekingsnummer geven het volgnummer aan van het record dat voor dat boekingsnummer wordt gebruikt. Het record met gegevens over investeringen bevindt zich dus in BUDGET3, en heeft het volgnummer 12.

Gemakshalve worden de boekingsnummers steeds als een stringwaarde ingevoerd. Met behulp van de functies LEFT$ en RIGHT$ kunnen het bestandsnummer en het recordnummer betrekkelijk eenvoudig gescheiden worden.

Het volgende schema geeft de indeling van de records, en de naam en lengte van de bijbehorende buffervariabelen:

```
N$  = BOEKINGSNUMMER (4)
AF$ = BESCHRIJVING (20)
BF$ = BEGROTINGSBEDRAG (JAARBASIS) (4)
SF$ = SALDO (INKOMSTEN/UITGAVEN V.A. 1 JANUARI) (4)
```

Iedere maand wordt een nieuw sequentieel bestand gemaakt. Voor de maand januari heeft dit bestand de naam MAAND1, voor februari de naam MAAND2, enz. In zo'n bestand worden de gegevens over alle inkomsten en uitgaven voor die maand – de zogenaamde *transacties* – bijgehouden. We noemen zo'n bestand dan ook een *transactiebestand*. Aan het einde van iedere maand wordt het transactiebestand verwerkt en worden de direct-toegankelijke bestanden bijgewerkt.

De recordindeling voor zo'n transactiebestand met de bijbehorende buffervariabelen is als volgt*:

---

* Terwille van de algemeenheid, en om een enigszins realistisch beeld te geven, zijn in dit bestand meer gegevenssoorten gedefinieerd dan we voor onze toepassing nodig hebben.

K = NUMMER BETAALKAART/KASBEWIJS
D$ = DATUM (6)
X$ = NAAM BEGUNSTIGDE/INKOMSTENBRON (20)
N$ = BOEKINGSNUMMER (4)
G = BEDRAG IN HFL

We zetten de periodiek te ondernemen acties nog even op een rijtje:

1. Aan het begin van ieder jaar worden direct-toegankelijke bestanden gemaakt (BUDGETm) waarin de beginstatus van alle boekingen wordt opgeslagen. Deze beginstatus omvat o.a. een begrotingsbedrag (op jaarbasis) voor de desbetreffende rekening.
2. Iedere maand wordt een sequentieel bestand gemaakt (MAANDn) waarin alle inkomsten en uitgaven in die maand worden opgeslagen.
3. Iedere maand worden de direct-toegankelijke bestanden bijgewerkt aan de hand van het overeenkomstige sequentiële bestand voor die maand.
4. Op elk gewenst tijdstip kan een overzicht worden gemaakt van de BUDGET-bestanden.

We zijn nu zo ver dat we de inleidende module kunnen schrijven. Deze ziet er als volgt uit:

```
100 REM   PERSOONLIJKE FINANCIELE ADMINISTRATIE
110 REM   TOEPASSING SEQUENTIEEL/DIRECT-TOEGANKELIJK BESTAND
120 REM
130 REM   VARIABELENLIJST
140 REM    - N$  = N1$ = BOEKINGSNUMMER (4)
150 REM    - B$  = BESCHRIJVING (20)
160 REM    - D$  = DATUM (6)
170 REM    - X$  = NAAM BEGUNSTIGDE/INKOMSTENBRON (20)
180 REM    - M$  = GESELECTEERDE MAAND (20)
190 REM    - K   = NUMMER BETAALKAART/KASBEWIJS
200 REM    - G   = BEDRAG BETAALKAART/KASBEWIJS
210 REM    - B   = BEGROTINGSBEDRAG
220 REM    - S   = SALDO (4)
230 REM    - F$  = NAAM SEQUENTIEEL BESTAND
240 REM    - F1$ = F2$ = NAAM D.T. BESTAND
250 REM    - R   = RECORDNUMMER
260 REM
270 REM   BESTANDEN
280 REM    - MAAND#  = SEQ. TRANSACTIEBESTAND (# = 1..12)
290 REM    - BUDGET# = D.T. BESTAND (# = VOLGNUMMER)
300 REM
310 REM   INITIALISATIE BESTANDEN
320 REM
325       MAXFILES = 2
330       LINE INPUT "NUMMER TE VERWERKEN MAAND: "; M$
340 REM
350       IF F$ = "MAAND" + M$
360 REM
370       OPEN F$ FOR INPUT AS 1
380 REM   D.T. BESTAND EN BUFFER WORDEN VOOR IEDERE TRANSACTIE GEOPEND/INGERICHT
390 REM
```

Toetsvraag

(a) Welke waarde krijgt F$ als de gebruiker naar aanleiding van opdracht 330 de waarde 3 invoert?

(a) MAAND3

N.B.: Merk op dat de ingevoerde waarde aan een *string*variabele wordt toegekend. Toekenning aan een numerieke variabele, met daarna conversie naar een stringwaarde met behulp van de STR$-functie zou resulteren in MAAND 3 (STR$ geeft een voorafgaande spatie als de getalwaarde positief is). Het programma zou dit bestand niet kunnen vinden.

We gaan verder met het verwerken van de transacties.
In het volgende fragment zorgt regel 430 voor de gebruikelijke controle op het einde van het bestand. Bij EOF kan het programma beëindigd worden (de transacties voor de desbetreffende maand zijn dan in de direct-toegankelijke bestanden verwerkt). In regel 440 wordt een volledig transactierecord gelezen, waarin o.a. het boekingsnummer (N$) staat). In regel 470 moet aan de hand van het eerste cijfer van N$ het nummer van het overeenkomstige direct-toegankelijke bestand worden bepaald. U wordt gevraagd de opdracht daarvoor te geven.

```
(a)  400 REM   VERWERKING TRANSACTIES
     410 REM
     420 REM   LEES TRANSACTIERECORD SEQ. BESTAND
     425 REM
     430       IF EOF(1) THEN 730
     440       INPUT #1, K, D$, X$, N$, G
     450 REM
     460 REM   BEPAAL NUMMER D.T. BESTAND EN INITIALISEER
     465 REM
     470       .....: REM *** BESTANDNUMMER → F2$

     480 REM
     490       LET F1$ = "BUDGET" + F2$
     500       OPEN F1$ AS 2
     510       FIELD 2, 4 AS NF$, 20 AS GF$, 4 AS BF$, 4 AS SF$
     520 REM
```

```
(a)  470       LET F2$ = LEFT$(N$,1)
```

In het volgende gedeelte moet op soortgelijke wijze uit het boekingsnummer het recordnummer worden bepaald en naar een getalwaarde worden geconverteerd. Dit getal wordt in regel 550 gecontroleerd op geldigheid, waarbij een ongeldig getal in het afdrukken van een foutboodschap en het afsluiten van het BUDGET#-bestand resulteert.

```
(a)   530 REM   BEPAAL RECORDNUMMER EN CONVERTEER NAAR GETALWAARDE
      535 REM
      540       .........................: REM *** RECORDNUMMER → R ***

      550       IF R <= LOF(2)/256 THEN 620
      560       PRINT "ONGELDIG NUMMER OP BETAALKAART OF KASBEWIJS ("; K; ")"
      570       PRINT "TRANSACTIE NIET VERWERKT"
      580       CLOSE 2
      590       GOTO 430                 : REM *** TERUG VOOR VOLGENDE TRANSACTIE ***
      600 REM

--------------------------
(a)   540       LET R = VAL(RIGHT$(N$,3))
```

In het laatste deel van het programma wordt het bij te werken record van het direct-toegankelijke bestand naar de buffer gekopieerd, bijgewerkt en weer teruggeschreven. Geef zelf de nodige opdrachten daarvoor.

```
(a)   610 REM   KOPIEER RECORD VAN DIRECT-TOEGANKELIJK BESTAND NAAR BUFFER,
      612 REM   MUTEER EN SCHRIJF TERUG
      615 REM
      620       .........................: REM *** KOPIEER RECORD NAAR BUFFER ***

      630       .........................: REM *** CONVERTEER SALDO NAAR
                                                 GETALWAARDE ***
      640       .........................: REM *** BEPAAL NIEUW SALDO ***

      650       .........................: REM *** MET LSET NAAR BUFFER ***

      660       .........................: REM *** BUFFER NAAR BESTAND ***

      665 REM
      670 REM   BESTAND AFSLUITEN EN TERUG VOOR VOLGENDE TRANSACTIE
      680 REM
      690       CLOSE 2
      700       GOTO 430
      710 REM
      720 REM   ALLE BESTANDEN AFSLUITEN EN STOPPEN
      725 REM
      730       CLOSE
      999       END

--------------------------
(a)   620       GET 2, R
      630       LET  S   = CVS(SF$)
      640       LET  S   = S + G
      650       LSET SF$ = MKS$(S)
      660       PUT 2, R
```

Hiermee zijn we aan het einde van dit programma gekomen. Een kritische noot mag echter niet ontbreken. Het programma leest namelijk doorlopend sequentiële transactierecords, en verwerkt ze in de desbetreffende direct-toegankelijke bestanden, tot het einde van het transactiebestand wordt bereikt. Daarbij worden de betrokken

bestanden steeds opnieuw geopend en gesloten, hetgeen voor het operating system de nodige rompslomp – oftewel *overhead* – met zich meebrengt. Dit zou te verbeteren zijn door de bestanden eerst te sorteren. Bovendien zouden controles kunnen worden ingebouwd die het onnodig sluiten en weer openen van hetzelfde direct-toegankelijke bestand voorkómen. Dit is in het volgende (alternatieve) fragment verwerkt. Een sterretje voor een regelnummer geeft een wijziging aan.

```
  400 REM   VERWERKING TRANSACTIES
  402 REM
* 404       LET F3$ = "X" : REM *** DUMMY WAARDE VOOR EERSTE KEER ***
  410 REM
  420 REM   LEES TRANSACTIERECORDS SEQ. BESTAND
  425 REM
  430       IF EOF(1) THEN 730
  440       INPUT #1, K, D$, X$, N$, G
  450 REM
  460 REM   BEPAAL NUMMER D.T. BESTAND EN INITIALISEER
  465 REM
  470       LET F2$ = LEFT$(N$,1)
* 480       IF  F2$ = F3$ THEN 540 : REM *** BIJ IDENTIEK BESTAND OPENEN OVERSLAAN ***
* 485       LET F3$ = F2$          : REM *** F3$ VOOR VOLGENDE KEER AANPASSEN ***
  490       LET F1$ = "BUDGET" + F2$
  500 REM   OPEN F1$ AS 2
  510       FIELD 2, 4 AS NF$, 20 AS GF$, 4 AS BF$, 4 AS SF$
  520 REM
  530 REM   BEPAAL RECORDNUMMER EN CONVERTEER NAAR GETALWAARDE
  535 REM
  540       LET R = VAL(RIGHT$(N$,3))
  550       IF R <= LOF(2)/256 THEN 620
  560       PRINT "ONGELDIG NUMMER OP BETAALKAART OF KASBEWIJS ("; K; ")"
  570       PRINT "TRANSACTIE NIET VERWERKT"
* 580 REM
  590       GOTO 430                : REM *** TERUG VOOR VOLGENDE TRANSACTIE ***
  600 REM
  610 REM   KOPIEER RECORD VAN DIRECT-TOEGANKELIJK BESTAND NAAR BUFFER; MUTEER
                EN SCHRIJF TERUG
  615 REM
  620       GET 2, R
  630       LET  S   = CVS(SF$)
  640       LET  S   = S + G
  650       LSET SF$ = MKS$(E)
  660       PUT 2, R
  665 REM
* 670       GOTO 430
  680 REM
* 690 REM   TERUG VOOR VOLGENDE TRANSACTIE
* 695 REM
  700       GOTO 430
  710 REM
```

Toetsvraag

(a) Slechts een deel van deze toepassing is nu gerealiseerd. Welke andere programma's zijn nodig?

(a) Programma's voor:
1. Het aanmaken van de MAAND#-bestanden aan de hand van kasboekgegevens, aantekeningen van betaalkaarten, enz.
2. Het muteren van de MAAND#-bestanden.
3. Het initialiseren van de BUDGET#-bestanden.
4. Het toevoegen van nieuwe rekeningen aan de BUDGET#-bestanden.
5. Het afdrukken van de BUDGET#-bestanden.
6. Het controleren op mogelijke saldo-overschrijdingen (zogenaamde *uitzonderingsrapportage*).

8.3 TEN SLOTTE

Wat direct-toegankelijke bestanden betreft hebben we ons in dit boek meestal beperkt tot records met vaste lengte. In § 7.3 werd reeds gezegd dat MSX BASIC voorziet in de mogelijkheid van records met variabele lengte. Daardoor kan de beschikbare schijfruimte efficiënter worden benut. Een andere mogelijkheid om de schijfbezetting te optimaliseren is verschillende logische records onder te brengen in één fysiek record. Aangezien dit in de meeste gebruikershandleidingen uitvoerig wordt beschreven, gaan we er hier niet op in.
We hopen overigens dat u - dankzij dit boek - in staat zult zijn uw eigen gebruikershandleiding beter te begrijpen, en dat u meer oog hebt gekregen voor de mogelijkheden van uw computer!

8.4 TOETSVRAGEN

1. De eerste toepassing in dit hoofdstuk betrof voorraadbeheer. Hierbij werden de eigenlijke voorraadgegevens opgeslagen in een direct-toegankelijk bestand. Aan de hand van het artikel- of produktnummer kon het adres van het bijbehorende record in het direct-toegankelijke bestand worden bepaald.
Neem nu aan dat dit systeem met een derde bestand is uitgebreid, waarin alle handelingen of *transacties* betreffende het voorraadbestand zijn geregistreerd. Daarbij zijn twee transactietypen gedefinieerd:

   type 1 = transactie waarbij aan de voorraad wordt toegevoegd;

   type 2 = transactie waarbij uit de voorraad wordt weggenomen.

   Het transactiebestand is sequentieel georganiseerd en heeft de volgende recordindeling:

```
T   = TRANSACTIETYPE (1 OF 2)
J$  = DATUM (6)
N$  = BONNUMMER (5)
P2$ = PRODUKT-/ARTIKELNUMMER (4)
H1  = HOEVEELHEID (TOENAME OF AFNAME VAN VOORRAAD)
```

Schrijf een programma dat de hoeveelheden (H1) van het transactiebestand verwerkt in het voorraadbestand.

2. Breid het programma van opgave 1 zo uit dat, na het verwerken van alle transacties, het voorraadbestand kan worden onderzocht op produkten waarvan de momentele voorraad kleiner is dan de kritische voorraad. Ten behoeve van bijbestellingen van deze produkten dient op verzoek een lijst te worden gemaakt.

8.5 ANTWOORDEN OP DE TOETSVRAGEN

1.
```
100 REM   OPGAVE 1, HOOFDSTUK 8
105 REM
110 REM   DIT PROGRAMMA WERKT DE MOMENTELE VOORRADEN
120 REM   IN HET VOORRAADBESTAND BIJ AAN DE HAND VAN
130 REM   DE HOEVEELHEDEN IN HET TRANSACTIEBESTAND
135 REM
140 REM   VARIABELENLIJST
140 REM    - R$  = TE WIJZIGEN RECORD/INVOER
150 REM    - P1$ = PRODUKTNUMMER (4)
160 REM    - R   = RECORDNUMMER
170 REM    - P$  = PRODUKTNUMMER (4)
190 REM    - D$  = BESCHRIJVING (20)
200 REM    - L$  = LEVERANCIER  (20)
210 REM    - K   = KRITISCHE VOORRAAD
220 REM    - H   = BESTELHOEVEELHEID
230 REM    - M   = MOMENTELE VOORRAAD
240 REM    - C   = INKOOPPRIJS
250 REM    - V   = VERKOOPPRIJS PER VERPAKKINGSEENHEID
260 REM    - T   = TRANSACTIETYPE
270 REM    - J$  = DATUM (6)
280 REM    - N$  = BONNUMMER (5)
290 REM    - P2$ = PRODUKT-/ARTIKELNUMMER (4)
300 REM    - H1  = HOEVEELHEID (TOENAME OF AFNAME)
310 REM
320 REM   BESTANDEN
330 REM    - POINT    = SEQUENTIEEL "POINTER" BESTAND
340 REM    - VOORRAAD = DIRECT TOEG. VOORRAADBESTAND
350 REM    - TRANSACT = SEQUENTIEEL TRANSACTIEBESTAND
360 REM
```

```
370 REM   INITIALISATIES
375 REM
377       MAXFILES = 3
380       OPEN "POINT" FOR INPUT AS 1
390       OPEN "VOORRAAD" AS 2
400       FIELD 2, 4 AS PF$, 20 AS DF$, 20 AS LF$, 4 AS KF$, 4 AS JF$,
                   4 AS MF$, 4 AS CF$, 4 AS VF$
410       OPEN "TRANSACT" FOR INPUT AS 3
420 REM
430 REM   LEES TRANSACTIEBESTAND
435 REM
440       IF EOF(3) THEN 850
450       INPUT #3, T, J$, N$, P2$, H1
460 REM
470 REM   LEES RECORD POINTER BESTAND
475 REM
480       IF EOF(1) THEN 790
490       INPUT #1, P1$, R
500       IF P1$ <> P2$ THEN 480
510 REM
520 REM   LEES VOORRAADRECORD EN WERK HOEVEELHEID BIJ
525 REM
530       IF R > LOF(2)/256 THEN 750
540       GET 2, R
550       LET M = CVS(MF$)
560       IF T = 2 THEN 600
570       LET M = M + H1
580       GOTO 620
590 REM
600       LET M = M - H1
610       IF M < 0 THEN LET M = M + H1 : GOTO 690      : REM *** FOUT ***
620       LET MF$ = MKS$(M)
630       PUT 2, R
640 REM
650 REM   STEL POINTER BESTAND IN OP BEGIN EN HERHAAL
655 REM
660       CLOSE 1 : OPEN "POINT" FOR INPUT AS 1 : GOTO 440
670 REM
680 REM   FOUTOPVANG (1)
685 REM
690       PRINT "NEGATIEVE HOEVEELHEID. TRANSACTIE D.D. ";
700       PRINT J$; " MET PRODUKTNR. "; P2$;
710       PRINT "NIET VERWERKT"
720       GOTO 660
730 REM
740 REM   FOUTOPVANG (2)
745 REM
750       PRINT "FOUT RECORDNUMMER IN POINTER BESTAND"
760       GOTO 800
770 REM
780 REM   FOUTOPVANG (3)
785 REM
790       PRINT "PRODUKTNUMMER NIET IN TRANSACTIEBESTAND";
800       PRINT "TRANSACTIE D.D. "; J$; "MET PRODUKTNUMMER "; P2$;
810       PRINT "NIET VERWERKT"
820       GOTO 440
830 REM
840 REM   AFSLUITING
845 REM
850       CLOSE
```

2.
```
860 REM
870 REM   BESTELLIJST
880 REM
890       LINE INPUT "BESTELLIJST MAKEN? (J/N): "; R$
900       IF LEFT$(R$,1) <> "J" THEN 1060
910       OPEN "VOORRAAD" AS 2
920       FOR X = 1 TO LOF(2)/256
930           GET 2
940           LET K = CVS(KF$)
950           LET J = CVS(JF$)
960           LET M = CVS(MF$)
970           LET C = CVS(CF$)
980           IF  M > K THEN 1050
990           PRINT "BESTELLING PRODUKTNR. "; PF$
1000          PRINT "OMSCHRIJVING       : "; DF$
1010          PRINT "LEVERANCIER        : "; LF$
1020          PRINT "HUIDIGE VOORRAAD   : "; M
1030          PRINT "BESTELHOEVEELHEID  : "; H
1040          PRINT "VORIGE INKOOPPRIJS : "; C
1050      NEXT X
1060      CLOSE
9999      END
```

EINDTOETS

1. Schrijf een programma waarmee een sequentieel schijfbestand TELEF1 kan worden gemaakt. De records moeten als een enkele string worden opgebouwd uit de volgende gegevens:
   - achternaam (maximaal 15 tekens)
   - voornaam/voorletter(s) (maximaal 15 tekens)
   - kengetal telefoon (maximaal 5 cijfers)
   - abonneenummer telefoon (maximaal 7 cijfers).

2. Schrijf een programma waarmee de inhoud van het bestand TELEF1 (zie opgave 1) kan worden afgedrukt. De recordvelden dienen daarbij afzonderlijk onder de volgende kolomhoofden te verschijnen:

   ACHTERNAAM VOORNAAM KENGETAL ABONNEENR.

   eerste deel:

```
100 REM   EINDTOETS, OPGAVE 2
110 REM
120 REM   VARIABELENLIJST
130 REM    - R$ = SAMENGESTELDE RECORDSTRING
140 REM
150 REM   BESTAND
160 REM    - TELEF1 SEQUENTIEEL/SCHIJF
165 REM
```

3. Schrijf een programma dat alle records van TELEF1 (zie opgave 1) met een gegeven kengetal selecteert en de inhoud daarvan afdrukt. Het kengetal moet door de gebruiker worden ingevoerd. Na het verwerken van een serie records dient het programma op verzoek een nieuwe serie records, met een ander kengetal, te kunnen selecteren en afdrukken.

   eerste deel:

```
100 REM   EINDTOETS, OPGAVE 3
110 REM
120 REM   VARIABELENLIJST
130 REM    - K$ = KENGETAL
140 REM    - R$ = SAMENGESTELDE RECORDSTRING
150 REM    - D$ = DIALOOG
160 REM
170 REM   BESTANDEN
180 REM    - TELEF1 SEQUENTIEEL/SCHIJF
185 REM
```

4. Schrijf een programma waarmee de inhoud van het direct-toegankelijke bestand ARTBES (zie § 7.11, opgaven 1 en 2) kan worden afgedrukt.

   eerste deel:

```
100 REM   EINDTOETS, OPGAVE 4
110 REM
120 REM   VARIABELENLIJST
130 REM    - NB$      = ARTIKELNUMMER (4)
140 REM    - XB$      = OMSCHRIJVING (20)
150 REM    - LB$      = LEVERANCIER  (20)
160 REM    - K,  KB$ = KRITISCHE VOORRAAD
170 REM    - B,  BB$ = BESTELHOEVEELHEID
180 REM    - M,  MB$ = MOMENTELE VOORRAAD
190 REM    - I,  IB$ = INKOOPPRIJS
200 REM    - V,  VB$ = EENHEIDSPRIJS (VERKOOP)
210 REM    - R$       = RESPONS
220 REM
230 REM   BESTAND
240 REM    - ARTBES (DIRECT-TOEGANKELIJK)
250 REM
```

5. Schrijf een programma om in het bestand ARTBES (zie vorige opgave) de eenheidsprijs voor verkoop met 10% te verhogen. Het programma dient het artikelnummer, de oude prijs en de nieuwe prijs af te drukken.

   eerste deel:

```
100 REM   EINDTOETS, OPGAVE 5
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$, NB$ = ARTIKELNUMMER (4)
140 REM    - X$, XB$ = OMSCHRIJVING (20)
150 REM    - L$, LB$ = LEVERANCIER  (20)
160 REM    - K,  KB$ = KRITISCHE VOORRAAD
170 REM    - B,  BB$ = BESTELHOEVEELHEID
180 REM    - M,  MB$ = MOMENTELE VOORRAAD
190 REM    - I,  IB$ = INKOOPPRIJS
200 REM    - V,  VB$ = EENHEIDSPRIJS (VERKOOP)
210 REM    - R$       = RESPONS
220 REM    - V1       = NIEUWE EENHEIDSPRIJS
230 REM
240 REM   BESTAND
250 REM    - ARTBES (DIRECT TOEGANKELIJK)
260 REM
```

1.
```
100 REM   EINDTOETS, OPGAVE 1
110 REM
120 REM   VARIABELENLIJST
130 REM    - A$ = ACHTERNAAM                  (15)
140 REM    - V$ = VOORNAAM OF -LETTER(S)      (15)
150 REM    - K$ = KENGETAL                    (5)
160 REM    - N$ = ABONNEENUMMER               (7)
170 REM    - R$ = SAMENGESTELDE RECORDSTRING  (42)
180 REM
190 REM   BESTAND
200 REM    - TELEF1 = SEQUENTIEEL SCHIJFBESTAND
210 REM
220 REM   INITIALISATIES
230 REM
240       CLEAR 500
250       OPEN "TELEF1" FOR OUTPUT AS 1
260 REM
270 REM   GEGEVENSINVOER
280 REM
290       PRINT "GEEF 'STOP' ALS U KLAAR BENT MET INVOER"
300 REM   LEZEN, CONTROLEREN EN "UITVULLEN" VAN ACHTERNAAM
310       GOSUB 520
312       IF LEFT$(A$,4) = "STOP" THEN 480
320 REM   IDEM, VOORNAAM
330       GOSUB 620
335 REM
340       IDEM, KENGETAL
350       GOSUB 720
355 REM
360 REM   IDEM, ABONNEENUMMER
370       GOSUB 820
380 REM
390 REM   RECORD OPBOUWEN, WEGSCHRIJVEN EN HERHALEN
400 REM
410       LET R$ = A$ + V$ + K$ + N$
420       PRINT #1, R$
430       CLS
440       GOTO 290
450 REM
460 REM   AFSLUITING
470 REM
480       CLOSE
490       PRINT "BESTAND GESLOTEN"
492 REM
494       END
500 REM   MODULE VOOR ACHTERNAAM
510 REM
520       LINE INPUT "ACHTERNAAM (MAX. 15 TEKENS): "; A$
530       IF LEN(A$) = 0 THEN
             PRINT "GEEN ACHTERNAAM INGEVOERD"       : GOTO 520
540       IF LEN(A$) > 15 THEN
             PRINT "MAXIMAAL 15 TEKENS. OPNIEUW AUB" : GOTO 520
550       IF LEN (A$) < 15 THEN
             LET A$ = A$ + " "                       : GOTO 550
560       RETURN
570 REM
```

```
600 REM   CONTROLEMODULE VOORNAAM/-LETTER(S)
610 REM
620       LINE INPUT "VOORNAAM/-LETTER(S) (MAX. 15 TEKENS) : "; V$
630       IF LEN(V$) = 0 THEN
             PRINT "GEEN VOORNAAM/-LETTER(S) INGEVOERD"  : GOTO 620
640       IF LEN(V$) > 15 THEN
             PRINT "MAXIMAAL 15 TEKENS. OPNIEUW AUB"     : GOTO 620
650       IF LEN(V$) < 15 THEN
             LET V$ = V$ + " "                           : GOTO 650
660       RETURN
670 REM
700 REM   CONTROLEMODULE KENGETAL
710 REM
720       LINE INPUT "KENGETAL (MAX. 5 CIJFERS)           : "; K$
730       IF LEN(K$) = 0 THEN
             PRINT "GEEN KENGETAL INGEVOERD"             : GOTO 720
740       IF VAL(K$) = 0 THEN
             PRINT "UITSLUITEND CIJFERS AUB"             : GOTO 720
750       IF LEN(K$) > 5 THEN
             PRINT "MAXIMAAL 5 CIJFERS. OPNIEUW AUB"     : GOTO 720
760       IF LEN(K$) < 5 THEN
             LET K$ = K$ + " "                           : GOTO 760
770       RETURN
780 REM
800 REM   CONTROLEMODULE ABONNEENUMMER
810 REM
820       LINE INPUT "ABONNEENUMMER (MAX. 7 CIJFERS)      : "; N$
830       IF LEN(N$) = 0 THEN
             PRINT "GEEN ABONNEENUMMER INGEVOERD"        : GOTO 820
840       IF VAL(N$) = 0 THEN
             PRINT "UITSLUITEND CIJFERS AUB"             : GOTO 820
850       IF LEN(N$) > 7 THEN
             PRINT "MAXIMAAL 7 CIJFERS. OPNIEUW AUB"     : GOTO 820
860       IF LEN(N$) < 7 THEN
             LET N$ = N$ + " "                           : GOTO 860
870       RETURN
880 REM
999       END
```

N.B.: In plaats van in afzonderlijke modules kunnen de controles uiteraard ook rechtstreeks in de hoofdtekst van het programma worden opgenomen. Deze zogenaamde *modulaire* opbouw is ons inziens echter duidelijker.

2.
```
100 REM   EINDTOETS, OPGAVE 2
110 REM
120 REM   VARIABELENLIJST
130 REM    - R$ = SAMENGESTELDE RECORDSTRING
140 REM
150 REM   BESTAND
160 REM    - TELEF1 SEQUENTIEEL/SCHIJF
165 REM
170 REM   INITIALISATIES
175 REM
180       CLEAR 500
190       OPEN "TELEF1" FOR INPUT AS 1
195 REM
200 REM   VERWERKING
205 REM
210       PRINT "ACHTERNAA""; TAB(16); "VOORNAAM"; TAB(30) "TELEFOON"
220       PRINT
230 REM
240       IF EOF(1) THEN 290
250          INPUT #1, R$
260          PRINT LEFT$(R$,15); TAB(16); MID$(R$,16,11); TAB(26);
                  MID$(R$,31,5); "-"; MID$(R$,36,7)
270       GOTO 240
280 REM
290 REM   AFSLUITING
300 REM
310       CLOSE 1
320       PRINT "EINDE GEGEVENS; BESTAND GESLOTEN"
999       END
```

3.
```
100 REM   EINDTOETS, OPGAVE 3
110 REM
120 REM   VARIABELENLIJST
130 REM    - K$ = KENGETAL
140 REM    - R$ = SAMENGESTELDE RECORDSTRING
150 REM    - D$ = DIALOOG
160 REM
170 REM   BESTANDEN
180 REM    - TELEF1 SEQUENTIEEL/SCHIJF
185 REM
190 REM   INITIALISATIES
200 REM
210       CLEAR 500
220       OPEN "TELEF1" FOR INPUT AS 1
230 REM
240 REM   INVOER KENGETAL
250 REM
260       LINE INPUT "KENGETAL(3 OF 5 CIJFERS): "; K$
265       IF LEN(K$) = 3 THEN K$ = K$ + "  " : REM *** 2 SPATIES TOEVOEGEN ***
270       IF LEN(K$) <> 5 THEN PRINT "INVOERFOUT.";
             "INGEVOERD: "; K$; : GOTO 260
280 REM
290 REM   VERWERKING
300 REM
310       CLS
320       PRINT "ACHTERNAAM"; TAB(16); "VOORNAAM"; TAB(30); "TELEFOON"
330       PRINT
340       IF EOF(1) THEN 410
350          INPUT #1, R$
360          IF K$ <> MID$(R$,31,5) THEN 340
370          PRINT LEFT$(R$,15); TAB(16); MID$(R$,16,11); TAB(26); K$; "-";
                MID$(R$,36,7)
380       GOTO 340
390 REM
400       PRINT
410       PRINT "EINDE VOOR KENGETAL "; K$
420       PRINT
430       LINE INPUT "NOG EEN KENGETAL? (J/N): "; D$
440       IF LEFT$(D$,1) = "N" THEN 520
450 REM
460 REM   POINTER POSITIONEREN EN HERHALEN
470 REM
480       CLOSE 1 : OPEN "TELEF1" FOR INPUT AS 1 : GOTO 260
490 REM
500 REM   AFSLUITING
510 REM
520       CLOSE 1
530       PRINT
540       PRINT "BESTAND GESLOTEN"
999       END
```

4.
```
100 REM   EINDTOETS, OPGAVE 4
110 REM
120 REM   VARIABELENLIJST
130 REM    - NB$      = ARTIKELNUMMER (4)
140 REM    - XB$      = OMSCHRIJVING (20)
150 REM    - LB$      = LEVERANCIER  (20)
160 REM    - K,  KB$ = KRITISCHE VOORRAAD
170 REM    - B,  BB$ = BESTELHOEVEELHEID
180 REM    - M,  MB$ = MOMENTELE VOORRAAD
190 REM    - I,  IB$ = INKOOPPRIJS
200 REM    - V,  VB$ = EENHEIDSPRIJS (VERKOOP)
210 REM    - R$       = RESPONS
220 REM
230 REM   BESTAND
240 REM    - ARTBES (DIRECT-TOEGANKELIJK)
250 REM
260 REM   INITIALISATIES
270 REM
280       OPEN "ARTBES" AS 1
290       FIELD 1, 4 AS NB$, 20 AS XB$, 20 AS LB$, 4 AS KB$, 4 AS BB$,
                   4 AS MB$, 4 AS IB$, 4 AS VB$
300 REM
310 REM   LEES RECORDS, CONVERTEER EN DRUK AF
320 REM
330       PRINT "GEEF  RETURN  VOOR VOLGEND RECORD"
340       PRINT
350       FOR X = 1 TO LOF(1)/256
360           GET 1
370           LET K = CVS(KB$)
380           LET B = CVS(BB$)
390           LET M = CVS(MB$)
400           LET I = CVS(IB$)
410           LET V = CVS(VB$)
420           PRINT NB$; XB$; LB$; K; B; M; I; V
430           LINE INPUT R$
440       NEXT X
450 REM
460 REM   AFSLUITING
470 REM
480       PRINT : PRINT "EINDE BESTAND"
490       CLOSE 1
500       PRINT "BESTAND GESLOTEN"
999       END
```

5.

```
100 REM   EINDTOETS, OPGAVE 5
110 REM
120 REM   VARIABELENLIJST
130 REM    - N$, NB$ = ARTIKELNUMMER (4)
140 REM    - X$, XB$ = OMSCHRIJVING (20)
150 REM    - L$, LB$ = LEVERANCIER  (20)
160 REM    - K,  KB$ = KRITISCHE VOORRAAD
170 REM    - B,  BB$ = BESTELHOEVEELHEID
180 REM    - M,  MB$ = MOMENTELE VOORRAAD
190 REM    - I,  IB$ = INKOOPPRIJS
200 REM    - V,  VB$ = EENHEIDSPRIJS (VERKOOP)
210 REM    - R$      = RESPONS
220 REM    - V1      = NIEUWE EENHEIDSPRIJS
230 REM
240 REM   BESTAND
250 REM    - ARTBES (DIRECT TOEGANKELIJK)
260 REM
270 REM   INITIALISATIES
280 REM
290       OPEN "ARTBES" AS 1
300       FIELD 1, 4 AS NB$, 20 AS XB$, 20 AS LB$, 4 AS KB$, 4 AS BB$,
                   4 AS MB$, 4 AS IB$, 4 AS VB$
310 REM
320 REM   VERWERKING
330 REM
340       PRINT "ART.NR"; TAB(12); "OUDE PRIJS"; TAB(25); "NIEUWE PRIJS"
350       PRINT
360       FOR X = 1 TO LOF(1)%256
370           GET 1, X
380           LET V  = CVS(VB$)
390           LET V1 = V + (V * 0.1)
400           LSET VB$ = MKS$(V1)
410           PRINT NB$; TAB(12); V; TAB(25); V1
420           PUT 1, X
430       NEXT X
440 REM
450 REM   AFSLUITING
460 REM
470       CLOSE 1
480       PRINT "PRIJZEN GEWIJZIGD, BESTAND GESLOTEN"
999       END
```

APPENDIX A

BEKNOPT OVERZICHT VAN DE IN DIT BOEK GEBRUIKTE BASIC-OPDRACHTEN

ASC(X$)	: Geeft de ASCII-getalwaarde van het eerste teken van X$.
CHR$(X)	: Converteert de getalwaarde X naar het overeenkomstige ASCII-teken.
CLOSE n	: Sluit de in-/uitvoerbuffer verbonden met bestandnummer n.
CVI(BF$)	: Convert to integer. Converteert bij een direct-toegankelijk bestand de stringwaarde in de buffervariabele BF$ naar een geheel getal.
CVS(BF$)	: Convert to single precision. Converteert bij een direct-toegankelijk bestand de stringwaarde in de buffervariabele BF$ naar een (gewoon) gebroken getal van enkele precisie.
CVD(BF$)	: Convert to double precision. Idem, dubbele precisie.
EOF(n)	: End of file. Logische functie die 'waar' is als het einde van een sequentieel bestand met nummer n is bereikt.
FIELD #n	: Inrichten van de buffer van een direct-toegankelijk bestand met nummer n, waarbij de buffer in een of meer stringvariabelen wordt opgedeeld.
GET n, r	: Laad de buffer van een direct-toegankelijk bestand n met de inhoud van record r.
INPUT #n, X, Y$	: Lees twee gegevens van het sequentieel bestand n, en ken deze toe aan de variabelen X en Y$.
LEFT$(X$,n)	: Selecteert de linker n tekens in X$.
LEN(X$)	: Geeft het aantal tekens in X$

LINE INPUT : Inlezen van één alfanumeriek gegeven, zonder gebruik van aanhalingstekens als afbakeningssymbolen.

LOF(n) : Geeft de lengte van bestand n in bytes.

LPRINT : Schrijven naar printer.

LSET BF$: Left set. Toekennen van een gegeven aan buffervariabele BF$, waarbij het gegeven links wordt aangesloten als zijn lengte kleiner is dan de lengte van de buffervariabele.

MAXFILES = n : Bepaalt het aantal bestanden (n), dat gelijktijdig in een programma geopend mag zijn (n = 0,1,...,15). De default-waarde van n is 1.

MIDS(X$,m,n) : Vervangen of selecteren van m tekens van X$, te beginnen bij het teken in positie n.

MKI$(X) : Make integer. Converteert de numerieke waarde X naar een twee-byte integerwaarde ten behoeve van opname in de buffer van een direct-toegankelijk bestand.

MKS$(X) : Make single precision. Idem, vier-byte enkele precisiewaarde.

MKD$(X) : Make double precision. Idem, acht-byte dubbele precisiewaarde.

OPEN "xxx" FOR M AS n : Opent een sequentieel bestand met naam xxx en nummer n. M staat voor mode en kan zijn: INPUT, OUTPUT of APPEND (schijf).

OPEN "xxx" AS n : Opent een direct-toegankelijk bestand met naam xxx en nummer n.

PRINT #n : Schrijven naar sequentieel bestand met nummer n.

PUT n, r : Schrijf inhoud van buffer van direct-toegankelijk bestand n naar record r van dat bestand (transport van intern naar extern geheugen).

RSET BF$: Als LSET, met dien verstande dat gegeven zonodig rechts wordt aangesloten.

RIGHT$(X$,n) : Selecteert rechter n tekens van X$.

STR$(X) : Geeft de alfanumerieke representatie van de getalwaarde X.

VAL(X$) : Geeft de numerieke representatie van de string X$.

APPENDIX B

ASCII-CODE TABEL

Decimale waarde	Teken	Decimale waarde	Teken	Decimale waarde	Teken
000	NUL	040	(	080	P
001	SOH	041	)	081	Q
002	STX	042	*	082	R
003	ETX	043	+	083	S
004	EOT	044	'	084	T
005	ENQ	045	-	085	U
006	ACK	046	.	086	V
007	BEL	047	/	087	W
008	BS	048	0	088	X
009	HT	049	1	089	Y
010	LF	050	2	090	Z
011	VT	051	3	091	[
012	FF	052	4	092	\
013	CR	053	5	093	]
014	SO	054	6	094	↑
015	SI	055	7	095	←
016	DLE	056	8	096	`
017	DC1	057	9	097	a
018	DC2	058	:	098	b
019	DC3	059	;	099	c
020	DC4	060	<	100	d
021	NAK	061	=	101	e
022	SYN	062	>	102	f
023	ETB	063	?	103	g
024	CAN	064	@	104	h
025	EM	065	A	105	i
026	SUB	066	B	106	j
027	ESC	067	C	107	k
028	FS	068	D	108	l
029	GS	069	E	109	m
030	RS	070	F	110	n
031	US	071	G	111	o
032	SP	072	H	112	p
033	!	073	I	113	q
034	"	074	J	114	r
035	#	075	K	115	s
036	$	076	L	116	t
037	%	077	M	117	u
038	&	078	N	118	v
039	´	079	O	119	w

Decimale waarde	Teken	Decimale waarde	Teken
120	x	124	|
121	y	125	}
122	z	126	~
123	{	127	DEL

Afkortingen ASCII-code tabel

NUL	Null	DLE	Data Link Escape
SOH	Start of Heading	DC1	Device Control 1
STX	Start of Text	DC2	" " 2
ETX	End Of Text	DC3	" " 3
EOT	End of Transmission	DC4	" " 4
ENQ	Enquiry	NAK	Negative Acknowledge
ACK	Acknowledge	SYN	Synchronous Idle
BEL	Bell	ETB	End of Transmission Block
BS	Backspace	CAN	Cancel
HT	Horizontal Tabulation	EM	End of Medium
LF	Line Feed	SUB	Substitute
VT	Vertical Tabulation	ESC	Escape
FF	Form Feed	FS	File Separator
CR	Carriage Return	GS	Group Separator
SO	Shift Out	RS	Record Separator
SI	Shift In	US	Unit Separator
SP	Space	DEL	Delete

APPENDIX C

FOUTMELDINGEN

Foutmelding	Code	Verklaring
Bad file name	56	(Foute programma- of bestandsnaam) U hebt een verkeerde naam gebruikt om een programma of een bestand aan te duiden.
Bad file number	52	(Fout bestandsnummer) Het nummer dat u gebruikt verwijst naar een bestand dat nog niet geopend is met de instructie OPEN, of het nummer is hoger dan het aantal bestanden dat met MAXFILES is gedefinieerd.
Can't CONTINUE	17	(Kan niet doorgaan) U probeert, door CONT in te typen, door te gaan met een programma dat: - onderbroken is door een foutmelding; - gewijzigd is nadat de uitvoering is afgebroken; - niet bestaat.
Device I/O error	19	(Invoer- of uitvoerfout) Tijdens het inlezen of wegschrijven van een bestand of een programma is een fout geconstateerd.
Direct statement	57	(Directe opdracht) Tijdens het inlezen van een ASCII-bestand is een directe opdracht gevonden.
Division by zero	11	(Delen door nul) Bij het uitvoeren van een berekening ontdekt de computer een deling door 0 of een machtsverheffing van 0 met een negatieve exponent. De nul kan ook het resultaat zijn van een voorgaande berekening.

Field overflow	50	(Veld loopt over) Het aantal bytes, toegekend met de instructie FIELD, is groter dan 256.
File already open	54	(Bestand al geopend) U vraagt de computer met de instructie OPEN een bestand te openen dat al geopend is, of u probeert met KILL een bestand te wissen dat geopend is.
File not open	59	(Bestand niet geopend) U probeert een bestand in te lezen of weg te schrijven dat nog niet is geopend met de instructie OPEN.
Illegal direct	12	(Niet toegestane directe opdracht) U hebt een instructie ingetoetst die in de directe stand niet is toegestaan.
Illegal function call	5	(Niet toegestane functie-aanroep) Bij het aanroepen van een functie is een verkeerde argumentwaarde meegegeven. Dit kan het geval zijn bij: - een negatieve of te grote index, bijvoorbeeld A(-2); - nul of een negatieve waarde bij een LOG-functie; - een negatieve waarde bij een SQR-functie; - Een onjuiste waarde bij één van de volgende instructies: MID$, LEFT$, RIGHT$, INP, OUT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, DELETE, INSTR$, ON....GOTO of ON....GOSUB.
Input past end	55	(Invoer voorbij einde) U probeert gegevens te lezen uit een bestand dat al helemaal gelezen is. Gebruik de functie EOF om deze fout te voorkomen.
Internal error	51	(Interne fout) Er doet zich een onverwachte situatie voor die niet kan worden opgelost door de BASIC-interpreter.
Line buffer overflow	25	(Regelbuffer loopt over) U hebt een te lange regel ingetoetst.

Missing operand	24	(Operand ontbreekt) Een expressie bevat een functie, een instructie of een commando zonder operand, dat wil zeggen zonder getal of string waarop de bewerking kan worden uitgevoerd.
NEXT without FOR	1	(NEXT zonder FOR) De variabele in de NEXT-instructie correspondeert niet met de variabele in de laatste FOR-instructie, of de computer komt een NEXT-instructie tegen zonder voorafgaande FOR-instructie.
No RESUME	21	(Geen RESUME) Na het afhandelen van een fout ontbreekt de instructie RESUME (= hervat het programma).
Out of DATA	4	(Geen DATA meer) Er wordt een READ-instructie uitgevoerd terwijl er geen gegevens meer zijn die nog gelezen kunnen worden.
Out of memory	7	(Geen geheugen meer) Een programma is te groot of bevat te veel FOR-lussen, GOSUB-instructies of variabelen. Het kan ook te ingewikkelde uitdrukkingen bevatten.
Out of string space	14	(Geen stringruimte meer) Er zijn meer alfanumerieke variabelen dan de gereserveerde stringruimte kan bevatten. Gebruik de instructie CLEAR om meer stringruimte te reserveren.
Overflow	6	(Register loopt over) Het resultaat van een berekening is groter dan bij de gekozen variabele kan worden weergegeven.
Redimensioned array	10	(Array opnieuw gedimensioneerd) Er zijn twee DIM-instructies voor dezelfde variabele gegeven, of een variabele die al in gebruik is wordt geDIMensioneerd.
RESUME without error	22	(RESUME zonder foutmelding) De computer wordt met een RESUME-instructie gevraagd het programma te hervatten zonder dat de instructie ON ERROR GOTO is uitgevoerd.

RETURN without GOSUB	3	(RETURN zonder GOSUB) De computer komt de instructie RETURN tegen zonder dat er een GOSUB aan vooraf is gegaan.
Sequential I/O only	58	(Alleen sequentiële in- en uitvoer) U probeert in een sequentieel bestand te lezen of te schrijven alsof het een 'random'-bestand (willekeurig toegankelijk bestand) is.
String formula too complex	16	(Stringexpressie te complex) De alfanumerieke uitdrukking is te lang of te ingewikkeld.
String too long	15	(String te lang) Er is geprobeerd meer dan 255 alfanumerieke tekens in een string te stoppen.
Subscript out of range	9	(Index buiten het bereik) Er wordt een poging gedaan een element te gebruiken met een index buiten het geDIMensioneerde gebied.
Syntax error	2	(Syntaxis-fout) Een commando, instructie of functie bevat een spelfout, een onjuiste interpunctie of zondigt tegen de taalregels van MSX BASIC.
Type mismatch	13	(Verkeerd type) Aan een numerieke variabele wordt een alfanumerieke waarde toegekend of omgekeerd, of aan een functie is een alfanumerieke waarde toegekend in plaats van een numerieke waarde.
Unidentified line number	8	(Onbekend regelnummer) Een commando of een instructie verwijst naar een niet bestaande programmaregel.
Unidentified user function	18	(Onbekende gebruikersfunctie) Er wordt een gebruikersfunctie aangeroepen voordat die is gedefinieerd met de instructie DEF.
Unprintable error	23 26...49 60...255	(Foutmelding zonder omschrijving) Deze foutcodes zijn in MSX BASIC niet gedefinieerd. U mag ze gebruiken om uw eigen foutmeldingen te definiëren.

Verify error	20	(Verificatiefout) Deze foutmelding treedt op als het programma in het geheugen verschilt met het programma op cassette na het uitvoeren van de instructie CLOAD?.

Bron: gebruiksaanwijzing voor de PHILIPS MSX-thuiscomputer VG 8020

APPENDIX D

GERESERVEERDE MSX BASIC WOORDEN

De volgende woorden zijn gereserveerd voor MSX BASIC en mogen niet als namen voor variabelen worden gebruikt en ook niet als deel van een naam.

ABS
AND
AS
ASC
ATN
AUTO
BASE
BEEP
BIN$
BLOAD
BSAVE
CALL
CDBL
CHR$
CINT
CIRCLE
CLEAR
CLOAD
CLOSE
CLS
COLOR
CONT
COPY
COS
CSAVE
CSNG
CSRLIN
CVD
CVI
CVS
DATA
DEF
DEFINT
DEFDBL
DEFSNG
DEFSTR
DEFUSR
DELETE
DIM
DRAW
DSKF
ELSE
END
EOF
EQV
ERASE
ERL
ERR
ERROR
EXP
FIELD
FILES
FIX
FN
FOR
FRE
GET
GOSUB
GOTO
HEX$
IF
IMP
INKEY$
INP
INPUT
INPUT$
INSTR
INT
INTERVAL
KEY
KILL
LEFT$
LEN
LET
LINE
LIST
LLIST
LOAD
LOC
LOCATE
LOF
LOG
LPOS
LPRINT
LSET
MAXFILES
MERGE
MID$
MKD$
MKI$
MOD
MOTOR
MKS$
NAME
NEW
NEXT
NOT
OCT$
OFF
ON
OPEN
OR
OUT
PAD
PAINT
PDL
PEEK
PLAY
POINT
POKE
POS
PRINT
PSET
PRESET
PUT
READ
REM
RENUM
RESTORE
RESUME
RETURN
RIGHT$
RND
RSET
RUN
SAVE
SCREEN
SGN
SIN
SOUND
SPACE$
SPC
SPRITE
SPRITE$
SQR
STEP
STICK
STOP
STR$
STRIG
STRING$
SWAP
TAB
TAN
THEN
TIME
TROFF
TRON
USING
USR
VAL
VARPTR
VDP
VPEEK
VPOKE
WAIT
WIDTH
XOR

APPENDIX E

MSX BASIC BESTURINGSOPDRACHTEN (COMMANDO'S)

AUTO : Geeft na elke RETURN automatisch een volgend regelnummer.
Voorbeelden:

```
AUTO          : regelnummers 10,20,30,40,..., enzovoort
AUTO 100, 50  : regelnummers 100,150,200,..., enzovoort
```

COPY : kopieert één of meer bestanden op dezelfde diskette of naar een andere diskette. MSX-2 computers kennen veel speciale COPY-opdrachten voor het kopiëren van grafische schermen.

DELETE : Verwijdert programmaregels uit het geheugen.
Voorbeelden:

```
DELETE 40     : verwijdert regel 40
DELETE 40-100 : verwijdert regels 40 t/m 100
DELETE -40    : verwijdert alles t/m regel 40
```

(L)FILES : Drukt een lijst af met een opgave van alle bestanden op de diskette in de hoofddrive (drive A)
FILES "B:*.*" drukt een overzicht van alle bestanden op de B-drive af.

KILL : Verwijdert een bestand van diskette.
Voorbeeld: KILL "PROG-4.BAS".

LIST : Drukt de tekst van het programma in het geheugen op het scherm af.
Voorbeelden:

```
LIST          : alle programmaregels
LIST 120      : alleen regel 120
LIST 150-     : alle regels vanaf regel 150
LIST -200     : alle regels t/m regel 200
LIST 150-200  : alle regels vanaf regel 150 t/m regel 200
```

LLIST : Drukt de tekst van een programma in het geheugen op de printer af.

(C)LOAD : Laadt een programma van cassetteband of van diskette.

MERGE : Voegt een programma van diskette of cassetteband toe (of in) aan het programma in het geheugen. Bij gelijke regelnummers wint het bestand het van de regels in het geheugen. Het bestand moet een ASCII-bestand zijn.
Voorbeeld: MERGE "SORT" of "MERGE "CAS:SORT".

NAME : Geeft een diskettebestand een andere naam.
Voorbeeld: NAME "PRO6-4.BAS" AS "VALUTA.BAS"
De laatste bestandsnaam wordt de nieuwe naam.

NEW : Verwijdert het programma dat in het geheugen staat.

RENUM : Hernummert de programmaregels van het programma in het geheugen.
Voorbeelden:

RENUM	: hernummert alle regels. De eerste regel wordt regel 10; de volgende steeds 10 hoger.
RENUM 300,,50	: hernummert alle regels. De eerste regel wordt regel 300; de volgende steeds 50 hoger.
RENUM 1000,900,50	: hernummert de programmaregels beginnend met regel 900. Deze regel krijgt nummer 1000; alle volgende steeds 50 hoger.

RUN : Start de uitvoering van het programma dat in het geheugen staat.
Voorbeelden:

RUN	: start programma vanaf de eerste regel
RUN 100	: start programma vanaf regel 100
RUN "PROG5-7"	: laadt programmabestand met de naam PROG5-7 in het geheugen en start de uitvoering ervan.

(C)SAVE : Kopieert het programma dat in het geheugen staat op (cassette) diskette.

WIDTH : Stelt de breedte in van een beeldschermregel.
MSX-1 computers: maximaal WIDTH 40
MSX-2 computers: maximaal WIDTH 80

INDEX

ACADEMIC SERVICE INFORMATICA UITGAVEN

AUTOMATISERING EN COMPUTERS

Computers en onze informatiemaatschappij - M.A. Arbib
Computers in de negentiger jaren - G.L. Simons
De informatiemaatschappij - Jan Everink
Op weg naar een risicoloze maatschappij? - Jan Holvast
Basiskennis informatieverwerking - Jan Everink
AIV, Automatisering van de informatieverzorging - Th.J.G. Derksen & H.W. Crins
BIV, Basis van de geautomatiseerde informatieverzorging - Th.J.G. Derksen & H.W. Crins
Organisatie, informatie en computers - David M. Kroenke
Computers in de basisschool - H. Lamers & J.A. Wegkamp
Effectieve toepassingen van computers - M. Peltu

MICROCOMPUTERS

Microcomputers thuis en op school - K.P. Goldberg & D. Sherwood
Bouw zelf een expertsysteem in BASIC - Chris Naylor
Programmeercursus MicrosoftBASIC - Nok van Veen
Werken met bestanden in BASIC - LeRoy Finkel & Jerald R. Brown
Programmeercursus BASIC op de Commodore 64 - Nok van Veen
Werken met bestanden op de Commodore 64 - G. Fisher, L. Finkel & J.R. Brown
Het Electron en BBC Micro boek - Jim McGregor & Alan Watt
Werken met bestanden op de Apple - LeRoy Finkel & Jerald R. Brown
Programmeercursus ApplesoftBASIC - Nok van Veen & Ad van Delft
Programmeercursus MSX BASIC - Nok van Veen
Werken met bestanden in MSX BASIC - LeRoy Finkel & Jerald R. Brown
40 grafische programma's - voor de Commodore 64; voor de Electron en BBC; voor de ZX Spectrum; voor de Apple II, IIe en IIc; in MSX BASIC - Marcel Sutter

MICROPROCESSORS EN ASSEMBLEERTALEN

Procescomputers, basisbegrippen - J.E. Rooda & W.C. Boot
Cursus Z-80 assembleertaal - Roger Hutty
6502 assembleertaal en machinecode voor beginners - A.P. Stephenson
EXAT-handboek - Micro-Teach

BESTURINGSSYSTEMEN

Inleiding besturingssystemen - A.M. Lister
Theorie en praktijk van besturingssystemen - J.L. Peterson & A. Silberschatz
Systeemprogrammatuur en softwareontwikkeling voor microcomputers - E. Verhulst
Bedrijfssystemen - EIT-serie, deel 4
CP/M, het operating system voor microcomputers - J.N. Fernandez & R. Ashley
CP/M 86, een besturingssysteem voor 16 bit microcomputers - J.N. Fernandez & R. Ashley
CP/M voor gevorderden - A. Clarke e.a.
PC DOS, het besturingssysteem van de IBM PC - R. Ashley & J.N. Fernandez
MS DOS, het besturingssysteem voor 16 bit microcomputers - R. Ashley & J.N. Fernandez
UNIX, het standaard operating system - G.J.M. Austen & H.J. Thomassen
De UNIX programmeeromgeving - B.W. Kernighan & R. Pike

PERSONAL COMPUTERS EN PROGRAMMAPAKKETTEN

De IBM PC en zijn toepassingen - Laurence Press
Werken met bestanden in IBM- en GW-BASIC - J.R. Brown & LeRoy Finkel
40 grafische programma's in IBM- en GW-BASIC - Marcel Sutter
Werken met VisiCalc - C. Klitzner & M.J. Plociak Jr.
Multiplan, een hulpmiddel bij de bedrijfsvoering - D.F. Cobb e.a.
Werken met Lotus 1-2-3 - G.T. LeBlond & D.F. Cobb
Lotus 1-2-3: Tips, Trucs en Tegenvallers - D. Andersen & D.F. Cobb
Lotus 1-2-3: Financiële macro's - Thomas W. Carlton
Symphony deel I en II - D.P. Ewing & G.T. LeBlond
dBASE III handboek - George Tsu-der Chou
WordStar stap voor stap - Ruth Ashley & Judi N. Fernandez

PROGRAMMEREN

Een methode van programmeren - Edsger W. Dijkstra & W.H.J. Feijen
Programmeren, met ontwerpen van algoritmen (met Pascal) - J.J. van Amstel
Voortgezet programmeren, het ontwerpen van datastructuren en algoritmen - J.J. van Amstel & J.A.A.M. Poirters

Problemen oplossen met de computer - R.G. Dromey
Inleiding tot het programmeren, deel 1 en 2 - J.J. van Amstel e.a.
Het Groot Pascal Spreukenboek - Henry F. Ledgard e.a.
Software engineering, het bouwen van grote programma's - I. Sommerville
JSP Jackson structureel programmeren - Henk Jansen
JSP Uitwerkingenboek, JSP Procedureboek - Henk Jansen

PROGRAMMEERTALEN

Aspecten van programmeertalen - J.J. van Amstel & J.A.A.M. Poirters
Programmeertalen, een inleiding - J.J. van Amstel e.a.
Colloquium programmeertalen - red. J.A.A.M. Poirters & G.J. Schoenmaker
BASIC - EIT-serie, deel 3
Cursus BASIC, een practicum handleiding voor BASIC op de PRIME - R. Bloothoofd e.a.
Een programmeercursus in BASIC - Nok van Veen & René Wissing
Cursus Pascal - A. van der Sluis & C.A.C. Görts
Cursus eenvoudig Pascal - A. van der Sluis & C.A.C. Görts
Inleiding programmeren in Pascal - C. van de Wijgaart
Modula-2 - E. Verhulst
Systeemontwikkeling met Ada - Grady Booch
Cursus COBOL - A. Parkin
Cursus FORTRAN 77 - J.N.P. Hume & R.C. Holt
De programmeertaal C - L. Ammeraal
Flitsend Forth - Alan Winfield
Logisch LOGO - Auke Sikma
Programmeren in LISP - L.L. Steels
Micro-PROLOG, programmeren in logica - K.L. Clarke & F.G.McGabe
Inleiding PROLOG - W. Burnham & A. Hall

BESTANDSORGANISATIE, DATABASE EN GEGEVENSANALYSE

Bestandsorganisatie - R.J. Lunbeck & F. Remmen
Database, een inleiding - C.J. Date
Databases, grondslagen voor de logische structuur - F. Remmen
SQL in de praktijk - H.B. Eilers e.a.
Het SQL Leerboek - Rick F. van der Lans
Gegevensanalyse - R.P. Langerhorst

INFORMATIE-ANALYSE EN SYSTEEMONTWERP

Inleiding systeemanalyse en systeemontwerp - W.S. Davis
Systeemontwikkeling zonder zorgen - Paul T. Ward
Systeemontwikkeling volgens SDM - H.B. Eilers
Samenvatting SDM - Pandata
Systeemontwikkeling volgens JSD - Michael Jackson
Informatie-analyse volgens NIAM - J.J.V.R. Wintraecken
Information engineering - J. Blank
Het ontwerpen van interactieve toepassingen en computernetwerken - J.A. Scheltens
EDP Audit - C. de Backer
Management informatiesystemen - G.B. Davis & M.H. Olson
Prototyping, een instrument voor systeemontwerpers - red. T. Hoenderkamp & H.G. Sol
Het ontwikkelen van informatiesystemen met prototyping - R. Vonk
Simulatie, een moderne methode van onderzoek - S.K.T. Boersma & T. Hoenderkamp

EXPERTSYSTEMEN EN KUNSTMATIGE INTELLIGENTIE

Kunstmatige intelligentie - Patrick H. Winston
Expertsystemen - Henk de Swaan Arons & Peter van Lith
Ontwikkelingen in expertsystemen - red. A. Nijholt & L.L. Steels

THEORETISCHE INFORMATICA EN SYSTEEMPROGRAMMATUUR

Systeemprogrammatuur - H. Alblas
Vertalerbouw - H. Alblas e.a.

OPERATIONELE RESEARCH

Operationele research - Y.M.I. Dirickx e.a.
Lineaire programmering als hulpmiddel bij de besluitvorming - S.W. Douma

INFORMATIE OVER DEZE PUBLIKATIES BIJ:

Academic Service, Postbus 81, 2870 AB Schoonhoven, tel. 01823-6577

Vooral door de opkomst van de microcomputer heeft het gebruik van BASIC een enorme vlucht genomen. Met die toegenomen belangstelling heeft ook het aantal boeken dat over deze taal verschijnt min of meer gelijke tred gehouden.
Van die boeken komen slechts weinig verder dan een elementaire uiteenzetting van de taal en/of de toepassing daarvan in betrekkelijk eenvoudige situaties. Het werken met bestanden - gegevensverzamelingen die op een bepaalde manier op bijvoorbeeld diskettes (floppies) kunnen worden opgeslagen - is bijvoorbeeld één van de aspecten die grotendeels worden verwaarloosd.
Dit boek heeft tot doel deze leemte aan te vullen. Uitvoerig, maar duidelijk en praktisch wordt besproken hoe bestanden in MSX BASIC opgebouwd en onderhouden kunnen worden. Stap voor stap leert de lezer boekhoudkundige zaken, ledenlijsten, inventaris- en voorraadlijsten etc. met de computer bij te houden. Tientallen voorbeelden laten zien hoe men hierbij te werk kan gaan. En passant komen ook 'stilistische' onderwerpen zoals top-down, modulair en gestructureerd programmeren ter sprake, waardoor het netjes en 'zindelijk' programmeren wordt bevorderd. Met name aan het gestructureerd programmeren wordt uitgebreid aandacht besteed.
Het boek kan ook gebruikt worden voor MSX BASIC op MSX-2 computers.

ISBN 90 6233 215 3