van Veen **Programmeercursus MSX BASIC**

Programmeercursus MSX BASIC

Nok van Veen

ACADEMIC SERVICE

OOK VOOR MSX-2

PROGRAMMEERCURSUS MSX BASIC

In deze reeks zijn verschenen:

Programmeercursus Microsoft BASIC - Nok van Veen
Programmeercursus BASIC op de Commodore 64 - Nok van Veen
Programmeercursus Applesoft BASIC - Nok van Veen & Ad van Delft

Verschenen voor MSX-computers:

Werken met bestanden in MSX BASIC - LeRoy Finkel & Jerald R. Brown
40 grafische programma's in MSX BASIC - Marcel Sutter

Programmeercursus MSX BASIC

Nok van Veen

ACADEMIC SERVICE

OOK VOOR MSX-2

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Veen, Nok van

Programmeercursus MSX BASIC / Nok van Veen ; [ill. André de Regt]. - Den Haag : Academic Service. - Ill.
Met index.
ISBN 90-6233-173-4
SISO 365.3 SVS 7.12.3 UDC 681.3.06:800.92 MSX-Basic
UGI 650
Trefw.: MSX-Basic (programmeertaal).

Uitgegeven door: Academic Service
Postbus 81
2870 AB Schoonhoven
Illustraties: André de Regt
Omslagtekening: George Bazan
Omslagontwerp: Leo Bolt
Druk: Krips Repro Meppel
Bindwerk: Meeuwis, Amsterdam
ISBN 90 6233 173 4

Inhoud

Voorwoord

MSX BASIC en MSX-(1 en 2)-computers

MSX is de afkorting voor MicroSoft eXtended. MSX BASIC is namelijk een uitbreiding van MBASIC, de algemene BASIC-versie van Microsoft, die op veel microcomputers draait. MSX-computers hebben niet alleen de eigenschap dat MSX BASIC-programma's overdraagbaar zijn, maar ook dat de hardware, zoals bijvoorbeeld de Z-80 microprocessor, de printeraansluiting en de disk drive interface, gestandaardiseerd is.
Tot het begin van 1986 kunnen we gewoon over 'MSX-computers' spreken. De een heeft wat meer RAM-geheugen dan de andere, maar allemaal hebben ze een 16K videogeheugen voor het opslaan van gegevens over tekst, kleur, sprites enzovoort, voor de vier schermen (SCREEN 0, 1, 2 en 3), die in MSX BASIC gebruikt kunnen worden. Nu zijn er echter ook MSX-2-computers te koop, waardoor de bovengenoemde MSX-computers noodgedwongen MSX-1-computers worden genoemd. Beide typen blijven naast elkaar bestaan. MSX-2-computers kunnen hetzelfde als MSX-1 computers, maar dan nog iets meer. Deze uitbreiding uit zich vooral in de grafische mogelijkheden. Er is een nieuwe videoprocessor gebruikt, waarmee een videogeheugen van maar liefst 128K bytes wordt bestuurd. Naast de bestaande vier schermen is er een 80-kolommen scherm bijgekomen en nog vier grafische schermen, die een veel hogere schermresolutie toestaan.
Het is echter niet de bedoeling in dit voorwoord alle nieuwtjes van MSX-2-computers op te noemen. Integendeel, het gaat in dit boek om het leren programmeren. Het maakt hierbij niet uit of er met een MSX-1- of MSX-2-computer wordt gewerkt.
Nu eerst iets over programmeren en dan iets over het boek.

Programmeren

In veel 'MSX BASIC'-boeken vinden we alleen korte omschrijvingen van de diverse 'MSX BASIC'-opdrachten, aangevuld met enkele voorbeelden. Als je echter wilt leren programmeren dan kan je niet volstaan met te weten wat de diverse programma-opdrachten doen. Programmeren is veel meer dan het opschrijven van een aantal (MSX BASIC)-programma-opdrachten. In feite heet dit laatste coderen en is de laatste stap in het hele programmeerproces.

Programmeren omvat alle activiteiten die nodig zijn om voor de oplossing van een bepaald probleem een computerprogramma te schrijven. Deze activiteiten zijn ruw samengevat:

Het opstellen van een goede probleembeschrijving
(wat moet er opgelost worden?)

Het analyseren en structureren van het probleem
(hoe zit het probleem in elkaar; is het te splitsen in een aantal kleinere problemen?)

Het zoeken van een 'goede' oplossingsmethode
(hoe gaan we het probleem oplossen?)

Het schrijven van de programmatekst
(hoe coderen we het programma?)

Programmeren is dus heel wat meer dan het schrijven van de tekst van een computerprogramma.

Het boek

Dit boek is een programmeercursus, een cursus MSX BASIC. De programmeermethode, die in dit boek wordt aangeleerd, leidt tot het schrijven van goed-gestructureerde en vooral goed-leesbare programma's. Aan de leesbaarheid van de vele, in dit boek opgenomen, programmavoorbeelden is maximale zorg besteed; het is immers een vereiste dat, in een boek als dit, de lezer aan de hand van de programmatekst kan zien hoe een programma met behulp van diverse structuren is opgebouwd. Bij het ontwerpen van een programma wordt namelijk eerst de structuur van het programma bepaald en pas daarna wordt de programmatekst gecodeerd. Het is heel belangrijk dat in de programmatekst de eerder ontworpen structuur te herkennen is. Een goede structuur en leesbaarheid zijn bovendien een voorwaarde om een programma later zonder veel moeite te kunnen wijzigen of uitbreiden; zeker geldt dit als de programma's groter en ingewikkelder worden.

Na een korte inleiding (hoofdstuk 1) wordt een aantal specifieke programmastructuren (hoofdstukken 2, 3, 4) behandeld. Achtereenvolgens komen de sequentiële, de herhalings- en de keuzestructuur aan bod. Door hierbij niet te veel verschillende MSX BASIC opdrachten te gebruiken wordt de nadruk op het programmeren en op het bepalen van de programmastructuur gelegd en niet op de MSX BASIC-opdrachten zelf, alhoewel deze laatste natuurlijk wel nodig zijn om de diverse programmastructuren te coderen. In deze hoofdstukken komt toch een groot aantal MSX BASIC-opdrachten aan de orde en worden vele voorbeeldprogramma's, waaronder een sprite-ontwerpprogramma, met joystick, graphics en geluid gegeven. Ook wordt de lezer wegwijs gemaakt in het videogeheugen.

In hoofdstuk 5 wordt ingegaan op de top-down programma-ontwerpmethode. Aangetoond wordt dat deze methode van 'stapsgewijze verfijning' een voorwaarde is voor het kunnen ontwerpen van goedgestructureerde programma's. Hierbij wordt gebruik gemaakt van programmastructuurdiagrammen (Nassi Shneiderman) om de structuur van programma's in kaart te brengen.

Vanaf hoofdstuk 6 wordt de nadruk meer gelegd op de MSX BASIC-opdrachten zelf. Aan de hand van veel educatieve en ontspannende toepassingen laten we zien hoe je deze opdrachten kunt gebruiken. Hoofdstuk 6 bevat onder meer een groot educatief simulatieprogramma, waarin zowat alle grafische opdrachten, die MSX BASIC kent, gebruikt worden. In hoofdstuk 7 gaan we met lijsten en tabellen (arrays) werken en worden onder andere drie sorteermethoden geprogrammeerd. Hoofdstuk 8 is geheel gewijd aan tekstbewerking. In hoofdstuk 9 leren we werken met gegevensbestanden en programmabestanden op cassetteband en diskette.

Voor het hele boek geldt dat het maken van de opdrachten in de tekst bijdraagt tot een verdieping van de kennis. Het daadwerkelijk maken van deze opdrachten wordt dan ook sterk aanbevolen. De antwoorden van deze opdrachten zijn in Appendix A opgenomen. Naast deze tekstopgaven is aan het einde van bijna elk hoofdstuk een groot aantal oefenopgaven opgenomen. De antwoorden van deze opgaven zijn te vinden in Appendix B.

Bij bijna alle voorbeeldprogramma's is gestreefd naar een goed evenwicht tussen berekeningen en andere toepassingen zoals tekstbewerking en graphics.

Tot slot het belangrijkste: de lezer! Voor wie is dit boek bestemd? Wel, voor iedereen die netjes wil of 'moet' leren programmeren en die hierbij een MSX-computer kan gebruiken.
Omdat het boek naast de programmeertaal MSX BASIC ook het programmeren zelf centraal stelt kan het boek ook als leer- en studieboek bij een practicum of opleiding programmeren gebruikt worden.

Dit boek laat zien dat je met MSX BASIC goed-gestructureerde programma's kunt ontwerpen. Diegenen, die in BASIC leren programmeren op de manier zoals in dit boek is gevolgd zullen geen moeite hebben met het leren programmeren in echte gestructureerde talen als Pascal en Comal.

Tot slot

Bij het schrijven van dit boek en het testen van de programma's is gebruik gemaakt van MSX-computers van verschillend fabrikaat. Graag wil ik hierbij de firma's AVT Electronics en Brandsteder Electronics danken voor het spontaan in bruikleen geven van MSX-apparatuur.

De illustraties zijn van de hand van André de Regt. Het vakkundige tik- en zetwerk is verzorgd door Elsina Baselmans. Léon Schell heeft het manuscript doorgewerkt en gecorrigeerd. Elsina, André en Léon, bedankt!

Nok van Veen Voorburg, januari 1986

1 MSX BASIC, Een Eerste Kennismaking

Het schrijven van een programma bestaat uit het formuleren en achter elkaar plaatsen van een aantal opdrachten. Als we een BASIC-programma schrijven moeten we ons aan de in BASIC geldende taalregels houden. Deze taalregels noemen we de *syntax*. Deze BASIC syntax vertelt ons welke woorden we in BASIC kunnen gebruiken (zie Appendix C), hoe deze woorden gespeld moeten worden en hoe we met deze woorden opdrachten (zinnen) kunnen formuleren.

Wanneer wij in ons dagelijks leven niet geheel volgens de taalregels van onze eigen moedertaal handelen, zal dat in het algemeen geen ernstige consequenties hebben. Bij het praten in BASIC ligt dit heel anders. Als wij in een BASIC-programma iets schrijven wat niet geheel in overeenstemming is met de BASIC-taalregels, zal de computer, waarop we het programma laten draaien, niet begrijpen wat we bedoelen. Doorgaans geeft hij in zo'n geval netjes aan waarom hij het niet begrijpt; hij geeft een foutmelding. Bovendien geeft hij in bijna alle gevallen aan waar hij de taalfout in het programma ontdekt heeft, zodat wij de fout snel kunnen opzoeken en verbeteren. In dit boek zullen we leren BASIC-programma's te schrijven zonder taalfouten, 'syntactisch goede programma's' dus.

Om een taal goed te kunnen gebruiken is het echter niet alleen voldoende de woorden uit die taal en de taalregels van die taal te kennen. Ook zullen we moeten weten wat in bepaalde situaties de betekenis van de woorden en/of zinnen is. In BASIC betekent dit dat we moeten weten wat een opdracht doet en hoe we opdrachten moeten rangschikken en groeperen zodat de opdrachten precies doen wat wij willen. Dit noemen we de **semantiek** (= betekenis) van de taal. Semantiek is iets heel anders dan syntax. Bekijk de volgende zin, uit de Nederlandse taal, eens:

DE AUTO WORDT ONTSLAGEN

Deze zin voldoet geheel aan de taalregels (syntaxis) van de Nederlandse taal; de woorden zijn goede 'Nederlandse' woorden (ze zijn goed gespeld) en ook de zinsbouw is goed. Maar hoe zit het met de semantiek van de zin, wat is de betekenis ervan? Op het eerste

gezicht zouden we kunnen zeggen: fout!, auto's kunnen niet worden ontslagen. Maar als het een zin zou zijn uit een kinderverhaal, waarin auto's de plaats van mensen innemen, heeft de zin opeens wel betekenis. Semantiek is dus iets dat afhankelijk is van de situatie, waarin we een woord, een zin of een opdracht gebruiken.

Het leren van de taalregels van BASIC is een kwestie van veel programmeren.

Het aanbrengen van de juiste semantiek in een programma, met andere woorden het kiezen van de juiste opdrachten en het rangschikken van deze opdrachten zodat het programma precies dat doet wat het moet doen, wordt bevorderd door 'netjes' te programmeren. Wat 'netjes' programmeren inhoudt hopen we in dit boek duidelijk te maken.

Het programmeren begint in hoofdstuk 2. Dit eerste hoofdstuk is echt een eerste kennismaking met BASIC op een MSX-computer, toegespitst op het rekenen in BASIC en het werken met tekst. Toch eindigen we dit hoofdstuk met een echt BASIC programmaatje.

Voordat we beginnen nog iets over MSX BASIC. Als we een MSX-computer aanzetten dan zien we één van de volgende twee meldingen op het scherm. De bovenste zien we als we Disk BASIC gebruiken; er is dan een diskettestation (disk drive) op de computer aangesloten. Werken we zonder drive dan zien we de onderste melding. Dit voorbeeld is voor een MSX-1-computer met 80 K geheugen.

```
MSX BASIC version 1.0
Copyright 1983 by Microsoft
24455 Bytes free
Disk BASIC version 1.0
Ok
```

```
MSX BASIC version 1.0
Copyright 1983 by Microsoft
28815 Bytes free
Ok
```

De 'Ok' is de MSX-BASIC-prompt die aangeeft dat de computer klaar staat om een opdracht van ons te ontvangen. We zullen het in dit boek vaak hebben over BASIC, waarbij we dan natuurlijk bijna altijd MSX BASIC bedoelen.

Als we een MSX-computer aanzetten, dan staat het beeldscherm automatisch ingesteld op 24 regels van maximaal 37 tekens (stand SCREEN 0). Druk vast de hoofdlettervastzettoets (capslock) in,

want we zullen eerst alleen hoofdletters gebruiken bij het intikken van opdrachten. In de loop van het boek gaan we over op kleine letters èn hoofdletters en laten we zien welke verschillende beeldscherminstellingen mogelijk zijn.

1.1 DE MSX-COMPUTER ALS ZAKREKENMACHINE

We kunnen de MSX-computer als zakrekenmachine gebruiken. Stel dat we het sommetje 25 + 18 willen uitrekenen. Op een zakrekenmachientje gaat dat zo:

[2] [5] [+] [1] [8] [=] 43 ← uitkomst

Op de MSX-computer gaat dit anders. Wij zetten 'm aan. Even daarna zien we dat hij Ok op het scherm afdrukt èn

wij tikken in: PRINT 25 + 18 ◁

hij drukt af: 43

Ok

dit betekent:
"DRUK OP DE RETURN-TOETS"

uitkomst optelling

de computer wacht tot wij weer iets intikken

Op het toetsenbord tikken we de tekst **PRINT 25 + 18** in en daarna drukken we op de **RETURN**-toets. Het drukken op de **RETURN**-toets is, in dit geval, het teken voor de computer om de opdracht PRINT 25 + 18 uit te voeren. Het resultaat van deze opdracht is dat de getallen 25 en 18 bij elkaar worden opgeteld en dat de som op het beeldscherm wordt afgedrukt. De **RETURN**-toets vervult hierbij dus de functie van de =-toets op een zakrekenmachine. Omdat de PRINT-opdracht direct na het indrukken van de **RETURN**-toets wordt uitgevoerd, spreken we in dit geval van een directe BASIC-opdracht. Na het uitvoeren van zo'n directe opdracht is de computer de opdracht 'vergeten'. De opdracht wordt niet in het geheugen van de computer bewaard. We zullen straks zien hoe we de computer kunnen instrueren om opdrachten wel te bewaren.

Laten we de directe **PRINT**-opdracht gebruiken om te demonstreren hoe een MSX-computer rekent.

onze notatie	zijn notatie
15 + 10	PRINT 15 + 10
7,5 - 4,2	PRINT 7.5 - 4.2
8 × 13	PRINT 8 * 13
$\frac{15}{4}$	PRINT 15/4
3^4	PRINT 3 ∧ 4

We zien dat de computer (dat wil zeggen MSX BASIC in de computer) voor de rekenkundige bewerkingen de volgende tekens gebruikt:

machtsverheffen : ∧
vermenigvuldigen : *
delen : / (ook \ en MOD, zie § 2.2 en § 7.4)
optellen : +
aftrekken : -

Bovendien zien we dat hij een punt gebruikt in een gebroken getal, terwijl wij gewend zijn daarvoor een komma te gebruiken.

We zijn snel geneigd in BASIC-opdrachten notaties te gebruiken waaraan we zelf gewend zijn, maar waar BASIC geen weg mee weet! Zo schrijven wij zelf voor het produkt 4 maal 5 maal 6 nog wel eens 4·5·6 . BASIC begrijpt niet wat er met 4·5·6 bedoeld wordt (wellicht een dubbeldecimaal getal?!). We moeten 4 maal 5 maal 6 dan ook in BASIC schrijven als 4*5*6 .

Hoe zit het nu als we in een rekensommetje een aantal rekenkundige bewerkingen moeten uitvoeren?
Uit de algebra weten we dat we het sommetje:

$$5 \times (4 + 11) - \frac{24}{3}$$

in een bepaalde volgorde moeten uitrekenen, namelijk eerst haakjes uitwerken en dan:

Meneer Van Dalen Wacht op Antwoord, dus

$5 \times 15 - \frac{24}{3} =$
$75 - 8 =$
67

De computer zal dit sommetje ook zo oplossen. Maar pas op ..., hij rekent niet altijd zoals wij dat doen.

BASIC houdt zich aan de volgende regels:

I Haakjes uitwerken
II Machtsverheffen, Worteltrekken
III Vermenigvuldigen, Delen
IV Optellen, Aftrekken

- Allereerst worden de haakjes uitgewerkt, waarbij een paar haakjes dat binnen een ander paar haakjes voorkomt eerst wordt uitgewerkt.
- Machtsverheffen en Worteltrekken hebben geen voorrang ten opzichte van elkaar, maar wel ten opzichte van vermenigvuldigen, delen, optellen en aftrekken.
- Vermenigvuldigen en Delen hebben geen voorrang ten opzichte van elkaar, maar wel ten opzichte van optellen en aftrekken.
- Optellen en Aftrekken hebben geen voorrang ten opzichte van elkaar.
- Als twee rekenkundige bewerkingen geen voorrang ten opzichte van elkaar hebben, worden ze in een BASIC-opdracht uitgevoerd in de volgorde waarin ze in de rekenkundige uitdrukking voorkomen, van links naar rechts dus. Bekijk het volgende voorbeeld eens:

zoals wij het doen		zoals BASIC het doet	
$\frac{5}{2 \times 4}$	geeft	PRINT 5/2 * 4	geeft
$\frac{5}{8} = 0{,}625$		2.5 * 4 = 10	

We zien dat BASIC zich houdt aan de hierboven gegeven regels. BASIC geeft, als alle haakjes (I) en de machten en wortels (II) zijn uitgewerkt geen prioriteit aan vermenigvuldigen of delen (III). Beide zijn even belangrijk. BASIC werkt de uitdrukking 5/2 * 4 van links naar rechts uit, dus eerst delen (5/2) en dan vermenigvuldigen (2.5 * 4). Als wij de computer duidelijk willen maken dat 2 * 4 in de noemer moet staan, moeten we haakjes gebruiken. We hadden dan de volgende PRINT-opdracht moeten geven: PRINT 5/(2 * 4).

opdracht 1.1

Bereken de uitkomsten van de volgende PRINT-opdrachten.

a. PRINT 4 *(5/2) + 1
b. PRINT 2 *(1 + 3)/4
c. PRINT 3 + 2 ^ 3

d. PRINT (4 - 1) ^ 2 * 3
e. PRINT (1 + 2/3)/5 + 2/3
f. PRINT (6/3) * 2 ^ 2

opdracht 1.2

Geef voor elk van de volgende uitdrukkingen een PRINT-opdracht in BASIC.

a. $5 \times 3 + 2^4 - 5$

b. $4 \times (3 + 1) - \frac{5}{6}$

c. $2^3 \times 3^2 - \frac{8}{5+1}$

d. $3 \times (5^2 - 3)^2$

e. $\frac{4}{5} \times 8 + (1 - 2 \times 3)^2$

f. $\frac{3}{(4-3\times2)} \times 10$

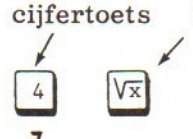
1.2 REKENKUNDIGE (BASIC) FUNCTIES IN MSX BASIC

Tegenwoordig heeft zelfs het meest primitieve rekenmachientje een worteltoets. Als we op die toets drukken wordt de vierkantswortel uit het getal, dat op dat moment in het afleesvenster (display) staat, berekend. Een dergelijke toets noemen we een functietoets. Met zo'n functietoets laten we de rekenmachine een aantal voorgeprogrammeerde opdrachten uitvoeren en dat scheelt ons een hoop werk. Enkele andere, u wellicht welbekende functietoetsen zijn:

BASIC kent ook de mogelijkheid om van dergelijke 'voorgeprogrammeerde' functies gebruik te maken. In plaats van een toets gebruiken we een bepaalde afkorting, meestal bestaande uit drie letters. Laten we dit illustreren aan de hand van een voorbeeld.

Denk erom: Hier worden niet de functietoetsen F1 t/m F10 op het toetsenbord van een MSX-computer bedoeld. Het gebruik van deze toetsen komt later aan de orde.

Het uitrekenen van de wortel uit 4 ($\sqrt{4}$) gaat op een rekenmachine als volgt:



wij tikken in:

rekenmachine drukt af: 2

Op de computer moeten we de wortel uit 4 als volgt berekenen:

```
wij tikken in:  PRINT SQR(4) ◁
hij drukt af :  2
                Ok
```

denk aan de RETURN-toets!

In plaats van het indrukken van één worteltoets moeten we op de computer heel wat toetsen (om precies te zijn 13) indrukken om de wortel uit 4 te berekenen. In deze BASIC-opdracht stellen de letters SQR de worteltoets voor.

DENK EROM:	De computer doet niets als we niet op de RETURN-toets drukken

De functie SQR betekent in BASIC de vierkantswortel van de waarde die we tussen haakjes opgeven. We noemen deze waarde het argument van de functie. SQR is een afkorting (mnemonic) voor SQuare Root ofwel VierKantsWortel. (In een Nederlandstalige uitgave van BASIC zal deze functie dan ook VKW heten!) Het argument van een rekenkundige functie mag ook een rekenkundige uitdrukking zijn, bijvoorbeeld:

```
PRINT SQR(4 * 3 + 2 ^ 3)
```

dit noemen we het argument van de functie

De SQR functie vereist een niet-negatief argument. Bij een negatief argument geeft de computer een foutmelding:

```
wij tikken in:  PRINT SQR(4 * 3 - 15) ◁
hij drukt af:   Illegal function call  ← ← foutmelding
                Ok
```

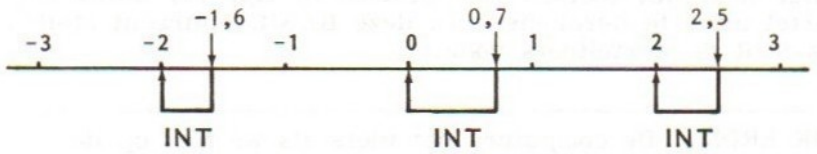
Deze foutmelding betekent dat we een functie hebben aangeroepen met voor het argument een waarde die niet is toegestaan bij deze functie.

SQR(X) =	de vierkantswortel uit X; X is een niet-negatief getal of een rekenkundige uitdrukking met een niet-negatieve waarde

We zullen nog twee rekenkundige functies bespreken die we in de volgende hoofdstukken in de diverse programma's nogal eens zullen tegenkomen.

Allereerst de functie **INT**. INT is een afkorting voor **INT**eger, het Engelse woord voor geheel getal. Deze functie maakt van het opgegeven argument een geheel getal. Een paar voorbeelden:

```
INT(2,5) = 2
INT(0,7) = 0
INT(-1,6) = -2
```



INT rondt dus altijd naar beneden af tot een geheel getal, anders gezegd: voor een bepaald argument X is

INT(X)	het grootste gehele getal kleiner dan of gelijk aan X

Ook mag X bij INT een rekenkundige uitdrukking zijn; zo is INT(3/2 + 5) gelijk aan INT(6.5), dus 6.

De tweede functie die we nu al even willen bespreken is de functie **RND**. RND is een afkorting voor **RaNDom**, hetgeen willekeurig betekent. De functie RND geeft ons een willekeurig getal, een toevalsgetal. Deze functie zullen we straks nodig hebben als we een spelletje gaan programmeren, maar ook zullen we RND tegenkomen in meer serieuze toepassingen. Laten we eens kijken wat voor soort toevalsgetallen RND ons geeft.

Wij tikken in: `PRINT RND(1)` ◁

hij drukt af: `.59521943994623`

`Ok`

Hier volgt nog een aantal getallen die we met **RND** opgeroepen hebben:

```
.10658628050158
.76597651772823
.57756392935958
.73474759503023
.18426812909758
.37075377905223
```

Hmmm..... blijkbaar geeft **RND(1)** een **toevalsgetal tussen 0 en 1.**

Zouden we met RND het gooien met een dobbelsteen kunnen nabootsen? De uitkomst van een worp met een dobbelsteen is 1, 2, 3, 4, 5 of 6 ogen.

Laten we eens proberen van een getal tussen 0 en 1 één van de gehele getallen 1, 2, 3, 4, 5 of 6 te maken.

RND(1) = een toevalsgetal tussen 0 en 1

Door het argument 1 en het toevalsgetal tussen 0 en 1 dat RND(1) oplevert zou je kunnen concluderen dat RND(2) een toevalsgetal tussen 0 en 2 en RND(10) een toevalsgetal tussen 0 en 10 zouden geven. Dit is echter niet zo. De twee énen hebben niets met elkaar te maken. We hadden in plaats van 1 ook elk ander positief getal als argument voor RND kunnen opgeven. We komen hierop later terug.

Als RND een getal tussen 0 en 1 geeft dan moet een zes keer zo groot getal tussen 0 en 6 liggen. Laten we eens kijken:

wij voeren in: `PRINT 6 * RND(1)` ◁

hij drukt af:
```
 5.6972490990935
Ok
```

De uitvoer ligt inderdaad tussen 0 en 6. We hebben het nog een aantal keren geprobeerd; dit was de uitkomst:

```
.63951768300948
4.5958591063694
3.4653835761575
4.4084855701814
1.1056087745855
2.2245226743134
```

Het aantal ogen van een worp is een geheel getal, geen gebroken getal. We moeten dus het decimale gedeelte kwijt. Juist!..... daar gebruiken we INT voor:

wij voeren in: `PRINT INT(6 * RND(1))` ◁

hij drukt af:
```
 2
Ok
```
◁ We laten dit in de navolgende voorbeelden meestal weg

Inderdaad, een geheel getal. We zien dat je als argument van een functie een uitdrukking mag nemen die zelf weer een functie bevat.

Zo krijg je een functie van een functie (RND zit in het argument van INT).

Als 6 * RND(1) een getal tussen 0 en 6 is, zal INT(6 * RND(1)) een geheel getal uit de reeks 0, 1, 2, 3, 4, 5 zijn. Dit lijkt al aardig op de aantallen ogen die je met een dobbelsteen kunt gooien. Bij een 'echte' dobbelsteen is het aantal ogen dat je kunt gooien echter

1, 2, 3, 4, 5, 6 en niet
0, 1, 2, 3, 4, 5

We zien dat elke waarde in de bovenste rij steeds één groter is dan de overeenkomstige waarde in de onderste rij. Laten we die één er gewoon bij optellen:

wij voeren in: `PRINT INT(6 * RND(1) + 1)` ◁

hij drukt af: 3

We hebben dit nog eens 10 keer herhaald. We vonden:

5 6 4 5 2 4 3 1 2 5

Dus:

INT(6*RND(1) + 1) = het aantal ogen van een worp met een dobbelsteen

We zullen in de loop van het boek nog wel meer rekenkundige functies tegenkomen. Nu volgen nog enkele opdrachten.

opdracht 1.3

Geef een PRINT-opdracht om de volgende sommen op de computer uit te rekenen.

a. $\sqrt{7,5}$

b. $\sqrt{4^2 + 5^2}$

c. $\sqrt{(4 + 5)^3}$

d. $\sqrt{7} + \sqrt{8}$

e. $\sqrt{\sqrt{5} + 7}$

opdracht 1.4

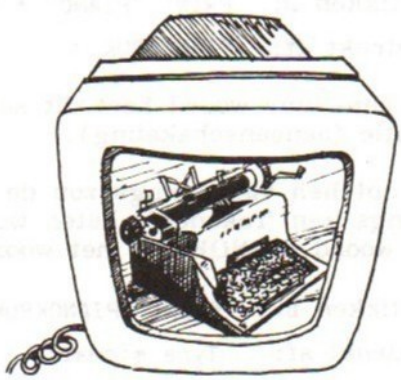
Geef een PRINT-opdracht om:

a. Het grootste gehele getal kleiner of gelijk aan $5 \times 2{,}7 + 6{,}3^2$ af te drukken.
b. Een getal X (groter dan of gelijk aan $\frac{1}{2}$ naar boven afronden) af te ronden.
c. Het grootste gehele getal kleiner dan of gelijk aan de wortel uit het getal 520 op het scherm af te drukken.
d. Het gehele getal dat het dichtst bij de wortel uit 433 ligt te berekenen.
e. Een toevalsgetal met de waarde -2, -1, 0, 1 of 2 te trekken.

We hebben in de voorbeelden en opdrachten allemaal 'ongevaarlijke' getallen gebruikt. Wat er zoal komt kijken bij het werken met minder mooie getallen (heel grote en heel kleine) komt nog uitvoerig ter sprake.

1.3 MSX BASIC EN TEKST

MSX BASIC gaat met tekst bijna net zo gemakkelijk om als met getallen. We kunnen met een directe PRINT-opdracht tekst op het beeldscherm laten afdrukken; dat gaat zo:



wij tikken in: `PRINT "TEKST"` ◁

hij drukt af: `TEKST`

Alles wat tussen aanhalingstekens staat wordt opgevat als een stuk tekst dat afgedrukt moet worden. Dat wat tussen aanhalingstekens staat noemen we een string (letterlijk is een string een aaneengeregen rij). Hier volgen een aantal voorbeelden van een (tekst)string:

een naam : KLAAS
een kenteken: HN-09-NH
onzin : AB#8%JK?*4

Elke string bestaat uit een combinatie van een of meer

- Cijfers (0, 1, 2, 3, 4, ..., 9)
- Letters (A, B, C, ..., Z; a, b, c, ..., z)
- Speciale tekens (+, -, *, /, #, %, komma, punt, spatie, ...)

opdracht 1.5

Welk teken mag, denkt u, nooit als onderdeel van een string voorkomen?

Op het toetsenbord van een MSX-computer vinden we een groot aantal toetsen. We beginnen echter met het gebruiken van de letter- en cijfertoetsen voor het intikken van eenvoudige opdrachten. De tekst die we gaan intikken bestaat eerst alleen uit hoofdletters, dus druk de hoofdlettervastzettoets (Capslock) maar in. Na enige tijd zullen we overgaan op het gebruik van kleine letters en hoofdletters. Met de SHIFT (hoofdletter-) toets, de GRPH-toets en de CODE-toets kunnen we een groot aantal tekens, waaronder grafische tekens, bijzondere tekens zoals é, ê, ë of ü gebruiken. In uw MSX-handleiding vindt u wellicht een overzicht van alle mogelijkheden.

MSX BASIC kent zeer fraaie mogelijkheden voor het werken met en manipuleren van tekst. Zo kunnen we twee teksten bij elkaar optellen:

wij tikken in: `PRINT "PIANO" + "KRUK"` ◁

hij drukt af: `PIANOKRUK`

Met een duur woord heet dit samenvoegen van twee strings **concatenatie** (aaneenschakeling).

Dit optellen van strings zou de indruk kunnen geven dat je met strings kan 'rekenen'. Laten we het eens proberen; we gaan van het woord PIANOKRUK het woord KRUK maken:

wij tikken in: `PRINT "PIANOKRUK" - "PIANO"` ◁

hij drukt af: `Type mismatch`

Gaat niet! MSX BASIC begrijpt het niet en geeft een **foutmelding**. Dit is blijkbaar niet de manier om in BASIC tekst in stukken te knippen. Toch kan het wel, maar dan zullen we van een voorgeprogrammeerde (ingebouwde) **tekstfunctie** gebruik moeten maken. Deze functie zal als 'waarde' de laatste vier letters van het woord PIANOKRUK moeten krijgen, KRUK dus. Zo'n tekstfunctie is er inderdaad; het is de RIGHT$-functie. Met deze functie kunnen we de RECHTer 'zoveel tekens' uit een string laten afdrukken. Hieronder zien we de PRINT-opdracht waarmee we de gevraagde laatste 4 (RECHTer 4) tekens uit de string "PIANOKRUK" kunnen afdrukken.

wij tikken in: `PRINT RIGHT$("PIANOKRUK",4)` ◁

hij drukt af: `KRUK`

De RIGHT$-functie is een functie met twee argumenten. Het eerste argument is een string, het tweede een positief geheel getal. De waarde van de RIGHT$-functie is in dit geval de string KRUK. Het dollarteken achter RIGHT betekent dat het om een tekstfunctie gaat. Een functienaam zonder dollarteken zoals RND en INT betekent een rekenkundige functie, dat wil zeggen een functie waarvan de waarde een getal is. Als we de 'laatste zoveel' tekens uit een string kunnen halen, kunnen we vast ook de 'eerste zoveel' tekens uit een string halen!

opdracht 1.6

Met welke opdracht (functie) zouden we de LINKer vijf letters uit PIANOKRUK op het beeldscherm kunnen laten afdrukken?
Probeer dit probleem niet op te lossen met de RIGHT$-functie. BASIC heeft hiervoor een andere functie. Zoudt u kunnen bedenken welke functie dit kan zijn?

```
 4 + 9      is een rekenkundige uitdrukking
"4 + 9"     is een string

PRINT  4 + 9      geeft 13, de waarde van de rekenkundige
                            uitdrukking
PRINT "4 + 9"     geeft 4 + 9, de waarde van de string
```

Een string kan ook uit cijfers en speciale tekens bestaan. Je bent snel geneigd om aan letters te denken als het om tekst gaat (en een string is op te vatten als een stukje tekst). Hieronder zien we een voorbeeld van het gebruik van cijfers in een string.

In BASIC kunnen we in één PRINT-opdracht een combinatie van strings en rekenkundige uitdrukkingen opnemen:

wij tikken in: `PRINT "4 + 9 ="; 4 + 9 ◁`

hij drukt af: `4 + 9 = 13`

In dit voorbeeld wordt 4 + 9 als stukje tekst gebruikt (in "4 + 9 =") maar ook als rekenkundige uitdrukking (achter de puntkomma). Je zou kunnen zeggen:

```
    de waarde van   4 + 9      is het getal 13
èn
    de waarde van  "4 + 9 ="   is de tekst 4 + 9 =
```

Hiermee kunnen we de uitkomst van een berekening door een bepaalde tekst vooraf laten gaan. Tussen de tekst en de rekenkundige uitdrukking staat een puntkomma (;); nog een voorbeeld

wij tikken in: `PRINT "AAA ZIJN";LEN("AAA");"A'S"` ◁

hij drukt af: `AAA ZIJN 3 A'S`

Deze PRINT-opdracht bevat twee strings, te weten "AAA ZIJN" en "A'S". Daartussen staat een rekenkundige uitdrukking, in dit geval een rekenkundige functie. Uit dit voorbeeld zult u begrepen hebben wat de functie LEN doet. LEN is een afkorting voor LENgth en bepaalt de lengte van de, als argument, tussen haakjes opgegeven string.

LEN is een voorbeeld van een functie waarvan het argument een string is maar waarvan de waarde een getal is. In de loop van het boek zullen we nog meer van dit soort functies tegenkomen. Ook komen we functies tegen waarvan het argument een getal of rekenkundige uitdrukking is en waarvan de waarde een string is.

LEN("string") = Aantal tekens in de tussen "-tekens opgegeven string

opdracht 1.7

Druk op het beeldscherm af dat de wortel uit negen drie is; met een passende tekst.

Laten we eens kijken of we het voorgaande kunnen gebruiken om een klein BASIC-programmaatje te maken.

1.4 EEN ECHT BASIC PROGRAMMA

De volgende vier regels vormen samen een BASIC programma:

```
10 PRINT "5 + 9 =";  5 + 9
20 PRINT "5 * 9 =";  5 * 9
30 PRINT "DE WORTEL UIT 8 IS";SQR(8)
40 PRINT "AAA ZIJN";LEN("AAA");"A'S"
```

Elke regel is een opdracht uit het programma. Laten we eens kijken wat er gebeurt als we deze vier regels in de computer invoeren.

We toetsen de eerste regel in:

wij tikken in: `10 PRINT "5 + 9 =";  5 + 9 ◁`

hij drukt af: ▮ ← niets! (alleen de cursor)

Na het invoeren van regel 10 staat de cursor aan het begin van de volgende regel. In plaats van het uitvoeren van de PRINT-opdracht slaat de computer regel 10 in zijn geheugen op. Als we toch willen dat de computer de opdracht uitvoert, moeten wij hem daartoe 'opdracht' geven. De BASIC opdracht voor het laten uitvoeren van een, in het geheugen opgeslagen, BASIC programma is **RUN**:

wij tikken in: `RUN ◁`

hij drukt af: `5 + 9 = 14`

RUN is een directe opdracht. Deze opdracht wordt direct uitgevoerd. Toch is RUN een ander soort directe opdracht dan de directe PRINT-opdracht uit 1.1. De directe opdracht **RUN** heet een **besturingsopdracht** of **commando**. Ook besturingsopdrachten, zoals **RUN**, **LIST**, **COPY** enzovoort, kunnen als programmaregels, dus met een regelnummer, voorkomen. We zullen het verschil tussen een directe opdracht en een programma-opdracht aan de hand van PRINT laten zien:

```
   PRINT "5 + 9 =";  5 + 9     is een directe opdracht
10 PRINT "5 + 9 =";  5 + 9     is een programma-opdracht
```

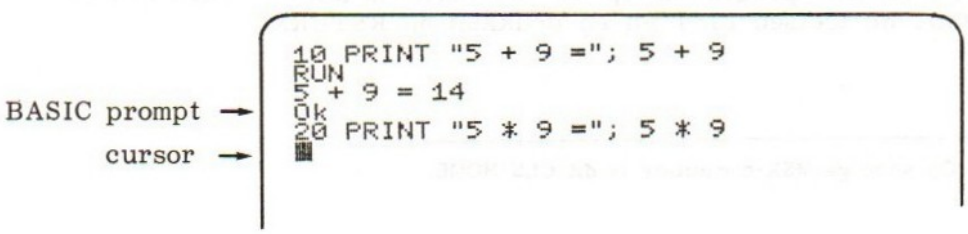
Ziet u het verschil? Een opdracht die begint met een **regelnummer** is een programma-opdracht, die pas wordt uitgevoerd als we daartoe met het commando **RUN** opdracht gegeven hebben.

Terug naar ons programma. Regel 10 staat nog steeds in het geheugen. We gaan nu de volgende regel invoeren:

Wij tikken in: `20 PRINT "5 * 9 =";  5 * 9 ◁`

hij drukt af: ▮ ← weer niets!!

Ook nu geen uitvoer als reactie op het invoeren van regel 20. Regel 20 staat nu samen met regel 10 in het geheugen. Op het scherm zien we nu:



Boven regel 20 ziet u Ok staan. Dit is de **MSX BASIC prompt.** Hiermee geeft de computer aan dat het onze beurt is om iets in te tikken. Het blokje (▮) onder regel 20 heet de **cursor.** De cursor geeft de positie op het scherm aan waar het eerstvogende, door ons ingetikte teken zal worden afgedrukt. We zullen deze prompt en cursor niet altijd in de voorbeelden laten zien.

Als we nu RUN intoetsen gebeurt het volgende:

```
wij tikken in:  RUN ◁
hij drukt af:    5 + 9 = 14
                 5 * 9 = 45
                Ok
```

Het programma in het geheugen bestaat nu uit twee regels, de regels 10 en 20. Beide opdrachten worden uitgevoerd als we RUN intoetsen; zij vormen met zijn tweetjes het hele programma. We toetsen nu de laatste twee regels in:

```
wij tikken in:  30 PRINT "DE WORTEL UIT 8 IS";SQR(8) ◁
                40 PRINT "AAA ZIJN";LEN("AAA");"A'S" ◁
```

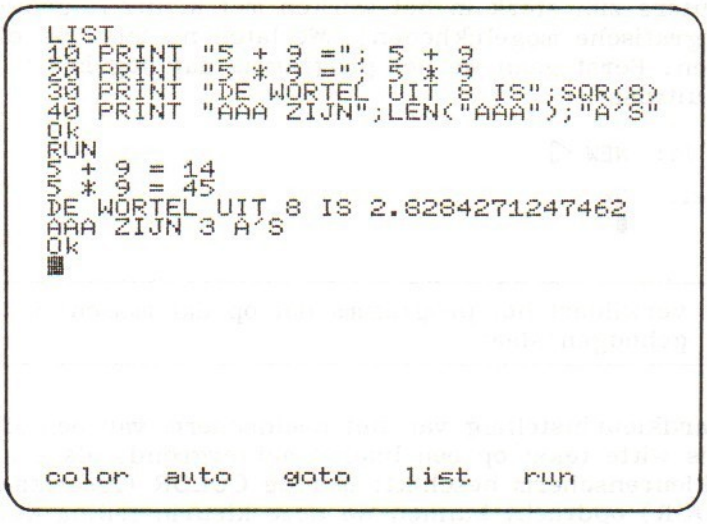
Deze twee programmaregels worden gewoon toegevoegd aan de twee die al in het geheugen staan, zodat er nu vier in staan (10, 20, 30 en 40).

Op het beeldscherm is het intussen al aardig druk geworden. De computer beschikt over een toets waarmee we het scherm leeg (schoon) kunnen maken. We zullen straks zien dat er ook een BASIC opdracht is waarmee we, vanuit een programma, het scherm schoon kunnen maken. Nu doen we het met een druk op de daarvoor bestemde toets op het toetsenbord. Dit is een combinatie van twee toetsen, de **SHIFT**- en de **CLR/HOME**-toetsen *. Als we alleen op **CLR/HOME** drukken dan gaat de cursor linksbovenaan het scherm staan maar wordt het scherm niet gewist! Zo, het scherm is schoon. Het programma staat echter nog steeds in het geheugen!

Jammer dat we het scherm gewist hebben, want we willen eigenlijk nog even zien hoe het programma er uitziet. Er is een commando (besturingsopdracht) **LIST**, waarmee we een, in het geheugen opgeslagen, programma op het beeldscherm kunnen laten afdrukken. We toetsen LIST en en drukken op RETURN:

---

* **Op sommige MSX-computers is dit CLS/HOME.**



Als reactie op het intikken van het woord LIST en het indrukken van de RETURN-toets drukt de computer het programma, dat op dat moment in het geheugen staat, af; dit geeft dus in dit geval de programmatekst van de regels 10, 20, 30 en 40. Daarna drukt hij Ok af en kunnen wij de volgende opdracht intikken. Hierboven zien we dat we vervolgens RUN hebben ingetoetst en op de RETURN-toets gedrukt hebben. Hierdoor wordt het programma, dat op dat moment in het geheugen staat, uitgevoerd. De uitvoer van dit programma wordt door de computer op het beeldscherm afgedrukt. We zien dat elke PRINT-opdracht op een nieuwe beeldschermregel begint af te drukken.

RUN	start het uitvoeren van een in het geheugen opgeslagen programma
LIST	drukt de programmatekst van een, in het geheugen opgeslagen, programma op het beeldscherm af

Dit vierregelige programma is een echt BASIC programma; daar is dan ook alles mee gezegd. In het volgende hoofdstuk gaan we echter pas 'echt' leren programmeren. Toch kennen we nu al de diverse soorten BASIC-opdrachten, te weten:

directe opdrachten	, zoals	PRINT 5 ^ 2 - 4
programma-opdrachten	, zoals	10 PRINT "5 + 9 ="; 5 + 9
functies	, zoals	SQR, INT, RND, RIGHT$, LEN
besturingsopdrachten	, zoals	RUN, LIST

MSX-computers zijn sterk in het werken met kleuren, geluid en **graphics** (grafische mogelijkheden). We laten nu iets van die kleuren zien. Eerst gaan we het programma dat nog in het geheugen staat uitwissen:

wij tikken in: `NEW ◁`

hij drukt af: `Ok`

NEW	verwijdert het programma dat op dat moment in het geheugen staat

De standaardkleurinstelling van het beeldscherm van een MSX-computer is witte tekst op een blauwe achtergrond, als u althans over een kleurenscherm beschikt! Met de **COLOR** (Amerikaans voor COLOUR) opdracht kunnen we deze kleurinstelling veranderen. Bovendien kunnen we de boven- en onderrand op het scherm een andere kleur dan de achtergrond geven. We hebben dus:

een kaderkleur (boven- en onderrand)
een achtergrondkleur (rechthoekig veld)
een afdrukkleur

Als we de MSX-computer aanzetten dan is de standaardbeeldscherminstelling de zogenaamde **SCREEN** 0-stand. Dit zijn standaard 24 regels met standaard 37 tekens. Met **WIDTH** 40◁ kunnen we hiervan 40 tekens per regel maken. In deze stand is de randkleur altijd die van de achtergrond. Wat we ook in de COLOR-opdracht zetten, het kader zal altijd de kleur van de achtergrond krijgen. In de standen **SCREEN** 1, **SCREEN** 2 en **SCREEN** 3 kunnen we het kader (de boven- en onderrand) wel een aparte kleur geven. Deze drie kleuren kunnen we met één **COLOR**-opdracht instellen. MSX-computers bieden zestien kleuren met codes 0 t/m 15.

Code	kleur	Code	kleur
0	Transparant	8	Middelrood
1	Zwart	9	Lichtrood
2	Middelgroen	10	Donkergeel
3	Lichtgroen	11	Lichtgeel
4	Donkerblauw	12	Donkergroen
5	Lichtblauw	13	Paars
6	Donkerrood	14	Grijs
7	Hemelsblauw	15	Wit

De standaardinstelling is **COLOR** 15,4,4 Voorgrond wit (15), achtergrond donkerblauw (4) en kader donkerblauw (4).

wij tikken in: `COLOR 15, 4, 5: SCREEN 1` ◁

hij drukt af: lichtblauw kader

(SCREEN 1 geeft 32 tekens op een regel in plaats van 37)

wij tikken in: `10 PRINT "MOOIE KLEUR?"` ◁

Als we dit programma (nou ja,... programma!) draaien door RUN◁ in te toetsen, na eerst het scherm schoongemaakt te hebben (SHIFT + CLR/HOME), dan wordt de tekst "MOOIE KLEUR?" wit op de donkerblauwe achtergrond afgedrukt.

wij tikken in: `SHIFT + CLR/HOME`
`COLOR 10: RUN` ◁

hij drukt af: `MOOIE KLEUR?` (in donkergeel)

wij tikken in: `SHIFT + CLR/HOME`
`COLOR 11, 13: RUN` ◁

hij drukt af `MOOIE KLEUR?` (in lichtgeel op een paarse achtergrond)

wij tikken in: `SHIFT + CLR/HOME`
`COLOR 1, 15, 6: RUN` ◁

hij drukt af: `MOOIE KLEUR?` (zwart op witte achtergrond met donkerrood kader)

Straks zullen we zien hoe we de kleur door een programma kunnen laten veranderen. Voordat we verder gaan is het goed even stil te staan bij de functietoetsen F1 t/m F10 bovenaan op het toetsenbord. Onderop het scherm zien we bij het aanzetten van de computer de tekst:

```
color    auto    goto    list    run
```

Als we op SHIFT drukken zien we vijf andere BASIC opdrachten en commando's. Deze 10 BASIC-woorden zitten 'verstopt' onder de functietoetsen F1 t/m F5 (met SHIFT F6 t/m F10). Drukken we bijvoorbeeld op F5 dan zien we de tekst "run" op het beeldscherm verschijnen en direct wordt een eventueel programma, dat in het geheugen staat, gedraaid.

Bij de bovenstaande drie COLOR-voorbeelden hadden we dus steeds op de F5-toets kunnen drukken in plaats van het intikken van RUN ◁ na de dubbele punt. Als we een programma op het scherm willen afdrukken kunnen we intikken LIST ◁, maar we kunnen net zo goed toets F4, gevolgd door de RETURN-toets, indrukken. De functietoetsen besparen ons dus een hoop 'tikwerk'.

We kunnen de tekst 'onder' deze functietoetsen zelf veranderen. In het volgende hoofdstuk gaan we programma's maken, waarin nogal vaak de BASIC-opdrachten **PRINT**, **INPUT** en **END** voorkomen. Als we de tekst van deze drie opdrachtnamen nu onder de functietoetsen **F1**, **F2** en **F3** programmeren dan scheelt dat straks een hoop tikwerk. Dit gaat als volgt:

wij tikken in:
```
KEY OFF ◁                    (verwijdert tekst onderaan het
                              beeldscherm)
KEY 1, "PRINT" ◁
KEY 2, "INPUT" ◁             (herprogrammeert de functie-
KEY 3, "END" ◁                toetsen F1, F2 en F3)
KEY ON ◁
```

hij drukt af:
```
PRINT   INPUT   END   list   run      (op de onderste
                                       regel)
```

Als we de computer uitzetten en vervolgens weer aan, krijgen we de oude functietoetsinstelling weer terug. Maak oefenopgave 10 als u een programma wilt maken voor het programmeren van functietoetsen en kijk in hoofdstuk 9 hoe u dit programma op cassetteband of op diskette moet opslaan en inlezen om het vaker te kunnen gebruiken.

De antwoorden op de opdrachten uit de tekst vindt u in Appendix A. Denkt u erom dat deze antwoorden gegroepeerd zijn per opdrachtnummer en niet per hoofdstuk. Eerst komen dus de antwoorden van de met .1 genummerde opdrachten, dan die genummerd met .2, vervolgens die genummerd met .3, enzovoorts. Hiermee wordt voorkomen dat u bij het opzoeken van het antwoord van een opdracht (per ongeluk) het antwoord van de volgende opdracht onder ogen krijgt!

Nu volgt nog een aantal oefenopgaven. De antwoorden van deze opgaven zijn wel gegroepeerd per hoofdstuk en u vindt ze in Appendix B.

OEFENOPGAVEN HOOFDSTUK 1

1. Geef een directe PRINT-opdracht voor het uitrekenen van de volgende sommetjes:

   a. $4{,}5^3$ b. $\frac{4 \times 7^2}{3}$ c. $6 \times (15{,}5 - 8{,}3)$

d. $\frac{3}{(5 + 14)}$ e. $\sqrt{15}$ f. $\sqrt{11 + \frac{4}{2{,}5^3 \times 3{,}5}}$

2. Een auto rijdt met een constante snelheid van 95 km/uur.
   a. Geef een PRINT-opdracht om uit te rekenen hoeveel kilometer de auto heeft afgelegd na een rit van $6\frac{1}{2}$ uur.
   b. Hoeveel minuten zit de bestuurder achter het stuur als de auto 175 kilometer heeft afgelegd? Geef de PRINT-opdracht.

3. Op een dag is het in Californië 95° Fahrenheit. Als je van een aantal graden Fahrenheit tweeëndertig aftrekt, dit resultaat met vijf vermenigvuldigt en ook nog eens door negen deelt, krijg je het aantal graden Celsius. Was het warm (°C) in Californië? Geef een PRINT-opdracht met passende tekst.

4. Onze lichaamstemperatuur is ongeveer 36,8°C. Hoe warm (°F) is de gemiddelde Amerikaan? Geef een PRINT-opdracht.

5. Een vast bedrag B dat we tegen een vaste rente van i% per jaar, gedurende n jaar, bij een bank 'vastzetten' groeit in die n jaar tot een eindkapitaal K dat gelijk is aan:

   $$K = B \times (1 + \frac{i}{100})^n$$

   Geef een PRINT-opdracht om uit te rekenen wat het eindkapitaal is als we 75 gulden tegen 8% per jaar gedurende 10 jaar vastzetten. Rond dit bedrag tot op hele guldens af. Geef vóór de uitkomst een passende tekst.

6. Voor een spelletje hebben we een toevalsgetal uit de reeks 0, 2, 4, 6, 8, 10 nodig, een even niet-negatief getal kleiner dan of gelijk aan tien. Geef een PRINT-opdracht waarmee we zo'n toevalsgetal kunnen oproepen.

7. Als we een getal met twee decimalen (bijvoorbeeld 8,57) op één decimaal willen afronden dan kunnen we dit doen door het getal met tien te vermenigvuldigen, vervolgens een half erbij op te tellen, van dit resultaat het decimale deel weg te halen en wat we overhouden door 10 te delen. Geef een PRINT-opdracht om het getal 8,57 op deze manier op één decimaal af te ronden.

8. Wat denkt u dat de uitkomst is van de volgende PRINT-opdracht:

```
PRINT LEN(RIGHT$("KLAVERTJEDRIE",4))
```

9. Geef de uitvoer van het volgende BASIC-programma:

```
10 PRINT "5 + 4" + "9"
20 PRINT "5 + 4" + " +" + " 9"
30 PRINT "5 + 4" + " =";  10 - 1
40 PRINT "5 + 4" + " =" + " 10 - 1"
```

10. Maak voor het programmeren van de functietoetsen F1, F2, F3 met respectievelijk de tekst **print**, **input** en **end**, een BASIC-programma. Zorg ervoor dat u de tekst tussen aanhalingstekens met kleine letters intikt en de rest met hoofdletters. Kijk in hoofdstuk 9 hoe u dit programma op cassetteband of op diskette kunt bewaren zodat u niet steeds opnieuw alle KEY-opdrachten moet intikken als u de functietoetsen met bepaalde 'teksten' wilt programmeren. U draait dan gewoon het eerder opgeslagen 'functietoetsprogrammeer'-programma.

2 Programmavariabelen, Eenvoudige Programma's

Dit hoofdstuk begint met het behandelen van het begrip programmavariabele. In de wiskunde staat het begrip variabele centraal: in de functie $x \rightarrow 2x + 3$ noemen we x de onafhankelijke variabele. De functiewaarde (2x + 3) is afhankelijk van de waarde van de variabele x. Ook in ons dagelijkse taalgebruik komen we het begrip variabele tegen; een voorbeeld hiervan is de zin:

"Dat kost mij elke maand een x-bedrag"

Meestal wordt met het begrip variabele een getalvariabele bedoeld, hetgeen betekent dat de waarde die zo'n variabele kan aannemen een getal is, een numerieke waarde. In dit hoofdstuk zullen we zien dat we in BASIC programma's naast **getalvariabelen** ook **tekstvariabelen** gebruiken. Een tekstvariabele is een variabele waarvan de waarde een stuk tekst is, een aantal tekens. Tijdens het behandelen van het begrip 'programmavariabele' zullen we af en toe een kijkje nemen in het **geheugen** van de MSX-computer.

Na de introductie van het begrip programmavariabele gaan we enkele eenvoudige **'conversationele'** programma's schrijven. BASIC is uitstekend geschikt voor het schrijven van conversationele programma's. Een conversationeel programma is een programma dat in 'samenspraak' met de gebruiker een aantal opdrachten uitvoert. De gebruiker zit achter de computer en reageert op vragen die hem of haar door het programma worden gesteld. Het programma drukt de vragen (of instructies) op het beeldscherm af. De gebruiker reageert hierop met het intoetsen van een getal, een stuk tekst of gewoon door op een bepaalde toets te drukken. Dergelijke programma's worden ook wel **interactieve** programma's genoemd; er is interactie tussen gebruiker en programma. Nog mooier gezegd: "De gebruiker voert een dialoog met het programma".

Uit hoofdstuk 1 zal duidelijk geworden zijn dat, als we de computer iets willen laten doen, wij op de **RETURN**-toets moeten drukken. Dit moet als we bijvoorbeeld een programma willen draaien (RUN en **return**), als we iets willen uitrekenen (PRINT 4 * 3 + 7 en **return**) of als we een programmaregel in de computer willen opslaan (10 PRINT "5 + 9 ="; 5 + 9 èn **return**) kortom altijd als we willen

dat de computer op onze invoer reageert. In de komende voorbeelden zullen we steeds minder vaak aangeven wanneer de RETURN-toets dient te worden ingedrukt.

2.1 PROGRAMMAVARIABELEN

Voordat we met programmeren beginnen eerst iets over het geheugen van de MSX-computer *. Het geheugen van veel MSX-computers is 80K. Dit wil zeggen dat het geheugen uit 80 × 1024 = 81.920 bytes bestaat. Een byte is de geheugenruimte waarin één teken (een letters, een cijfer, een spatie, een leesteken of een grafisch teken) opgeslagen kan worden. Als we het hele geheugen van 80K bytes (K betekent Kilo, maar dan kilo's van 1024 en niet van 1000!) zouden kunnen gebruiken, dan kunnen we daar ongeveer de tekst van bijna 23 A-4'tjes (60 regels met 60 tekens) in opslaan. Van die 80K is echter 16K gereserveerd voor het video-RAM. Dit is het deel van het geheugen waarin alle beeldschermbytes (kleur, sprites, enzovoort) in staan. Dit stuk geheugen kunnen we niet voor BASIC-programma's gebruiken. Blijven dus over 64K bytes. Hiervan wordt 32K ingenomen door de BASIC-ROM. (ROM = Read Only Memory). Dit BASIC-ROM bevat de BASIC-vertolker. Dit is grof gezegd, het in de machine ingebouwde BASIC-systeem. Blijft dus over 32K voor onze BASIC-programma's. Als we de MSX-computer met 32K RAM (RAM = Random Access Memory) voor het opslaan en draaien van BASIC-programma's echter aanzetten, dan zien we niet 32 × 1024 = '32768 Bytes free' maar slechts '28815 Bytes free' (zie pagina 2). (Bij gebruik van MSX-Disk BASIC is dit zelfs maar 24455 bytes. We hebben dus 28815 : 1024 = iets meer dan 28K geheugenruimte om BASIC-programma's en gegevens voor deze programma's in op te slaan. Dit zal echter in de meeste gevallen ruimschoots genoeg zijn.

We kunnen de nog resterende vrije BASIC-ruimte ook door de computer laten afdrukken. Dit gaat zo:

wij tikken in: `PRINT FRE(0)` ◁

hij drukt af: `28815`

Als we deze opdracht direct na het aanzetten intikken dan geeft de computer aan dat er nog 28815 geheugenplaatsen (bytes) vrij zijn. We zullen de FRE-opdracht straks gebruiken om te kijken hoeveel ruimte een programma in beslag neemt.

Een BASIC-programma bestaat uit één of meer BASIC-opdrachten die elk van een regelnummer voorzien zijn. Het kleinste BASIC-

* We gaan hierbij uit van MSX-1 computers.

programma bestaat uit één opdracht. Laten we als voorbeeld de opdracht uit regel 10 van het programmaatje uit § 1.4 nemen. We nemen nu als regelnummer 50, dan kunnen we er straks nog een aantal opdrachten (met lagere regelnummers) vóór zetten.

```
50 PRINT "5 + 9 ="; 5 + 9
```

Je zou dit programma een programma voor het afdrukken van de som van twee getallen kunnen noemen. Dit programma is echter alleen geschikt voor het afdrukken van de som van de getallen vijf en negen. Eigenlijk zou dit programma algemeen toepasbaar moeten zijn; het zou geschikt moeten zijn voor het afdrukken van de som van twee willekeurige getallen. In het algemeen treffen we bij het draaien van een programma de volgende situatie aan:

De **invoer** voor een computerprogramma is in het algemeen variabel; we kunnen hetzelfde programma steeds met een andere invoer draaien en krijgen dan een bij de invoer passende uitvoer. Welke **uitvoer** we krijgen hangt af van de **opdrachten** in het programma. Als we ons éénregelige programma (regel 50) geschikt willen maken voor het afdrukken van twee willekeurige getallen, moeten we in regel 50 de 5 en de 9 vervangen door ....., precies, door variabelen, in dit geval getalvariabelen, dus door iets wat steeds andere waarden kan aannemen.

Een programmavariabele in BASIC geven we aan met een **naam**. Zo'n naam moet altijd met een letter beginnen en bestaat uit een combinatie van letters en/of cijfers. Dit zijn goede BASIC namen voor getalvariabelen:

A ; AB ; A1 ; SOM ; GTAL ; PI ; RENTE

We moeten echter oppassen want we kunnen niet zomaar elke willekeurige naam voor een variabele kiezen.

- Alle BASIC-woorden (zoals **PRINT**, **INPUT**, **RUN**, **LIST**, enzovoorts) kunnen niet als naam voor een variabele gekozen worden. In Appendix C staat een overzicht van deze namen.
- Een BASIC-woord (zie Appendix C) mag ook niet als deel van een naam voorkomen. Zo mogen we de naam LENGTE niet gebruiken want hierin zit het BASIC-woord **LEN**; ook SPRINT mag niet want daar zit **PRINT** en **INT** in.
- De MSX-computer kijkt alleen naar de eerste twee letters van een naam; alle andere letters van de naam onthoudt hij wel maar hij herkent ze niet. Dit betekent dat de variabelen PRIJS1 en

PRIJS2 voor de computer dezelfde namen zijn, terwijl hij P1 en P2 wel als verschillende namen ziet.

De meest eenvoudige naam voor een getalvariabele is dus een naam bestaande uit slechts één letter. Laten we in regel 50, zie vorige bladzijde, in plaats van de 5 en de 9 dergelijke namen introduceren:

```
50 PRINT "A + B ="; A + B
```

Laten we eens aannemen dat de variabele A de waarde 5 en de variabele B de waarde 9 heeft, dan is dit de uitvoer van dit éénregelige programma:

```
RUN
A + B = 14
```

Omdat "A + B =" een string is en A + B een rekenkundige uitdrukking zal eerst de string afgedrukt worden (de tekst A + B =) en daarachter de waarde van de rekenkundige uitdrukking A + B (14 dus). Wat A en B ook moge zijn, altijd zal "A + B =" worden afgedrukt. We willen echter niet A + B = 14 maar 5 + 9 = 14 laten afdrukken. Dit kan als we regel 50 als volgt wijzigen:

```
50 PRINT A; "+"; B; "="; A + B
```

Nu zal achtereenvolgens worden afgedrukt:

- de waarde van A
- een + teken
- de waarde van B
- een = teken
- de waarde van A + B

De waarden die aan A en B worden toegekend zouden we de invoer voor het programma kunnen noemen. De vraag is nu: hoe krijgen A en B respectievelijk de waarde 5 en 9? Welnu, dat kan in BASIC op vele manieren. Een van die manieren is de variabelen in het programma zelf waarden te geven. Laten we met deze methode beginnen. We breiden ons programma met twee opdrachten uit. In de eerste opdracht (regel 30) krijgt A de waarde 5, in de tweede (regel 40) krijgt B de waarde 9:

```
30 A = 5
40 B = 9
50 PRINT A; "+"; B; "="; A + B
```

De opdrachten in regel 30 en 40 noemen we **toekenningsopdrachten.**

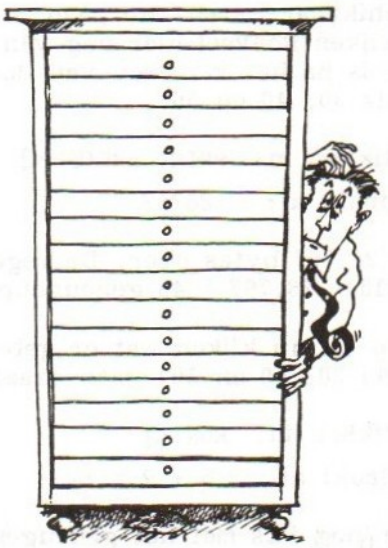
We kunnen regel 30 en regel 40 gewoon intikken. De computer zal ze in zijn geheugen vóór regel 50 plaatsen. We mogen ook gerust eerst regel 40 en dan regel 30 invoeren. De computer rangschikt de programmaregels volgens oplopende regelnummers, ongeacht de volgorde waarin wij die regels invoeren.

Met een toekenningsopdracht geven we een variabele een waarde. De naam van de variabele staat altijd links van het = teken. De toe te kennen waarde staat altijd rechts van het = teken.

```
A = 4  spreken we uit als  'A wordt vier'
```

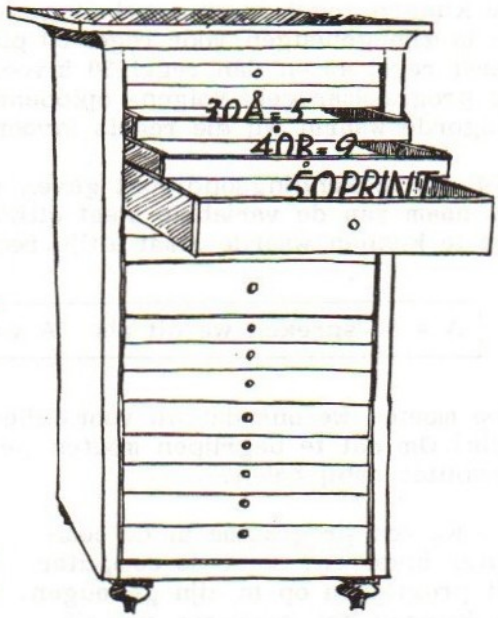
Hoe moeten we ons dat nu voorstellen, zo'n naam van een variabele? Om dat te begrijpen moeten we even het geheugen van de computer erbij halen.

Als we een programma in de computer invoeren, slaat de computer dit programma op in zijn geheugen. We kunnen het geheugen van de computer voorstellen als een enorm grote ladenkast met allemaal even grote (liever gezegd even kleine) laatjes. We weten nu dat veel MSX-computers 81.920 (80K) van deze laatjes in hun geheugen hebben, waarvan wij er in BASIC zo'n 28815 (of bij disk-BASIC 24455) kunnen gebruiken. (We kunnen er eerlijk gezegd nog iets meer gebruiken. Hoe? Dat wordt niet in dit boek behandeld!) Als we een programma ingevoerd hebben, ligt dit programma verdeeld over een aantal van die laatjes opgeslagen in het geheugen.



Zo'n standaardgeheugenla in een MSX-computer is precies groot genoeg om 8 bits (een bit is een 0 of een 1) op te kunnen slaan. Met 8 bits kunnen we $2^8 = 256$ verschillende patronen van nullen en énen maken. Als elk van zo'n patroon een teken (cijfer, letter, leesteken) voorstelt dan kunnen we hiermee dus 256 verschillende tekens coderen. Een standaardgeheugenla bevat dus één teken. Het opslaan van regel 30 (30 A = 5) kost dus al heel wat van deze standaardlaatjes. 8 bits wordt een byte genoemd.

Als we even aannemen dat de computer een aantal standaardlaatjes tot een grotere lade kan samenvoegen, zien we in de tekening hiernaast hoe de regels 30, 40 en 50 in het geheugen liggen opgeslagen.



Hoeveel geheugenruimte nemen deze drie regels nu in beslag? We weten dat we met de functie FRE(0) de nog beschikbare geheugenruimte kunnen opvragen. Aan het begin van deze paragraaf zagen we dat vlak na het aanzetten van de computer 28.815 geheugenplaatsen of bytes (1 byte = 8 bits) beschikbaar waren. We gaan nu kijken hoeveel hier nog van over is na het invoeren van de regels 30, 40 en 50:

Wij tikken in: PRINT FRE(0) ◁

hij drukt af: 28767

Nog 28.767 bytes over. De regels 30, 40 en 50 nemen dus 28.815 - 28.767 = 48 geheugenplaatsen in beslag.

Laten we nu kijken wat er gebeurt als we het programma (de regels 30, 40 en 50) gaan draaien:

wij tikken in: RUN ◁

hij drukt af 5 + 9 = 14

Is er nog iets met het geheugen gebeurt? We gaan kijken:

wij tikken in: PRINT FRE(0) ◁

hij drukt af: 28745

Na het draaien van het programma is er nog minder geheugen (28.745 bytes) beschikbaar. Hoe kan dat? Wel, voordat het programma gedraaid werd, stonden alleen de regels 30, 40 en 50 in het geheugen. Na het intikken van RUN begint de computer met het uitvoeren van deze drie regels. In regel 30 staat dat hij de waarde 5 moet toekennen aan de variabele A. Wat er in feite gebeurt, is dat hij op een of andere geheugenla het etiket "A" plakt en de waarde 5 in deze la doet. Dat kost dus ruimte. Zo

krijgen we ook een B-la met inhoud 9; dat kost nog meer ruimte. Het uitvoeren van regel 50 kost geen extra ruimte omdat er géén 'A + B'-la gereserveerd wordt; de waarde 14 (A + B) wordt niet bewaard, maar alleen uitgerekend en afgedrukt. We zien dus dat na afloop van het programma de computer nog twee laden gevuld heeft: de A-la en de B-la en dat kost blijkbaar 22 bytes.

Om te laten zien dat na afloop van het programma de A-la nog bestaat doen we dit:

wij tikken in: PRINT A ◁

hij drukt af: 5 ← de A-la is er dus nog!

Als we nu dit programma met andere waarden voor A en B willen draaien, dan hoeven we slechts de regels 30 en 40 opnieuw in te tikken. Regel 50 kunnen we onveranderd laten staan:

wij tikken in: 30 A = 25 ◁
40 B = 14 ◁
RUN ◁
hij drukt af: 25 + 14 = 39

We zullen straks zien hoe we met een **conversationeel** programma op een slimmere manier de waarden van A en B kunnen veranderen.

De namen A en B zijn namen voor getalvariabelen. Er zijn echter ook tekstvariabelen. De computer moet onderscheid kunnen maken

tussen deze twee typen variabelen, al was het alleen maar om de gebruiker er op te wijzen dat niet alles wat met getalvariabelen mag ook toegestaan is voor tekstvariabelen en omgekeerd. Voordat we laten zien hoe namen voor tekstvariabelen er uitzien gaan we nog iets dieper in op getalvariabelen; maar eerst nog een opdracht.

opdracht 2.1

Schrijf een programma voor het vermenigvuldigen van twee willekeurige getallen. Geef aan hoe de opdrachten er uitzien voor het vermenigvuldigen van de getallen 15 en 25. Gebruik in het programma twee getalvariabelen. De uitvoer moet er zo uitzien:

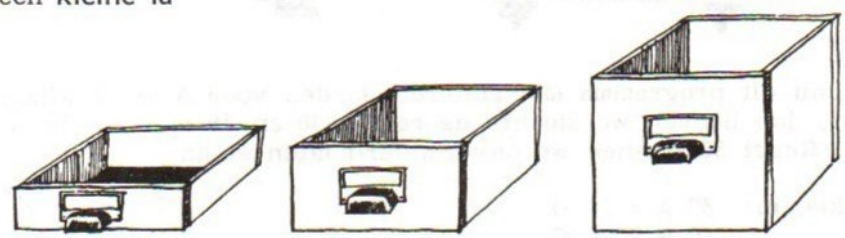
15 MAAL 25 IS 375

2.2 DRIE SOORTEN GETALVARIABELEN

In de vorige paragraaf hebben we gezegd dat de naam van een getalvariabele door de computer geassocieerd wordt met een stukje geheugen waarin de waarde van die variabele kan worden opgeslagen. Dit stukje geheugen hebben we voorgesteld als een lade van een bepaalde afmeting.

BASIC kent voor het opslaan van getalwaarden drie soorten laden, te weten

- een standaardla
- een middelgrote la
- een kleine la



KLEIN MIDDELGROOT STANDAARD

Hoe weet de MSX-computer nu welk type la hij voor een bepaalde variabele moet reserveren? Wel, hij kiest altijd de standaardla, tenzij wij hem zelf vragen om een kleine, dan wel een middelgrote la te kiezen. In het hierna volgende programmafragment kiest de computer een standaardla voor het opslaan van de waarde van de variabele GTAL.

Laten we eens kijken wat voor soort waarden we in zo'n standaardla kunnen opbergen.

```
10 GTAL = 10/7
20 PRINT GTAL
RUN
 1.4285714285714
```

In dit programma kennen we aan de variabele GTAL de waarde $\frac{10}{7}$ toe en laten vervolgens deze waarde afdrukken. De uitvoer is een decimaal getal van 14 cijfers. Laten we de variabele GTAL eens een andere waarde geven (we veranderen alleen regel 10!):

```
10 GTAL = 25/11
RUN
 2.2727272727273
```

Weer een decimaal getal van veertien cijfers. Nog één:

```
10 GTAL = 10000/6
RUN
 1666.6666666667
```

Alweer veertien cijfers, afgerond op het laatste cijfer.

Als we de inhoud van zo'n standaardla laten afdrukken, krijgen we blijkbaar altijd een decimale uitkomst van maximaal veertien cijfers, afgerond op het laatste cijfer.

BASIC spreekt niet over een 'standaardla' maar over een **double precision variabele**. Waarom double? Wel, omdat er ook single precision variabelen bestaan. Kijk maar:

```
10 GTAL! = 25!/11!          (25/11 mag ook)
20 PRINT GTAL!
RUN
 2.27273
```

en

```
10 GTAL! = -10000!/6!       (-10000/6 mag ook)
RUN
-1666.67
```

Door achter de naam van de variabele en eventueel achter de daaraan toe te kennen waarde het teken ! te zetten, drukt de computer plotseling 6 cijfers af. We noemen GTAL! een **single precision variabele**. Minder cijfers betekent minder decimalen dus minder nauwkeurig rekenen.

U begrijpt het al, een single precision variabele komt overeen met een middelgrote la. De geheugenruimte die een single precision variabele in beslag neemt is twee keer zo klein als de ruimte voor een double precision variabele. Gebruik dit soort variabelen dan ook alleen als met een kleinere nauwkeurigheid gewerkt kan worden.

In dit voorbeeld hebben we een negatieve waarde aan GTAL! toegekend. U ziet dat de **PRINT**-opdracht nu een minteken afdrukt. Bij positieve getallen wordt echter geen plusteken afgedrukt. Bovendien wordt na elk afgedrukt getal altijd één spatie afgedrukt.

Nu zult u wellicht denken dat de kleine la overeenkomt met een half precision variabele, maar dan hebt u het mis. De kleine la komt overeen met een **integer variabele**. Een integer variabele is een variabele waarvan de waarde altijd een geheel getal is. We geven een integer variabele aan door achter de naam van de variabele het teken % te zetten. De computer zorgt er dan voor dat zo'n variabele ook altijd een geheeltallige waarde heeft:

```
10 GTAL% = 25/11
20 PRINT GTAL%
RUN
 2

en

10 GTAL% = 10000/6
RUN
 1666
```

Inderdaad zien we dat nu steeds een geheel getal als waarde van GTAL% wordt afgedrukt. Het %-teken werkt net als **INT**. Een integer waarde neemt een kwart van de ruimte van een standaardwaarde in beslag. Dit komt omdat er geen decimaal gedeelte te onthouden valt!

Vooral bij grote programma's is het raadzaam, daar waar mogelijk deze geheeltallige variabelen te gebruiken; het scheelt immers heel wat geheugenruimte. Kort samengevat:

GTAL%	-	integer variabele
GTAL!	-	single precision variabele
GTAL	-	double precision variabele

Voor alle volledigheid vermelden we nog dat we in plaats van GTAL ook GTAL# mogen gebruiken. Het hekje betekent dat de variabele een double precision variabele is. Vrijwel altijd laten we echter dit hekje weg. De MSX-computer werkt dus standaard in double precision.

Programmeurs spreken dikwijls over **'reals'** en **'integers'** als ze **numerieke-**, respectievelijk **geheeltallige-** variabelen bedoelen.

opdracht 2.2

Bekijk het volgende programmafragment:

```
10 X=5.6
20 XA%=X
30 XA%=X + .5
40 XA%=INT(X) + .5
50 XA%=X - 7
```

Welke waarde krijgt de variabele XA% in de regels 20, 30, 40 en 50? (Zie voor INT § 1.2.)

Tot slot van deze paragraaf nog een klein programmaatje met alleen maar geheeltallige variabelen:

```
10 T%=100
20 N%=12
30 G%=INT(T%/N%)
40 R%=T%-G%*N%
50 PRINT T%;":";N%;"=";G%;"rest";R%
```

opdracht 2.3

a) Wat is de uitvoer van bovenstaand programma als we het zouden intikken en draaien?

b) Hoe kunnen we dit programma draaien met voor T% en N% respectievelijk de waarden 35 en 8.

We kunnen in MSX BASIC ook 'geheeltallig delen' met \ en MOD. Tik in:

```
PRINT "100 : 12 ="; 100\12; "REST"; 100 MOD 12
```

100\12 geeft de waarde van G% in bovenstaand programma, terwijl 100 MOD 12 de waarde van R% oplevert.

Door, waar dat kan, geheeltallige variabelen te gebruiken maken we een programma sneller. We zullen deze tijdwinst in hoofdstuk 10 met behulp van de, in de MSX-computer, aanwezige klok, meten.

De waarde die we aan een geheeltallige variabele kunnen toekennen loopt van -32.768 tot en met +32.767.

2.3 TEKSTVARIABELEN

Een tekstvariabele of stringvariabele is een variabele waaraan we als waarde een stuk tekst kunnen toekennen. Een variabele wordt een tekstvariabele door achter de naam een **dollarteken** ($) te zetten. In het volgende programma ziet u een voorbeeld:

```
10 W1$="MICRO"
20 W2$="COMPUTER"
30 PRINT W1$;" + ";W2$;" geeft ";
          W1$ + W2$

run
MICRO + COMPUTER geeft MICROCOMPUTER
```

De variabelen W1$ en W2$ zijn tekstvariabelen.
In regel 10 wordt aan W1$ de waarde MICRO toegekend en in regel 20 aan W2$ de waarde COMPUTER. Regel 30 beslaat 2 beeldschermregels. Geef na de ; geen RETURN, maar gebruik de spatiebalk om onder de W op de tweede regel te komen en tik daar W1$ + W2$. Geef dan pas RETURN.

Hoe zit het met het toekennen van geheugenruimte aan tekstvariabelen? Voor tekstvariabelen bestaan geen kleine en standaardladen! Als de computer in een programma ziet dat een tekstvariabele een bepaalde waarde krijgt, wordt op dat moment voor die tekstvariabele een geheugenla gemaakt die precies groot genoeg is om de string, die als waarde aan de tekstvariabele wordt toegekend, in op te bergen. (Met een duur woord noemen we dit dynamische stringallocatie!)
Wel is het aantal tekens in een string, die als waarde aan een tekstvariabele wordt toegekend, aan een maximum gebonden. Bij MSX-computers is de systeemgekozen waarde 255 tekens. De standaardruimte die MSX BASIC reserveert voor het opslaan van alfanumerieke waarden (strings) is 200 tekens. Met CLEAR 500 kunnen we deze ruimte bijvoorbeeld vergroten tot 500 tekens.

Na de toekenningsopdracht W1$ = "MICRO" heeft de computer een geheugenlade gereserveerd precies groot genoeg om de string MICRO in op te bergen. Op deze lade plakt de computer een etiket met het opschrift W1$ en in de lade stopt hij de string MICRO.

opdracht 2.4

In het volgende programmafragment wordt een aantal tekstvariabelen en tekstfuncties (zie § 1.3) gebruikt. Kunt u opschrijven wat de uitvoer van dit programma zal zijn?
Tik dit programma in met de CAPS/LOCK uit (lampje uit). De tekst tussen "-tekens zal hierdoor in kleine letters blijven staan. BASIC-woorden en de namen van variabelen worden in de computer in hoofdletters veranderd. U ziet dit als u na het invoeren in kleine letters de opdracht LIST geeft.

```
10 W1$="programma"
20 AANTAL%=LEN(W1$)
30 LINKS$=LEFT$(W1$,1)
40 RECHTS$=RIGHT$(W1$,1)
50 PRINT "raad het woord"
60 PRINT "het zijn";AANTAL%;"letters"
70 PRINT "de eerste is een ";LINKS$
80 PRINT "de laatste is een ";RECHTS$
```

Nu volgt nog een aantal belangrijke opmerkingen over het geven van namen aan variabelen.

- Zorg dat de naam van een variabele iets zegt over de betekenis van de variabele. Gebruik bijvoorbeeld RENTE in plaats van alleen een R; SOM in plaats van S, enzovoorts. Dit verhoogt de leesbaarheid van de programma's. Ook anderen moeten begrijpen waar het programma over gaat!

- Gebruik aan de andere kant ook weer niet te lange namen. Gebruik bijvoorbeeld ARTNUM in plaats van ARTIKELNUMMER. Lange namen kosten meer geheugenruimte dan korte namen en maken de programma's (onnodig) langer.
  In principe mag een naam 255 tekens lang zijn; maar pas op, MSX-computers kijken alleen naar de eerste twee letters. Zorg er daarom voor dat de naam van twee verschillende variabelen nooit dezelfde twee beginletters hebben! Als wij wel soms wat lange namen gebruiken, is dat bedoeld om de betekenis van zo'n variabele via zijn naam nog eens extra duidelijk te maken.

- Gebruik nooit als naam voor een variabele een van de woorden (opdrachten, functies) uit de BASIC-taal zelf, dit is verboden. U vindt alle BASIC-woorden in Appendix C.

- Als we in een programma een variabele gebruiken waaraan nog geen waarde (door het programma) is toegekend, bezit zo'n variabele wel een door de computer aan die variabele toegekende waarde. We spreken dan van een **default-waarde**; een standaard-waarde zou je kunnen zeggen. Voor een getalvariabele is dit de waarde nul en voor een tekstvariabele is dit de lege string. De lege string is een string zonder tekens. (De lege string geven we aan met "".)

default-waarde of systeemgekozen waarde is een waarde die de computer zelf kiest

Denk erom dat de lege string ("") iets anders is dan de string die bestaat uit één spatie (" "). Een spatie is ook een teken! We geven nu een programmavoorbeeld waarin de bovengenoemde default-waarden tot uitdrukking komen. Stel dat we het volgende tweeregelige programma draaien:

```
10 PRINT GTAL
20 PRINT LEN(W1$)
RUN
 0
 0
```

De eerste nul is de defaultwaarde van GTAL. De tweede nul is de lengte van de tekstvariabele W1$. Daar de defaultwaarde van W1$ de lege string is, is de lengte ervan 0 (de lege string is een string met nul tekens). Er zit niets in de lege string! Laten we nu aan W1$ als waarde één spatie toekennen en het programma nog eens draaien. We 'frommelen' een nieuwe regel tussen regel 10 en regel 20:

```
15 W1$ = " "
RUN
 0
 1
```

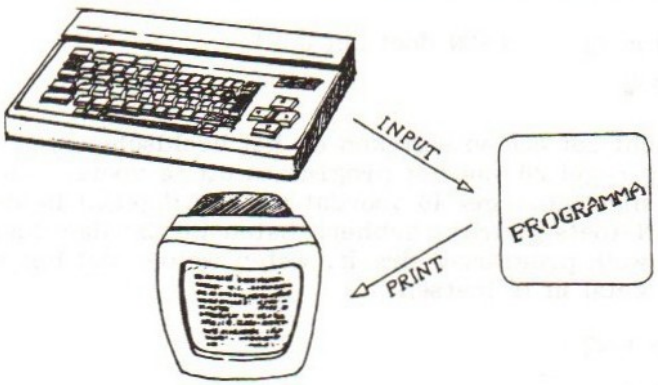
hier staat een spatie

Nu is de lengte van W1$ 1; namelijk die ene spatie. Een spatie telt dus gewoon voor één teken.

2.4 EEN PROGRAMMA VRAAGT OM INVOER

In § 2.3 hebben we gezien hoe we met behulp van toekenningsopdrachten (30 A = 5 of 10 W1$ = "MICRO") bepaalde waarden aan variabelen kunnen toekennen. Hierbij is dus al in het programma aangegeven welke waarden de verschillende variabelen zullen hebben als we het programma gaan draaien.

In BASIC is er ook een manier om tijdens het draaien van een programma aan programmavariabelen waarden toe te kennen. Dit is één van de sterke punten van BASIC!



De BASIC-opdracht waarmee we dit kunnen doen is de **INPUT**-opdracht. Met de INPUT-opdracht kan vanuit een programma aan de gebruiker, die op dat moment achter de computer zit, gevraagd worden een bepaalde waarde in te toetsen. Met de INPUT-opdracht kunnen we dus een **conversationeel** (interactief) programma maken. Uitkomsten van berekeningen of tekstbewerkingen worden met PRINT-opdrachten op het beeldscherm afgedrukt. Hierop kan het programma dan weer door middel van INPUT-opdrachten de gebruiker om een reactie vragen. Op deze manier ontstaat een dialoog tussen programma en gebruiker.

Als voorbeeld voor het gebruik van de INPUT-opdracht nemen we het programma van p. 26. Hier is dit programma nog eens:

```
30 A = 5
40 B = 9
50 PRINT A; "+"; B; "="; A + B
```

We gaan er een interactief programma van maken:

```
30 INPUT A
40 INPUT B
50 PRINT A; "+"; B; "="; A + B
60 END
```

We zien de INPUT-opdrachten in de regels 30 en 40. We hebben nog een nieuwe opdracht toegevoegd, de END-opdracht in regel 60. Deze opdracht heeft niets met interactieve programma's te maken. We spreken alleen af dat we van nu af aan elk programma beëindigen met een END-opdracht. De END-opdracht betekent gewoon EiNDe programma. De computer weet dan dat er geen opdrachten meer uitgevoerd behoeven te worden.

Hoe werkt de INPUT-opdracht?
Welnu, dat merken we vanzelf als we het bovenstaande programma gaan draaien (dan moeten we het programma zelf natuurlijk eerst intoetsen!):

wij tikken in: run ◁ (RUN doet het ook!)

hij drukt af: ? ▮

De INPUT-opdracht zet een vraagteken op het beeldscherm. De computer is bezig regel 30 van het programma uit te voeren. De computer begint niet aan regel 40 voordat wij iets ingetikt hebben en op de RETURN-toets gedrukt hebben. Laten we dat dan doen. Omdat wij weten welk programma draait, weten we ook dat het de bedoeling is een getal in te toetsen.

wij tikken in: ? 5 ◁

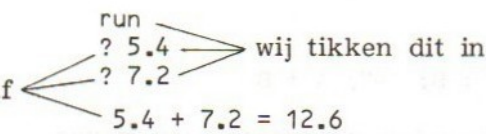
hij drukt af: ? ▮

Opnieuw drukt de computer een vraagteken af; dit als gevolg van het uitvoeren in de INPUT-opdracht in regel 40. Wij moeten nu weer iets doen (het tweede getal intoetsen):

wij tikken in: ? 9 ◁

hij drukt af: 5 + 9 = 14
Ok

De computer drukt het sommetje af (regel 50). Daaronder zien we dat de computer de BASIC-prompt (Ok) afdrukt ten teken dat het programma beëindigd is en dat we iets anders kunnen gaan doen. Laten we het programma nog eens draaien, we tikken gewoon weer run in:



We kunnen dit interactieve programma steeds weer opnieuw met telkens andere invoer draaien. In plaats van het intikken van run ◁ kunnen we ook de toetsen SHIFT/F5 indrukken!

opdracht 2.5

Wat zou er gebeuren als we achter het vraagteken een string (zoiets als "hallo") zouden intikken, terwijl de computer een getal verwacht? Eerst beantwoorden, daarna proberen!

2.5 INPUT OPDRACHT MET AANWIJZING

Stel dat het optelprogramma uit de vorige paragraaf een programma is uit een pakket computerlessen dat leerlingen van een lagere school kunnen gebruiken als extra oefenstof. De leerling moet dan maar weten dat achter het vraagteken een getal moet worden ingetikt en niet bijvoorbeeld een string! Het zou aardig zijn de leerling een aanwijzing te geven over datgene wat ingetoetst moet worden. Zo'n aanwijzing (ook wel **promptstring** genoemd) kan in een **INPUT**-opdracht worden opgenomen. We hebben in het optelprogramma in de **INPUT**-opdrachten in regel 30 en in regel 40 zo'n aanwijzing opgenomen:

```
30 INPUT "tik het eerste getal in";A
40 INPUT "tik het tweede getal in";B
50 PRINT A; "+"; B; "="; A + B
60 END
```

Na de aanwijzing moet een ; staan. Als we het programma draaien met voor A en B de getallen 5 en 9, zien we deze uitvoer:

wij tikken in: `run`

hij drukt af:
```
tik het eerste getal in? 5
tik het tweede getal in? 9
 5 + 9 = 14
```
> dit tikken wij in

Wellicht is het nog informatiever om op het scherm te laten afdrukken wat het programma doet. Daartoe tikken we regel 20 in. Het programma ziet er dan zo uit:

```
20 PRINT "Ik tel twee getallen op"
30 INPUT "tik het eerste getal in";A
40 INPUT "tik het tweede getal in";B
50 PRINT A; "+"; B; "="; A + B
60 END
```

In de praktijk komt het vaak (heel vaak) voor dat we een programma onder ogen krijgen dat door een ander gemaakt is. Het kan erg moeilijk zijn (soms is het onmogelijk!) een programma van een ander te lezen, laat staan te begrijpen wat het programma doet. De programmamaker kan hier iets aan doen door in het programma een aantal commentaarregels op te nemen. Een commentaarregel is gewoon een BASIC opdracht, de REMARK-opdracht, met een regelnummer. Deze REM-opdrachten worden niet door de computer uitgevoerd, ze worden alleen in de programmatekst opgenomen en als zodanig ook afgedrukt. REM-opdrachten kunnen overal in het programma worden opgenomen; vooraan om te vertellen wat het programma doet, wie het geschreven heeft, informatie voor de gebruiker, enzovoorts. Ook kan commentaar midden in een programma

worden opgenomen om bepaalde programmatechnische zaken nader toe te lichten. Laten wij ons programma ook voor commentaar voorzien:

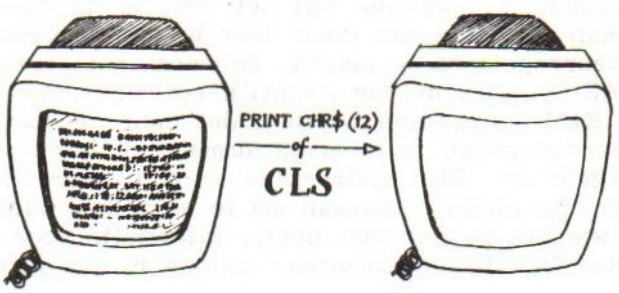
```
10 REM demonstratie van een inter-
       actief programma
20 PRINT "Ik tel twee getallen op"
30 INPUT "tik het eerste getal in";A
40 INPUT "tik het tweede getal in";B
50 PRINT A; "+"; B; "="; A + B
60 END
```

Het commentaar dat we in een REM-opdracht opgenomen hebben beslaat twee beeldschermregels. Gewoon na het afbreekstreepje de spatiebalk blijven indrukken tot de cursor onder de 'd' staat op de volgende regel en dan de rest tikken. Daarna pas RETURN. Een opdracht, inclusief regelnummer, mag maximaal 255 tekens bevatten!

In interactieve programma's komt het vaak voor dat vanuit een programma opdracht aan de computer moet kunnen worden gegeven om het scherm schoon te maken. De gebruiker die achter de computer zit kan dit te allen tijde zelf doen door tegelijkertijd de SHIFT- en CLR/HOME-toetsen in te drukken, maar een gebruiker kan natuurlijk niet weten op welke momenten het programma het nodig vindt het scherm schoon te maken.

Elke computer, dus ook een MSX-computer, kan vanuit een programma het beeldscherm **schoonmaken**. Aangezien een programma niet, zoals wij, over vingers beschikt om toetsen aan te slaan, moet er een manier zijn om het effect van het indrukken van een bepaalde toets in een programma na te bootsen zonder dat wij zelf, tijdens het uitvoeren van dat programma, daadwerkelijk die toets moeten indrukken.

Het is mogelijk om met een PRINT-opdracht en met behulp van de CHR$-functie (deze functie komt later in het boek uitvoeriger aan de orde) vanuit een BASIC-programma bepaalde besturingsopdrachten als scherm schoonmaken (Form Feed) of nieuwe regel (Line Feed) te geven. Bij MSX-computers is PRINT CHR$(12) de opdracht voor het schoonmaken van het beeldscherm. Omdat dit



een nogal lange opdracht is, voorziet MSX BASIC in de CLS-opdracht. Laten we deze CLS-opdracht (als regel 15) opnemen plus nog twee 'lege' PRINT-opdrachten (als regels 25 en 45), waardoor twee blanco regels worden afgedrukt, hetgeen de tekst op het beeldscherm wat beter leesbaar maakt. Uiteindelijk ziet het interactieve programma er dan zo uit:

```
10 REM demonstratie van een inter-
       actief programma
15 CLS ' of print chr$(12)
20 PRINT "Ik tel twee getallen op"
25 PRINT
30 INPUT "tik het eerste getal in";A
40 INPUT "tik het tweede getal in";B
45 PRINT
50 PRINT A; "+"; B; "="; A + B
60 END
```

Ziet u de ' na CLS in regel 15? Na een ' mogen we commentaar opnemen. De computer ziet alles na een ' als commentaar. Als we dit programma zouden invoeren en draaien, zien we op het beeldscherm:

```
Ik tel twee getallen op

tik het eerste getal in? 5
tik het tweede getal in? 9

 5 + 9 = 14
Ok
```

opdracht 2.6

Wat moeten we in bovenstaande situatie doen wanneer we het programma nog eens, maar dan met andere invoergetallen, willen draaien?

Laten we even aannemen dat we na het draaien van bovenstaand programma nog even achter de computer blijven zitten. Op het beeldscherm zien we als laatste afgedrukte tekens de BASIC prompt (Ok) en de cursor (▮). Dit betekent dat de computer niet meer bezig is met het uitvoeren van een BASIC programma maar gewoon wacht op datgene wat wij zullen gaan doen. We kunnen nu alles doen wat in BASIC is toegestaan. Laten we eens een aantal suggesties doen:

- We zouden met de directe opdracht **print a ◁** de laatste waarde die variabele A gekregen heeft kunnen laten afdrukken.

- We zouden een **nieuw programma** kunnen gaan intikken.
  Als we dit willen, moeten we wel bedenken dat het oude programma nog steeds in het geheugen staat en dat, als we andere regelnummers in het nieuwe programma gaan gebruiken, het oude en nieuwe programma 'gemixed' worden, hetgeen tot zeer vreemde uitvoer aanleiding kan geven.
  We moeten er altijd voor zorgen dat, voordat we een nieuw programma gaan intikken, we met een schoon geheugen beginnen. Dit doen we door voor het intikken van de eerste programmaregel de besturingsopdracht **new** in te tikken en op **RETURN** te drukken.
- We zouden de programmatekst nogmaals op het scherm kunnen laten afdrukken (**list** ◁).
- We zouden het programma nog eens kunnen draaien (**run** ◁).
- We zouden het programma, dat in het geheugen staat, op magneetband of op een diskette kunnen opnemen.
- We zouden de computer gewoon 'uit' kunnen zetten, maar dan zijn we het programma wel kwijt!

èn nog veel meer!

In Appendix C vindt u een overzicht van alle besturingsopdrachten (commando's) die we in BASIC kunnen geven.

We kunnen met een **INPUT** opdracht natuurlijk ook een **waarde** aan een **tekstvariabele** toekennen:

```
10 input "Tik een woord in";wrd$
20 print "Dit woord bestaat uit";
         len(wrd$); "letters."
30 end
run
Tik een woord in? pereboom
Dit woord bestaat uit 8 letters.
Ok
```

opdracht 2.7

In de **SCREEN 2**-stand bestaat het scherm van een MSX-computer uit 256 puntjes horizontaal (genummerd 0 t/m 255) en 192 puntjes verticaal (genummerd 0 t/m 191). Met de opdracht **LINE (X1,Y1)-(X2,Y2)** kunnen we op dit scherm een lijn trekken tussen de punten **P1** en **P2** met coördinaten **(X1,Y1)** en **(X2,Y2)**. X1 en X2 zijn het aantal puntjes horizontaal vanuit de linkerbovenhoek van het scherm; Y1 en Y2 het aantal puntjes verticaal (naar beneden). Schrijf een programma dat voor twee punten op het scherm de X- en Y-coördinaat inleest (met invoeraanwijzing) en dat tussen deze twee punten op het SCHERM (sorry **SCREEN 2**) een lijn trekt.

2.6 MEER PRINT MOGELIJKHEDEN EN PROGRAMMAVOORBEELDEN

Tot nu toe hebben we gezien dat elke nieuwe PRINT-opdracht op een nieuwe regel begint af te drukken. We kunnen de computer echter ook opdracht geven om na het afdrukken op dezelfde regel te blijven. We doen dit door de PRINT-opdracht af te sluiten met een **puntkomma** (;); een voorbeeld:

```
10 A=10
20 B=25
30 PRINT A;
40 PRINT B;
50 PRINT A*B
60 END
run
 10  25  250
```

De drie PRINT-opdrachten drukken alles op één regel af. Getallen worden afgedrukt met een spatie (positief getal) of een minteken (negatief getal) ervoor en altijd met een spatie erachter.

We kunnen de drie PRINT-opdrachten uit bovenstaand programma ook als één PRINT-opdracht schrijven:

```
10 A=10
20 B=25
30 PRINT A; B; -A*B
40 END
run
 10  25 -250
```

Als we in plaats van puntkomma's alleen **komma's** gebruiken, gebeurt het volgende (standaardscherm met 37 tekens per regel):

```
10 A$="hier"
20 B$="daar"
30 PRINT A$, B$, A$, B$
40 END

run
hier           daar
hier           daar
```

Door de komma's in de **PRINT**-opdracht worden de waarden (in dit voorbeelden tekstwaarden) op twee vaste regelposities gezet. Per regel is dit op positie 1 en op positie 15. De volgende waarde komt weer op positie 1 van de volgende regel, enzovoort. De **TAB**-toets geeft overigens 4 tabs op een regel.

Anders dan bij getallen worden er bij teksten geen spaties afgedrukt:

```
10 A$="rij"
20 B$="jij"
30 C$="of"
40 D$="ik"
50 PRINT A$; B$; C$; A$; D$
60 END

run
rijjijofrijik
```

We moeten de spaties tussen de woorden of in de woorden zelf opnemen of in de **PRINT**-opdracht. Laten we voor de laatste oplossing kiezen; we geven alleen de gewijzigde regel 50:

```
50 PRINT A$;" ";B$;" ";C$;" ";A$;
          " ";D$
60 END

run
rij jij of rij ik
```

opdracht 2.8

We kunnen ook in A$, B$ en C$ een spatie opnemen in plaats van de vier spaties (" ") in regel 50. Regel 10 wordt dan bijvoorbeeld 10 A$ = "RIJ ". Wat zou meer geheugenruimte kosten als we het programma in het geheugen zouden opnemen, de spaties in A$, B$ en C$ of in regel 50?

Nu volgt nog een aantal programmavoorbeelden:

programma 1 Intikken en draaien! Om regel 10 en 40 op één beeldschermregel te krijgen moet eerst **width 40** ◁ ingetikt worden

```
10 PLAY"CEGR64GAR64AG2FR64FER64EDR64DC2"
20 PLAY "GR64GFR64FER64EDR64D"
30 PLAY "GR64GFR64FER64EDR64D"
40 PLAY"CEGR64GAR64AG2FR64FER64EDR64DC2"
```

programma 2 Een voorbeeld met tekstbewerking

```
10 REM Dit programma drukt de middel-
       ste letter af van een woord
       met een oneven aantal letters
15 CLS
20 INPUT "Toets een woord in";WRD$
25 N%=LEN(WRD$)
30 M%=INT(N%/2)+1
35 LHELFT$=LEFT$(WRD$,M%)
40 HEBBES$=RIGHT$(LHELFT$,1)
45 PRINT
50 PRINT "De middelste letter uit "
55 PRINT WRD$;
60 PRINT " is een "; HEBBES$
65 END
```

Regel 10 beslaat drie regels op het beeldscherm. Een genummerde BASIC-regel mag, inclusief regelnummer, maximaal 255 tekens lang zijn. Denk erom dat na het laatste woord in de eerste regel van de opdracht 10 niet op de RETURN-toets gedrukt wordt. We doen dit pas na het woord 'letters' aan het einde van de derde regel. Door herhaald op de spatiebalk te drukken gaat de cursor vanzelf naar de volgende regel.
Laten we dit programma even toelichten:

Regel 10	:	Dit is een commentaarregel waarmee de computer niets doet; hij dient alleen voor diegenen die de programmatekst lezen.
Regel 15	:	Schoon scherm.
Regel 20	:	De computer vraagt aan ons om een woord in te toetsen, dat in de variabele WRD$ gestopt wordt.
Regel 25	:	N% wordt het aantal letters in het ingetoetste woord.
Regel 30	:	M% wordt de plaats van de middelste letter in het ingetoetste woord, dat een oneven aantal letters moet bevatten.
Regel 35	:	LHELFT$ krijgt als waarde de LinkerHELFT (inclusief de middelste letter) van het ingetoetste woord.
Regel 40	:	HEBBES$ krijgt als waarde het meest rechtse (het laatste dus) teken uit LHELFT$; dit is dus de middelste letter in het ingetoetste woord.
Regels 50, 55, 60	:	De middelste letter uit het ingetoetste woord wordt afgedrukt.

(De puzzelaars kunnen nu vast naar oefenopgave 9 kijken.)

opdracht 2.9

Zouden we in plaats van de naam HEBBES$ ook één van de volgende namen mogen gebruiken? MIDDEN$ of MIDDELSTE$ of MIDLETTER$?

programma 3 Toevalsgetallen trekken

```
100 REM Toevalsgetallen
110 CLS
120 PRINT "Dit programma trekt toe-"
130 PRINT "valsgetallen uit de gehe-"
140 PRINT "le getallen 0,1,2..t/m N"
150 PRINT
160 INPUT "Geef N een waarde"; N
170 PRINT
180 PRINT "Ik trok:";
          INT((N+1)*RND(1))
190 END
```

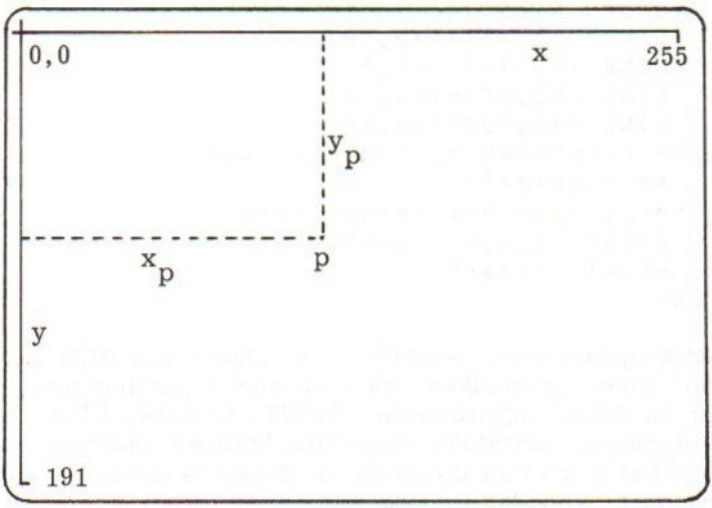
dit wordt vlak voor de oefenopgaven uitgelegd

Voor de verandering hebben we andere regelnummers gekozen. Het is verstandig de regelnummers niet opvolgend te kiezen, zoals 10, 11, 12, 13, ... Je kunt dan nooit meer regels tussenvoegen.

Als je een programma schrijft en je wilt het aan een ander laten zien dan moet de programmatekst duidelijk en goed gestruktureerd opgesteld worden anders kan het voor die ander weleens heel lastig worden om te begrijpen wat het programma doet. Dit geldt zeker voor programma's in een boek als dit; immers vele lezers moeten aan de hand van de programmatekst van de in dit boek opgenomen programma's kunnen nagaan wat de programma's doen. Met enige inspanning en passen en meten kunnen we op het beeldscherm een goed overzichtelijke programmatekst opstellen. Een beperking is de regelbreedte van 37 (40) tekens. Hierdoor zullen vaak meer regels nodig zijn. Dit is in principe niet erg als we de tekst dan maar netjes over deze regels verdelen. Vaak is het ook duidelijk als we de programmatekst ten opzichte van het regelnummer iets laten inspringen. Wij zullen aan de hand van het volgende programma laten zien wat een overzichtelijke programmatekst is en wat niet.

Het volgende programma vraagt om het intoetsen van de coördinaten van vier punten. Hierna vraagt het programma om het intikken van drie codes (0-15) voor respectievelijk de voorgrondkleur (afdrukkleur), de achtergrondkleur en de kaderkleur (boven- en onderrand). Daarna tekent het programma een vierhoek met als

hoekpunten de vier punten, waarvan de coördinaten ingetoetst moesten worden. Het grafische scherm (**SCREEN 2** in regel 210) bestaat uit 256 puntjes horizontaal (genummerd 0 t/m 255) en 192 puntjes verticaal (genummerd 0 t/m 191). De oorsprong van het grafische scherm ligt in de linkerbovenhoek. Horizontaal naar rechts



geven we de x-as aan. Verticaal naar beneden ligt de y-as. In dit coördinatenstelsel kunnen we een punt P aangeven met (x_p,y_p); x_p is de afstand van punt P tot de linkerrand (de y-as), terwijl y_p de afstand is van P tot de bovenrand (de x-as). We geven eerst het programma.

programma 4

```
100 REM voorbeeld van een grafisch
        programma. Dit programma te-
        kent een vierhoek, waarvan de
        coordinaten van de hoekpunten
        door de gebruiker moeten wor-
        den ingetikt.
110 CLS
120 'coordinaten invoeren
130   INPUT "XA,YA "; XA,YA
140   INPUT "XB,YB "; XB,YB
150   INPUT "XC,YC "; XC,YC
160   INPUT "XD,YD "; XD,YD
170 'grafisch scherm(0-255,0-192) en
     kleuren instellen
180   INPUT"geef de voorgrondkleur"
            ;VKLEUR
```

```
190   INPUT"geef de achtergrondkleur"
          ;AKLEUR
200   INPUT"geef de randkleur"
          ;RKLEUR
210   COLOR VKLEUR,AKLEUR,RKLEUR :
      SCREEN 2
220 'tussen A,B,C,D lijnen trekken
230   LINE (XA,YA)-(XB,YB)
240   LINE (XB,YB)-(XC,YC)
250   LINE (XC,YC)-(XD,YD)
260   LINE (XD,YD)-(XA,YA)
270 'de rechthoek rustig bekijken
280   A$=INPUT$(1)
290 'terug naar het tekstscherm
300   COLOR 15,4,4 : SCREEN 0
310   PRINT "klaar"
320 END
```

Door de REM-opdrachten, waarbij we in plaats van REM ook een ' (apostroph) mogen gebruiken, zien we direct wat het programma doet. Door de BASIC-opdrachten (**INPUT, COLOR, LINE**) één positie te laten inspringen, ontstaan duidelijke 'blokjes' opdrachten. De programmatekst is zo dus duidelijk en direct is duidelijk wat het programma doet. Voordat we nog wat nader uitleg geven over de opdrachten in het programma geven we eerst een programma dat hetzelfde doet maar waarvan de programmatekst niet erg overzichtelijk is, zeg maar 'nauwelijks te lezen!'.

```
100 CLS:INPUT"XA,YA";XA,YA:INPUT"XB,Y
B";XB,YB:INPUT"XC,YC";XC,YC:INPUT"XD,
YD";XD,YD
110 INPUT"geef de voorgrondkleur"; VK
LEUR:INPUT"geef de achtergrondkleur";
AKLEUR:INPUT"geef de randkleur"; RKLE
UR
120 COLOR VKLEUR,AKLEUR,RKLEUR:SCREEN
 2
130 LINE (XA,YA)-(XB,YB):LINE (XB,YB)
-(XC,YC):LINE (XC,YC)-(XD,YD):LINE (X
D,YD)-(XA,YA)
140 A$=INPUT$(1):COLOR 15,4,4:SCREEN
0:PRINT"klaar":END
```

Dit programma doet precies hetzelfde als het vorige programma. We hebben echter alle commentaarregels eruit verwijderd; we hebben niet ingesprongen en we hebben veel opdrachten achter één regelnummer gezet (in MSX BASIC mogen we tot 255 tekens, inclusief het regelnummer gaan). Wat nog het ergste is, we zien niet eens meer wat nu precies de regelnummers zijn. Dit is natuurlijk wel een erg overdreven voorbeeld, maar het geeft wel een idee hoe onleesbaar

je een programma kunt maken. Dit programma neemt natuurlijk minder geheugen in beslag, maar is even snel als het vorige.

Laten we nog eens teruggaan naar het 'nette' programma. We zien dat in een INPUT-opdracht (regels 130 t/m 160) ook twee variabelen tegelijkertijd ingelezen kunnen worden. Toets dan ook steeds twee getallen, gescheiden door een komma, in. Geef ook altijd eerst de COLOR-opdracht en dan pas de SCREEN-opdracht. Een COLOR-opdracht na de SCREEN-opdracht kan soms niet de gewenste uitwerking hebben. De opdracht A$ = INPUT$(1) in regel 280 heeft tot gevolg dat het programma daar wacht tot wij 1 toets (geeft niet welke) hebben ingedrukt. Pas daarna gaat het programma verder met regel 300, waarin naar het 'gewone' tekstscherm (24 regels met maximaal 40 posities) wordt teruggekeerd. Terwijl de computer bij regel 280 wacht kunnen wij rustig naar de getekende vierhoek kijken.

Vervang in dit programma de regels 230 t/m 260 door één regel

```
230  LINE (XA,YA)-STEP (XB,YB)-
     STEP (XC,YC)-STEP (XD,YD)-STEP (XA,YA)
```

en kijk wat er gebeurt.

Vervang de regels 230 t/m 260 door deze regel:

```
230  LINE (XA,YA)-(XD,YD),VKLEUR,B
```

en kijk hoe door de toevoeging ',B' de computer nu een rechthoek (Box) tekent met punt A als linkerbovenhoek en punt D als rechterbenedenhoek. Kijk wat er gebeurt als u achter de B nog een F zet.

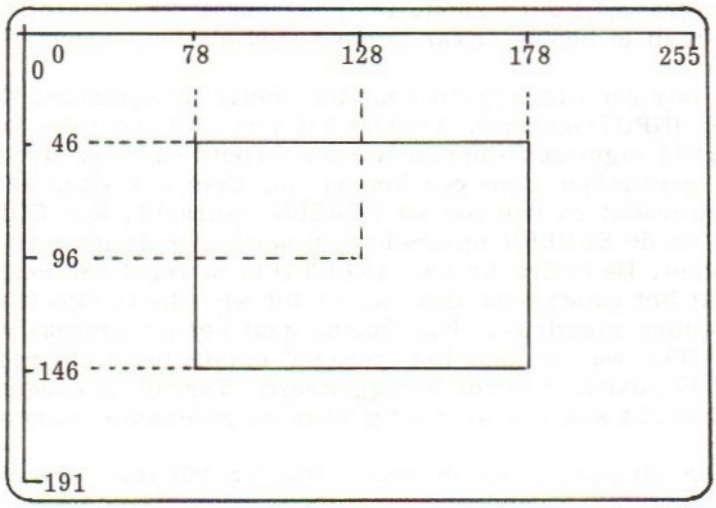
Neem even aan dat u inderdaad de regels 230 t/m 260 hebt vervangen door de volgende regel:

```
230  LINE (XA,YA)-(XD,YD),VKLEUR,B
```

We willen nu een vierkant tekenen in het midden van het scherm, we draaien het programma en tikken voor XA en YA en XD en YD respectievelijk 78,46 en 178,146 in (zie figuur volgende bladzijde). Hierdoor zou je verwachten midden in het scherm een vierkant met zijden van 100 (beeldpunten) te zien. Het is echter geen vierkant maar een rechthoek! De computer gebruikt niet dezelfde schaal voor de horizontale en verticale richting. Het blijkt dat de horizontale schaal 1,4 maal zo groot is als de verticale. Laten we dus alle horizontale coördinaten (de X-coördinaten dus) door 1.4 delen:

```
230  LINE (XA/1.4,YA)-(XD/1.4,YD),VKLEUR,B
```

En zie, nu is het een vierkant. Maar het staat nu niet meer in het midden van het scherm! Dit komt omdat de X-coördinaat van het



midden ook door 1,4 gedeeld is. De X-coördinaat van het nieuwe midden is dus nu 128/1,4 = 91,43 (afgerond op twee decimalen). Het is een beetje raar om de X-coördinaat van een punt op het scherm met een gebroken getal aan te geven; dit is immers altijd een geheel getal tussen 0 en 255! Laten we daarom als nieuwe X-coördinaat voor het midden nemen: INT(128/1.4 + .5) hetgeen 91 oplevert. De .5 zorgt voor het eventueel naar boven afronden van de nieuwe X-coördinaat. Het nieuwe midden ligt dus nu bij X = 91 en niet meer bij X = 128. Als we het nieuwe midden toch precies in het midden van het beeldscherm willen houden dan moeten we bij het nieuwe midden dus 128 - 91 = 37 optellen. Als we dit allemaal toepassen in regel 230 krijgen we

```
230  LINE (37+INT(XA/1.4+.5),YA)-
          (37+INT(XD/1.4+.5),YD),VKLEUR,B
```

Als we dus horizontaal en verticaal dezelfde schaal willen hebben, moeten we alle X-coördinaten in LINE-opdrachten altijd op de bovenstaande manier aanpassen.

opdracht 2.10

We kunnen het vierkant dat door de hierboven gegeven regel 230 wordt getekent op twee manieren inkleuren. De eerste manier is door iets aan de LINE-opdracht in regel 230 toe te voegen. De andere manier is om een nieuwe opdracht te gebruiken. Kijk in uw gebruikershandleiding of u die andere manier kunt vinden en geef aan hoe dat dan moet. Neem aan dat voor (XA,YA) respectievelijk voor (XD,YD) ingetikt wordt 78,46 en 178,146.

Het is duidelijk dat in een programma met erg veel LINE-opdrachten het erg onhandig is om steeds weer in plaats van de X-coördinaat, die we even gewoon XP zullen noemen, in te moeten tikken: 37 + INT(XP/1.4+.5). Het zou veel gemakkelijker zijn om een 'afkorting' voor deze berekening te kunnen gebruiken; tenslotte gebruiken we ook namen als SQR, INT, RND en LEN voor opdrachten, waarachter ook wel enig rekenwerk schuil gaat. De functies SQR, INT, RND en LEN zijn echter al in de computer aanwezig; we kunnen ze direct gebruiken. In BASIC kunnen we ook zelf zulke functies maken, alhoewel we ze jammer genoeg niet zoals SQR, INT, RND en LEN, permanent in de computer kunnen opslaan. We kunnen functies maken in een programma. Alleen in het programma, waarin we die functies maken, kunnen we ze gebruiken. De namen van zelfgemaakte functies moeten altijd met de twee letters FN beginnen.

We gaan nu zelf een functie maken die voor een gegeven X-coördinaat XP voor ons uitrekent wat 37 + INT(XP/1.4+.5) is. De naam van zo'n functie moet met FN beginnen. Daarachter kunnen we zelf letters toevoegen. Laten we het zelf ook bij een drieletterige naam houden, bijvoorbeeld

```
FNX
```

We weten dat de INT-, SQR-, RND- en LEN-functie allemaal een zogenaamd argument hebben; dit is de waarde of de rekenkundige uitdrukking die we tussen haakjes achter de functienaam opgeven. We hebben voorbeelden gezien als

```
INT(N%/2), SQR(8), RND(1) en LEN(WRD$)
```

Onze functie FNX zal ook een argument moeten hebben. Dit is namelijk de X-coördinaat waarvoor de berekening 37 + INT(X-coördinaat/1.4+.5) moet worden uitgerekend.

We zouden het zo kunnen opschrijven:

```
FNX(XP) = 37 + INT(XP/1.4+.5)
```

Hoe weet de computer nu welke zelfgemaakte functies we willen gebruiken? Welnu, dat moeten wij hem vertellen door aan het begin van een programma de functie(s) te DEFiniëren. We doen dat met de DEF-opdracht.

In het voorgaande programma (zie p. 47-48) nemen we als regel 175 op:

```
175  DEF FNX(XP) = 37+INT(XP/1.4+.5)
```

De computer weet vanaf dat moment dat, als hij de naam FNX met tussen haakjes een bepaalde waarde of rekenkundige uitdrukking

tegenkomt, hij voor die opgegeven waarde het rekenvoorschrift (achter het = teken) in regel 175 moet uitvoeren. De regels 230 t/m 260 uit het programma wijzigen we vervolgens in:

```
230  LINE (FNX(XA),YA)-(FNX(XB),YB)
240  LINE (FNX(XB),YB)-(FNX(XC),YC)
250  LINE (FNX(XC),YC)-(FNX(XD),YD)
260  LINE (FNX(XD),YD)-(FNX(XA),YA)
```

Als we het programma nu draaien zullen de coördinaten van een vierkant (bijvoorbeeld (0,0), (0,100), (100,100) en (100,0) die we intikken ook tot gevolg hebben dat er een echt vierkant getekend wordt en niet een rechthoek die in horizontale richting breder is dan in verticale richting.

We zullen nog wel meer eigengemaakte functies tegenkomen. Belangrijk is dat u nooit een zelfgemaakte functie gebruikt vóór u hem met een DEF FN ... opdracht hebt geDEFinieerd. In zo'n DEF FN ... opdracht mogen we voor het argument 'tussen haakjes' elke variabele gebruiken die we maar willen. In ons voorbeeld is het XP, maar het zou net zo goed A, ARG, JAN of zelfs XA of XB kunnen zijn. De computer houdt de programmavariabelen en de variabelen die als argument in zelfgemaakte functies voorkomen strikt gescheiden ook al hebben ze dezelfde naam!

opdracht 2.11

Maak een zelfgemaakte functie FND die het aantal ogen nabootst dat je kan krijgen als je een keer met een dobbelsteen gooit. Deze FND is een functie zonder argument!

2.7 FOUTEN EN FOUTMELDINGEN

Tot nu toe is er nog weinig gezegd over de fouten die bij het invoeren van programma's en gegevens gemaakt kunnen worden en ook niet over fouten die bij het draaien van een programma kunnen optreden.

Zo kunnen we bijvoorbeeld bij het intoetsen van de programmatekst gewoon tikfouten maken. Stel dat in programma 1 uit de vorige paragraaf regel 30 als volgt is ingetoetst:

```
30 PLAU "GR64GFR64FER64EDR64D"
```

In plaats van PLAY is PLAU ingetoetst (de U zit naast de Y op het toetsenbord). Als we dit programma zo zouden draaien, zien we dit:

```
Syntax error in 30
Ok
```

In regel 30 heeft de computer een taalfout ontdekt en drukt dit ook af. Het uitvoeren van het programma wordt afgebroken.

We kunnen deze fout gemakkelijk herstellen. We tikken in LIST 30 en gaan vervolgens met de cursortoetsen naar de U in regel 30. Als de cursor bovenop de U staat tikken we een Y en geven RETURN. Regel 30 staat vanaf dat moment goed in het geheugen en als we RUN ◁ intikken wordt 'altijd is Kortjakje ziek' gespeeld.

Soms krijgen we de gelegenheid om tijdens de uitvoering van een programma een fout te herstellen. Stel dat het programma 3 uit 2.6 geen taalfouten bevat en stel dat we het programma draaien. We zien dan op het beeldscherm:

```
Dit programma trekt toe-
valsgetallen uit de gehe-
le getallen 0,1,2..t/m N

Geef N een waarde?
```

We moeten nu voor N een geheel getal intoetsen. Stel dat we (dit zal niet zo snel gebeuren) voor N geen getal maar een letter intoetsen, dan gebeurt dit:

```
Geef N een waarde? ja
?Redo from start
Geef N een waarde?
```

De computer vraagt nu om nieuwe invoer. Het programma wordt dus in dit geval niet afgebroken; we kunnen door nu wel een geheel getal in te tikken de programma-uitvoering laten hervatten.

Een fout met 'fatale' gevolgen (als je het afbreken van een programma fataal mag noemen) kan optreden in programma 4 uit de vorige paragraaf. In dat programma wordt een vierhoek getekend nadat we de coördinaten van de vier hoekpunten en de kleuren voor de voorgrond, achtergrond en de rand hebben ingetoetst. De toegestane kleurcodes zijn de gehele getallen 0,1,2,... t/m 15. Toetsen we echter voor de randkleur bijvoorbeeld per ongeluk 16 in, dan gebeurt dit:

```
XA,YA ? 5,5
XB,YB ? 5,150
XC,YC ? 150,150
XD,YD ? 150,5
geef de voorgrondkleur? 1
geef de achtergrondkleur? 4
geef de randkleur? 16
Illegal function call in 210
Ok
```

Het programma wordt afgebroken in regel 210. Dit is de opdracht:

```
210 COLOR VKLEUR,AKLEUR,RKLEUR: SCREEN 2
```

De computer struikelt blijkbaar over de waarde 16 die RKLEUR door ons heeft gekregen. Dat deze fout kan optreden komt omdat dit programma verre van volmaakt is. Een 'goed' programma zal nagaan of de door ons ingetikte waarden niet tot dergelijke foutmeldingen aanleiding zouden kunnen geven. Hoe we dergelijke tests in kunnen bouwen leren we in hoofdstuk 4.

De bovengenoemde fouten zijn fouten die vrij snel aan het licht komen en die ook vrij snel te verhelpen zijn. Er is echter een andere categorie fouten die veel erger is. We zouden dit 'denkfouten' van de programmeur kunnen noemen. Een programma dat dit soort fouten vertoont doet iets anders dan de schrijver denkt dat het programma zal doen. Dergelijke fouten komen slechts aan het licht door het programma terdege te testen met alle mogelijke invoerwaarden en in alle mogelijk op te treden situaties. We geven één voorbeeld van een dergelijke denkfout. Stel we werken met het programma 3 uit de vorige paragraaf. Als we dit programma draaien, zien we dit:

```
Dit programma trekt toe-
valsgetallen uit de gehe-
le getallen 0,1,2..t/m N

Geef N een waarde? 100

Ik trok: 60
Ok
```

Zo op het oog is er niets aan de hand. Het toevalsgetal 60 is inderdaad een geheel getal uit de reeks 0,1,2,3,...,100. Stel nu dat de programmamaker een denkfout heeft gemaakt door regel 180 als volgt te programmeren:

```
180 PRINT "Ik trok:";
              INT(N*RND(1))
```

hier zit de denkfout

Als nu voor N 100 ingetoetst wordt dan heeft N*RND(1) een waarde tussen 0 en 100, want N = 100 en RND(1) ligt tussen 0 en 1 (nooit 0 of 1!). Als we van een getal tussen 0 en 100 de INT functie nemen, krijgen we een geheel getal uit de reeks 0,1,2,3,..., 99 en dus niet een getal uit de reeks 0,1,2,3,...,100. De waarde 100 zal dus nooit als toevalsgetal getrokken worden (vandaar dat we ook (N+1)*RND(1) moeten nemen).

Deze denkfout is alleen op te sporen indien we het programma vele keren draaien (in gemiddeld één op de 101 keer zal het getal 100 worden getrokken) en daarbij constateren dat het getal 100 nooit wordt getrokken. In dit geval toch een lastige zaak, want de getallen zijn toevallig; maar ja ..., toevallig goed of toevallig fout?

Door **netjes te programmeren** kan een groot aantal van dergelijke denkfouten (ook wel **logische fouten** genoemd) in programma's worden voorkomen. Wat 'netjes' programmeren is hopen we in de komende hoofdstukken te ontdekken. In Appendix G is een lijst met foutmeldingen, die de computer kan geven, opgenomen.

OEFENOPGAVEN HOOFDSTUK 2

1. Welke van de volgende namen zouden namen kunnen zijn van variabelen in een BASIC-programma? Als het geen goede naam is, geef dan aan waarom niet. Als het wel kan, geef dan het soort variabele aan.

a. RENTE	b. A11!	c. A-2	d. WOORD1$
e. VIJF%	f. L2$	g. SOM	h. PI#
i. 2A	j. Z3C$	k. AANTAL!	l. STOP

2. a. Schrijf een programma dat een willekeurig getal inleest en dit getal, op helen afgerond, op het scherm afdrukt. Zorg voor een passende tekst bij zowel de invoer als de uitvoer.

   b. Definieer in dit programma een functie FNG (van FuNctie-Geheel) die een gegeven argument op helen afrondt en gebruik deze functie bij het afdrukken van het afgeronde getal.

3. Wat is de uitvoer van het volgende programma? Geef aan hoeveel spaties tussen de getallen worden afgedrukt.

```
10 A=10
20 B=4
30 PRINT A+B;-A/B;INT(A/B+.5)
40 PRINT A^B,(A+2*B)/2+A
50 END
```

4. a. Schrijf een programma dat een getal (tussen 0 en 95) inleest en dat een vierkant tekent met als middelpunt het midden van het hoge resolutiescherm (**SCREEN 2** met coördinaten 128,91 voor met midden) en met als lengte van elk van de vier zijden het zojuist ingelezen (door ons ingetikte) getal. Zorg voor een gele achtergrond, zwarte zijden en paarse randen.

   b. Hoe kunt u het hele vierkant zwart maken?

   c. En hoe kan het vierkant de kleur van de boven- en onderrand van het scherm krijgen?

   d. Kunt u ook een cirkel tekenen met als straal het ingetoetste getal en als middelpunt het midden van het beeldscherm?

5. a. Schrijf een programma dat een RaNDom getal afdrukt dat geheel is (afgerond) en ligt tussen 10 en 20 (10 en 20 mogen ook voorkomen).

   b. Schrijf een programma dat vraagt om het intoetsen van twee getallen A en B, waarbij B groter is dan A en dat een geheel toevalsgetal afdrukt dat tussen A en B ligt (A en B zelf mogen ook meedoen!).

6. De omtrek van een fietswiel is $\pi.d$; d is de diameter van het fietswiel en π is het getal pi; $\pi = 3{,}141592654$ (en nog meer decimalen). Bij één omwenteling van het fietswiel legt de fiets dus een afstand van $\pi.d$ (de omtrek van het wiel) af. Schrijf een programma dat vraagt om het intoetsen van de diameter van het fietswiel in inches (26 inch is een veel voorkomende maat); 1 inch = 2,54 cm. Vervolgens drukt het programma de afgelegde afstand in meters af bij een omwenteling van het wiel. In- en uitvoer met passende tekst!

7. Schrijf een programma dat als het draait het volgende tafereel oplevert:

```
run
Uw telefoonnummer is? 070-870954
Dan is uw netnummer 070
En uw abonneenummer 870954
Ok
```

(netnummer is altijd drie cijfers, abonneenummer altijd zes cijfers).

8. Als je bij het getal 3,468 vijfduizendsten optelt, krijg je: 3,468 + 0,005 = 3,473. Als je deze uitkomst met honderd vermenigvuldigt, krijg je: 3,473 × 100 = 347,3. Als je van deze uitkomst het decimale gedeelte weghaalt, krijg je: 347. Deel je dit nu door honderd (347/100 = 3,47), dan heb je het oorspronkelijke getal 3,468 afgerond op twee cijfers achter de komma (op honderdsten dus). Pas dit principe in het hierna te schrijven programma toe.

   Als we een geldbedrag B tegen I% rente per jaar voor N jaar vastzetten, is dit bedrag na N jaar aangegroeid tot een eindkapitaal K dat gelijk is aan $B \times (1 + I/100)^N$.

   Schrijf een programma dat B, I en N inleest en dat het eindkapitaal K, afgerond tot op centen nauwkeurig afdrukt. Het beginkapitaal B is een bedrag in guldens.
   In- en uitvoer moeten worden voorzien van een passende tekst.

9. Dit is een lastige opgave!
   In programma 2 uit § 2.6 op pagina 45 worden de variabelen N%, M%, LHELFT$ en HEBBES$ gebruikt voor het afdrukken van de middelste letter uit een ingetoetst woord. We kunnen echter de regels 40 t/m 80 uit dit programma vervangen door één PRINT-opdracht waarmee de middelste letter wordt afgedrukt. De bovengenoemde variabelen zijn hierbij niet nodig. Geef die ene PRINT-opdracht.
   Hint: in BASIC mag je een functie van een functie van een .... nemen.

3 Herhalingsstructuur in een Programma

3.1 DE LUSSTRUCTUUR

In de programma's die we tot nu toe gezien hebben worden alle opdrachten (programmaregels) netjes na elkaar uitgevoerd; vanaf het laagste regelnummer tot en met het hoogste regelnummer. We noemen dit een **sequentiële programmastructuur**. Er zijn echter nog meer structuren in een programma mogelijk. In dit hoofdstuk bespreken we de **herhalingsstructuur**; ook wel lusstructuur genoemd. Een (programma)lus is een manier (structuur) om binnen een programma eenzelfde groep opdrachten meerdere malen uit te laten voeren. We gaan drie soorten programmalussen bespreken, te weten:

de **DOE-ZOVEEL-KEER** lus
de **DOE-ZOLANG** lus èn
de **HERHAAL-TOTDAT** lus

Bij de DOE-ZOVEEL-KEER lus staat van tevoren vast hoeveel maal een bepaalde groep opdrachten (de lus) moet worden uitgevoerd. Bij de DOE-ZOLANG en de HERHAAL-TOTDAT lussen is het uitvoeren van de lusopdrachten afhankelijk van een bepaalde voorwaarde. Er moet getest worden of aan de voorwaarde voldaan is. Wordt er niet meer aan de voorwaarde voldaan dan worden de lusopdrachten niet meer uitgevoerd. Het verschil tussen de DOE-ZOLANG en de HERHAAL-TOTDAT lussen is het moment waarop deze voorwaarde getest wordt. Bij een DOE-ZOLANG lus is dit voor het uitvoeren van de lusopdrachten, bij een HERHAAL-TOTDAT lus wordt na het uitvoeren van de lusopdrachten getest of er nog aan de voorwaarde voldaan is.

We zullen de drie soorten lussen met een voorbeeld illustreren. We gaan een programma maken dat meerdere keren om het intoetsen van een getal vraagt en dat na het intoetsen van zo'n getal het getal zelf, het kwadraat van het getal en de derde macht van het getal op het beeldscherm afdrukt.

I de DOE-ZOVEEL-KEER lus

Stel dat we het programma zo willen maken dat we precies 10 getallen kunnen intoetsen. Van tevoren is dan dus bekend (10 keer) hoeveel maal het programma ons om een getal moet vragen en hoeveel maal de eerste, tweede en derde macht van dat getal afgedrukt moet worden. We zouden de lusstructuur als volgt kunnen formuleren:

```
doe 10 keer:
        vraag om een getal
        druk 1e, 2e en 3e macht van het getal af
```

Hier moeten we een constructie in BASIC van maken. De twee lusopdrachten zijn niet moeilijk te programmeren:

```
doe 10 keer:
        INPUT "Toets een getal in"; G
        PRINT G; G∧2; G∧3
```

Nu nog een BASIC-opdracht voor 'doe 10 keer:'. In BASIC gaat dit als volgt:

```
FOR I = 1 TO 10
   INPUT "Toets een getal in"; G
   PRINT G; G∧2; G∧3
NEXT I
```

In plaats van een DOE-ZOVEEL-KEER lus spreken we van een FOR-NEXT lus. I is de lusteller of lusvariabele, die bij 1 begint en die na het uitvoeren van de lusopdrachten steeds met 1 wordt opgehoogd. Als I de waarde 11 gekregen heeft wordt de lus beëindigd. De opdrachten die tussen de FOR- en NEXT-opdrachten in staan vormen de lusbody.

Als we voor de bovenstaande vier opdrachten regelnummers zetten en als we vervolgens deze vier regels als een programma intoetsen en draaien dan zal de computer ons precies 10 keer vragen om een getal in te toetsen en zal hij precies 10 keer de 1e, 2e en 3e macht van elk getal afdrukken.

Om de leesbaarheid te vergroten laten we de lusopdrachten inspringen ten opzichte van de FOR- en NEXT-opdrachten. Zo is direct duidelijk welke opdrachten de eigenlijke lus vormen. Het is verboden om de lusteller I in de lus zelf een andere waarde te geven; wel mogen we de waarde van I in de lus gebruiken. We zullen hier in § 3.2 nog genoeg voorbeelden van zien.

II de DOE-ZOLANG lus

Bij deze lusconstructie staat niet van tevoren vast hoeveel maal de lusopdrachten uitgevoerd moeten worden. Dit hangt af van een voorwaarde die, voordat de lus wordt uitgevoerd, getest wordt. Deze voorwaarde wordt door de programmeur (wij dus) zelf bedacht en geprogrammeerd. Stel dat wij de computer alleen de 1e, 2e en 3e macht van het ingetoetste getal willen laten afdrukken zolang dat ingetoetste getal niet nul is. Zolang wij niet het getal nul intoetsen op de vraag "Toets een getal in?" zullen de drie machten van het getal afgedrukt worden. Door het intoetsen van een 0 zorgen wij er zelf voor dat de lus beëindigd wordt; de computer weet van tevoren niet wanneer wij die 0 gaan intoetsen! We gaan de lusstructuur ontwerpen:


```
zolang er geen nul is ingetoetst:
        lusopdrachten
```

Hoe weet de computer of er wel of niet een nul is ingetoetst? Dat weet hij pas als hij ons eerst vraagt om iets in te toetsen, dus:

```
vraag om getal
zolang er geen nul is ingetoetst:
        lusopdrachten
```

Hoe moeten de lusopdrachten er uitzien? Wel, als er inderdaad geen nul is ingetoetst dan moeten de drie machten van het getal worden afgedrukt en moet het volgende getal gevraagd worden, dus:

```
vraag om getal
zolang er geen nul is ingetoetst:
        druk 1e, 2e en 3e macht van het getal af
        vraag om getal
```

Deze twee laatste opdrachten vormen de lus; zij worden immers herhaald uitgevoerd zolang we geen nul intoetsen. De eerste opdracht 'vraag om getal' hoort niet tot de lus: deze dient alleen om ervoor te zorgen dat er wat te testen valt. Laten we van deze DOE-ZOLANG lus bijna BASIC maken:

```
INPUT "Toets een getal in"; G
zolang er geen nul is ingetoetst:
        PRINT G; G∧2; G∧3
        INPUT "Toets een getal in"; G
```

Nu moeten we de tweede opdracht nog in BASIC schrijven. In MSX BASIC moeten we dit oplossen met een IF-opdracht en een GOTO-opdracht. Laten we daartoe eerst regelnummers voor de opdrachten zetten:

```
10 INPUT "Toets een getal in"; G
20 zolang er geen nul is ingetoetst:
30         PRINT G; G∧2; G∧3
40         INPUT "Toets een getal in"; G
60 END
```

De END-opdracht is met opzet toegevoegd want we zullen 'm nodig hebben. De lus speelt zich af in de regels 20, 30 en 40. Hoe programmeren we deze lus in BASIC? De oplossing is om in regel 20 een IF-opdracht te gebruiken en in regel 50 een GOTO-opdracht in te voegen; we krijgen dan:

```
10 INPUT "Toets een getal in"; G
20 IF G=0 THEN 60
30   PRINT G; G^2; G^3
40   INPUT "Toets een getal in"; G
50 GOTO 20
60 END
```

Regel 20 vertelt de computer: Als G gelijk is aan nul ga dan verder met regel 60; dat wil zeggen stop de uitvoering van de lus. Als G ongelijk is aan nul moet hij gewoon verder met regel 30; dat wil zeggen de lus wordt uitgevoerd. Als laatste lusopdracht staat in regel 50 de GOTO-opdracht. Hiermee wordt de computer gedwongen verder te gaan met het regelnummer dat achter de GOTO is opgegeven. In de IF-opdracht in regel 20 noemen we 'G = 0' de bewering of voorwaarde.

Nu is het zo dat we de IF-opdracht en de GOTO-opdracht niet alleen gebruiken voor het programmeren van lusstructuren. Aangezien het belangrijk is om in een programma de diverse structuren te kunnen herkennen zullen wij in dit boek met behulp van commentaar (REM-opdracht) de lusstructuur aangeven; dit gaat zo:

```
10 INPUT "Toets een getal in"; G
15 REM*** doe zolang G niet nul is
20 IF G=0 THEN 60
30   PRINT G; G^2; G^3
40   INPUT "Toets een getal in"; G
50 GOTO 20
55 REM***
60 END
```

Tussen twee REM-opdrachten met drie *** bevindt zich voortaan altijd een DOE-ZOLANG-constructie. (Het vakjargon spreekt over een DO-WHILE loop). De IF-THEN-opdracht heet een voorwaardelijke sprongopdracht. Dat wat tussen IF en THEN staat is de voorwaarde waaraan voldaan moet zijn opdat het programma verdergaat met de programmaregel, genoemd achter THEN. In plaats van 'voorwaarde' zullen wij dikwijls over een 'bewering' spreken. In plaats van 'aan de voorwaarde moet voldaan zijn' zeggen we dan 'de bewering moet waar zijn'.

IF 'bewering' THEN n :	Als de 'bewering' waar is ga dan verder met programmaregel n.

In tegenstelling tot de IF-THEN-opdracht heet de GOTO-opdracht een onvoorwaardelijke sprongopdracht.

GOTO n :	Ga verder met programmaregel n.

Veelvuldig en onoordeelkundig gebruik van sprongopdrachten kan leiden tot een slechte programmastructuur; zelfs tot onleesbare programma's. Wij zullen de IF-THEN- en de GOTO-opdrachten in dit boek altijd in vaste patronen gebruiken. Zo zullen wij een DOE-ZOLANG lus altijd programmeren met één voorwaardelijke sprongopdracht (IF-THEN aan het begin van de lus) en met één onvoorwaardelijke sprongopdracht (GOTO aan het einde van de lus). **Nooit zullen wij van binnenuit de lus met een sprongopdracht naar een regelnummer buiten de lus springen!** In § 3.7 komt de DOE-ZOLANG lus uitvoerig ter sprake.

opdracht 3.1

Wat moeten we, als het onderstaande programma draait, intoetsen om het programma te laten beëindigen?

```
10 INPUT "Toets een string in"; S$
15 REM***
20 IF S$="h" THEN 60
30   PRINT S$; " bevat"; LEN(S$);
                "letters"
40   INPUT "Toets een string in"; S$
50 GOTO 20
55 REM***
60 END
```

III de HERHAAL-TOTDAT lus

Evenals bij een DOE-ZOLANG lus wordt er in een HERHAAL-TOTDAT lus getest of een bewering al dan niet waar is. Het verschil is echter dat bij een HERHAAL-TOTDAT lus deze test aan het **einde** van de lus plaatsvindt en niet, zoals bij een DOE-ZOLANG lus, aan het **begin** van de lus. Omdat de test pas aan het einde van de lus wordt uitgevoerd zullen de opdrachten in een HERHAAL-TOTDAT lus altijd tenminste één keer worden uitgevoerd. We kunnen het programma met het intoetsen en afdrukken van getaller ook met een HERHAAL-TOTDAT lus programmeren. We geven meteen het programma:

```
15 REM===
20    INPUT "Toets een getal in"; G
30    PRINT G; G^2; G^3
40 IF G<>0 THEN 20
45 REM===herhaal totdat G nul is
50 END
```

De HERHAAL-TOTDAT lus bevindt zich tussen de twee REM=== - opdrachten. De test of de bewering waar is staat in regel 40; de laatste lus-opdracht. De bewering die hier op 'waar zijn' getest wordt is de bewering 'G <> 0'. Dit is in BASIC de manier om de computer te laten testen of G ongelijk is aan nul. Is dit zo dan wordt de lus nog eens uitgevoerd (naar regel 20). Is G gelijk aan nul dan wordt gewoon de opdracht na de IF-opdracht uitgevoerd. Dit is een REM-opdracht die dus eigenlijk niet meetelt. Regel 50 wordt uitgevoerd hetgeen het einde van het programma betekent.

Evenals bij de DOE-ZOLANG lus hebben we zelf door middel van REM-opdrachten de lus geaccentueerd. We hebben dit gedaan met

regelnummers die op 5 eindigen. Diegenen die deze REM-opdrachten niet in hun programma's willen opnemen kunnen ze rustig weglaten; de IF- en GOTO-opdrachten verwijzen nooit naar deze REM-opdrachten! Worden ook de spaties weggelaten dan zal het volgende programma precies hetzelfde doen als het bovenstaande programma, alhoewel de tekst er iets anders uitziet:

```
20 INPUT "Toets een getal in"; G
30 PRINT G; G^2; G^3
40 IF G<>0 THEN 20
50 END
```

Naarmate programma's groter worden is het accentueren van de diverse lusstructuren (en andere structuren) aan te bevelen aangezien dit de leesbaarheid van de programma's vergroot.

De structuur van een HERHAAL-TOTDAT lus is eenvoudiger dan de structuur van een DOE-ZOLANG lus. Vaak kunnen beide structuren als alternatief voor elkaar worden gebruikt; de programmeur kiest dan zelf één van beide structuren. Soms kunnen we de HERHAAL-TOTDAT structuur niet gebruiken omdat er al bij de eerste keer dat de lus uitgevoerd wordt iets 'mis' kan gaan; bedenk daarom:

Een HERHAAL-TOTDAT lus wordt altijd tenminste één keer uitgevoerd

opdracht 3.2

Met het volgende programma kunt u kijken of er nog een betere afdrukkleur is dan wit.

```
15 REM===
20   CLS
30   COLOR 15
40   INPUT"Toets een kleurkode in";K
50   COLOR K
60   PRINT"Is dit een mooie kleur?"
70   A$=INPUT$(1)
```

```
80 IF A$<>" " GOTO 20
85 REM=== herhaal tot op de spatie-
         balk gedrukt wordt
90 COLOR 15 : END
```

a) Hoe kunt u dit programma beëindigen?

b) Als u het programma op de gewone manier beëindigt (zie a)), in welke kleur wordt dan de BASIC prompt Ok afgedrukt?

c) Is de herhaal-totdat structuur de juiste lusstructuur voor dit programma?

3.2 DE DOE-ZOVEEL-KEER LUS

Bekijk het volgende voorbeeldprogrammaatje:

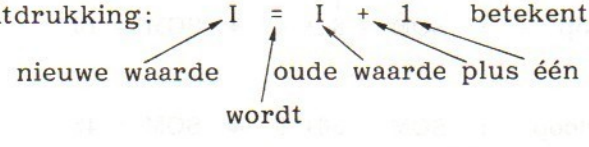
```
10 FOR I=1 TO 5
20     PRINT I;
30 NEXT I
40 PRINT "Na afloop is I"; I
50 END
```

Als we dit programma draaien is de uitvoer:

```
run
 1  2  3  4  5 Na afloop is I 6
Ok
```

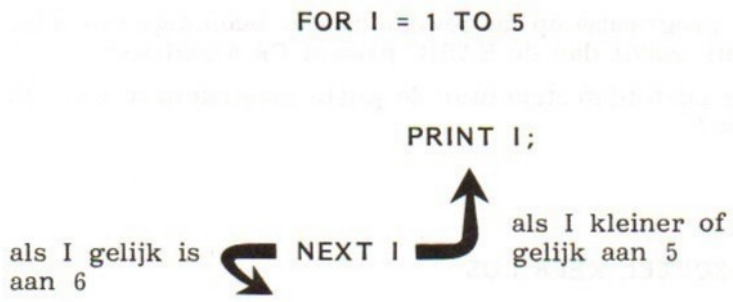
We zien dat bij elke doorloop van de lus de waarde van de lusteller wordt afgedrukt. Dit betekent dat bij de eerste doorloop de waarde van de variabele I gelijk is aan 1; bij de tweede doorloop gelijk is aan 2, enzovoorts. Het ophogen van de lusteller I gebeurt door de opdracht NEXT I. Deze opdracht verhoogt de waarde van I met 1, de stapgrootte van de FOR-NEXT lus, dus

NEXT I ⇒ I = I + 1



De computer verhoogt de inhoud van de geheugenla met het etiket I met 1. De naam van de variabele verandert niet, alleen de waarde ervan! We zien dat na beëindigen van de lus I de waarde 6 heeft.

Na het verhogen van de lusteller controleert de computer of de waarde van de lusteller groter is dan de in de FOR-opdracht opgegeven bovengrens. Is dit zo, dan gaat het programma verder met de opdracht volgend op de NEXT-opdracht, anders wordt de lus nog eens doorlopen; we zien dit in onderstaande tekening.

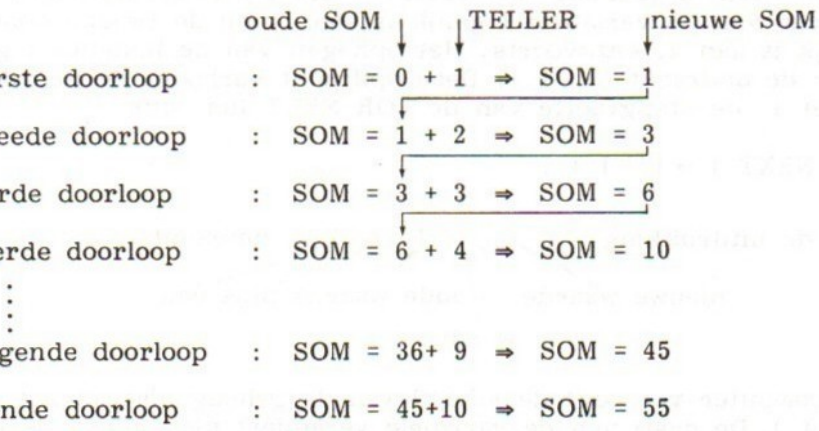


Waarvoor kunnen we de FOR-NEXT lus zoal gebruiken? We zullen een aantal toepassingen laten zien.

We kunnen een FOR-NEXT lus gebruiken voor het berekenen van de som van de gehele getallen 1 tot en met 10.

```
10 REM cumulatieve som bepalen
20 SOM=0 'dit is eigenlijk overbodig
30 FOR TELLER=1 TO 10
40    SOM=SOM + TELLER
50 NEXT TELLER
60 PRINT "1+2+3+4+5+....+10 is"; SOM
70 END
```

We beginnen met SOM = 0. Bij elke doorloop van de lus wordt bij de waarde van SOM de waarde van TELLER opgeteld, dus:



na afloop van de lus vinden we SOM = 55.

opdracht 3.3

Zou de waarde van SOM na afloop van de FOR-NEXT lus ook 55 zijn geweest indien we regel 20 (SOM = 0) hadden weggelaten? In het bovenstaande programma staat immers dat regel 20 overbodig is!

We kunnen van dit programma ook een interactieve versie maken:

```
10 REM Dit programma berekent de som
       van de gehele getallen 1,2,3,
       4,...t/m N,het gehele getal
       dat wij intikken
20 CLS
30 INPUT "Tik het getal N in"; N
40 SOM=0 'om te laten zien dat we met
         niets beginnen
50 FOR I=1 TO N
60    SOM=SOM + I
70 NEXT I
80 PRINT "De som van 1 t/m";N;
         "is"; SOM
90 END
```

opdracht 3.4

Wijzig dit programma zo dat het programma in plaats van de som van de getallen het rekenkundig gemiddelde van de N getallen afdrukt; met een passende tekst!

Nog een voorbeeld van het gebruik van een FOR-NEXT lus. We nemen als uitgangspunt het 'kortjakje'-deuntje uit hoofdstuk 2.

```
100 REM altijd is kortjakje ziek in
        acht octaven
110 A$="CEGR64GAR64AG2FR64FE" +
       "R64EDR64DC2"
120 B$="GR64GFR64FER64EDR64D"
130 FOR OCTAAF=1 TO 8
140    PLAY "O"+STR$(OCTAAF)+A$
150    PLAY B$+B$+A$
160 NEXT OCTAAF
170 END
```

De variabelen A$ en B$ (regels 110 en 120) bevatten de noten voor de eerste twee regels van kortjakje. De lus in de regels 130 t/m 160 speelt steeds de eerste vier regels van het versje in een bepaalde octaaf. We krijgen acht octaven te horen (dat kan op een

piano ook!). In regel 140 zien we dat van de lusvariabele OCTAAF, die de gehele getallen 1 t/m 8 doorloopt, eerst een string wordt gemaakt met behulp van de STR$-functie. Vervolgens wordt hiervoor de letter O (van Octaaf) gezet en worden de noten van de eerste regel van het versje erachter geplakt. Bij de eerste doorloop van de lus, OCTAAF = 1, staat er dus eigenlijk in regel 140:

```
PLAY "O1CEGR64GAR64.....R64DCZ"
```

Bij de tweede doorloop:

```
PLAY "O2CEGR64GAR64.....R64DCZ" , enzovoort.
```

Kijk in uw gebruikershandleiding als u meer wilt weten over de PLAY-opdracht. MSX BASIC beschikt over een eigen muziek(sub)-taal, de muziekmacrotaal. Hiermee kunt u het tempo, het volume en de klankkleur op een fraaie manier regelen.

Het is niet noodzakelijk de lusteller met 1 te laten beginnen. We mogen elke begin- en eindwaarde kiezen die we willen. We kunnen ook variabelen als begin- en eindwaarden kiezen en zelfs rekenkundige uitdrukkingen.

```
FOR I = 0 TO 5
FOR TELLER = 15 TO 45
FOR A = -5 TO N
FOR I = BEGIN TO EIND
FOR K = 1 TO 4*N+1
```

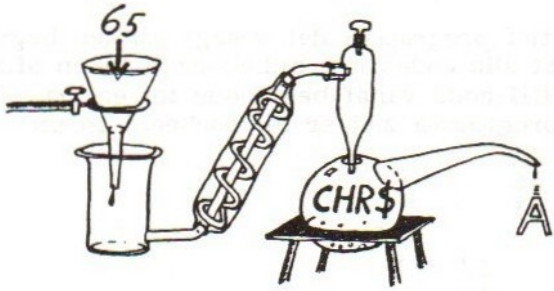
Laten we als beginwaarde van de lusteller 65 nemen en als eindwaarde 90. Dus

```
FOR I = 65 TO 90
```

Kijk nu eens naar de ASCII tabel in Appendix D. Als de computer tekens opslaat, transporteert, inleest of afdrukt, gebruikt hij de ASCII code van deze tekens en niet de tekens zelf! In de tabel ziet u bijvoorbeeld dat de hoofdletter A de code 65 heeft en de hoofdletter Z de code 90. Laten we bovenstaande FOR-opdracht nu eens zo schrijven:

```
FOR CODE = 65 TO 90
```

We gaan nu een lus maken waarbij de lusvariabele, in dit geval de variabele CODE, de ASCII codes van de letters A,B,C,D,... tot en met Z doorloopt. Als we nu met behulp van de ASCII code het daarbij behorende teken kunnen laten afdrukken, kunnen we alle letters van het alfabet met een FOR-NEXT lus laten afdrukken. Wel, we weten dat zo'n functie bestaat in BASIC: het is de CHR$-functie.



Het argument (dit kan een getal, een variabele of een uitdrukking zijn) van de CHR$-functie moet een waarde hebben tussen 0 en 255. De waarde (de uitkomst) van de CHR$-functie is een string met één teken, corresponderend met de ASCII code die door de waarde van het argument wordt bepaald. Laten we het alfabet eens afdrukken:

```
10 FOR I=65 TO 90
20     PRINT CHR$(I);
30 NEXT I
40 END
```

Als we dit programma draaien dan zien we:

```
run
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ok
```

Een paar bladzijden terug hebben we gezien hoe we cumulatief de SOM van de gehele getallen 1 t/m N kunnen bepalen. We kunnen ook de som van de letters a t/m z bepalen, al klinkt het gek:

```
10 REM Het alfabet,cumulatief
20 'we beginnen met een leeg alfabet
30 FOR CODE=97 TO 122
40     L$=CHR$(CODE) 'pak de volgende
                      letter
50     ALABET$=ALFABET$ + L$
       'voeg deze toe aan het alfabet
60 NEXT CODE
70 PRINT "Het alfabet:",,ALFABET$
80 END
```

Als we beginnen (regel 20) is het ALFABET leeg (default-string-waarde). In regel 40 bepalen we aan de hand van de ASCII-code de bij die code behorende letter. In regel 50 'vullen' we het ALFABET door steeds de volgende letter uit het alfabet aan ALFABET$ toe te voegen (concatenatie).

opdracht 3.5

Schrijf een interactief programma dat vraagt om een begincode en een eindcode en dat alle codes met bijbehorend teken afdrukt voor tekens met een ASCII-code vanaf begincode tot en met eindcode. Een RUN van dit programma ziet er bijvoorbeeld zo uit:

```
Begin code? 36
Eind code ? 63

 36 = $          37 = %
 38 = &          39 = '
 40 = (          41 = )
 42 = *          43 = +
 44 = ,          45 = -
 46 = .          47 = /
 48 = 0          49 = 1
 50 = 2          51 = 3
 52 = 4          53 = 5
 54 = 6          55 = 7
 56 = 8          57 = 9
 58 = :          59 = ;
 60 = <          61 = =
 62 = >          63 = ?
```

De twee kolommen komen overeen met de standaardtabulatie van het beeldscherm (, in de **PRINT**-opdracht).

Als we de som van de getallen 1,2,3,...,N kunnen uitrekenen, kunnen we ook het produkt van deze getallen berekenen. In de wiskunde heet het produkt

1×2×3×4× ... ×N : N faculteit; notatie N!

Het volgende programma berekent N!:

```
100 REM N-faculteit
110 CLS
120 PRINT "Dit programma berekent
           N-faculteit. Toets voor N
           een waarde in";
130 INPUT N
140 PRODUCT=1
150 FOR GTAL=1 TO N
160    PRODUCT=PRODUCT*GTAL
170 NEXT GTAL
180 PRINT
190 PRINT N;"-faculteit is"; PRODUCT
200 END
```

N faculteit (N!) is het aantal manieren om N objecten achter elkaar te plaatsen (het aantal permutaties van N objecten). Laten we eens kijken op hoeveel manieren we vier speelkaarten kunnen rangschikken:

```
Dit programma berekent
N-faculteit. Toets voor N
een waarde in? 4

 4 -faculteit is 24
```

Het aantal manieren om vier speelkaarten te rangschikken is $1 \times 2 \times 3 \times 4 = 24$.
Nu met zeventien kaarten:

```
Dit programma berekent
N-faculteit. Toets voor N
een waarde in? 17

 17 -faculteit is 3.55687428096E+14
```

Oei!.....wat is dit?

3.55687428096E+14 betekent $3{,}55687428096 \times 10^{14}$ dus 355.687.428.096.000. Waarden groter dan 99.999.999.999 worden door de computer in een soort wetenschappelijke notatie afgedrukt. De E in deze getallen staat voor Exponent, en wel de Exponent van 10. Dergelijke getallen heten getallen met **drijvende komma** (floating point getallen). Had u gedacht dat je 17 speelkaarten op zoveel verschillende manieren kan rangschikken?

Op hoeveel verschillende manieren kunnen we een heel kaartspel rangschikken? (52 faculteit.)

```
Dit programma berekent
N-faculteit. Toets voor N
een waarde in? 52
Overflow in 160
```

De computer 'loopt over'; 52 faculteit is te groot voor hem.
De computer kan werken met getallen tot 1.0E+64 (1×10^{64}).

3.3 FOR-NEXT MET STAPGROOTTE ONGELIJK ÉÉN

In de tot nu toe gegeven voorbeelden is steeds gewerkt met een stapgrootte 1; de lusteller werd steeds met 1 verhoogd. We kunnen deze ophoging echter net zo groot (of net zo klein) maken als we zelf willen. In de FOR-opdracht

```
FOR X = 1 TO 10 STEP 2
```

worden stappen van twee genomen, en in

```
FOR X = 15 TO 30 STEP .25
```

stapjes van een kwart, en in

```
FOR X = 30 TO 5 STEP -5
```

stappen van min vijf!

Denk erom dat bij een FOR-NEXT lus met een positieve stapgrootte en een eindwaarde die kleiner is dan de beginwaarde, bijvoorbeeld FOR I = 10 TO 4 STEP 2, de lus toch één keer wordt uitgevoerd. Dit geldt ook voor een negatieve stapgrootte en een eindwaarde die groter is dan de beginwaarde, bijvoorbeeld FOR I = 4 TO 10 STEP -1. Een voorbeeld:

```
10 REM negatieve stapgrootte
20 CLS
30 INPUT "Toets een woord in"; WRD$
40 PRINT
50 FOR I= LEN(WRD$) TO 1 STEP -1
60    PRINT LEFT$(WRD$,I)
70 NEXT I
80 END
```

Als we dit draaien dan zien we bijvoorbeeld dit:

```
Toets een woord in? afbouwen

afbouwen
afbouwe
afbouw
afbou
afbo
afb
af
a
Ok
```

opdracht 3.6

Schrijf een programma dat de volgende uitvoer oplevert:

```
run

 15 * 15 = 225
 25 * 25 = 625
 35 * 35 = 1225
 45 * 45 = 2025
Ok
```

3.4 FOR-NEXT LUS BINNEN EEN FOR-NEXT LUS

We mogen in BASIC binnen een FOR-NEXT lus weer een FOR-NEXT lus opnemen. We spreken dan van **'geneste'** FOR-NEXT lussen. Wel moeten beide lussen een andere lusvariabele gebruiken.
Stel we gaan zes keer gooien met een zuivere dobbelsteen. Na zes keer gooien bepalen we het gemiddelde van het aantal gegooide ogen. Dit gemiddelde drukken we af. Dit herhalen we tenslotte tien keer, zodat we uiteindelijk zestig maal gegooid hebben en tien keer een gemiddelde hebben afgedrukt. We zijn benieuwd of die gemiddelden een beetje 'bij elkaar in de buurt' liggen. Bij gebrek aan een echte dobbelsteen schrijven we even een simulatieprogrammaatje (simuleren is nabootsen).

Uit hoofdstuk 1 weten we dat het aantal ogen dat in een worp met een dobbelsteen gegooid wordt kan worden nagebootst met

```
INT(6*RND(1) + 1)
```

Als we dit zes keer doen, waarbij we steeds de totale som van het aantal gegooide ogen onthouden, kunnen we het gemiddelde van deze zes worpen berekenen volgens:

```
SOM=0
FOR WRP=1 TO 6
   SOM=SOM+INT(6*RND(1)+1)
NEXT WRP
PRINT "Het gemiddelde van 6 ";
"worpen:";INT(10*SOM/6+.5)/10
```

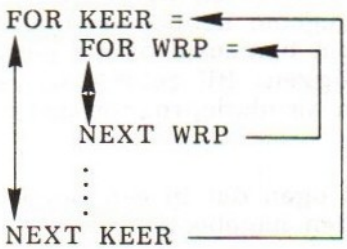
Als we dit tien keer willen herhalen, zetten we er gewoon nog een FOR-NEXT omheen:

```
FOR KEER=1 TO 10
   SOM=0
   FOR WRP=1 TO 6
      SOM=SOM+INT(6*RND(1)+1)
   NEXT WRP
   PRINT "Het gemiddelde van 6 ";
   "worpen:";INT(10*SOM/6+.5)/10
NEXT KEER
```

Als we er nu regelnummers voorzetten hebben we een BASIC programma:

```
100 REM 10 keer zes maal gooien met
          een dobbelsteen
110 CLS
120 FOR KEER=1 TO 10
130    SOM=0
140    FOR WRP=1 TO 6
150       SOM=SOM+INT(6*RND(1)+1)
160    NEXT WRP
170    PRINT "Het gemiddelde van 6 ";
       "worpen:";INT(10*SOM/6+.5)/10
180 NEXT KEER
190 END
```

Het is gebruikelijk de bij elkaar horende FOR en NEXT altijd recht onder elkaar te zetten. Dit maakt de programmatekst zeer overzichtelijk.



U ziet dat bij elke doorloop van de KEER-lus de variabele SOM opnieuw nul gemaakt wordt; dit moet omdat we steeds opnieuw voor elke zes keer gooien (WRP-lus) het gemiddelde willen uitrekenen.

Laten we eens kijken of de statistici gelijk hebben als zij beweren dat die gemiddelden niet zo gek ver uit elkaar zullen liggen:

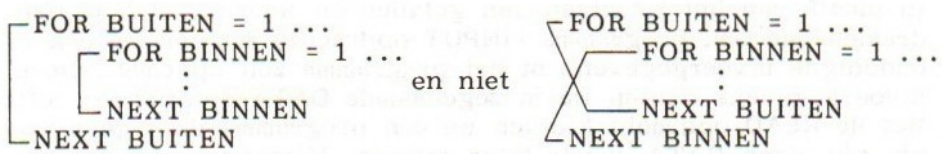
```
Het gemiddelde van 6 worpen: 3.5
Het gemiddelde van 6 worpen: 4
Het gemiddelde van 6 worpen: 2.5
Het gemiddelde van 6 worpen: 3.7
Het gemiddelde van 6 worpen: 2.5
Het gemiddelde van 6 worpen: 3.7
Het gemiddelde van 6 worpen: 1.3
Het gemiddelde van 6 worpen: 4.3
Het gemiddelde van 6 worpen: 3.3
Het gemiddelde van 6 worpen: 4.7
```

Inderdaad tamelijk dicht bij elkaar. U zult zien dat, als we per KEER bijvoorbeeld zestig W(O)RPEN simuleren, de tien gemiddelden nog dichter bij elkaar liggen.

opdracht 3.7

Wijzig het bovenstaande programma zo dat het aantal keren en het aantal worpen door de gebruiker kan worden ingetoetst. Pas daarbij ook de tekst in de PRINT-opdracht aan. Probeer dit ook eens op een printer af te drukken als u er een hebt! U hoeft alleen maar **L** voor **PRINT** te zetten, **LPRINT** dus.

Denk erom dat bij geneste FOR-NEXT lussen de binnenste lus altijd geheel binnen de buitenste lus valt, dus:



Nog een voorbeeld met een geneste FOR-NEXT lus:

```
100 REM diagonaalweb
110 CLS
120 COLOR 1,15,4 : SCREEN 2
130 Y1=0 : Y2=191
140 FOR X1= 16 TO 240 STEP 32
150     FOR X2=16 TO 240 STEP 32
160         LINE (X1,Y1)-(X2,Y2)
170     NEXT X2
180 NEXT X1
190 A$=INPUT$(1)
200 COLOR 15,4,4 : END
```

Kunt u raden wat u zult zien?

opdracht 3.8

Verander dit programma zo, dat u het aantal punten kunt intoetsen, alsmede de achtergrond-, voorgrond- en afdrukkleur. Zorg dat het plaatje blijft staan tot u op een toets drukt.

Wat niet mag is het veranderen van de lusvariabele binnen de lus zelf. De lusvariabele mag nooit in een toekenningsopdracht links van het = teken voorkomen. Dus dit mag niet:

```
FOR I = 1 TO 10
   PRINT I
   I = I + 1
NEXT I
```

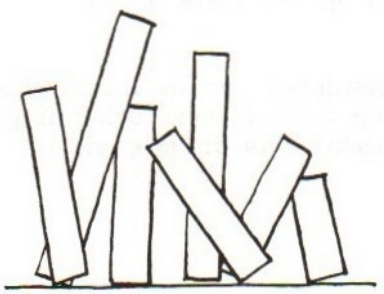
3.5 INLEZEN VAN GEGEVENS MET EEN FOR-NEXT LUS

Met behulp van een DATA-opdracht kunnen we invoergegevens voor een programma in het programma opnemen. Deze gegevens kunnen we vervolgens inlezen met de READ-opdracht

READ leest gegevens uit een DATA lijst

In plaats van het intoetsen van getallen en woorden tijdens het draaien van een programma (INPUT-opdracht) kunnen we ook de benodigde invoergegevens in het programma zelf opnemen. Deze invoergegevens moeten we in zogenaamde DATA-opdrachten zetten. Met de READ-opdracht kunnen we een programma deze gegevens uit zijn eigen DATA-regels laten inlezen. Hieronder zien we een voorbeeld. Dit programma tekent een eenvoudig staafdiagram met horizontaal liggende staven. De tekst, die voor elke staaf wordt afgedrukt, alsmede de lengte van de staaf leest het programma zelf uit de vier DATA-regels, die aan het einde van het programma zijn opgenomen.

```
10 REM Een liggend staafdiagram
20 CLS
30 READ N 'lees aantal staven
40 FOR STAAF=1 TO N
50    READ MERK$,AANTAL
60    PRINT MERK$;
70    LOCATE 8
80    PRINT STRING$(AANTAL,219)
90    PRINT
100 NEXT STAAF
110 END
120    DATA 6
130    DATA "volvo",12,"mazda",9
140    DATA "opel",2,"ford",19
150    DATA "toyota",14,"datsun",16
```



We mogen net zoveel DATA-regels opnemen als we nodig hebben. In principe mogen DATA-regels overal in een programma voorkomen. Vaak zie je ze aan het begin van een programma. Wij zullen ze altijd direct na de END-opdracht opnemen, een beetje apart dus.

In het bovenstaande programma zien we een aantal nieuwe opdrachten. We zullen het programma even kort behandelen.

De READ N opdracht in regel 30 zorgt ervoor dat N de waarde 6 krijgt. Dit is de waarde in de DATA-regel 120, de eerste DATA-regel in het programma. Het programma weet nu dat bij de volgende READ-opdracht de waarde uit de volgende DATA-regel gelezen moet worden. We mogen net zoveel waarden in een DATA-regel zetten als we maar willen, mits we ze door komma's scheiden. Niet alleen getalwaarden, maar ook tekstwaarden (strings), zoals we in de DATA-regels 130, 140 en 150 zien.
In regel 40 begint de FOR-NEXT lus, waarin we steeds een automerk en het aantal mensen dat een auto van dat merk bezit, inlezen. Dit inlezen gebeurt in regel 50 waar we in één READ-opdracht steeds twee waarden uit een DATA-regel lezen, in dit geval steeds een tekstwaarde (de merknaam) en een getalwaarde (het aantal bezitters van een auto van dat merk). Deze twee invoerwaarden zijn steeds gescheiden door een komma. We zien dat per DATA-regel twee maal de READ-opdracht moet worden uitgevoerd om alle gegevens uit de DATA-regel te lezen. Dit komt omdat elke DATA-regel vier waarden bevat, terwijl de READ-opdracht in regel 50 er maar steeds twee tegelijk inleest. Regel 60 drukt op een nieuwe regel de merknaam af. Door de ; blijft de cursor op de positie vlak na de merknaam staan. De LOCATE 8 regel zet de cursor op de achtste positie van de lopende regel. We kunnen met LOCATE kolom,rij de cursor op elke gewenste (kolom,rij) positie op het scherm zetten. Laten we de rij-positie weg, zoals in LOCATE 8 dan wordt de cursor op de aangegeven positie, van de rij waar de cursor dan is, gezet. De STRING$-functie drukt het opgegeven teken (219) een aantal (AANTAL) keer af. Teken 219 is de cursor.

STRING$(n,asciicode)	maakt een string met n dezelfde tekens, waarvan de ASCII-code is opgegeven

Dit programma laat zien hoe je met hele eenvoudige tekstgraphics (scherm 0) een staafdiagram kunt tekenen.

opdracht 3.9

De 'lengte' van zo'n staaf is beperkt; immers het scherm is maar 37 (hoogstens 40) tekens breed en het merk neemt ook nog wat ruimte in. Stel dat we 'rechts' van elke staaf nog drie posities op

het scherm over hebben. Voeg in een programmaregel iets toe waardoor rechts van elke staaf de lengte van de staaf wordt afgedrukt, dus het aantal 'cursortjes' waaruit de staaf is opgebouwd.

We zullen nog veel meer toepassingen met READ-DATA tegenkomen.

3.6 EEN KIJKJE IN HET GEHEUGEN

We kunnen de FOR-NEXT lus ook goed gebruiken om een kijkje in het geheugen te nemen. Hoe ziet dit MSX-geheugen eruit en wat zit er allemaal in? Het standaardgeheugen van een MSX-computer * is op te vatten als een ladenkast met 64 × 1024 (1K = 1024) = 65536 laden. Deze laden zijn genummerd van 0 t/m 65535. In elke lade kan één teken (8 bits) worden opgeslagen. In plaats van lade zullen we voortaan geheugenplaats zeggen. Het nummer van een geheugenplaats noemen we voortaan het adres. Naast dit standaardgeheugen bezit elke MSX-computer ook nog een 16K videogeheugen. Daarover straks meer.

In het standaardgeheugen van 64K zit de BASIC-vertolker, zeg maar het BASIC-systeem. Dit heet ook wel BASIC-ROM. Het is een stuk Read-Only-Memory dat door de fabrikant 'gevuld' is met MSX BASIC en waarin wij zelf niets kunnen veranderen. Read Only wil zeggen dat we er alleen uit kunnen lezen. Wat is dat nu, 'het BASIC-systeem'? Je zou kunnen zeggen dat de BASIC-ROM alle taalregels van BASIC en alle spelregels, waar wij ons in BASIC aan moeten houden, bevat. De BASIC-ROM zorgt ervoor dat alles wat wij intikken gecontroleerd wordt, bjivoorbeeld op een juiste spelling, en reageert als we een fout maken. Verder zorgt het voor het uitvoeren van allerlei handelingen als reactie op door ons ingetikte opdrachten of programma's. Van dit laatste zullen we een voorbeeld geven.

Als we de computer aanzetten zien we op de onderste regel van het scherm de volgende BASIC-woorden:

color, auto, goto, list en run

We weten nu dat elk van deze BASIC-woorden een woord is dat op het beeldscherm verschijnt als we op een van de functietoetsen F1, F2, ... of F5 drukken. Hoe weet de computer nu dat, als hij wordt aangezet, hij altijd (default) deze vijf woorden op de onderste regel moet afdrukken? Welnu, dat staat in zijn BASIC-ROM! Met de BASIC-opdracht PEEK kunnen we de inhoud van een geheugenplaats, dus ook van de BASIC-ROM bekijken. Letterlijk vertaald is to PEEK 'gluren'.

* We gaan even uit van een MSX-1 computer.

PRINT PEEK(n)	geeft de inhoud (een code tussen 0 en 255) van de geheugenplaats met adres n

Waar zouden die vijf woorden van de functietoetsen F1 t/m F5 staan? Wij weten het; bekijk daarom het onderstaande programma en datgene wat dit programma afdrukt.

```
10 REM peek in het geheugen
20 FOR ADRES=5033 TO 5099
30    PRINT PEEK(ADRES);
40 NEXT ADRES
50 END

run
 99  111  108  111  114  32  0  0  0
 0  0  0  0  0  0  0  97  117  116
 111  32  0  0  0  0  0  0  0  0  0
 0  0  103  111  116  111  32  0  0
 0  0  0  0  0  0  0  0  0  108  105
 115  116  32  0  0  0  0  0  0  0  0
  0  0  0  114  117  110
Ok
```

Met een FOR-NEXT lus gluren we even (PEEK) naar de inhoud van de adressen 5033 tot en met 5099 in de BASIC-ROM. Wat we zien drukken we op het scherm af. Het resultaat is een reeks van 68 getallen. Dit zijn codes die de inhoud vormen van de geheugenplaatsen met de adressen 5033 t/m 5099. Als we vertellen dat dit ASCII-codes zijn dan kunt u ze waarschijnlijk wel vertalen in voor ons herkenbare tekens (Appendix D). Door in regel 30 in bovenstaand programma **PEEK(ADRES)** te veranderen in **CHR$(PEEK(ADRES))** kunnen we dit vertaalwerk echter door de computer laten doen. Wat er dan afgedrukt wordt zien we nu:

```
10 REM peek in het geheugen
20 FOR ADRES=5033 TO 5099
30    PRINT CHR$(PEEK(ADRES));
40 NEXT ADRES
50 END

run
color auto goto list run
Ok
```

Het programma drukt precies de tekst van de onderste beeldschermregel af. Als 't goed is ziet u deze regel nu twee keer. De onderste beeldschermregel staat er altijd. Die daarboven is zojuist door het programma afgedrukt. De computer weet dus wat hij op

die onderste regel moet afdrukken omdat dat blijkbaar in zijn BASIC-ROM staat en wel in de geheugenplaatsen met de adressen 5033 t/m 5099.

Tik nu in: `KEY OFF ◁`

Hierdoor verdwijnt de tekst in de onderste beeldschermregel. Deze regel kunnen we nu zelf gebruiken. Wat zou er gebeurd zijn met de inhoud van de adressen 5033 t/m 5099? Daar komen we snel genoeg achter, want het PEEK-programma staat nog in het geheugen:

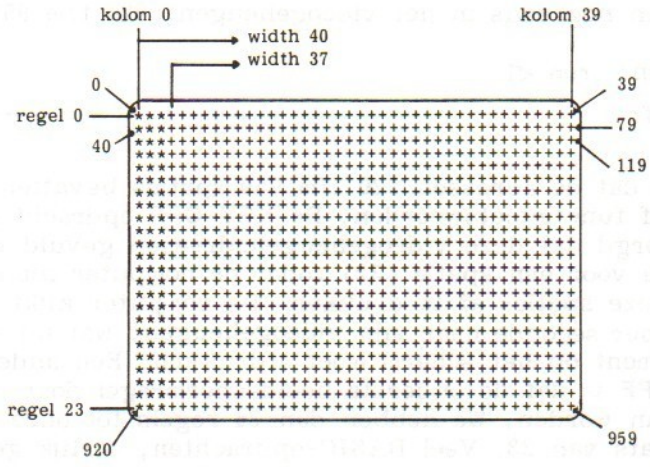
wij tikken in: `run ◁`

hij drukt af: `color auto goto list run`

We zien dat de BASIC-ROM nog steeds de vijf BASIC functietoetswoorden bevat. Dat is maar goed ook, anders zou de computer niet meer weten wat hij, vlak na het aanzetten, op de onderste beeldschermregel af moet drukken. Dat de onderste beeldschermregel leeg is moet dan blijkbaar ergens anders staan. Alles wat al dan niet op het scherm moet worden afgedrukt staat inderdaad ergens anders, namelijk in het VIDEO-geheugen. Dit is een apart 16K (bij MSX-2 128K) geheugen waar alle informatie over het schermgebruik (SCREEN 0, 1, 2 of 3; bij MSX-2 t/m SCREEN 9) over de achtergrond-, afdruk- en randkleur, over sprites en nog meer, in wordt opgeslagen. Elke schermindeling, dat wil zeggen de tekstschermen SCREEN 0 en 1 en de graphicsschermen SCREEN 2 en 3, heeft een eigen stukje videogeheugen.

In dit videogeheugen kunnen we ook gluren. Dit noemen we 'videopeeken', afgekort tot VPEEK. Als we in SCREEN 0 werken (het tekstscherm met 24 beedschermregels van maximaal 40 tekens per regel) dan bevatten de eerste 2948 geheugenplaatsen in het videogeheugen informatie over datgene wat op het scherm moet worden afgedrukt. Deze informatie bestaat uit de codes van de tekens die op de 24 regels van het scherm worden afgedrukt, over de afdrukkleur, over het teken dat de cursor voorstelt en nog veel meer.

Elke schermpositie, in SCREEN 0, heeft dus een videogeheugenadres, waarin de code staat van het teken dat op die schermpositie moet worden afgedrukt. Bij SCREEN 0 zijn dit maximaal $24 \times 40 = 960$ posities. Als deze 960 posities zouden corresponderen met de eerste 960 geheugenadressen in het videogeheugen (bij SCREEN 0) dan kunnen we uitrekenen waar in het geheugen de inhoud van de onderste beeldschermregel zou moeten staan.



Video-adressen voor schermposities in SCREEN 0.

Laten we de beeldschermregels nummeren van 0 t/m 23 (dit zijn 24 regels) en de regelposities (kolommen) van 0 t/m 39 (dit zijn 40 posities). De onderste beeldschermregel is dan regel 23. Als we de schermposities regel voor regel in het geheugen doornummeren dan komt de eerste positie van regel 23 overeen met video-adres $23 \times 40 = 920$. Het punt helemaal rechtsonder op het beeldscherm heeft dan adres $23 \times 40 + 39 = 959$.

Laten we eens gaan kijken of de tekst van de onderste beeldschermregel inderdaad in dit videogeheugen is terug te vinden. We zetten de computer eerst uit en weer aan of we tikken in: KEY ON ◁ zodat we de vijf functietoetswoorden weer onderaan het scherm zien staan. We tikken nu het volgende programma in en draaien het:

```
10 REM peek in het videogeheugen
20 FOR VADRES=920 TO 959
30    PRINT CHR$(VPEEK(VADRES));
40 NEXT VADRES
50 END

run
  color  auto   goto   list   run

Ok
```

Inderdaad bevatten de videogeheugenadressen 920 t/m 959 de tekst van de onderste beeldschermregel.

wij tikken in: KEY OFF ◁

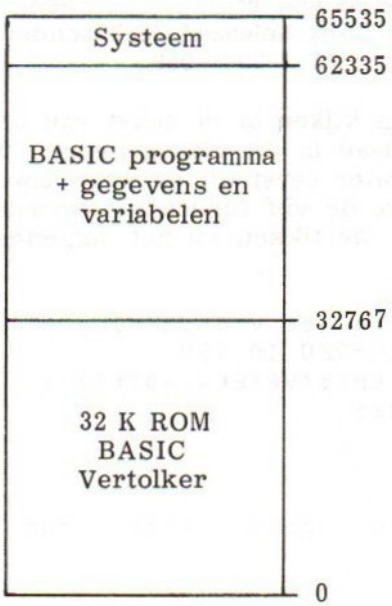
En we kijken nog eens in het videogeheugen (920 t/m 959):

wij tikken in: run ◁

hij drukt af:

← niets, dat wil zeggen 40 spaties

Nu zien we dat de adressen 920 t/m 959 spaties bevatten en niet meer de vijf functietoetswoorden. De KEY OFF opdracht heeft ervoor gezorgd dat deze videogeheugenadressen gevuld worden met de code voor een spatie (32) zodat de computer niets anders overblijft deze spaties af te drukken. De computer kijkt namelijk vele malen per seconde naar zijn videogeheugen, wat hij op een bepaald moment op het scherm moet afdrukken. Een ander effect van KEY OFF is dat de onderste beeldschermregel door ons gebruikt kan worden. We hebben dan 24 regels tot onze beschikking in plaats van 23. Veel BASIC-opdrachten, eerlijk gezegd bijna alle!, hebben tot gevolg dat er allerlei codes op diverse plaatsen in het geheugen worden veranderd. De opdracht voor het veranderen van de inhoud van die bepaalde geheugenplaatsen komt uit de BASIC-ROM als gevolg van een door ons ingetikte directe opdracht of als gevolg van het uitvoeren van een programma-opdracht. Bekijk de onderstaande geheugenindeling eens.



Geheugenindeling MSX-1 computer

De BASIC-ROM van MSX-1 computers bevindt zich op de adressen 0 t/m 32767; dit is precies 32K. De ruimte op de adressen 32768 t/m 62335 is bedoeld voor het opslaan van programma's, die wij intikken, en voor allerlei dingen die bij het draaien van die programma's nodig zijn. De adressen 62336 t/m 65535 worden door de BASIC-ROM gebruikt voor het bijhouden van allerlei systeemvariabelen. Zo'n systeemvariabele is bijvoorbeeld de kleur van de voorgrond, de afdrukkleur. In totaal bevat dit geheugengebied 256 systeemvariabelen, waarvan de waarde voortdurend verandert. De meeste gebruiken één of twee geheugenplaatsen (bytes), maar er zijn er ook die 160 bytes nodig hebben. Laten we heel kort aangeven hoe de BASIC-ROM bijvoorbeeld de systeemvariabele 'afdrukkleur' verandert. Zet de computer uit en weer aan. De standaardafdrukkleur is dan wit. De code voor wit is 15.

```
wij tikken in:  PRINT PEEK(62441) ◁
hij drukt af:    15
```

De 'afdrukkleur', beter gezegd de code van de afdrukkleur, wordt bijgehouden op het geheugenadres 62441 (we zeggen: in byte 62441). We gaan nu de afdrukkleur veranderen in zwart:

```
wij tikken in:  COLOR 1 ◁
hij drukt af:   Ok      (in 't zwart)

wij tikken in:  PRINT PEEK(62441) ◁
hij drukt af:    1
```

1 is de code voor zwart. De COLOR-opdracht die geïnterpreteerd wordt in de BASIC-ROM heeft dus de inhoud van het adres 62441 van 15 in 1 veranderd. Wat de BASIC-ROM kan, kunnen wij ook! Wij kunnen zelf ook de inhoud van geheugenplaatsen veranderen, behalve dan die in de BASIC-ROM. Het veranderen van geheugenplaatsen in het systeemgebied kan tot rare (en nare) dingen leiden. Wij zullen dit dan ook niet doen, maar beperken ons tot het experimenteren met het videogeheugen.

VPOKE n,code :	zet op het video-adres n de opgegeven code. Code moet tussen 0 en 255 zijn

Met de VPOKE-opdracht kunnen wij in het videogeheugen zelf codes POKEn.

```
wij tikken in:  SCREEN 0 ◁
                VPOKE 39,65
hij drukt af:    A   (in de rechterbovenhoek van het scherm)
```

wij tikken in: VPOKE 959,66

hij drukt af: B (in de rechterbenedenhoek van het scherm)

opdracht 3.10

Schrijf een programma om op de tweede beeldschermregel 40 *'tjes te VPOKEn. Kijk in Appendix D om de code van een * op te zoeken. Geef eerst een WIDTH 40 opdracht om de regelbreedte naar 40 tekens over de schakelen.

We gaan nu wat experimenteren. Eerst:

wij tikken in: COLOR 1,8,14 : SCREEN 1 ◁

hij drukt af: Het tekstscherm SCREEN 1 met witte rand, rode achtergrond en zwarte afdrukkleur.

We gaan nu de cursor van kleur veranderen:

wij tikken in: VPOKE BASE(6) + 31,1

hij drukt af: De cursor wordt in de randkleur afgedrukt.

BASE(6) is het videogeheugenadres van waaraf informatie over het gebruik van kleuren op het tekstscherm SCREEN 1 te vinden is. Op 31 geheugenplaatsen van dit begin (BASE(6) + 31) bevindt zich informatie over de kleur van de cursor. Door de code in deze geheugenplaats te veranderen, veranderen we de kleur van de cursor. We kunnen nog enkele grapjes uithalen. Maak het scherm schoon en tik achtereenvolgens in:

```
ABCDEFGHIJKLMNOPQRSTUVabcdefghijk
lmnopqrstuvwxyz0123456789:,;<>
```

Geef nu een RETURN, hetgeen een Syntax error zal opleveren. Niets van aantrekken, we gaan gewoon door. Veeg het scherm niet schoon!

wij tikken in: VPOKE BASE(6) + 4,2

hij drukt af: Een groene achtergrond, maar alle tekens behalve spaties blijven zwart op een rode achtergrond

wij tikken in: VPOKE BASE(6) + 5,4

hij drukt af: De tekens , ( ) en + worden wit afgedrukt op een blauwe achtergrond

wij tikken in: `VPOKE BASE(6) + 6,5`

hij drukt af: De cijfers 0, 1, 2, 3, 4, 5, 6 en 7 worden wit afgedrukt op een lichtblauwe achtergrond

wij tikken in: `VPOKE BASE(6) + 7,5`

hij drukt af: De cijfers 8 en 9 en de tekens : ; < en > worden nu ook wit op een lichtblauwe achtergrond

Met VPOKE BASE(6) + 8 (en 9, 10 en 11),8 worden alle hoofdletters in wit afgedrukt op een rode achtergrond.
Met VPOKE BASE(6) + 12 (en 13, 14 en 15),13 worden alle kleine letters in wit op een paarse achtergrond afgedrukt.
Zo kunnen we nog wel even doorgaan. Zet de computer nu uit en weer aan en tik het volgende programma in:

```
100 REM KlEuReNpAlEt
110 CLS
120 COLOR 1,15,15:SCREEN1
130 FOR I=8 TO 11
140     VPOKE BASE(6)+I,8
150     VPOKE BASE(6)+I+4,13
160 NEXT I
170 VPOKE BASE(6)+6,5
180 VPOKE BASE(6)+7,5
190 PRINT "HOOFDLETTERS WIT OP ROOD"
200 PRINT "kleine letters wit op paars"
210 PRINT "0,1,2,3,4,5,6,7,8,9 wit op blauw"
220 END
```

Over het videogeheugen zijn boeken vol te schrijven. Experimenteer gerust verder en ontdek nieuwe mogelijkheden.

3.7 DOE-ZOLANG- EN HERHAAL-TOTDAT-LUSSEN

DOE-ZOLANG- en HERHAAL-TOTDAT-lussen worden gebruikt in situaties waarin niet van tevoren vaststaat hoeveel keer een programmalus doorlopen zal worden. Het beëindigen van het uitvoeren van de lus hangt af van wat er binnenin de lus gebeurt. Vaak krijgt een bepaalde variabele in de lus een bepaalde waarde waardoor de lus wordt beëindigd. Er wordt dan aan het begin van de lus (DOE-ZOLANG-lus) of aan het einde van een lus (HERHAAL-TOTDAT-lus) getest of die variabele die bepaalde waarde heeft. Vaak kan in één bepaalde situatie zowel een DOE-ZOLANG- als een HERHAAL-TOTDAT-lus geprogrammeerd worden. Maar niet altijd; hiervan zullen we een voorbeeld laten zien.

We willen een programma maken dat voor elk niet-negatief getal dat iemand intoetst, de wortel van dat getal afdrukt. Er moet meer dan één getal ingetoetst kunnen worden; dus hebben we een lus-constructie nodig. Door het intoetsen van een negatief getal wordt het programma beëindigd. Aangezien het programma van tevoren niet weet wanneer er een negatief getal zal worden ingetoetst kunnen we geen FOR-NEXT-lus gebruiken. We zouden de lus als volgt kunnen formuleren:

> HERHAAL het vragen om het intoetsen van een getal en het afdrukken van de wortel uit dit getal TOTDAT een negatief getal is ingetoetst

In een programma zou dit er zo uitzien:

```
15 '===
20   INPUT "Toets een getal in"; G
30   PRINT "De wortel uit"; G;
           "IS"; SQR(G)
40   PRINT
50 IF G >= 0 THEN 20
55 '===herhaal totdat G negatief is
60 END
```

In regel 50 wordt de bewering 'G >= 0' (G groter dan of gelijk aan nul) op waarheid getest. Is deze bewering waar dan vraagt het programma om het volgende getal; anders gaat het programma verder met de opdracht na de IF-opdracht hetgeen het beëindigen van de lus en daarmee van het programma inhoudt. Stel dat we dit programma invoeren en draaien:

```
run
Toets een getal in? 9
De wortel uit 9 IS 3

Toets een getal in? 20
De wortel uit 20 IS 4.4721359549996

Toets een getal in? -4
De wortel uit-4 IS
Illegal function call in 30      ◄ foutmelding
Ok
```

Voor G = -4 krijgen we in regel 30 een foutmelding; het argument in een SQR-functie moet niet-negatief zijn. Alhoewel het intoetsen van een negatief getal als **'stopwaarde'** voor dit programma voor de hand ligt gaat er toch iets fout; tenminste als je het beëindigen van een programma door middel van een fout(melding) niet als een nette manier beschouwt om een programma te laten beëindigen. We zouden dit programma wel 'netjes' kunnen laten beëindigen door ervoor te zorgen dat na het intoetsen van het negatieve getal de wortel uit dat getal niet meer wordt berekend. Dat kan, maar dan moeten we in plaats van de HERHAAL-TOTDAT-lus een DOE-ZOLANG-lus gebruiken. We zouden deze DOE-ZOLANG-lus als volgt kunnen formuleren

vraag om het intoetsen van het eerste getal
DOE ZOLANG een niet-negatief getal is ingetoetst
het afdrukken van de wortel uit dit getal en
het vragen om het intoetsen van het volgende getal

In een programma ziet dat er zo uit:

```
10 INPUT "Toets een getal in"; G
15 '***doe zolang G niet negatief is
20 IF G < 0 THEN 70
30   PRINT "De wortel uit"; G;
           "IS"; SQR(G)
40   PRINT
50   INPUT "Toets een getal in"; G
60 GOTO 20
65 '***
70 END
```

Als we dit programma draaien dan zal het programma na het intoetsen van bijvoorbeeld -4 netjes worden beëindigd.

```
run
Toets een getal in? -4
Ok
```

Voor deze toepassing verdient een DOE-ZOLANG-lus dus de voorkeur boven een HERHAAL-TOTDAT-lus.

We kunnen dit wortel-probleem ook goed gebruiken om een programma te laten zien met zowel een HERHAAL-TOTDAT-lus als een DOE-ZOLANG-lus. Stel dat we niet kiezen voor een negatief getal als stopwaarde maar voor het getal nul als stopwaarde en dat we als eerste aan het gebruik van een HERHAAL-TOTDAT-lus denken. We krijgen dan:

```
15 '===
20   INPUT "Toets een getal in"; G
30   PRINT "De wortel uit"; G;
          "IS"; SQR(G)
40   PRINT
50 IF G <> 0 THEN 20
55 '===herhaal totdat G nul is
60 END
```

In regel 50 wordt getest of G ongelijk is aan nul. Zo ja, dan wordt de lus nog eens uitgevoerd; zo nee, dan stopt het programma. Ook nu zal het verkeerd aflopen als we een negatief getal intoetsen. Tussen regel 20 en 30 zullen we moeten testen of een negatief getal is ingetoetst. Als dit inderdaad gebeurd is dan moet het programma vragen om een nieuw getal in te toetsen. Toetsen we dan weer een negatief getal in dan moet er weer om een nieuw getal worden gevraagd. Het programma moet dit spelletje net zo lang volhouden tot wij wel een positief getal of het getal nul intoetsen. Dit wordt dus een lus en wel een DOE-ZOLANG-lus want er is immers al iets ingetoetst (regel 20) dat vóór het vragen om een nieuw getal eerst getest moet worden. In het volgende programma hebben we deze DOE-ZOLANG-lus tussen de regels 20 en 30 geperst!

```
15 '===
20   INPUT "Toets een getal in"; G
21   '***doe zolang G negatief
22   IF G >= 0 THEN 30
23     INPUT "G moet >= 0 zijn"; G
24   GOTO 22
25   '***
30   PRINT "De wortel uit"; G;
          "IS"; SQR(G)
40   PRINT
50 IF G <> 0 THEN 20
55 '===herhaal totdat G nul is
60 END
```

We hebben nu dus een DOE-ZOLANG-lus (***) binnen een HERHAAL-TOTDAT-lus (===).

Laten we de REM-opdrachten weg en springen we ook niet meer in en laten we het programma door de computer (LIST) op het scherm afdrukken dan zien we dit:

```
20 INPUT "Toets een getal in"; G
22 IF G >= 0 THEN 30
23 INPUT "G moet >= 0 zijn"; G
24 GOTO 22
30 PRINT "De wortel uit"; G;"IS"; SQR
(G)
40 PRINT
50 IF G <> 0 THEN 20
60 END
```

Als je dit zo ziet herken je toch niet direct een DOE-ZOLANG-lus binnen een HERHAAL-TOTDAT-lus. Met andere woorden: de structuur van het programma komt zo niet tot uitdrukking in de wijze waarop de programmatekst wordt afgedrukt.
Toch is het erg belangrijk dat de diverse programmastructuren in een programma in de programmatekst tot uitdrukking komen. Het vergroot de leesbaarheid van de tekst en het is onontbeerlijk als we later een programma moeten aanpassen, uitbreiden of verbeteren.

Opdracht 3.11

Bekijk het volgende programma eens:

```
10 LINE INPUT "Uw zin is: "; Z$
15 '***doe zolang niet 'stop' is
    ingetoetst
20 IF Z$ = "stop" THEN 60
30      PRINT "'";Z$; "' is";LEN(Z$);
           "tekens lang"
40      LINE INPUT "Volgende zin"; Z$
50      GOTO 20
55 '***
60 PRINT "Ik stop ook"
70 END
```

In regel 10 wordt gevraagd een zin in te toetsen. Met **LINE INPUT** mogen we alles, ook komma's en spaties, in de zin opnemen; dat mag met alleen **INPUT** niet! (Zie hoofdstuk 6.)

De DOE-ZOLANG-lus (regels 15 tot en met 55) regelt dat de computer alleen de regels 30 en 40 (de lus-body) uitvoert als de bewering Z$ = "stop" niet waar is.

Programmeer dit programma met een HERHAAL-TOTDAT-lus. Wat is het verschil met het gegeven programma wanneer we beide programma's zouden draaien?

Tot slot nog twee programmavoorbeelden waarin alle drie lusstructuren verwerkt zijn. Het zijn beide voorbeelden van low-resolution graphics. Het eerste programma werkt met SCREEN 0, het tweede met SCREEN 1.

Het eerste programma schiet zwarte gaatjes in een wit vlak. Het witte vlak (kleur 15) en de zwarte (kleur 1) gaatjes krijgen we door COLOR 1,15,15 : SCREEN 0. In regel 130 zetten we met LOCATE de cursor op een willekeurige kolompositie tussen 0 en 36 (37*RND(1)) en een willekeurige rij (24*RND(1)). Op die toevalspositie drukken we de cursor (teken met ASCII code 219) af. De INKEY$-functie in regel 150 zorgt ervoor dat, als we op een toets drukken, de code van deze toets in A$ wordt bewaard. Regel 160 is de laatste opdracht in de HERHAAL-TOTDAT-lus. Als er geen toets is ingedrukt, dat wil zeggen als er niets in A$ zit (de bewering A$ = "" is dan waar), dan wordt het volgende schot afgevuurd. Drukken we wel een toets in (A$ = "" is dan niet waar), dan krijgen we het bekende blauwe scherm weer terug.

```
100 REM gaatjes schieten
110 CLS : KEY OFF
120 COLOR 1,15,15 : SCREEN 0
125 '===
130   LOCATE 37*RND(1),24*RND(1)
140   PRINT CHR$(219);
150   A$=INKEY$
160 IF A$ = "" THEN 130
165 '===herhaal tot een toets is
    ingedrukt
170 COLOR 15,4,4:KEY ON :SCREEN0
180 END
```

opdracht 3.12

Verander dit programma zo dat er ook een toevalsteken op elke toevalspositie wordt afgedrukt. Neem tekens met een ASCII code groter dan 31.

In het tweede programmavoorbeeld laten we een 'kopbal' bovenaan het scherm los en laten hem op de onderrand 'stuiteren'. Als de kopbal uitgestuiterd is, stopt het programma en blijft de bal op de onderrand liggen. Het stuiteren vindt plaats in kolom 16 van het 32-tekens brede SCREEN 1. De randkleur is blauw (kleur 4), de achtergrond is wit (kleur 15) en de kopbal is groen (kleur 2).

We weten nu dat we op elke positie van het scherm een bepaald teken kunnen laten verschijnen als we de code van dat teken op de juiste plaats in het VIDEOgeheugen (V)POKEn. In Appendix D

zien we na de ASCII tabel nog een tabel met videocodes. Door deze codes in het videogeheugen te POKEn worden de daarbij horende videotekens afgedrukt.

Als 'kopbal' nemen we videoteken 2. Dit is een donker 'gezichtje' met twee witte ogen, neus en mond. De vraag is nu: waar bevindt zich in het videogeheugen het gebied waar we voor SCREEN 1 de schermposities kunnen programmeren? Alle MSX-computers beschikken over de BASE-functie, die voor een bepaalde SCREEN-stand een aantal beginposities in het videogeheugen aangeeft, van waar af bepaalde informatie te vinden is. Voor het tekstscherm in SCREEN 1 geeft BASE(5) het adres van de videogeheugenplaas die correspondeert met de linkerbovenhoek van het 32 tekens brede SCREEN 1-scherm. We hoeven niet te weten wat de waarde van BASE(5) is om alle schermposities aan te kunnen geven. Linksboven is gewoon BASE(5) en één positie naar rechts is BASE(5) + 1. Zo is de eerste positie op de tweede beeldschermregel BASE(5) + 32 enzovoort. Wilt u wel weten wat BASE(5) is,

tik dan in: `PRINT BASE(5)` ◁

hij drukt af: 6144

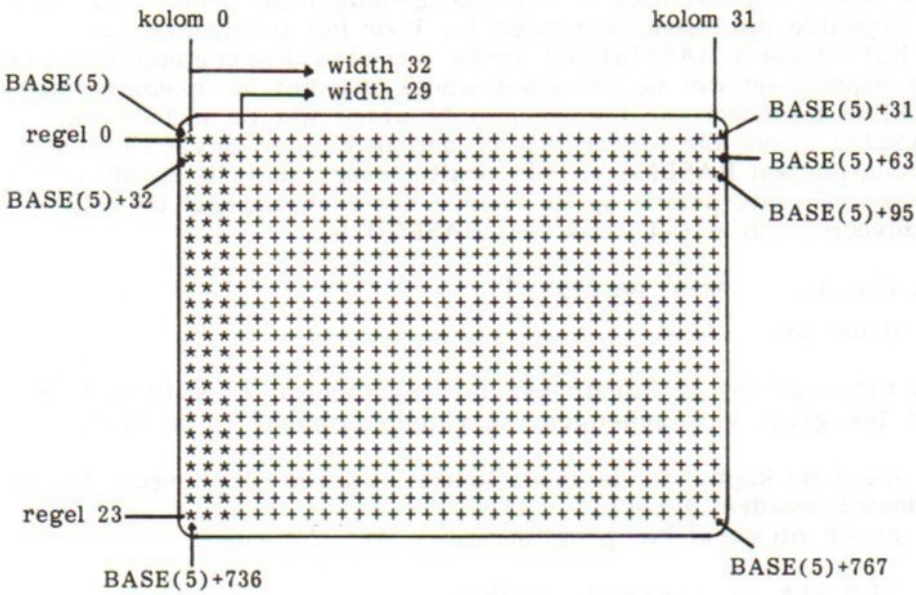
Dit betekent dat de inhoud van het tekstscherm in SCREEN 1, in het 16K grote videogeheugen, te vinden is vanaf adres 6144.

We laten de kopbal in de 16-de kolom stuiteren. Hij begint dus op videoschermadres BASE(5) + 15.
We geven nu eerst het programma.

```
100 REM stuiterende kopbal
110 COLOR 12,15,4 : SCREEN1
120 CLS :KEY OFF : B=BASE(5)
130 BOVEN=B+15
140 VPOKE BOVEN,2
145 '===
150  FOR CEL=BOVEN TO B+719 STEP 32
160     VPOKE CEL,32
170     VPOKE CEL+32,2
180     FOR TEL=1 TO 10:NEXT TEL
190  NEXT CEL
200 'kopbal is beneden
210 'lees hoe hoog het komt
220  READ BOVEN:BOVEN=BOVEN+B
225  '***doe zolang kopbal niet boven
230   IF CEL=BOVEN THEN 290
240     VPOKE CEL+32,32
250     VPOKE CEL,2
260     FOR TEL=1 TO 10:NEXT TEL
270     CEL=CEL-32
280   GOTO 230
285  '***
```

```
290 IF BOVEN <> 719+B THEN 150
295 '===herhaal tot kopbal is
        uitgestuiterd
300 END
310 DATA 47,79,143,207,271,367,
        463,591,687,719
```

SCREEN 1 ziet er in het videogeheugen zo uit:



Video-adressen voor schermposities in SCREEN 1; BASE(5) = 6144.

Als we SCREEN 1 inschakelen hebben we de standaard 29 posities per regel. Met **WIDTH 32** kunnen dit er maximaal 32 worden, vandaar dat de rechterbovenhoek van het scherm het video-adres BASE(5) + 31 heeft en is BASE(5) + 2 de eerste positie van het 29-tekens scherm; de eerste 3 worden dan niet gebruikt (zie tekening).

Het bovenstaande programma programmeert een stuiterende 'kopbal' die steeds minder hoog stuitert en ten slotte 'uitgestuiterd' op de onderrand van het scherm blijft liggen. De variabele **BOVEN** bevat steeds het video-adres van de schermpositie van waaruit de bal naar beneden valt. De startpositie is bovenaan het scherm op de 16-de regelpositie. In de bovenstaande tekening zien we dat het adres in het videogeheugen (voor SCREEN 1), dat overeenkomt met de linkerbovenhoek van het scherm, gegeven wordt door BASE(5). Dit is in het 16K videogeheugen het adres 6144. De

16-de regelpositie op de bovenste beeldschermregel heeft dus adres BASE(5) + 15. In regel 130 zien we dat BOVEN de waarde B + 15 krijgt, met B = BASE(5). BOVEN is dus gelijk aan het video-adres van de schermpositie waaruit de bal gaat vallen. De bal (videocode 2) wordt in regel 140 op dit adres gePOKEd. Hij is nu klaar om te vallen.

Dit vallen moet herhaald worden totdat de kopbal is uitgestuiterd, vandaar de HERHAAL-TOTDAT-lus in de regels 145 t/m 295.

De bal valt in de regels 150-190. Dit gebeurt in een FOR-NEXT-lus waarin de lusvariabele CEL de adressen van de beeldschermposities in de 16-de kolom doorloopt van BOVEN naar beneden. 'beneden' is adres B + 719. Dit is de 16-de regelpositie op de op-één-na onderste beeldschermregel.
We kunnen het vallen nabootsen door de bal op zijn huidige positie uit te vegen, dat wil zeggen door op die positie een spatie te zetten (code 32), en 32 adressen verder (CEL + 32) opnieuw de bal te VPOKEn. Als we het vallen niet zouden vertragen dan zouden we niet eens kunnen zien wat er gebeurt. Vandaar dat we de computer in regel 180 in zichzelf tot 10 laten tellen voordat de bal één regel verder valt. In regel 200 is de bal beneden en moet dus weer naar BOVEN. Hoe ver hij komt lezen we uit de DATA-regel (regel 220). Dit BOVEN-adres verhogen we met B (BASE(5)). We starten de opwaartse beweging op adres CEL + 32. Dit is altijd het adres BASE(5) + 751.

Wat we moeten doen om de bal omhoog te laten gaan is ZOLANG de bal nog niet BOVEN is, hem uitvegen en één regel hoger afdrukken. Dit doen we net zolang tot CEL (de plaats van de bal) gelijk is aan BOVEN.

Deze DOE-ZOLANG-lus bevindt zich in de regels 225-285. Omdat we voor dit omhoogstuiteren geen FOR-NEXT-lus gebruiken maar een DOE-ZOLANG-lus, moeten we in de lus zelf de CEL-aanwijzer steeds met 32 verlagen (regel 270) om de plaats (het video-adres) aan te geven waar de volgende 'kopbal' afgedrukt moet worden.

Dit voorbeeld laat dus zien dat we met de diverse lusstructuren heel mooi graphics kunnen programmeren door gebruik te maken van het videogeheugen en de VPOKE-opdracht. In plaats van met VPOKE kunnen we dit programma ook schrijven met de LOCATE-opdracht. Hierbij hoeven we niets van de video-adressen en van de BASE-functie af te weten. Als we LOCATE gebruiken veranderen de regels 130 en 140 in

```
130 LOCATE 13,0,0
140 PRINT CHR$(1); CHR$(66);
```

Regel 130 zet de cursor op de eerste regel op de 16-de positie (0,0 is de eerste positie!) en schakelt de cursor uit (laatste 0). Regel 140 drukt op die positie het teken CHR$(1); CHR$(66); af. Dit is het teken dat de kopbal voorstelt (zie Appendix D). In plaats van video-adressen werken we nu met regelnummers. De bal stuitert nu in de 16-de kolom tussen de beeldschermregels 0 en 23. De variabele BOVEN wordt dus nu een regelnummer. De DATA-regel zal dus ook regelnummers moeten bevatten. Hier volgt het programma met LOCATE. Het is belangrijk om te zorgen dat de cursor hierbij niet wordt afgedrukt, want anders wordt het beeld wat onrustig. Laat die derde nul in de LOCATE-opdrachten maar eens weg en kijk hoe het er dan uitziet!

```
100 REM stuiterende kopbal
110 COLOR 12,15,4 : SCREEN1
120 CLS :KEY OFF
130 LOCATE 13,0,0
140 PRINT CHR$(1);CHR$(66);
145 '===
150  FOR REGEL=BOVEN TO 22
160     LOCATE 13,REGEL,0 :
          PRINT CHR$(32);
170     LOCATE 13,REGEL+1,0 :
           PRINT CHR$(1);CHR$(66);
180     FOR TEL=1 TO 10:NEXT TEL
190  NEXT REGEL
200 'kopbal is beneden
210 'lees hoe hoog het komt
220  READ BOVEN
225  '***doe zolang kopbal niet boven
230   IF REGEL=BOVEN THEN 290
240     LOCATE 13,REGEL+1,0 :
         PRINT CHR$(32);
250     LOCATE 13,REGEL,0 :
        PRINT CHR$(1);CHR$(66);
260     FOR TEL=1 TO 10:NEXT TEL
270     REGEL=REGEL-1
280   GOTO 230
285  '***
```

```
290 IF BOVEN <> 22 THEN 150
295 '===herhaal tot kopbal is
        uitgestuiterd
300 END
310 DATA  1 ,2  ,4,  6,  9, 12,
          15, 19, 21, 22
```

Bedenk dat de LOCATE-opdracht rekening houdt met de 29- en 32-tekenstand in SCREEN 1. De 16-de kolom (regelpositie) wordt in de 29-tekenstand LOCATE 13,... De laatste puntkomma's in de PRINT-opdrachten in de regels 140, 160, 170, 240 en 250 zijn erg belangrijk!

opdracht 3.13

a. Breid het programma uit zodat de bal, nadat hij is uitgestuiterd, langzaam naar de rechterbenedenhoek rolt.

b. Hoe kunnen we de bal naarmate hij naar rechts rolt steeds langzamer laten rollen?

3.8 WAARHEIDSRELATIES IN BASIC

In het dagelijks leven horen we veel beweringen waarin twee of meer waarden met elkaar worden vergeleken; hier zijn er een paar:

1. Jan verdient minder dan $ 25.000,-- per jaar.
2. Wies is tenminste even oud als Roderik.
3. Hij geeft meer uit dan jullie allemaal bij elkaar.
4. Ik heb niet schoenmaat 43.
5. Er staat STOP op dat bord.

Wiskundig zouden we deze beweringen als volgt kunnen opschrijven:

1.	$x < 25000$	x is jaarsalaris
2.	$x \geq y$	x en y zijn leeftijden
3.	$x > (a+b+c+d+e)$	x, a, b, c, d, e zijn uitgaven
4.	$x \neq 43$	x is schoenmaat
5.	dit wordt moeilijk om wiskundig op te schrijven	

In zo'n bewering staat een relatie tussen twee of meer waarden. Zo'n waarde kan een constante zijn (25000 of 43), een variabele (x of y) of een uitdrukking (a+b+c+d+e). Deze waarden in een bewering worden gescheiden door een teken (<, ≥, ≠) dat we een **relationele operator** noemen.

In BASIC kunnen we van de volgende relaties en relationele operatoren gebruik maken:

relatie	BASIC notatie	wiskundig symbool
is gelijk aan	=	=
is ongelijk aan	<>	≠
is groter dan	>	>
is kleiner dan	<	<
is groter dan of gelijk aan	>=	≥
is kleiner dan of gelijk aan	<=	≤

Er is echter een verschil tussen wiskunde en BASIC.

In de wiskunde gebruiken we in wiskundige relaties variabelen als x, y, x, a, b, enzovoorts. Bovendien hebben bepaalde variabelen in de wiskunde een bepaalde betekenis; zo worden x en y vaak voor de horizontale en verticale coördinaten van een punt in het platte vlak gebruikt, terwijl a, b en c meestal de betekenis van coëfficiënt hebben.

In BASIC programma's gebruiken we daarentegen vaak een naam voor een variabele die iets zegt over de 'inhoud van die variabele' zoals bij TELLER%, SOM en ANTW$. Bovendien geven we met % of $ het soort variabele aan.

In een wiskundige relatie vergelijken we meestal getalwaarden met elkaar. In een BASIC relatie kunnen we ook teksten met elkaar vergelijken (ANTW$ = "JA").

We geven nu een aantal BASIC relaties die als voorwaarde in een lus-opdracht (of in een IF-opdracht, zie volgende hoofdstuk) kunnen worden gebruikt. Steeds wordt getest of zo'n relatie of bewering waar (true) is of niet waar (false). We kunnen zo'n logische relatie daarom een 'waarheidsrelatie' noemen.

waarheidsrelatie	waar als
SOM = 40	de waarde van SOM veertig is
TELLER% <= 100	de waarde van TELLER% kleiner dan of gelijk aan honderd is
WRD$ <> "STOP"	de inhoud (waarde) van WRD$ niet de tekst STOP is
LEN(A$) > 15	de variabele A$ meer dan vijftien tekens bevat
SQR(X) - 4*Y < 0	de waarde van de rekenkundige uitdrukking $\sqrt{x} - 4y$ negatief is.

opdracht 3.14

Geef de BASIC-notatie voor de BASIC-relaties die 'waar' zijn als bovengenoemde relaties 'niet waar' zijn. (De relatie SOM = 40 is niet waar als de relatie SOM <> waar is!)

We besluiten dit hoofdstuk met een DOE-ZOLANG-lus met een waarheidsrelatie waarin twee tekstwaarden met elkaar vergeleken worden.

```
100 REM Alfabetische volgorde
110 CLS
120 PRINT "Mijn woord is automaat"
130 PRINT
140 PRINT "Toets een woord in dat"
144 PRINT "alfabetisch kleiner is"
150 PRINT
160 INPUT "Uw woord"; WRD$
165 '***doe zolang woord niet alfabe-
       tisch kleiner is dan automaat
170 IF WRD$ < "automaat" THEN 220
180   PRINT "Dit is niet kleiner!!!"
190   PRINT
200   INPUT "Ander woord"; WRD$
210 GOTO 170
215 '***
220 PRINT WRD$;" is inderdaad "
224 PRINT "kleiner dan automaat"
230 END
```

Laten we dit programma eens draaien:

```
Mijn woord is automaat

Toets een woord in dat
alfabetisch kleiner is

Uw woord? automatisch
Dit is niet kleiner!!!

Ander woord? autodeur
autodeur is inderdaad
kleiner dan automaat
Ok
```

De computer vergelijkt twee tekstwaarden met elkaar door de ASCII codes van de overeenkomstige tekens met elkaar te vergelijken (ASCII codes staan in Appendix D).

De letter A heeft ASCII code 65, de letter B de code 66; 65 < 66, dus de A komt voor de B.

De tekst "A1" levert de codes 65, 49; de code voor het cijfer 1 als teken is 49!
De tekst "A2" levert de codes 65, 50; de code voor het cijfer 2 als teken is 50!

```
65 = 65  en
49 < 50  dus  "A1" < "A2"
```

Nog wat voorbeelden:

```
KAM   komt voor  KAMER
KAR   komt voor  KAS
AAA   komt voor  AAAA
DUIKEN  komt voor  DUIVEN
          "K" < "V"
```

We hebben al gezien dat de computer teksten moet kunnen vergelijken om te kunnen nagaan of relaties als WRD$ < "automaat" waar zijn of niet waar zijn.

Een andere toepassing, waarbij het vergelijken van teksten noodzakelijk is, is een lijstje met namen door de computer in alfabetische volgorde laten zetten. We zullen in dit boek laten zien hoe dit 'sorteren' geprogrammeerd kan worden.

opdracht 3.15

Vul op de stippeltjes een relationele operator (>, <, <=, >= of <>) in waardoor de bewering waar is.

```
a.              4^2  .....  18
b.            "JAN"  .....  "JACOB"
c.            "A1A"  .....  "A1B"
d.         "IK LOOP"  .....  "IK ZING"
e.  "JAN DE NOOIJE"  .....  "JAN DEN OOIJE"
```

Pas op voor het programmeren van zogeheten 'eindloze' lussen. Dit zijn lussen waar het programma nooit meer uit komt. Een programma dat in een 'eindloze' lus zit zal doorgaans geen foutmelding geven. Betreft het ook nog een lus, waarin geen in- of uitvoeropdrachten staan, dan zien we als gebruiker niet eens dat er iets niet in orde is, Wel krijgen we natuurlijk argwaan wanneer door een opdracht na de lus iets op het scherm zou moeten worden afgedrukt en we zien maar niets! Maar als een ander dit programma zou draaien, krijgt die dan ook argwaan?

Een eindloze lus ontstaat als de bewering in de IF-opdracht in een lus altijd 'waar' (HERHAAL-TOTDAT) of altijd niet waar (DOE-ZOLANG) blijft; als het goed is gebeurt er op een bepaald moment in de lus iets waardoor de bewering 'niet waar' of 'waar' wordt.

Een voorbeeld van een eindloze lus, waarin overigens wel iets afgedrukt wordt, zien we hieronder.

```
10 REM demo eindloze lus
20 CLS : I=1
30 INPUT "Wat wordt N"; N
40 '===
50  PRINT I;I^2;I^3
60 IF I <> N THEN 50
70 '===herhaal tot I=N
80 END
```

Doordat we gekozen hebben voor een HERHAAL-TOTDAT-lus in plaats van een FOR-NEXT-lus (FOR I = 1 TO N) en doordat we in de HERHAAL-TOTDAT-lus iets vergeten zijn, zitten we (als we het programma zouden draaien) in een eindloze lus! Bij elke doorloop van de lus heeft I de waarde 1. We zien dus alleen maar regels met drie enen op het scherm (1, 1^2, 1^3). Dit gaat 'eeuwig' (eindeloos) door als voor N een waarde ongelijk aan 1 is ingetoetst. Breek het programma af met de CTRL/STOP-toets.

opdracht 3.16

a. Voeg aan het voorafgaande programma een opdracht toe (die kennelijk vergeten is), waardoor van de eindloze lus een eindige lus wordt gemaakt.

b. Wat gebeurt er als we voor N een negatieve waarde of nul intoetsen?

De BASIC programma's uit het vorige hoofdstuk bevatten opdrachten die gewoon de een na de ander moeten worden uitgevoerd. We spreken dan van een **sequentiële programmastructuur.**
In dit hoofdstuk hebben we kennis gemaakt met de **herhalingsstructuur in een programma.**
In het volgende hoofdstuk wordt de **keuzestructuur** (als dit ... dan dat ...) behandeld.

OEFENOPGAVEN HOOFDSTUK 3

De eerste 8 opgaven hebben voornamelijk betrekking op de FOR-NEXT-lus.

1. Welke waarden heeft de lusvariabele bij elke doorloop van de FOR-NEXT-lus?

   a. FOR A = 1 TO 5
   b. FOR B = 0 TO 6 STEP 2
   c. FOR TELLER = 10 TO 50 STEP 15
   d. FOR TELLER = -10 TO 10 STEP 10
   e. FOR X = 10 TO 0 STEP -1
   f. FOR H = 5 TO 3
   g. FOR I = 1 TO 1 STEP 0

2. Schrijf een FOR-opdracht voor een FOR-NEXT-lus waarmee de som van de oneven gehele getallen tussen 1 en N kan worden berekend.

3. Schrijf het eerste deel van een programma waarin het scherm eerst wordt schoongemaakt en waarin vervolgens 40 sterretjes (*) op de bovenste schermregel worden afgedrukt.
   Hoe krijgen we deze sterretjes op de 10e regel?

4. Schrijf een programma dat vraagt om het intoetsen van één van de letters a, b, c, d, e, ... of m en dat de tien letters uit het alfabet, volgend op de ingetoetste letter, afdrukt.
   Neem met het oog op opgave 10, regelnummers die steeds met 10 oplopen. Gebruik de CHR$- en ASC-functies. ASC doet het omgekeerde van CHR$. ASC("a") = 97, terwijl CHR$(97) = "a".

5. MSX-computers hebben een inwendige klok, die we in onze programma's kunnen gebruiken. Deze klok is de functie TIME. We kunnen deze klok zelf op nul zetten als we dat willen. We doen dat bijvoorbeeld zo:

   ```
   10 TIME = 0
   ```

   We kunnen ook de tijd laten afdrukken:

   ```
   20 PRINT TIME
   ```

   Deze klok loopt altijd en zal na het op nul zetten gewoon doorlopen. Aan het einde van een programma kunnen we dan laten afdrukken hoe lang het programma erover heeft gedaan.
   De klok wordt elke 1/50-ste seconde één verder gezet. Voor het netjes afdrukken van UREN, MINUTEN en SECONDEN moet er echter wel iets met TIME gedaan worden, want TIME telt

in 1/50-ste seconden. Een minuut is voor TIME dus 50 × 60 = 3000 tijdseenheden.
Schrijf een programma dat vraagt om het intoetsen van een getal dat aangeeft hoe lang de computer in zichzelf moet tellen. Na dit tellen moet hij afdrukken hoe lang hij over dit tellen gedaan heeft. De uitvoer moet er zo uitzien:

```
IK HEB TOT ... GETELD
DIT DUURDE ... MINUTEN
           ... SECONDEN
```

Het programma moet dit doen voor verschillende door ons op te geven getallen (lus).

6. Schrijf een programma dat in SCREEN 1 de kopbal uit het 'stuiterprogramma' tegelijkertijd in de kolommen 1, 3, 5, 7, .. t/m 27 naar beneden laat vallen en weer omhoog laat komen. Laat de kopballen steeds tot bovenaan stuiteren. Maak er een eindloze lus van zodat alleen het indrukken van CTRL/STOP het programma kan beëindigen.
Omdat de computer maar één opdracht tegelijk kan uitvoeren moet de eerste kopbal eerder vallen dan de laatste. Kunt u het zien?

7. Wat is er niet goed in het volgende programma?

```
10 FOR I=1 TO 10
20     PRINT I;
30     FOR K= 2 TO 5
40         PRINT I*K,I/K
50     NEXT I
60 NEXT K
70 END
```

8. Gebruik twee geneste FOR-NEXT-lussen en de VPOKE-opdracht om een programma te schrijven dat op de bovenste schermregel 40 keer de letter A POKEt, op de tweede regel 40 B's, op de derde regel 40 C's, ..., en op de vierentwintigste regel 40 X'en. Gebruik SCREEN 0.

9. Geef voor elk van de volgende beweringen de BASIC-notaties zoals die in een IF-opdracht, in een DOE-ZOLANG- of in een HERHAAL-TOTDAT-lus voor zouden kunnen komen.

   a. SOM heeft hoogstens de waarde 525
   b. VERSCHIL is negatief
   c. Voor WRD$ is de string "GADOOR" ingetoetst
   d. Het verschil tussen A en B is tenminste nul
   e. De wortel uit B^2-4AC is positief
   f. Het aantal gegooide ogen (met een dobbelsteen) is minstens 2.

10. In opgave 4 wordt gevraagd één van de letters a, b, c, ... of m in te toetsen. Voeg aan het programma een DOE-ZOLANG-opdracht toe waarmee wordt getest of niet één van de letters n, o, p, ... of z is ingetoetst. Het programma mag niet verder gaan voordat een van de bovengenoemde letters is ingetikt. Bij overtreding moet er worden afgedrukt: Niet boven een m!
Geef alleen de opdrachten die u wilt toevoegen. Neem aan dat de gebruiker inderdaad één van de letters a, b, ..., z intoetst en geen ander teken, zoals cijfers, leestekens of besturingstekens.

11. Breid het programma van opgave 4 (10) uit zodat de gebruiker meerdere malen een letter kan intoetsen. Neem aan dat de gebruiker tenminste één letter wil intoetsen (HERHAAL-TOT-DAT). De uitvoer moet er bijvoorbeeld zo uitzien:

```
Beginnen? (ja/nee)? ja
Wat is uw letter? a
De volgende 10 letters:
b c d e f g h i j k
Nog een keer? ja
Wat is uw letter? r
Niet boven een m!? w
Niet boven een m!? m
De volgende 10 letters:
n o p q r s t u v w
Nog een keer? nee
Einde programma
Ok
```

12. Schrijf de volgende DOE-ZOVEEL-KEER-lus als een DOE-ZOLANG-lus en als een HERHAAL-TOTDAT-lus die dezelfde uitvoer opleveren.

```
FOR I = 1 TO 10
    PRINT I;
NEXT I
```

13. Wat is er mis?

```
100 INPUT "Toets een woord in";W$
110 '***
120  IF W$="stop" THEN 190
130    FOR I=1 TO LEN(W$)
140       PRINT LEFT$(W$,I)
150    NEXT I
160    PRINT "Zo bouw je iets op"
170  GOTO 120
180 '***
190 PRINT "Ik stop ook"
200 END
```

14. Schrijf een programma dat voor bedragen tussen een bepaald begin- en eindbedrag afdrukt: bedrag, nettobedrag en BTW (19%). De uitvoer van dit programma ziet er bijvoorbeeld uit zoals hieronder is afgedrukt.

    Denk erom!, de bedragen zijn op centen afgerond. Aangenomen wordt dat begin- en eindbedragen en stapgrootte als gehele getallen ingevoerd worden.

```
Beginnen (j/n)? j

Beginbedrag? 20

Eindbedrag ? 60

Stapgrootte? 5


bedrag   netto bedrag   19%btw

 20         16.81         3.19
 25         21.01         3.99
 30         25.21         4.79
 35         29.41         5.59
 40         33.61         6.39
 45         37.82         7.18
 50         42.02         7.98
 55         46.22         8.78
 60         50.42         9.58

Nog een tabel? nee
Ok
```

15. Schrijf een programma dat in SCREEN 0 de omtrek van een vierkant tekent door met het cursorteken (code 219) de vier zijden te tekenen. Neem elke zijde 20 tekens lang. Binnen in het vierkant ontstaan dan 18 × 18 = 324 plaatsen om een teken af te drukken. Druk in dit veld van 324 posities net zolang willekeurige tekens af totdat in de linkerbovenhoek van het vierkant een ? wordt afgedrukt.

4 Keuzestructuur in een Programma

De programma's uit hoofdstuk 2 hebben alle een **sequentiële programmastructuur**, hetgeen wil zeggen dat de opdrachten in deze programma's gewoon achter elkaar, de een na de ander, worden uitgevoerd. Die sequentiële structuur hebben we zelf aangebracht; we hebben in de programma's namelijk alleen opdrachten gebruikt waarvan de computer weet dat na zo'n opdracht gewoon de daaropvolgende moet worden uitgevoerd.

In hoofdstuk 3 hebben we echter gezien hoe we door bepaalde constructies te gebruiken (**FOR-NEXT, DOE-ZOLANG** en **HERHAAL-TOTDAT**) programma's kunnen maken met naast de sequentiële structuur ook een **herhalingsstructuur**. We kunnen dus met bepaalde BASIC opdrachten (IF en GOTO) de programmabesturing, dat wil zeggen de volgorde waarin de opdrachten moeten worden uitgevoerd beïnvloeden.

In dit hoofdstuk bespreken we nog een belangrijke groep programmabesturingsopdrachten en wel de **keuzeopdrachten**. Een keuzeopdracht is zoiets als "ALS dit waar is DOE dat ..." of "ALS dit waar is DOE dat ... ANDERS DOE dat ...". Met een keuzeopdracht kunnen we afhankelijk van het al dan niet 'waar' zijn van een bewering de computer instrueren bepaalde opdrachten uit te voeren.

Deze keuzeopdracht zijn we al tegengekomen, het is de **IF-opdracht**. In het vorige hoofdstuk hebben we echter de IF-opdracht alleen gebruikt voor het programmeren van programmalussen. De vraag hierbij was steeds: "moet de lus nog een keer doorlopen worden of niet?".

We zullen in dit hoofdstuk een ander gebruik van de IF-opdracht laten zien en wel als er afhankelijk van een bepaalde situatie iets wel of niet gedaan moet worden of als er afhankelijk van een bepaalde situatie dit of dat gedaan moet worden. Hierbij gaat het niet om de vraag of programmalussen wel of niet doorlopen moeten worden. Toch komen we in de programma-voorbeelden wel lussen tegen met uiteraard de daarbij horende IF-opdrachten. Om op diverse plaatsen de rol van de IF-opdracht goed te kunnen onderscheiden zullen we spreken over lus-IFfen en echte-IFfen; we

hebben dus een echte-IF en een lus-IF. In dit hoofdstuk gaat het dus over de echte-IF alhoewel we de lus-IF ook gebruiken.

4.1 ALS ... DAN (IF ... THEN)

In het volgende programmaatje zien we de meest eenvoudige keuze-opdracht, de IF...THEN...opdracht (een echte-IF). Het programma vraagt steeds om het intoetsen van een getal (positief of negatief). Het programma bepaalt echter alleen de som van de positieve getallen. Als een nul wordt ingetoetst wordt het programma beëindigd na het afdrukken van de som van de ingetoetste positieve getallen.

```
10 SOM=0
20 INPUT "Uw eerste getal:"; GTAL
25 '***doe zolang G(e)TAL niet nul
30 IF GTAL=0 THEN 70
35 '
40   IF GTAL > 0 THEN SOM=SOM + GTAL
45 '
50   INPUT "Uw volgende getal:";GTAL
60 GOTO 30
65 '***
70 PRINT "De som van alle positieve";
         " getallen is"; SOM
80 END
```

We lichten de echte IF-opdracht er uit:

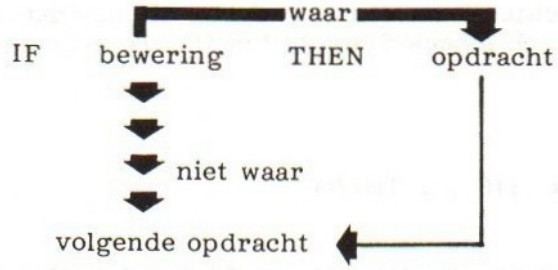
```
40 IF GTAL > 0 THEN SOM = SOM + GTAL
```

bewering — opdracht

Tussen IF en THEN staat een bewering en na THEN staat een BASIC opdracht. We lezen deze opdracht als volgt:

Als GTAL positief is tel dan dit GTAL bij SOM op.

De opdracht achter THEN wordt alleen uitgevoerd als de bewering waar (true) is. Is de bewering niet waar (false) dan wordt gewoon de opdracht na de IF-opdracht, regel 50 dus, uitgevoerd. We kunnen dit ook zo aangeven:



De computer heeft de keuze tussen het uitvoeren van de opdracht volgend op (onder) de IF-opdracht en het uitvoeren van de opdracht achter THEN.

opdracht 4.1

Geef bij elk van de volgende IF-opdrachten aan wanneer de computer de opdracht achter THEN uitvoert.

a. IF A$ <> "STOP" THEN ...
b. IF B-A < 0 THEN ...
c. IF LEN(WRD$) > 5 THEN ...
d. IF GTAL = INT(GTAL) THEN ...

Een lus-IF is heel gemakkelijk van een echte-IF te onderscheiden. In bovenstaand programma is:

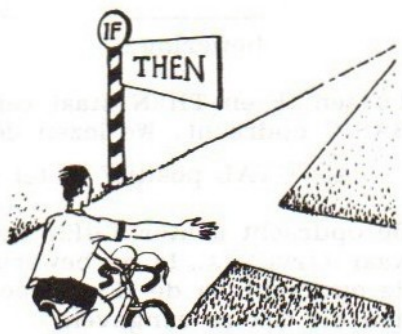
```
de lus-IF    :   IF GTAL = 0 THEN 70        èn
de echte-IF  :   IF GTAL > 0 THEN SOM = SOM + GTAL
```

Een lus-IF heeft dus altijd een regelverwijzing achter THEN, terwijl in een echte-IF altijd een BASIC-opdracht achter THEN staat.

In regel 40 van het programma op bladzijde 105:

```
40 IF GTAL > 0 THEN SOM = SOM + GTAL
```

staat achter THEN slechts één opdracht, namelijk de toekenningsopdracht SOM = SOM + GTAL. Vaak is het echter nodig **na THEN meer dan een opdracht** te laten uitvoeren. Deze opdrachten behoren nog wel tot de IF-opdracht, dat wil zeggen ze moeten onder hetzelfde regelnummer vallen als het regelnummer van de IF-opdracht. Hoe we dit voor elkaar krijgen zien we in de volgende paragraaf.



4.2 MEER DAN ÉÉN OPDRACHT ACHTER THEN

Stel dat we met het programma uit § 4.1 niet de som van de positieve getallen willen uitrekenen maar het gemiddelde van de ingetoetste positieve getallen (met negatieve getallen doen we in het programma voorlopig niets!). We zullen dan een TELLER moeten gebruiken die we bij een positief getal steeds met 1 ophogen. Als we namelijk weten hoeveel positieve getallen er ingetoetst zijn en we weten ook de som van die getallen dan kunnen we het gemiddelde van die getallen uitrekenen. De TELLER moet op nul beginnen. We zullen dus aan het begin van het programma de opdracht TELLER = 0 moeten opnemen en in de IF-opdracht, achter THEN, de opdracht TELLER = TELLER+1. Deze laatste opdracht moet dus in regel 40 worden opgenomen. Hier staat echter al een opdracht, namelijk SOM = SOM + GTAL.

De oplossing is: neem meer opdrachten in een regel op. Dit kan in BASIC door tussen twee opdrachten een dubbelepunt te zetten. Zo zouden we in regel 10 na de opdracht SOM = 0 ook nog de opdracht TELLER = 0 mogen opnemen. Regel 10 wordt dan:

```
10 SOM = 0 : TELLER = 0
```

Dit doen we ook in regel 40:

```
40     IF GTAL > 0 THEN SOM = SOM + GTAL :
                        TELLER = TELLER + 1
```

Nu worden de twee opdrachten na THEN beide uitgevoerd als het ingetoetste G(E)TAL positief is. Breng deze wijzigingen in het programma aan en verander regel 70 in:

```
70 PRINT "Het gemiddelde van "; TELLER,
         "positieve getallen is ";
          SOM/TELLER
```

opdracht 4.2

Wat zal er gebeuren als we na het intikken van run als eerste GTAL (sorry, GETAL) een nul intoetsen?

We hebben in het bovenstaande programma de twee opdrachten achter THEN op twee regels gezet, zodat ze precies onder elkaar op het scherm worden afgedrukt. We zullen in hoofdstuk 6 zien dat, als we een printer gebruiken, we de opdrachten rustig achter elkaar kunnen zetten. Op het scherm kunnen we dit ook doen, maar dan komt een deel van de tweede opdracht op de regel waar

THEN staat en een ander deel op de volgende regel tussen de regelnummers 40 en 50. Dit geeft een rommelige aanblik en daarom doen we het zoals hierboven is aangegeven.

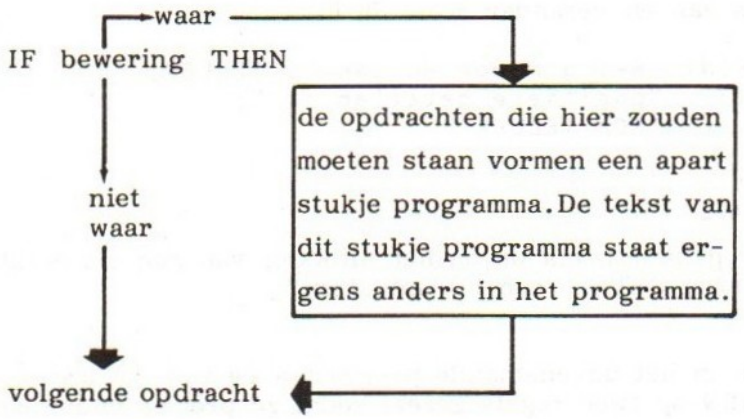
opdracht 4.3

In het programma uit § 4.1 moet bij het intoetsen van een negatief getal de tekst "MET NEGATIEVE GETALLEN DOE IK NIETS!" worden afgedrukt. Welke opdracht voegt u aan het programma toe?

We kunnen niet eeuwig blijven doorgaan met het toevoegen van dubbelepunten en nieuwe opdrachten achter THEN. In MSX BASIC mag een programmaregel uit maximaal 256 tekens bestaan. Voor een beeldscherm met 37 tekens per regel houdt dit in dat we elke opdracht, dus ook de IF-opdracht, over ruim zes regels mogen verdelen.

Zo erg is deze beperking van maximaal 256 tekens per programmaregel (per regelnummer dus) nu ook weer niet, want als er na THEN 'zoveel' moet worden gedaan, is datgene wat moet worden gedaan bijna altijd op te vatten als een apart stukje programma, anders gezegd "als een aantal bij elkaar horende opdrachten met een min of meer eigen taak", een deeltaak van de totale taak die het programma moet uitvoeren. Hieronder zien we deze situatie.

We hebben in zo'n situatie behoefte aan een constructie waarmee we in een programma **aparte stukjes programma** kunnen opnemen die we op een door ons gewenst moment kunnen laten uitvoeren.



In de bovenstaande IF-opdracht zal zo'n stukje programma alleen moeten worden uitgevoerd als de bewering die tussen IF en THEN staat waar is. Is deze bewering niet waar dan wordt gewoon de

opdracht met het eerstvolgende regelnummer uitgevoerd. Ook als de bewering waar is wordt uiteindelijk de opdracht volgend op de IF-opdracht uitgevoerd, alleen zijn dan wel eerst de opdrachten achter THEN uitgevoerd!

Vaak is een voorbeeld de beste uitleg. We zullen laten zien hoe we in BASIC zo'n apart stukje programma kunnen maken door in het 'positieve-gemiddelde'-programma de opdrachten achter IF GTAL > 0 THEN ... als apart stukje programma op te nemen.

Opmerking

In de komende hoofdstukken zult u nogal eens programma's tegenkomen met opdrachten (vooral IF-opdrachten) die meer dan één regel beslaan maar toch slechts één regelnummer bezitten. Als we dit soort lange opdrachten intikken, moet u erom denken de cursor naar een volgende regel te brengen zonder op RETURN te drukken; dat doet u immers pas als de hele opdracht (meerdere regels dus) ingetikt is!

4.3 IF ... THEN GOSUB ...

```
10 SOM=0 : TELER=0
20 INPUT "Uw eerste getal:"; GTAL
25 '***doe zolang G(e)TAL niet nul
30 IF GTAL=0 THEN 70
35 '
40   IF GTAL > 0 THEN GOSUB 1000
45 '
50   INPUT "Uw volgende getal:";GTAL
60 GOTO 30
65 '***
70 PRINT "Het gemiddelde van";TELLER,
         "positieve getallen is";
         SOM/TELLER
80 END
85 '
1000 'Subroutine positieve invoer
1005 '
1010   SOM=SOM + GTAL
1020   TELLER=TELLER + 1
1025 '
1030 RETURN
```

Het enige wat 'echt' nieuw is, is de GOSUB 1000-opdracht achter THEN, die in de plaats is gekomen van de twee opdrachten die er eerst stonden. Deze opdrachten zien we nu in de regels 1010 en 1020 staan. Nu zijn ze echter opgenomen als programmaregels in een apart stukje programma. Zo'n apart stukje programma noemen we een

SUBROUTINE

Als begin van de subroutine zien we regel 1000, een commentaar-regel. We hebben het stukje programma (de subroutine) de naam "positieve invoer" gegeven (in de REM-opdracht).

Regel 1030 sluit de subroutine af. De laatste opdracht in een subroutine is altijd de opdracht **RETURN**. Voor de duidelijkheid laten we de opdrachten in een subroutine iets inspringen. Deze RETURN heeft niets te maken met de RETURN-toets op het toetsenbord. Het is een BASIC-opdracht (net als PRINT, INPUT, enzovoorts) die als afsluiting van een subroutine dient; een programmaregel dus.

Hoe werkt zo'n subroutine nu? Kijk daarvoor eerst naar regel 40:

```
40    IF GTAL > 0 THEN GOSUB 1000
```

Als GTAL positief is, wordt de computer opgedragen verder te gaan met programmaregel 1000. Dit is de eerste regel uit de subroutine "positieve invoer".

GOSUB 1000	betekent het aanroepen van de subroutine die op regel 1000 begint

De programmabesturing gaat nu naar regel 1000.
Na regel 1000 worden de regels 1010 en 1020 uitgevoerd; daar is alles tenslotte om begonnen.
Na regel 1020 wordt regel 1030 uitgevoerd. Dit is de RETURN-opdracht. De computer krijgt opdracht terug te keren naar het punt vanwaar de subroutine werd aangeroepen.

RETURN	betekent keer terug naar de opdracht volgend op de GOSUB-opdracht waarmee de subroutine voor het laatst is aangeroepen

In bovenstaand voorbeeld betekent de RETURN-opdracht in regel 1030 dat de computer verder gaat met regel 50. Regel 50 is immers de opdracht volgend op de GOSUB-opdracht waarmee de routine voor het laatst is aangeroepen (dit was immers regel 40).

We zien dat de subroutine na de END-opdracht is opgenomen. De END-opdracht wil dus niet zeggen dat er geen programmaregels meer kunnen volgen, maar heeft alleen tot doel de computer te vertellen dat hij geen programma-opdrachten meer hoeft uit te voeren.

De regelnummers in subroutines zijn hoge nummers; dit komt omdat de subroutines als laatste onderdelen van een programma worden opgenomen. Daar we vaak van tevoren niet weten hoelang een programma wordt, is het gebruikelijk voor subroutines hoge regelnummers te gebruiken. We kunnen dan nog alle kanten op!

In het volgende hoofdstuk komen we uitgebreid op het gebruik van subroutines terug. We zullen zien dat subroutines onmisbaar zijn bij het 'netjes' programmeren.

opdracht 4.4

Het hierna volgende programma is een spelprogramma. De computer bedenkt een positief getal kleiner dan honderd en vraagt of u zijn geheime getal X wilt raden. Als u het geraden hebt drukt hij het aantal beurten af waarin u het geraden hebt. Hebt u het nog niet geraden, dan drukt hij af "hoger" of "lager". Dit afdrukken wordt geregeld in een subroutine. Jammer genoeg zijn bepaalde stukken programma weggevallen. Kunt u invullen wat er heeft gestaan?

```
100 REM getallen raden
105 '
110 INPUT "Klaar ? (ja/nee)"; .....
115 '***doe zolang speler wil spelen
120 IF ANTW$....."nee" THEN 250
130   X=INT(100*RND(1))
140   TELLER=.....
150   CLS
160   INPUT "Raad mijn getal"; GK
165   '***doe zolang niet geraden
170   IF .....=X THEN 210
180     TELLER=TELLER+1 : GOSUB.....
190     INPUT "Raad mijn getal"; GK
200   GOTO 170
205   '***
210   PRINT
220   PRINT "Goed geraden in";
            ........; "beurt(en)"
230   INPUT "Nog een keer"; .....
240 GOTO 120
245 '***
250 END
```

```
255 '
1000 'Subroutine spelaanwijzing
1005 '
1010   PRINT
1020   IF GK > X THEN PRINT "lager"
1030   IF GK...X THEN PRINT .......
1040   PRINT
1045 '
1050 RETURN
```

We zien in bovenstaand programma dat de GOSUB in regel 180 als gewone BASIC opdracht wordt gebruikt. Deze opmerking is bedoeld voor diegenen die wellicht de indruk krijgen dat GOSUB iets te maken heeft met de IF-opdracht, omdat we het idee van een subroutine aan de hand van de IF...THEN-opdracht geïntroduceerd hebben. GOSUB heeft echter niets te maken met de IF-opdracht. Het is gewoon een BASIC opdracht die we overal in een programma kunnen gebruiken. Alles over GOSUB en subroutines wordt in het volgende hoofdstuk behandeld.

In de volgende paragraaf zien we een uitbreiding van de IF-THEN constructie.

4.4 ALS ... DAN ... ANDERS ... (IF ... THEN ... ELSE ...)

Bij het programmeren komt het dikwijls voor dat in een programma iets gedaan moet worden als een bewering waar is en iets anders als dezelfde bewering niet waar is. We spreken in zo'n geval van een IF...THEN...ELSE-situatie. In de voorgaande paragrafen hadden we alleen de IF...THEN-situatie waarin alleen iets gedaan moest worden als een bewering waar was. Was de bewering niet waar dan werd gewoon de volgende opdracht uitgevoerd. In een IF...THEN...ELSE-situatie moet eerst gekozen worden uit twee mogelijke opdrachten, waarvan er één zal worden uitgevoerd, alvorens met de volgende opdracht verder wordt gegaan. Hieronder zien we een programmavoorbeeld met zo'n IF...THEN...ELSE-situatie. Als we een geheel getal intoetsen dan (THEN) drukt de computer af dat een geheel getal is ingetoetst. Zo niet (ELSE) dan drukt hij af dat het een gebroken getal was. In dit programma is de IF ... THEN ... ELSE-situatie geprogrammeerd met behulp van twee IF-THEN-opdrachten:

```
100 REM Gebroken of Geheel?
105 '
110 FOR I=1 TO 10
120  INPUT "Getal"; GTAL
130  IF GTAL=INT(GTAL)
      THEN PRINT GTAL;"is geheel"
140  IF GTAL<>INT(GTAL)
      THEN PRINT GTAL;"is gebroken"
150  PRINT
160 NEXT I
170 END
```

Dit programma vraagt precies tien keer om het intoetsen van een geheel of een gebroken getal. Met twee IF...THEN-opdrachten test het programma of het ingetoetste getal geheel dan wel gebroken is.

Als het ingetoetste getal een geheel getal is (dan is de bewering GTAL = INT(GTAL) waar), drukt het programma het ingetoetste getal af met daarachter de tekst "is geheel" (**THEN**). Is het getal niet geheel (GTAL <> INT(GTAL) is waar) dan drukt het programma het ingetoetste getal af met daarachter de tekst "is gebroken" (**ELSE**).

Als de bewering GTAL = INT(GTAL) waar is dan
is de bewering GTAL <> INT(GTAL) niet waar en omgekeerd!

Voor elk ingetoetst getal zal dus òf de PRINT-opdracht in regel 130 òf die in regel 140 worden uitgevoerd. Nooit zullen beide PRINT-opdrachten worden uitgevoerd, omdat beide beweringen nooit tegelijkertijd waar kunnen zijn.

Het zou mooi zijn als we de twee opdrachten:

```
130  IF GTAL=INT(GTAL)
      THEN PRINT GTAL;"is geheel"
140  IF GTAL<>INT(GTAL)
      THEN PRINT GTAL;"is gebroken"
```

door één IF-opdracht zouden kunnen vervangen. Dit kan ook en wel door:

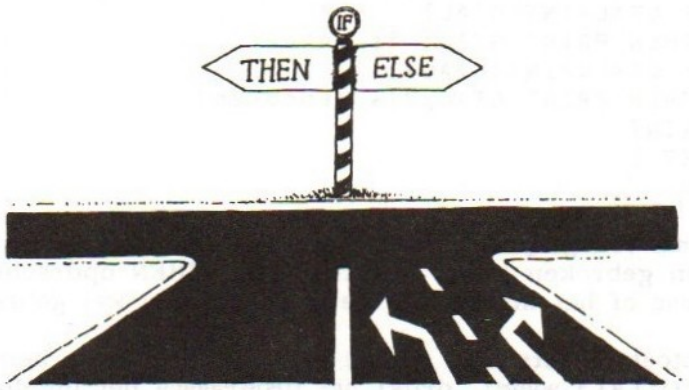
```
135  IF GTAL=INT(GTAL)
      THEN PRINT GTAL;"is geheel"
      ELSE PRINT GTAL;"is gebroken"
```

of door deze:

```
135  IF GTAL<>INT(GTAL)
       THEN PRINT GTAL;"is gebroken"
       ELSE PRINT GTAL;"is geheel"
```

Het is eigenlijk heel eenvoudig:

ALS bewering waar DAN dit ANDERS dat



opdracht 4.5

Bekijk het volgende programmafragment:

```
40 .....
50 IF A < B THEN PRINT A; "is kleiner dan";
                        B
60 IF A >= B THEN PRINT A; "is groter dan of";
                         " gelijk aan"; B
70 .....
```

Maak van deze twee IF-opdrachten een IF-THEN-ELSE-constructie.

In de programma's geven we, zoals u gemerkt hebt, de commentaarregels die eventueel weggelaten kunnen worden regelnummers die op een 5 eindigen. Bij opslag op magneetband of op diskette nemen deze commentaarregels, net zoals alle andere programmaregels, ruimte in beslag. Het valt echter aan te bevelen deze commentaarregels niet zonder meer weg te laten; zeker als een programma na verloop van tijd aangevuld of aangepast moet worden zijn ze erg handig.

Nog een voorbeeld van het gebruik van IF-THEN-ELSE:

```
100 REM Rekenles eerste klas
105 '
110 N=0
120 FOR SOMMETJE=1 TO 10
130     A%=INT(9*RND(1)+1)
140     B%=INT(9*RND(1)+1)
150     PRINT A%; "+"; B%; "=";:
        INPUT ANTW%
160     IF ANTW%=A% + B%
          THEN PRINT "Goed!" : N=N+1
          ELSE PRINT "FOUT!!"
170     PRINT "Druk op RETURN voor de"
            ;" volgende som";
180     INPUT A$
190 NEXT SOMMETJE
200 PRINT
210 PRINT "De score is"; N; "van de";
          " tien goed"
220 IF N <= 5
      THEN PRINT "Dit is onvoldoende"
      ELSE PRINT "Dit is voldoende"
230 END
```

De eersteklassertjes kunnen met dit programma tien sommetjes maken. In elk sommetje moeten twee (gehele) getallen onder de 10 worden opgeteld. Het programma houdt de score bij.

opdracht 4.6

Wat moeten we veranderen om optelsommen met getallen tussen de 10 en 20 te laten afdrukken? (10 mag wel, maar 20 mag niet voorkomen.)

Achter THEN en ELSE mogen we ook weer een IF-opdracht zetten. Dit komt voor als we een bewering die waar is (THEN) of niet waar is (ELSE) verder willen uitsplitsen. Stel dat we de eersteklassertjes wat meer willen aanmoedigen door niet te volstaan met de mededeling onvoldoende of voldoende, maar door, als zij acht of hoger scoren, de tekst "Dit is zeer goed!" af te drukken. Als zij 6 of 7 scoren dan "Dit is goed" en 5 of lager "Dit is onvoldoende". We krijgen dan:

```
Als score kleiner dan of gelijk aan 5
        Dan druk af "Dit is onvoldoende"
        Anders  Als de score kleiner is dan 8
                        Dan druk af "Dit is goed"
                        Anders druk af "Dit is zeer goed!"
```

In BASIC doen we dit als volgt:

```
220 IF N <= 5
      THEN PRINT "Dit is onvoldoende"
      ELSE IF N <= 7
             PRINT "Dit is goed"
             ELSE  "Dit is zeer goed"
```

Afhankelijk van de waarde van N wordt slechts één van de drie PRINT-opdrachten uitgevoerd.

Achter THEN en ELSE komt ook vaak een GOSUB-opdracht voor. In § 4.3 hebben we een programma gezien met een demonstratie van de IF...THEN GOSUB-constructie. We geven nu een ander programma voor het demonstreren van de constructie

```
IF .....
   THEN GOSUB .....
   ELSE GOSUB .....
```

Het nu volgende programma bevat twee subroutines (vierkanten en cirkels). In 'vierkanten' worden op het grafische scherm (SCREEN 2) 60 vierkanten getekend die allemaal een diagonaal hebben met een lengte tussen 10 en 85. Dit zien we in regel 1050 waar staat: 75*RND(1) + 10. Dit is een toevalsgetal tussen 10 en 85. Als we van een vierkant de lengte van de diagonaal delen door wortel 2 ($\sqrt{2}$) dan krijgen we de lengte van de zijde (zie SQR(2) in regel 1050). De variabele D geeft dus de lengte van een zijde van een vierkant. De opdracht K (van kleur) = INT(15*RND(1)) geeft één van de toevalsgetallen tussen 0,1,2,...14. Dit wordt de kleur van het vierkant dat door regel 1070 wordt getekend. We zorgen ervoor dat K niet 15 kan zijn, want dat is de kleur van de achtergrond (COLOR 1,15,15) en dat zie je toch niet. Om een mooi vierkant te krijgen moeten we de x-coördinaten delen door 1,4. Als we hier 37 bij optellen dan komt alles mooi in het midden. Dit wordt geregeld door de DEF FNX-functie in regel 110. De A$ = INPUT$(1)-opdracht in regel 1090 zorgt ervoor dat we even naar onze getekende 'Mondriaan' kunnen kijken.

Subroutine 'cirkels' tekent zo'n 30 cirkels met verschillende stralen en kleuren. Door de DOE-ZOLANG-lus in het 'hoofdprogramma' (regels 125-175) kunnen we net zolang kleuren tot we er genoeg van hebben.

```
100 REM vierkanten of cikels?
105 'Dit programma tekent een scherm
    vol vierkanten of cirkels. Maak
    uw keuze
110 DEF FNX(X)=INT(37+X/1.4+.5)
120 INPUT "Klaar(ja/nee)"; A$
125 '***doe zolang de gebruiker er
       niet genoeg van heeft
130 IF LEFT$(A$,1) <> "j" THEN 180
140   INPUT "v=vierkant,c=cirkel"; K$
150   IF K$="v"
        THEN GOSUB 1000
        ELSE GOSUB 2000
160   INPUT "Nog een keer(j/n)"; A$
170 GOTO 130
175 '***
180 COLOR 15,4,4 : SCREEN 0
190 KEY ON : END
195 '
1000 'Subroutine vierkanten
1005 '
1010  KEY OFF:COLOR 1,15,15:SCREEN 2
1020  FOR VIERKANT= 1 TO 60
1030  X=FNX(212*RND(1))
1040  Y=INT(132*RND(1))
1050  D=INT((75*RND(1)+10)/SQR(2))
1060  K=INT(15*RND(1))
1070  LINE(X,Y)-(X+D/1.4,Y+D),K,BF
1080  NEXT VIERKANT
1090  A$=INPUT$(1)
1095 '
1100 RETURN 'vierkanten
1105 '
2000 'Subroutine cirkels
2005 '
2010  KEY OFF:COLOR 1,15,15:SCREEN 2
2020  FOR CIRKEL= 1 TO 30
2030  X=FNX(215*RND(1))
2040  Y=INT(134*RND(1)+25)
2050  R=INT(45*RND(1)+15)
2060  K=INT(15*RND(1))
2070  CIRCLE (X,Y),R/1.4,K,,,1.4
2080  PAINT(X,Y),K
2090  NEXT CIRKEL
2100  A$=INPUT$(1)
2105 '
2110 RETURN 'cirkels
```

opdracht 4.7

Voeg aan het programma twee opdrachten toe waardoor vlak voor het tekenen van elk vierkant en elke cirkel een piep te horen is. (Kijk in uw MSX-handleiding of u zo'n opdracht kunt vinden.)

In de volgende paragraaf, waarin de meerkeuze-opdrachten aan bod komen, zullen we zien hoe we zelf door het indrukken van één van de functietoetsen het tekenen van de vierkanten en cirkels kunnen laten stoppen.

4.5 MEERKEUZESTRUCTUUR

Met 'meerkeuzestructuur' bedoelen we dat het programma, afhankelijk van de waarde van een variabele of afhankelijk van het indrukken van bepaalde toetsen of van het gebruik van een joystick, een bepaalde subroutine uitvoert. In MSX BASIC staat dit bekend als de ON...-opdrachten. Er zijn er heel wat: **ON ERROR, ON GOSUB, ON GOTO, ON INTERVAL, ON KEY, ON SPRITE, ON STOP** en **ON STRIG**. In uw MSX-handleiding vindt u deze opdrachten ook onder de titel 'onderbrekingen'. In dit boek zien we voorbeelden van **ON ERROR, ON GOSUB, ON KEY GOSUB, ON INTERVAL GOSUB** en **ON STRIG GOSUB**. Tot slot van dit hoofdstuk en eigenlijk een beetje als demonstratie van alles wat tot nu toe behandeld is geven we nog een programma met **ON STRIG GOSUB** voor het tekenen en berekenen van 8 × 8 **SPRITES**. Dit zijn figuurtjes van 8 bij 8 vierkantjes, die elk aan of uit kunnen zijn, waarmee de bekende 'computerspelletjes' werken. Eigenlijk zijn ON ... GOSUB, ON KEY GOSUB en ON STRIG GOSUB de enige 'echte' meerkeuze-opdrachten. De andere verwijzen altijd naar één subroutine (de interruptroutine), waarin een verdere afhandeling van de programma-onderbreking moet worden geprogrammeerd, en dit zijn de 'echte' programma-onderbrekingen.

ON ... GOSUB

Stel, dat we in het laatste programma uit de vorige paragraaf nog een subroutine 'driehoeken' (vanaf regel 3000) toevoegen. Om de gebruiker van het programma uit 'vierkanten', 'cirkels', driehoeken' en 'ophouden' te laten kiezen, drukken we een zogenaamd 'menu' op het scherm af:

```
100 REM vierkanten,cirkels en
        driehoeken
110 DEF FNX(X)=INT(37+X/1.4+.5)
120 KEY OFF: COLOR 1,15,15
130 SCREEN 0 : VERDER$="ja"
135 '***doe zolang verder=ja
140 IF VERDER$ = "nee" THEN 320
150   CLS
160   LOCATE 5,11
170   PRINT "1. Vierkanten"
180   LOCATE 5,13
190   PRINT "2. Cirkels    "
200   LOCATE 5,15
210   PRINT "3. Driehoeken"
220   LOCATE 5,17
230   PRINT "4. Ophouden   "
240   LOCATE 5,19
250   INPUT "Uw keuze: "; K
255   '***doe zolang k<1 of k>4
260   IF K>=1 OR K<=4 THEN 300
270     LOCATE 5,19
280     INPUT "Foute keuze,opnieuw";K
290   GOTO 260
295   '***
300   IF K=4
       THEN VERDER$="nee"
       ELSE ON K GOSUB 1000,2000,3000
310 GOTO 140
315 '***
320 COLOR 15,4,4 : SCREEN0
330 KEY ON : END
```

Zie voor het gebruik van OR (regel 260) paragraaf 4.6. Afhankelijk van of er een 1, 2, 3 of 4 ingetoetst wordt zal een van de subroutines 1000 (1), 2000 (2) of 3000 (3) aangeroepen worden, of zal het programma beëindigd worden (4). Bij een niet toegestane keuze moet de gebruiker het opnieuw proberen. Het is bij ON ... GOSUB opdrachten belangrijk dat de variabele tussen ON en GOSUB steeds één van de waarden 1, 2, 3, of ... zoveel als er subroutines zijn, kan hebben. Tussen ON en GOSUB mag ook een rekenkundige uitdrukking (zoiets als 2*K+1) staan. De subroutines in de lijst achter GOSUB hoeven niet per se met 1000, 2000, 3000, ... aangegeven te worden. Er zou ook mogen staan 1500, 2501, 4650, ... Het gaat dus om de volgorde in de lijst en niet om het regelnummer, waar de subroutine begint.

Voor diegenen die willen weten hoe de subroutine 'driehoeken' er uitziet drukken we hem nu af:

```
3000 'Subroutine driehoeken
3005 '
3010  KEY OFF:COLOR 1,15,15:SCREEN 2
3020  FOR DRIEHOEK= 1 TO 45
3030  X=FNX(212*RND(1))
3040  Y=INT(132*RND(1))
3050  H=INT((75*RND(1)+10)/SQR(2))
3060  K=INT(15*RND(1))
3070  LINE(X,Y+H)-(X+H/1.4,Y),K
3080  LINE-(X+2*H/1.4,Y+H),K
3090  LINE-(X,Y+H),K
3100  PAINT(X+H/1.4,Y+2*H/3),K
3110  NEXT DRIEHOEK
3120  A$=INPUT$(1)
3125 '
3130 RETURN 'driehoeken
```

ON KEY GOSUB

In het programma in 4.7 zien we een voorbeeld van het gebruik van ON KEY GOSUB. U vindt in de toelichting bij dit programma de nodige uitleg over deze opdracht.

4.6 LOGISCHE RELATIES EN WAARHEIDSVARIABELEN

De logische relaties die we tot nu toe in IF-opdrachten hebben gebruikt, zijn zogenaamde enkelvoudige logische uitdrukkingen zoals

```
CEL = BOVEN
A$ = " "
W$ <> "stop"
GTAL >= 0
```

Deze enkelvoudige logische uitdrukkingen bevatten één van de 'vergelijkende operatoren' =, >, <, <=, >= of <>.

We kunnen echter nog andere logische uitdrukkingen maken door van twee of meer enkelvoudige uitdrukkingen, met behulp van de zogeheten 'logische operatoren' AND, OR, NOT, XOR, EQU en IMP, samengestelde logische uitdrukkingen te maken. Wij zullen alleen voorbeelden met AND, OR en NOT laten zien.
In de linkerkolom zien we de samengestelde logische uitdrukking in 'gewoon' Nederlands, rechts in BASIC.

Samengestelde logische uitdrukkingen (beweringen)

ALS GTAL groter is dan 0 èn kleiner dan 25 DAN ...	IF GTAL > 0 AND GTAL < 25 THEN
ALS W$ de tekst STOP bevat òf de tekst stop DAN	IF W$ = "STOP" OR W$ = "stop" THEN
ALS TELLER ongelijk nul of A$ leeg is	IF TELLER <> 0 OR A$ = ""

We kunnen nog verder gaan:

'Als K tussen 10 en 20 of tussen 30 en 40 ligt dan ...' schrijven we als:

```
IF (K > 10 AND K < 20) OR (K > 30 AND K < 40) THEN ...
```

We gebruiken haakjes om de samengestelde uitdrukkingen opnieuw samen te stellen:

'Als K negatief is of als K tussen 100 en 1000 ligt' wordt:

```
IF (K < 0) OR (K > 100 AND K < 1000) THEN ...
```

In veel gevallen mogen we de haakjes weglaten, maar het is verstandiger ze wel te gebruiken; je weet maar nooit!

In het programma dat dit hoofdstuk afsluit zien we het gebruik van zo'n samengestelde logische uitdrukking in bijvoorbeeld de regels 11070 t/m 11100 en in regel 30060.

opdracht 4.8

Geef aan wanneer de volgende beweringen (logische expressies) waar zijn.

a. SOM > 0
b. A-B <= 5
c. A >= 3 AND B < 0
d. RENTE < 7 AND BEDRAG >= 1000
e. SOM < 25 AND SOM >= 32
f. TELLER >= 1 OR TELLER <= 5
g. TELLER < 10 OR TELLER > 20
h. ANTW$ <> "STOP" AND TELLER <= 50
j. ANTW$ = "JA" OR KEUS$ <> "STOP"

Tot slot behandelen we een speciaal gebruik van een numerieke variabele. We kunnen in een programma een numerieke variabele

slechts twee waarden toekennen, namelijk de waarde 0 of een waarde ongelijk 0. Als de variabele de waarde nul krijgt zeggen we de variabele is 'false'. Krijgt zo'n variabele een waarde ongelijk nul dan heet de variabele 'true'. We zouden dergelijke variabelen waarheidsvariabelen of logische variabelen kunnen noemen. Deze variabelen hebben de waarde 'waar' (true) of de waarde 'niet waar' (false).

Deze waarheidsvariabelen kunnen we in IF-opdrachten gebruiken en we kunnen er met AND, OR en NOT logische uitdrukkingen mee maken.
Kijk eens naar het volgende programma. Het is een programma uit opgave 13 van hoofdstuk 3, maar dan met een HERHAAL-TOTDAT-lus in plaats van een DOE-ZOLANG-lus:

```
100 '===
110    INPUT "Toets een woord in";W$
120    FOR I=1 TO LEN(W$)
130       PRINT LEFT$(W$,I)
140    NEXT I
150    PRINT "Zo bouw je iets op"
160 IF W$<>"stop" THEN 110
170 '===herhaal totdat w$ = stop
180 END
```

De IF-opdracht in regel 160 bevat de enkelvoudige logische uitdrukking W$<>"stop".
We kunnen dit programma ook programmeren door op de plaats van W$<>"stop" een waarheidsvariabele te zetten.
Bekijk het gewijzigde programma:

```
100 TERUG=-1
110 '===
120    INPUT "Toets een woord in";W$
130    IF W$="stop" THEN TERUG=0
140    FOR I=1 TO LEN(W$)
150       PRINT LEFT$(W$,I)
160    NEXT I
170    PRINT "Zo bouw je iets op"
180 IF TERUG THEN 120
190 '===herhaal totdat terug is false
200 END
```

Nu zien we in regel 180 (de voormalige regel 160) de opdracht:

```
180 IF TERUG THEN 120
```

De variabele TERUG is in dit programma gebruikt als waarheidsvariabele. In regel 100 maken we TERUG waar (true) door de waarde -1 eraan toe te voegen. In feite mogen we elke waarde,

ongelijk nul, nemen, maar omdat BASIC zelf altijd '-1' als 'waar' neemt hebben wij dat ook gedaan.

Als in regel 120 voor W$ de string 'stop' wordt ingetoetst maken we de variabele TERUG 'niet-waar' of false door de waarde nul eraan toe te kennen. De IF TERUG THEN 120 opdracht moeten we lezen als:

> Als TERUG 'waar' is ga dan verder met regel 120, zo niet dan gaan we door met de volgende opdracht (regel 190)

Als TERUG waar is dan is 'niet-TERUG' niet waar en als TERUG niet waar is dan is niet-TERUG waar. We kunnen in een IF-opdracht ook deze NIET- (of liever NOT-) waarheidsvariabelen gebruiken. Bekijken we het vorige programma, zoals we dat in hoofdstuk 3 (opgave 13) met een DOE-ZOLANG-lus hebben geprogrammeerd dan zien we

```
100 INPUT "Toets een woord in";W$
110 '***
120  IF W$="stop" THEN 200
130    FOR I=1 TO LEN(W$)
140       PRINT LEFT$(W$,I)
150    NEXT I
160    PRINT "Zo bouw je iets op"
170    INPUT "Volgende woord"; W$
180  GOTO 120
190 '***
200 PRINT "Ik stop ook"
210 END
```

We kunnen dit programma ook programmeren met de waarheidsvariabele VERDER in combinatie met de logische operator NOT. We geven eerst het gewijzigde programma:

```
100 VERDER=-1 ' maak verder true
110 INPUT "Toets een woord in";W$
120 IF W$="stop" THEN VERDER=0
130 '***doe zolang verder true is
140  IF NOT VERDER THEN 230
150    FOR I=1 TO LEN(W$)
160       PRINT LEFT$(W$,I)
170    NEXT I
180    PRINT "Zo bouw je iets op"
190    INPUT "Volgende woord";W$
200    IF W$="stop" THEN VERDER=0
210  GOTO 140
220 '***
230 PRINT "Ik stop ook"
240 END
```

We beginnen met VERDER = waar (regel 100). Als 'stop' is ingetoetst maken we VERDER 'niet-waar' (regel 120 of 200). In regel 140 testen we of NIET VERDER waar is. Is dit zo, dan wordt de lus beëindigd door met regel 230 verder te gaan. Is NIET VERDER 'niet-waar', dan wordt de lus uitgevoerd.
Nu is NIET VERDER waar als VERDER niet-waar is en NIET VERDER is niet-waar als VERDER waar is.
Eigenlijk kunnen we ook deze DOE-ZOLANG-versie zonder 'NOT' programmeren. Het is alleen zo dat in sommige situaties het 'logischer' is om NOT te gebruiken en in andere om NOT niet te gebruiken. Bij een DOE-ZOLANG-lus is het logisch om te zeggen IF NOT VERDER ... omdat deze test aan het begin van de lus wordt uitgevoerd, terwijl bij een HERHAAL-TOTDAT-lus het logischer is om te testen IF TERUG THEN, omdat deze test aan het einde van de lus plaatsvindt.

Het gebruik van deze logische variabelen en het samenstellen van logische uitdrukkingen met logische variabelen en de operatoren AND, OR, NOT, EQU, enzovoorts, is een moeilijk terrein en we zullen dit terrein niet verder dan de gegeven voorbeelden betreden. In het navolgende voorbeeldprogramma zien we het gebruik van logische variabelen in regel 11120 (JOYST) en in regel 240 (NOT KLAAR). Het systeem gebruikt voor 'waar' (true) altijd de waarde -1, vandaar dat wij dat ook doen (regels 11040 en 60010), en voor 'niet waar' (false) de waarde 0.

opdracht 4.9

Geef in elk van de volgende gevallen aan wanneer de subroutine 1000 zal worden uitgevoerd.

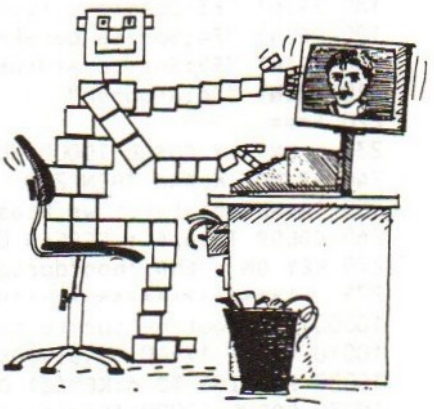
a. IF A THEN GOSUB 1000
b. IF RENTE - 4 THEN GOSUB 1000
c. IF RENTE AND SALDO THEN GOSUB 1000
d. IF NOT VERDER THEN GOSUB 1000
e. IF NOT (A AND B) THEN GOSUB 1000 (dit is een hele moeilijke).

4.7 PROGRAMMAVOORBEELD TER ILLUSTRATIE VAN DE TOT NU TOE BEHANDELDE ONDERWERPEN

Als afsluiting van het eerste deel van dit boek, waarin alle programmastructuren behandeld zijn, geven we een vrij groot programma waarmee u op het scherm 8 × 8 sprites kunt ontwerpen, wijzigen, berekenen, en op een printer kunt afdrukken. Als u de werking ervan door hebt kunt u het aanpassen om er 16 × 16 sprites mee te ontwerpen.

Een sprite is een grafisch figuurtje dat gebruikt kan worden in animaties. In MSX BASIC kunnen we maximaal 256 sprites tegelijkertijd op het scherm laten bewegen. Het programma is eigenlijk om u te laten zien hoe de tot nu toe behandelde onderwerpen in een groter programma gebruikt worden. Hoe een dergelijk groter programma op een verantwoorde en logische manier ontworpen kan worden behandelen we in het volgende hoofdstuk. Niets mag u er echter van weerhouden dit programma in te toetsen en er mee te experimenteren. Sla het dan wel eerst, direct na het intoetsen, op cassette of op diskette op (zie hoofdstuk 9), voor het geval dat er iets tijdens het experimenteren fout gaat.

Het programma gaat uit van het gebruik van een JOYSTICK met 'vuurknop', die is aangesloten op de eerste JOYSTICK-ingang aan de zijkant of voorkant van het toetsenbord van uw MSX-computer. Hebt u geen JOYSTICK dan moet u de volgende wijzigingen aanbrengen:



vervang regel 11010 door:

```
11010 STRIG(0) ON
```

vervang regel 11030 door:

```
11030 ON STRIG GOSUB 14000
```

vervang regel 11060 door:

```
11060 D = STICK(0)
```

voeg toe regel 11115:

```
11115 A$ = INKEY$: IF A$ = "s" OR A$ = "S"
                   THEN GOSUB 15000
```

vervang regel 11140 door:

```
11140 STRIG(0) OFF
```

Hierdoor wordt de joystick vervangen door de vier cursortoetsen en de vuurknop van de joystick door de S-toets. Hebt u een joystick met twee 'vuurknoppen', zoek dan even uit op welke knop het programma reageert.
Hebt u een MSX-printer, vervang dan in subroutine 50000 (sprite afdrukken) regel 50060 door:

```
50060 LPRINT CHR$(CODE)
```

en verwijder de regels 50015 en 50120.

Regel 50015 zorgt ervoor dat bij een EPSON-printer de regelafstand zo wordt ingesteld dat twee cursorblokjes zonder tussenruimte afgedrukt worden. Deze regelafstand is 24/216 inch. Wellicht kunt u op us MSX-printer dit ook doen. Kijk dan in de handleiding van uw printer bij ESCape codes en Line Spacing.
Regel 50120 zet de EPSON-printer weer op de standaardregelafstand.

Nu volgt het programma met daarna een korte toelichting.

```
100 'Sprites maken, wijzigen, berekenen en afdrukken
110 WIDTH 40 : KEY OFF : COLOR 1,15,15 : SCREEN 0
120 '*********************************
130 'Hoofdprogramma
140 FOR I=1 TO 6 :KEY(I) ON: NEXT I
150 LOCATE 5,18,0
160 PRINT "F1:Sprite maken     " : LOCATE 5,19
170 PRINT "F2:Sprite wijzigen  " : LOCATE 5,20
180 PRINT "F3:Sprite uitvullen " : LOCATE 5,21
190 PRINT "F4:Sprite berekenen " : LOCATE 5,22
200 PRINT "F5:Sprite afdrukken " : LOCATE 5,23
210 PRINT "F6:Stoppen"
220 '===
230   ON KEY GOSUB 10000,20000,30000,40000,50000,60000
240 IF NOT KLAAR THEN 230
250 '===herhaal tot we klaar zijn
260 COLOR 15,4,4 : SCREEN 0
270 KEY ON : END 'hoofdprogramma
275   '*********************************
10000 'Subroutine sprite maken
10010 VPOKE 17*40+1,45:VPOKE 17*40+2,45:VPOKE 17*40+3,62
10020 FOR I=1 TO 6:KEY(I) OFF :NEXT
10030 GOSUB 12000 'Spriteveld maken
10040 GOSUB 11000 'Joystick
10050 VPOKE 17*40+1,32:VPOKE 17*40+2,32:VPOKE 17*40+3,32
10060 FOR I=1 TO 6:KEY(I) ON: NEXT
10070 RETURN 'sprite maken
10075 '*********************************
11000 'Subroutine joystick
11010 STRIG(0) ON: STRIG(1) ON
11020 LOCATE 5,1,0:PRINT "Spatiebalk stopt het tekenen"
11030 ON STRIG GOSUB 14000,15000
11040 K=15 : R=7 : LOCATE K,R,1 : JOYST=-1
11050 '===
11060 D=STICK(1)
11070 IF D=1 AND R>7  THEN LOCATE K,R-1,1:R=R-1
11080 IF D=3 AND K<22 THEN LOCATE K+1,R,1:K=K+1
11090 IF D=5 AND R<14 THEN LOCATE K,R+1,1:R=R+1
11100 IF D=7 AND K>15 THEN LOCATE K-1,R,1:K=K-1
```

```
11110 FOR I=1 TO 80 : NEXT I
11120 IF JOYST THEN 11060
11125 '===herhaal tot joyst door spatiebalk afgezet wordt
11130 LOCATE 5,1,0:PRINT SPC(30)
11140 STRIG(0) OFF:STRIG(1) OFF
11150 RETURN 'joystick
11155 '*******************************
12000 'Subroutine spriteveld maken
12010 FOR I=0 TO 7
12020  FOR J=0 TO 14
12030    ADRES=280+I*40+15+J
12040    IF J<8 THEN VPOKE ADRES,45 ELSE VPOKE ADRES,32
12050  NEXT J
12060 NEXT I
12070 RETURN 'spriteveld maken
12080 '*******************************
13000 'Subroutine spriteveld herstellen
13010 FOR I=0 TO 7
13020  FOR J=0 TO 14
13030    ADRES=280+I*40+15+J
13040    IF J>7 THEN VPOKE ADRES,32 ELSE IF VPEEK(ADRES)=219
             THEN VPOKE ADRES,42 ELSE VPOKE(ADRES),45
13050  NEXT J
13060 NEXT I
13070 RETURN 'spriteveld herstellen
13075 '*******************************
14000 'Subroutine klaar met tekenen
14010   JOYST=0 'joyst wordt false
14020 RETURN
14025 '*******************************
15000 'Subroutie sprite ontwerpen
15010 LOCATE K,R,0
15020 IF VPEEK(R*40+K)=45 THEN PRINT "*" : LOCATE K,R,1
                        ELSE PRINT "-" : LOCATE K,R,1
15030 RETURN 'sprite ontwerpen
15035 '*******************************
20000 'Subroutine sprite wijzigen
20010 VPOKE 18*40+1,45:VPOKE 18*40+2,45:VPOKE 18*40+3,62
20020 FOR I=1 TO 6:KEY(I) OFF :NEXT
20030 GOSUB 13000 'spriteveld herstellenOk
20040 GOSUB 11000 'Joystick
20050 VPOKE 18*40+1,32:VPOKE 18*40+2,32:VPOKE 18*40+3,32
20060 FOR I=1 TO 6:KEY(I) ON: NEXT
20070 RETURN 'sprite wijzigen
20075 '*******************************
```

```
30000 'Subroutine sprite uitvullen
30010 VPOKE 19*40+1,45:VPOKE 19*40+2,45:VPOKE 19*40+3,62
30020 FOR I=1 TO 6:KEY(I) OFF :NEXT
30030 FOR I= 0 TO 7
30040  FOR J=0 TO 7
30050     ADRES=280+I*40+15+J
30060     IF VPEEK(ADRES)=219 OR VPEEK(ADRES)=42
            THEN VPOKE ADRES,219 ELSE VPOKE ADRES,32
30070  NEXT J
30080 NEXT I
30090 VPOKE 19*40+1,32:VPOKE 19*40+2,32:VPOKE 19*40+3,32
30100 FOR I=1 TO 6:KEY(I) ON: NEXT
30110 RETURN ' sprite uitvullen
30115 '*******************************
40000 'Subroutine sprite berekenen
40010 VPOKE 20*40+1,45:VPOKE 20*40+2,45:VPOKE 20*40+3,62
40020 FOR I=1 TO 6:KEY(I) OFF :NEXT
40030 FOR I=0 TO 7
40040  SPCODE=0
40050  FOR J=0 TO 7
40060     ADRES=280+I*40+15+J
40070     IF VPEEK(ADRES)=219 OR VPEEK(ADRES)=42
            THEN SPCODE=SPCODE+2^(7-J)
40080  NEXT J
40090  LOCATE 25,7+I,0: PRINT USING " ###"; SPCODE
40100 NEXT I
40110 VPOKE 20*40+1,32:VPOKE 20*40+2,32:VPOKE 20*40+3,32
40120 FOR I=1 TO 6:KEY(I) ON: NEXT
40130 RETURN 'sprite berekenen
40135 '*******************************
50000 'Subroutine sprite afdrukken
50010 VPOKE 21*40+1,45:VPOKE 21*40+2,45:VPOKE 21*40+3,62
50015 LPRINT CHR$(27);"3";CHR$(24);
50020 FOR I=1 TO 6:KEY(I) OFF :NEXT
50030 FOR I= 0 TO 7
50040  FOR J=0 TO 14
50050    ADRES=280+I*40+15+J:CODE=VPEEK(ADRES)
50060    IF CODE=219 THEN LPRINT CHR$(27);"m";CHR$(4);CHR$(140);
                     ELSE LPRINT CHR$(CODE);
50070  NEXT J
50080  LPRINT
50090 NEXT I
50100 VPOKE 21*40+1,32:VPOKE 21*40+2,32:VPOKE 21*40+3,32
50110 FOR I=1 TO 6:KEY(I) ON: NEXT
50120 LPRINT CHR$(27);"2";
50130 RETURN 'sprite afdrukken
50135 '*******************************
60000 'Subroutine stoppen
60010   KLAAR=-1 ' klaar wordt true
60020 RETURN
```

toelichting:

regel(s)	uitleg
110	Schermbreedte 40 tekens; functietoetsen uit; wit scherm (15) met zwarte afdrukkleur (1); scherm 0.
140	Lus voor het 'stand-by' maken van de functietoetsen F1 t/m F6
150	Zet cursor op 19-de rij, 6e positie en zet hem uit (0).
160-210	Druk keuzemenu op de 19e t/m 24ste regel af.
220-250	HERHAAL-TOTDAT-lus voor het reageren (ON KEY) op het indrukken van een van de functietoetsen F1 t/m F6. Bij het indrukken van F6 wordt (in subroutine 60000) de waarde van KLAAR gelijk aan -1 (true) waardoor NOT KLAAR 0 wordt (false) en de lus beëindigd wordt.
260-270	Terug naar blauw, wit scherm; functietoetsen op onderste regel afdrukken; programma beëindigen.
10000-10070	Hoofdsubroutine voor het maken van een sprite.
10010	(ook 20010, 30010, 40010, 50010) Zet een pijltje (-->, codes 45, 45 en 62) voor de gekozen functietoets in het menu.
10020	(ook 20020, 30020, 40020, 50020) Zet het reageren op functietoetsen tijdelijk af, zodat op het indrukken van één van deze toetsen tijdens het uitvoeren van de subroutine niet gereageerd wordt.
10030	Aanroepen van een subroutine voor het tekenen van 8 × 8 veld.
10040	Aanroepen van subroutine voor het activeren van de joystick (of van de cursortoetsen, zie vóór het programma).
10050	Uitwissen van de pijl (-->) met spaties (code 32) voor het menu.
10060	Weer activeren van functietoetsdetectie.
10070	Terugkeer naar hoofdprogramma (regel 240).
11000-11150	Subroutine voor het vullen van het 8 × 8 veld met *'tjes om zo een sprite te ontwerpen.

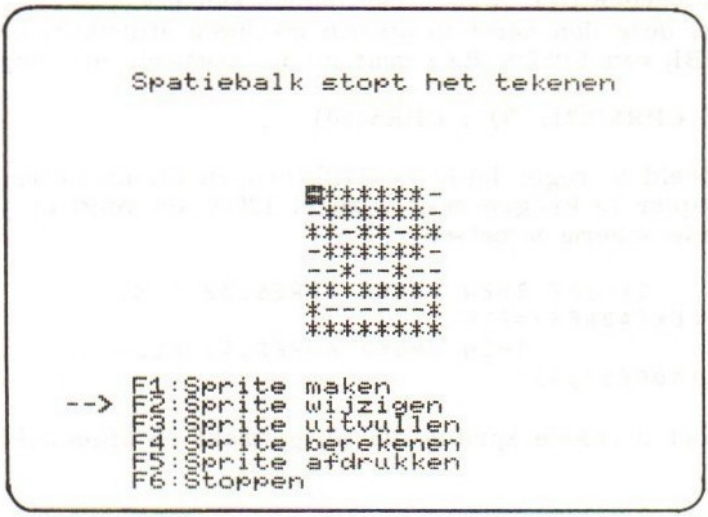
regel(s)	uitleg
11010	Activeren van spatiebalkdetectie (STRIG(0)) en van de vuurknop van de joystick (STRIG(1)). Dit is de 'eerste' vuurknop in geval uw joystick twee vuurknoppen heeft.
11020	Drukt mededeling bovenaan het scherm af.
11030	Als de spatiebalk wordt ingedrukt zal het programma naar subroutine 14000 gaan. Als de vuurknop van de joystick wordt ingedrukt dan naar subroutine 15000
11040	Zet de cursor 'aan' links bovenin het 8 × 8 veld; maakt de waarheidsvariabele JOYST true (-1).
11050-11120	Lus voor het reageren op de bewegingen van de joystick tot JOYST false wordt. Dit gebeurt als de spatiebalk wordt ingedrukt; immers dan wordt subroutine 14000 uitgevoerd.
11060	D krijgt een cijfer voor de beweging van de joystick. D = 1 naar boven, D = 3 naar rechts, D = 5 naar beneden en D = 7 naar links.
11070-11100	Als de cursor niet buiten het veld raakt dan wordt de cursor één naar boven (R = R-1), één naar rechts (K = K+1), één naar beneden (R = R+1) of één naar links (K = K-1) gezet. Als de cursor al aan de rand staat dan kan hij hierdoor niet uit het veld gebracht worden.
11110	Wachtlus voor het synchroniseren van de joystickbewegingen.
11120	Als JOYST waar dan lus nogmaals uitvoeren.
11130	Verwijder mededeling bovenaan het scherm.
11140	Zet spatiebalk en vuurknopdetectie af.
12000-12070	Tekent een spriteveld van 8 × 8 streepjes (-), met code 45 vanaf de 8e regel, 16e positie, tot en met 14e regel, 23e positie. Het adres van de linkerbovenhoek van het veld is in SCREEN 0 (BASE(0)) in VRAM het adres 280+15. Rechts naast het veld (I = 8 tot 14) worden spaties (32) afgedrukt om een eventuele berekening, die door subroutine 40000 daar is afgedrukt, te verwijderen.
13000-13070	Herstelt een eerder getekende sprite die door subroutine 30000 'uitgevuld' is.

regel(s)	uitleg
13040	Bevat een 'geneste' IF-THEN-ELSE, die ervoor zorgt dat als J > 7 een spatie wordt gezet (veegt alles rechts van het spriteveld uit) en als J <= 7 (ELSE) overal waar een ▮ (code 219) staat weer een * (code 42) wordt gezet en anders een streepje (-, code 45).
14000-14020	Deze subroutine wordt aangeroepen als bij het tekenen of wijzigen (F1 of F2) van een sprite op de spatiebalk gedrukt wordt.
14010	Maakt de waarheidsvariabele JOYST false (0).
15000-15030	Wordt aangeroepen als de vuurknop van de joystick ingedrukt wordt.
15010	Zet de cursor op positie K+1 van rij R+1 uit.
15020	Zet een * als op die positie een - staat en een - als er een * staat. Deze subroutine vult of verwijdert een hokje in het 8 × 8 spriteveld.
20000-20070	Deze subroutine wordt aangeroepen als functietoets F2 wordt ingedrukt.
20030	Roept de subroutine 13000 (zie hierboven) aan waardoor het oorspronkelijke ontwerp (met *'tjes) weer terugkomt en een eventuele spriteberekening (zie subroutine 40000) wordt uitgeveegd.
20040	Zie bij 10040.
30000-30110	Deze subroutine wordt met F3 aangeroepen en zet de met *'tjes getekende sprite (subroutine 10000) of gewijzigde sprite (subroutine 20000) om in een 'echte' sprite door de -'tjes te vervangen door spaties en de *'tjes te vervangen door ▮-jes. Dit is het cursorteken (code 219). Een sprite wordt namelijk uit ▮-jes opgebouwd. Omdat een ▮ het cursorteken is, kunnen we het niet gebruiken om de sprite te ontwerpen, anders weten we niet meer wat onze cursor is. Vandaar dat we de sprites eerst met *'tjes ontwerpen.
30060	Het lijkt overbodig om behalve een * (VPEEK(ADRES) = 42) ook een ▮ (VPEEK(ADRES) = 219) te veranderen in een ▮. Waarom een ▮ in een ▮ veranderen? Welnu, als iemand twee keer achter elkaar de functietoets F3 zou indrukken dan zou de zojuist 'uitgevulde' sprite door de tweede F3 uitgeveegd worden. Probeer het maar.

regel(s)	uitleg
40000-40130	Wordt aangeroepen als de functietoets F4 wordt ingedrukt. Deze subroutine 'berekent' de ontworpen sprite. Elke rij van 8 blokjes moet hierbij in een getal tussen 0 en 255 omgerekend worden. Hierbij worden de blokjes in een rij van links naar rechts genummerd met 7, 6, 5, 4, 3, 2, 1 en 0. Het rij-getal ontstaat door voor elk zwart hokje twee tot-de-macht van het nummer van het hokje te berekenen en deze machten op te tellen. Zo wordt  $\Rightarrow 2^7 + 2^6 + 2^3 + 2^1 = 202.$ 7 6 5 4 3 2 1 0
40070	Kijkt of een hokje zwart is (code 219) of een * bevat (code 42) en telt $2^{(7-J)}$ op bij de al berekende som van eerdere hokjes uit de rij. Als J = 0 dan 2^7, als J = 1 dan 2^6, ..., als J = 7 dan 2^0(1).
40090	Zet het berekende rijgetal als decimaal getal achter de rij (voor PRINT USING, zie hoofdstuk 6).
50000-50130	Drukt de getekende en eventueel berekende sprite op de printer af. Deze subroutine is geschreven voor een Epson-printer. Vóór het programma staat wat u moet veranderen als u een MSX-printer hebt. Met deze subroutine kunt u sprites afdrukken en bewaren zodat u (in hoofdstuk 10) nog weet welke getallen u in de spritedefinities moet opnemen.
60000-60020	Wordt aangeroepen als u de functietoets F6 (SHIFT/F1) indrukt. De waarheidsvariabele KLAAR wordt 'waar' (-1), waardoor de HERHAAL-TOTDAT-lus in regel 240 wordt beëindigd en het programma stopt.

U kunt het beste dit programma intoetsen, opslaan en ermee experimenteren. Lees dan deze toelichting nogmaals door om de werking van het programma door te krijgen.

Op het volgende scherm ziet u dat we op functietoets F2 gedrukt hebben om een zojuist ontworpen sprite te kunnen wijzigen:



Met de joystick (of de cursortoetsen) kunt u naar hartelust over het 8 × 8 spriteveld bewegen. Wilt u op een bepaalde plaats een hokje vullen druk dan op de vuurknop (of S-toets) van de joystick en u zult zien dat op die plaats een * afgedrukt wordt. Wilt u een * weghebben, zet dan met de joystick de cursor op die plaats en druk op de vuurknop; het *'tje zal weer in een - veranderen. U zult zien dat u niet van het veld afkunt.

Bent u klaar met het ontwerp, druk dan op de spatiebalk en het pijltje voor F1 (-->) zal verdwijnen, evenals de mededeling bovenaan het scherm. Als u geen --> meer voor één van de functietoetsen in het menu ziet staan kunt u opnieuw één van de functietoetsen F1 t/m F6 indrukken. Als u het --> ziet, dan reageert de computer niet op F1 t/m F6. U kunt nu de sprite wijzigen door F2 in te drukken. U kunt ook de sprite 'uitvullen' door F3 in te drukken. Bevalt hij niet, druk dan alsnog F2 in om hem te wijzigen. Druk F4 in als u de sprite wilt berekenen en F5 als u hem wilt afdrukken. Zorg dan wel dat de printer is aangesloten en aanstaat. Hebt u er genoeg van, druk dan op F6 (dit is SHIFT/F1).

Wilt u net zo'n mooie programma-layout krijgen als het hierboven afgedrukte programma, stel dan voor het intoetsen uw scherm in op 40 tekens. Bij regels met meer dan 40 tekens gaat u gewoon door met tikken, waardoor u automatisch op de volgende beeld-

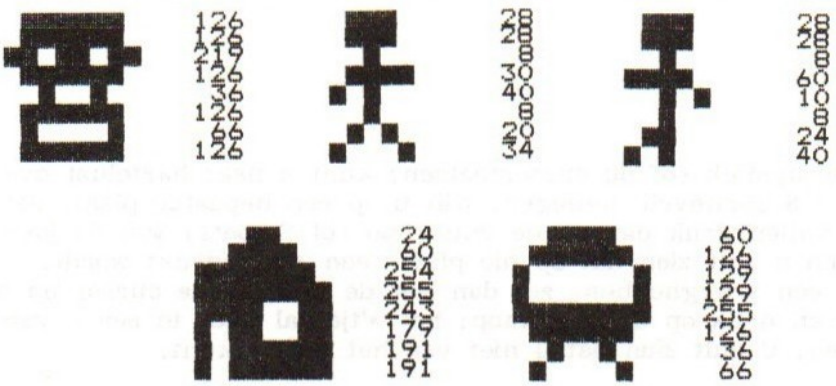
schermregel verder tikt. Wilt u vervolgens een mooie afdruk op uw printer, stel deze dan eerst in op een maximale afdrukbreedte van 80 tekens. Bij een EPSON RX+ gaat dit bijvoorbeeld als volgt:

```
LPRINT CHR$(27); "Q"; CHR$(80)
```

Om bijvoorbeeld in regel 13040 de THENnen en ELSEn netjes onder elkaar op papier te krijgen moet u regel 13040 als volgt op uw 40-posities brede scherm intoetsen:

```
13040    IF J>7 THEN VPOKE ADRES,32 ELSE
 IF VPEEK(ADRES)=219
                THEN VPOKE ADRES,42 ELSE
 VPOKE(ADRES),45
```

Hieronder ziet u enkele sprites die wij gemaakt en afgedrukt hebben:



Het volgende hoofdstuk gaat over 'netjes programmeren'. In het bovenstaande programma is de daar voorgestelde programmeeraanpak gevolgd. Dit betekent dat eerst een duidelijk ontwerp op papier gemaakt en uitgewerkt is alvorens de regels daadwerkelijk ingetoetst werden. Zo'n programma maak je niet zomaar van achter je machine!

Nu ook de keuze- en meerkeuze-opdrachten behandeld zijn kennen we alle bouwstenen om elke programmeeropdracht tot een goed einde te brengen. In bijna elk BASIC-programma komen we de sequentiële structuur, de herhalingsstructuur en de keuzestructuur tegen. Andere structuren zijn er niet. Dit wil natuurlijk niet zeggen dat we hiermee ook alle BASIC-opdrachten hebben behandeld; integendeel, we zullen nog een aantal nieuwe opdrachten tegenkomen.

In het volgende hoofdstuk zullen we zien hoe we de bovengenoemde drie programmastructuren kunnen toepassen voor het ontwikkelen van goed gestructureerde, maar bovenal goed leesbare programma's.

OEFENOPGAVEN HOOFDSTUK 4

Opgaven 1 t/m 4 IF-THEN; opgaven 5 t/m 9 IF-THEN-ELSE.

1. Schrijf een programma waarin twee IF...THEN... opdrachten voorkomen, en dat de volgende uitvoer oplevert:

```
run
a =? 5
b =? 8
 8 is groter dan 5
Ok
run
a =? 10
b =? 2
 10 is groter dan 2
Ok
```

2. Schrijf een programma dat steeds telefoonnummers inleest en alleen de telefoonnummers uit Den Haag en omgeving (netnummer: 070) telt. De telefoonnummers worden als strings ingevoerd, bijvoorbeeld 01717-7549 of 070-549860. Als de gebruiker wil stoppen toetst hij STOP of stop in als telefoonnummer. Het programma drukt dan het aantal 'Haagse' telefoonnummers af.

3. Schrijf een programma waarin honderd keer het gooien met een (zuivere) dobbelsteen wordt gesimuleerd. Na afloop van deze simulatie drukt het programma het aantal keren af dat meer dan drie ogen is gegooid en het aantal keren dat drie ogen of minder is gegooid.

4. Schrijf een stukje programma dat steeds een woord inleest en dat, wanneer de eerste letter van het woord gelijk is aan de laatste letter van het woord, de subroutine op regel 1000 aanroept. Als we STOP intoetsen wordt het programma beëindigd.

5. Wat is de uitvoer van het volgende programma:

```
10 FOR I= 1 TO 10
20    IF I<5
         THEN PRINT I;
         ELSE IF I<8 THEN PRINT 11-I;
                     ELSE PRINT I-5;
30 NEXT
40 END
```

6. Herschrijf het programma uit opgave 1 door in plaats van de twee IF-THEN-opdrachten één IF-THEN-ELSE-constructie te gebruiken.

7. Schrijf een programma dat een positief geheel getal N (N = 1,2,3,4,...) inleest en N! (N faculteit) berekent. N faculteit is het produkt 1×2×3×4×...×N. Het berekenen van N! moet in een subroutine gebeuren. N! mag echter alleen worden berekend als N kleiner dan of gelijk aan 25 is. Als voor N een waarde groter dan 25 wordt ingetoetst drukt het programma de tekst "N mag maximaal 25 zijn!" af.

8. Geef één logische uitdrukking om in een IF-opdracht te kunnen testen of de N uit opgave 7 inderdaad kleiner dan of gelijk aan 25 is maar tegelijkertijd ook groter dan of gelijk aan 1 is.

9. (Los deze opgave op door achter ELSE weer een IF te gebruiken of met twee IF-opdrachten achter elkaar!)
Schrijf een programma dat een salaris inleest en het verschuldigde bedrag aan belasting afdrukt. Dit bedrag moet op hele guldens worden afgerond. Bij een salaris beneden $ 7.500,-- is geen belasting verschuldigd; als het salaris tussen $ 7.500,-- en $ 25.000,-- ligt is 10% van het salaris aan belasting verschuldigd; als het salaris hoger is dan $ 25.000,-- is 25% belasting verschuldigd.

10. ON K GOSUB 1000, 2000, 3000 is een keuzeopdracht die voor K=1 de subroutine 1000, voor K=2 de subroutine 2000 en voor K=3 de subroutine 3000 aanroept.

    a. Schrijf drie IF-THEN-opdrachten die met zijn drieën hetzelfde effect hebben als bovengenoemde ON...GOSUB-opdracht.

    b. Schrijf één IF-THEN-ELSE-opdracht die hetzelfde effect heeft als bovengenoemde ON...GOSUB-opdracht.

11. Het komt vrij vaak voor dat als we de constructie
ON variabele GOSUB ..... willen gebruiken, de variabele niet precies de waarden 1 of 2 of 3 of ... heeft. Om dit voor elkaar te krijgen moet eerst een rekenkundige bewerking op de variabele worden toegepast. We kunnen dan in plaats van de variabele deze rekenkundige bewerking in de ON...GOSUB-opdracht zetten. Een voorbeeld:

Stel K = 0, 1 of 2 dan mogen we
ON K GOSUB... niet gebruiken, maar wel
ON K+1 GOSUB... want K+1 is 1, 2 of 3!

Geef voor elk van de volgende situaties aan welke rekenkundige bewerking in de ON...GOSUB nodig is om een foutmelding te vermijden.

a. K = 2; 4; 6; 8 ⇒ ON.....GOSUB
b. K = -1; 0; 1 ⇒ "
c. K = 1,4; 2,6; 3,7 ⇒ "
d. K = 1; 4; 9; 16 ⇒ "

12. Geef voor elk van de volgende situaties de logische uitdrukking (bewering) die we in een IF-opdracht zouden kunnen gebruiken.

a. SOM moet positief zijn maar kleiner dan 100
b. VERSCHIL moet tenminste 7 zijn maar hoogstens 12
c. LEEFTIJD moet kleiner zijn dan 18 of groter dan 65
d. ANTW$ moet "JA" zijn of GEVDEN$ moet "NEE" zijn
e. GTAL moet groter zijn dan 10 èn de wortel uit GTAL moet kleiner zijn dan 200
f. De som van A en B moet 10 zijn en het verschil van A en B moet 2 zijn
g. TELLER mag 1, 2 of 3 zijn
h. A moet positief zijn, B moet positief zijn maar A moet ongelijk zijn aan B.

5 Netjes Programmeren

In het eerste deel van dit hoofdstuk laten we zien hoe het ontwerpen, schrijven en testen van een programma volgens de methode van gestructureerd programmeren verloopt. Aan de hand van een eenvoudige programmeeropdracht, te weten 'het bepalen van het gemiddelde van een aantal getallen', wordt getracht de werkwijze en het nut van gestructureerd programmeren duidelijk te maken. Omdat gestructureerd programmeren meer inhoudt dan in dit boek wordt besproken, zullen we vanaf nu spreken over 'netjes' programmeren. 'Netjes' programmeren is het ontwerpen en schrijven van programma's die:

- modulair opgebouwd zijn
- heldere taalconstructies gebruiken
- een zeer goed leesbare programmatekst hebben
- goed gedocumenteerd zijn

Modulair opgebouwd wil zeggen dat het programma bestaat uit één hoofdmodule, waarin in grote lijnen de besturing van het hele programma geregeld wordt, en één of meer submodulen die binnen het programma een specifieke taak hebben. Deze submodulen worden zo gekozen dat ze min of meer onafhankelijk van de hoofdmodule en onafhankelijk van elkaar ontworpen en geprogrammeerd kunnen worden. Hoe ingewikkelder de programmeeropdracht is, des te meer submodulen er zullen ontstaan.

Heldere taalconstructies gebruiken wil zeggen dat we bij het bepalen van de programmastructuur gebruik zúllen maken van de FOR-NEXT, de HERHAAL-TOTDAT, de DOE-ZOLANG, de IF-THEN, de IF-THEN-ELSE en de ON...GOSUB constructies.
Een minder duidelijke taalconstructie is bijvoorbeeld het ongecontroleerd gebruik van GOTO-opdrachten, waarmee naar elke gewenste regel in het programma gesprongen kan worden. Onoordeelkundig gebruik van deze GOTO's leidt tot ondoorzichtige programmastructuren. Wij zullen dan ook alleen GOTO's gebruiken om de bovengenoemde programmastructuren te programmeren.

Wat met een leesbare programmatekst bedoeld wordt hebben we in de voorgaande hoofdstukken kunnen zien. Het betekent dat de 'programma-lay-out' zodanig is dat het programma voor iedereen

goed leesbaar is en dat de diverse structuren in het programma duidelijk tot uitdrukking komen. We kunnen dit bereiken door op diverse plaatsen de programmaopdrachten ten opzichte van elkaar te laten inspringen; door hier en daar blanco regels op te nemen en door op de juiste plaatsen commentaar in het programma op te nemen.

Goed **gedocumenteerd** houdt in dat naast het programma zelf (de programmatekst) documentatie bestaat, waarin het gebruik van het programma wordt toegelicht (handleiding); waar de in- en uitvoer van het programma en de betekenis van de variabelen wordt beschreven en waarin de structuur van het programma met behulp van een schematechniek in kaart wordt gebracht. In het tweede deel van dit hoofdstuk zullen we zien hoe we de structuur van een programma in een programma-structuurdiagram (PSD) kunnen weergeven.

5.1 WAAROM NETJES PROGRAMMEREN?

Het doel van 'netjes' programmeren is de totale programmeertaak zo te organiseren en de programmamodulen zo te ontwerpen dat de meeste fouten voorkomen worden. Hierbij doelen we niet zo zeer op de taalfouten (syntactische fouten) als wel op de logische fouten. Dit zijn fouten in de werking van het programma; het programma doet niet dat wat het moet doen! Door de modulaire opbouw, het gebruik van duidelijke taalconstructies en een overzichtelijke programmatekst kunnen eventuele fouten snel gelokaliseerd en verbeterd worden.

Een ander punt is dat modulair opgebouwde programma's beter te onderhouden zijn en gemakkelijker zijn aan te passen aan nieuwe wensen en veranderde omstandigheden. In de praktijk komt het niet zelden voor dat anderen dan de programmeur zelf een programma moeten onderhouden en het eventueel moeten aanpassen. Als een dergelijk programma niet modulair is opgebouwd, allerlei duistere programmeertechnische 'trucs' (omdat de programmeur dacht slim te zijn) bevat, een slecht leesbare programmatekst heeft en daarbij nog slecht gedocumenteerd is, is het niet verwonderlijk dat onderhoud, laat staan uitbreiding, van zo'n programma een onmogelijke opgave is.

Dikwijls wordt in een dergelijke situatie besloten een geheel nieuw programma te schrijven; dubbel werk dus. Door bij het programmeren de bovengenoemde vier punten niet uit het oog te verliezen kan een hoop ergernis, moeite en kosten worden bespaard.

Bij het ontwerpen en schrijven van een programma kunnen we bepaalde fasen in de uit te voeren werkzaamheden aanbrengen.

1. Probleembeschrijving

Probeer te formuleren wat het doel is van het te schrijven programma en wat er van het programma verlangd wordt.

2. Probleemanalyse

Het probleem (de programmeeropdracht) moet goed worden bestudeerd. Er moet een oplossingsmethode worden gekozen en eventuele complicaties moeten worden onderkend.

3. Module-indeling

In deze fase wordt het programma in modulen ingedeeld. Besloten moet worden of er naast de hoofdmodule nog submodulen moeten komen en zo ja voor welke specifieke taken.

4. Programmastructuur bepalen

Welke taalconstructies gaan we gebruiken? En hoe is in grote lijnen de werking van het programma? In deze fase kunnen we de programmastructuur met behulp van een schematechniek in kaart brengen. Deze fase wordt voor elk van de ontworpen modulen doorlopen.

5. Coderen van programmaopdrachten

Nu gaan we de programmatekst schrijven. Uitgangspunt is hierbij de eerder ontworpen en in kaart gebrachte programmastructuur. We zorgen voor een zeer duidelijke programmatekst en we vermijden 'slimme' of ingewikkelde (en daardoor voor anderen onlogische) programmatechnische oplossingen, ook al is de verleiding hiertoe soms groot! Dit coderen (wat veel mensen bedoelen als ze programmeren zeggen) wordt voor elk van de ontworpen programmamodulen afzonderlijk uitgevoerd.

De fasen 1, 2 en 3 zijn geheel onafhankelijk van de programmeertaal. Dit betekent dat we de werkzaamheden in deze fasen kunnen uitvoeren ongeacht de programmeertaal waarin we straks gaan coderen.
Bij fase 4 geldt dit eigenlijk ook, alhoewel de diverse talen verschillen in de manier waarop bepaalde structuren gerealiseerd kunnen worden.
Fase 5 is natuurlijk geheel taalafhankelijk, want daar schrijven we echt opdrachten in de taal waarin we programmeren. Bovendien programmeren we voor een bepaalde computer, waarbij het helaas zo is dat dezelfde taal (bijvoorbeeld BASIC) voor de diverse computermerken toch net iets anders werkt.

We zullen in de volgende paragraaf laten zien welke activiteiten zoal in bovengenoemde vijf fasen worden uitgevoerd. De programmeeropdracht is eenvoudig. Als er lezers zijn die straks denken 'Tsjonge, jonge.....wat een soesa voor zo'n klein probleempje', dan hebben ze gelijk. Het is echter de bedoeling de vijf fasen aan de hand van een voor iedereen herkenbare programmeeropdracht te demonstreren. En tenslotte moet u maar denken: 'Klein geleerd, GROOT gedaan'.

5.2 PROGRAMMAONTWERPMETHODEN

Het ontwerpen van programma's kan op vele manieren gebeuren. Wij zullen kiezen voor de methode van **stapsgewijze verfijning** van de programmeeropdracht. Deze methode staat bekend als de

TOP-DOWN ontwerpmethode

Uitgangspunt bij deze methode is de programmeeropdracht als geheel. De totale programmeertaak wordt opgesplitst in een aantal kleinere deeltaken, die op hun beurt weer kunnen worden onderverdeeld in nog kleinere deeltaken. Uiteindelijk blijft er een aantal goed afgebakende deeltaken over, terwijl steeds het totale overzicht behouden blijft.

Lijnrecht tegenover deze Top-Down ontwerpmethode staat de

BOTTOM-UP ontwerpmethode

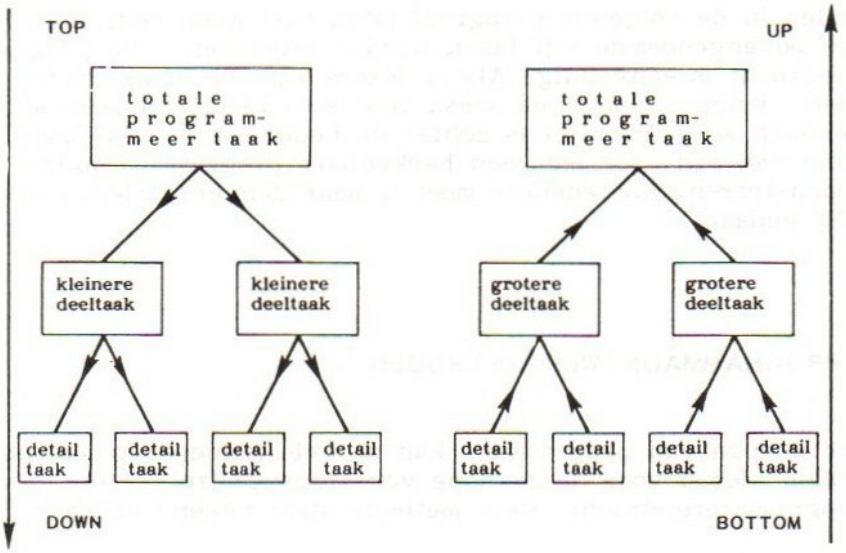
Ontwerpers en programmeurs die deze methode bij het programmeren gebruiken hebben een onweerstaanbare voorliefde voor details. Zij beginnen eigenlijk meteen te coderen en proberen daarna de gecodeerde stukken samen te voegen tot grotere stukken programma. Zij missen echter het totale overzicht en maken daardoor een programma dat onlogisch in elkaar zit en dat reeds voordat het klaar is op een aantal plaatsen is 'gerepareerd'. Niet zelden loopt de zaak helemaal vast en moet opnieuw worden begonnen.

In de tekening op bladzijde 142 is te zien hoe een Top-Down programmeur en een Bottom-up programmeur te werk gaan.

Wellicht zou je deze twee ontwerpmethoden ook zo kunnen typeren:

Top-Down = 'Een goed begin is het halve werk'

Bottom-up = 'Hardlopers zijn doodlopers'



Er zijn ook programmeurs die volgens een soort compromis tussen bovengenoemde methoden te werk gaan. Wij zullen echter in dit hoofdstuk laten zien hoe de TOP-DOWN methode in de praktijk kan worden toegepast.

5.3 TOP-DOWN ONTWERPMETHODE

In deze paragraaf doorlopen we aan de hand van een eenvoudige programmeeropdracht de in de vorige paragraaf genoemde vijf fasen.

De programmeeropdracht luidt:

> "Schrijf een programma voor het berekenen van het gemiddelde van een aantal getallen."

FASE 1 : Probleembeschrijving

Met een onnauwkeurig omschreven programmeeropdracht kunnen we niet werken. We proberen een zo volledig mogelijke probleembeschrijving boven tafel te krijgen. Laten we het eens proberen.

- Het doel van het programmeren is het berekenen van het rekenkundig gemiddelde van een aantal getallen.
  De programma-eisen zijn als volgt:
  - Het programma moet conversationeel zijn.
  - Er mogen alleen positieve getallen worden ingetoetst.
  - Bij het intoetsen van een bepaalde waarde wordt het programma beëindigd nadat het rekenkundig gemiddelde van de getallen op het beeldscherm is afgedrukt.

Hopelijk hebben we hiermee een goede en volledige probleemomschrijving gegeven. Het is in ieder geval een beschrijving waarmee te werken valt. In de hiernavolgende fase gaan we een oplossingsmethode kiezen en de programmeertaak in kleinere deeltaken onderverdelen.

opdracht 5.1

Probeer zelf een volledige probleembeschrijving te maken voor een programma dat een aantal woorden inleest (bijvoorbeeld alle woorden op een willekeurige bladzijde van dit boek) en afdrukt hoeveel e's en a's in elk ingetoetst woord voorkomen. Als het programma wordt beëindigd drukt het de percentages e's en a's onder alle ingetoetste letters (dus alle woorden bij elkaar) af.

FASE 2 : Probleemanalyse

Bij het analyseren van een probleem kunnen we heel goed als volgt te werk gaan:

1. Bepaal welke **uitvoer** nodig is.
2. Bepaal welke **invoer** hiervoor nodig is.
3. Bepaal de **bewerkingen** die nodig zijn om vanuit de invoer tot de uitvoer te komen.

Dit is een aanpak die niet alleen globaal voor het hele probleem (de totale programmeertaak) gevolgd kan worden maar ook bij het analyseren van de deeltaken die ontstaan bij het verfijnen van de totale taak.
Voor ons probleem ligt de zaak nogal eenvoudig:

1. De uitvoer is het rekenkundig gemiddelde van een aantal ingetoetste getallen.
2. De invoer hiervoor zijn de getallen waarvan het gemiddelde moet worden berekend.
3. De bewerkingen die nodig zijn om vanuit de ingetoetste getallen tot het gemiddelde van die getallen te komen zullen we nu gaan bepalen. We gaan een oplossingsmethode kiezen.

Als er over het rekenkundig gemiddelde van een aantal getallen gepraat wordt dan komt al snel de volgende formule in je gedachten op:

$$\bar{X} = \frac{1}{N}(x_1 + x_2 + \ldots + x_N) \quad , \quad \text{waarin}$$

$\bar{X}$ een notatie is voor het rekenkundig gemiddelde;

N het aantal getallen voorstelt;

en x_1, x_2, ... t/m x_N de afzonderlijke getallen voorstellen.

In een computerprogramma zouden we als volgt te werk kunnen gaan:
- Lees alle N getallen achter elkaar in en sla alle getallen in het computergeheugen op.
- Houd bij hoeveel getallen er zijn ingevoerd.
- Als -1 wordt ingevoerd bereken dan de som van alle in het geheugen opgeslagen getallen en deel deze som door het aantal ingetoetste getallen. We kiezen -1 omdat dit geen positief getal is.

Dit zou een goede oplossingsmethode zijn, indien er nog andere berekeningen met de ingevoerde getallen zouden moeten worden uitgevoerd waarbij het noodzakelijk is dat alle ingevoerde getallen afzonderlijk bewaard moeten worden. Dit wordt echter niet gevraagd!

Als we bovenstaande formule eens anders schrijven:

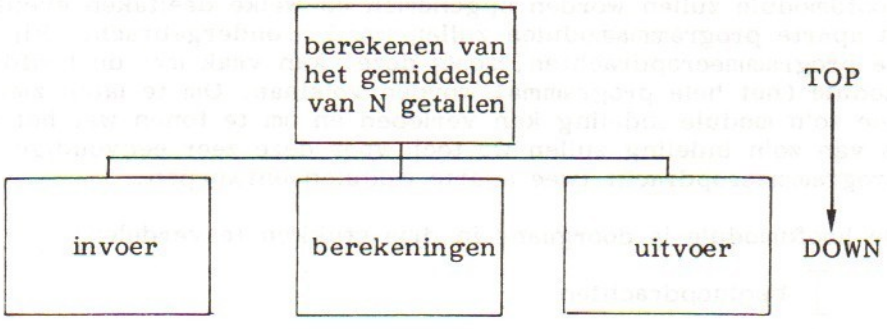
$$\bar{X} = \frac{SOM}{N} \quad , \quad \text{waarbij SOM de som van de ingetoetste getallen is en N het aantal ingetoetste getallen,}$$

dan zien we de enige twee waarden die nodig zijn om het gemiddelde te berekenen de som van de ingetoetste getallen en het aantal ingetoetste getallen zijn, dus SOM en N.

We kiezen daarom voor de laatste oplossingsmethode, waarbij twee **'cumulatieve'** waarden (SOM en N) worden berekend en bewaard. (Op deze manier werken trouwens ook de zakrekenmachines waarmee gemiddelden kunnen worden uitgerekend!) SOM bevat dus steeds de som van alle ingetoetste getallen, terwijl N steeds het aantal ingetoetste getallen als waarde heeft. Een cumulatieve waarde is een waarde die ontstaat door het steeds maar optellen van bepaalde waarden.

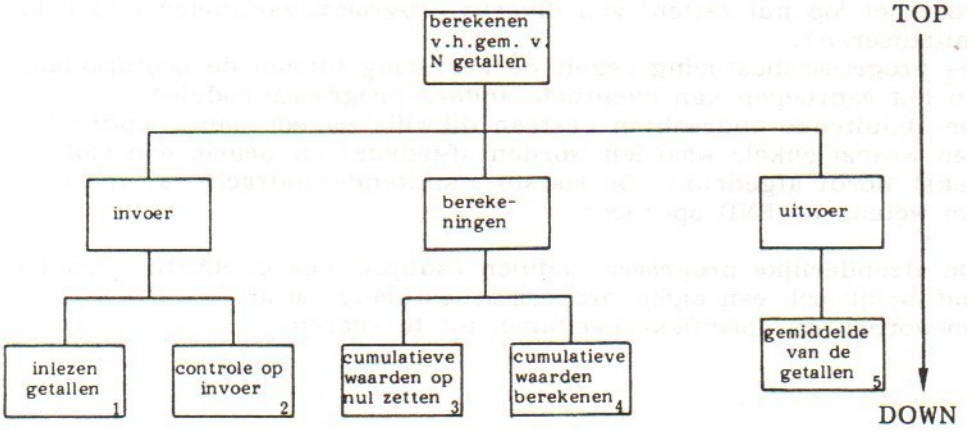
Nu we de oplossingsmethode hebben bepaald gaan we de programmeeropdracht (de programmeertaak) stapsgewijs verfijnen. Het **Top-Down ontwerpproces** begint! Boven aan de top staat de totale programmeertaak:

Dit blok geeft het doel (de globale taak) aan. We gaan de totale taak verfijnen (opsplitsen) in een aantal kleinere deeltaken. Deze deeltaken worden zo gekozen dat ze min of meer los van elkaar staan. De eerste verfijning geeft een ruwe indeling in deeltaken. In bijna elke programmeertaak (en deeltaak) kunnen we de volgende verfijning aanbrengen:



De eerste verfijning levert drie deeltaken op, te weten: invoer, berekeningen en uitvoer. De in- en uitvoer staan beschreven in de probleembeschrijving. De berekeningen hangen mede af van de oplossingsmethode die we gekozen hebben.

Als we de probleemomschrijving en de gekozen oplossingsmethode goed bekijken kunnen we nog een verfijning aanbrengen:



De deeltaken 1, 2, 3, 4 en 5 zoals die er nu staan zijn te overzien en goed hanteerbaar. We zullen zien dat niet iedere deeltaak 'domweg' een aparte programmamodule zal opleveren.

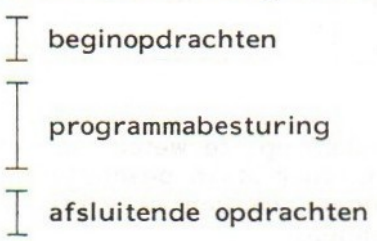
opdracht 5.2

Geef voor de programmeeropdracht uit opdracht 5.1 een opsplitsing in deeltaken.

FASE 3 : Module-indeling

In deze fase gaan we beslissen welke deeltaken uit fase 2 in de hoofdmodule zullen worden opgenomen en welke deeltaken eventueel in aparte programmamodulen zullen worden ondergebracht. Bij kleine programmeeropdrachten, zoals deze, kan vaak met de hoofdmodule (het hele programma) worden volstaan. Om te laten zien hoe zo'n module-indeling kan verlopen en om te tonen wat het nut is van zo'n indeling zullen we toch voor deze zeer eenvoudige programmeeropdracht twee aparte modulen ontwerpen.

De hoofdmodule is doorgaans in drie stukken te verdelen:

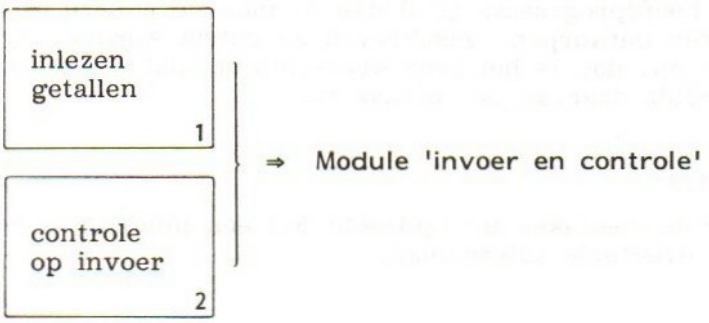


Bij beginopdrachten kunnen we denken aan het afdrukken van een tekst waarin het een en ander aan de gebruiker wordt verteld, en natuurlijk ook aan commentaaropdrachten. Maar ook aan opdrachten voor het 'op nul zetten' van diverse programmavariabelen (dit heet initialiseren).
De programmabesturing regelt de besturing binnen de hoofdmodule en het aanroepen van eventuele andere programmamodulen.
De afsluitende opdrachten bestaan dikwijls uit een aantal opdrachten waarin enkele waarden worden afgedrukt en waarin een slottekst wordt afgedrukt. De laatste afsluitende-opdracht is, zoals we weten, de END-opdracht.

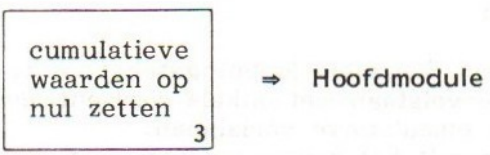
De afzonderlijke programmamodulen (subroutines in BASIC) hebben natuurlijk ook een eigen programmabesturing, maar zijn bovenal ontworpen om specifieke deeltaken uit te voeren.

Laten we voor onze programmeeropdracht eens een poging wagen. We proberen modulen te maken met een eigen taakomschrijving, die min of meer onafhankelijk zijn van andere modulen.

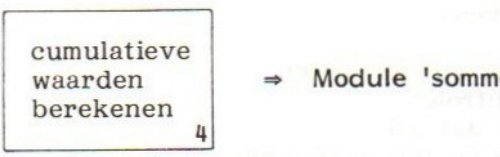
De invoer is onafhankelijk van de berekeningen en van de uitvoer. Laten we daar een aparte programmamodule voor maken, dus:



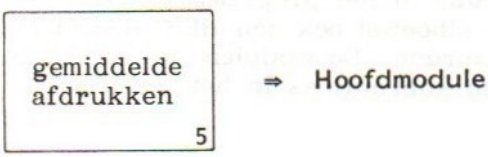
De volgende deeltaak 'cumulatieve waarden op nul zetten' is typisch iets voor het hoofdprogramma (beginopdrachten):



De deeltaak 'cumulatieve waarden berekenen' lijkt ons nogal ingewikkeld, dus daar maken we ook een aparte module voor:



Tenslotte is de deeltaak 'gemiddelde afdrukken' typisch iets voor het afsluiten van de hoofdmodule, dus:



Het nut van deze module-indeling ondervinden we in de volgende twee fasen. We kunnen ons nu namelijk eerst op de hoofdmodule concentreren zonder ons te bemoeien met de structuur en de inhoud van de andere modulen. We kunnen de programmatekst van de hoofdmodule schrijven en de hoofdmodule testen zonder dat de andere modulen geprogrammeerd zijn. Om het hoofdprogramma te kunnen testen nemen we gewoon 'lege' modulen (subroutines) op. Loopt het hoofdprogramma goed dan kunnen de andere modulen één voor één ontworpen, geschreven en getest worden. Treden er dan fouten op, dan is het zeer waarschijnlijk dat de laatst ontworpen module daarvan de oorzaak is.

opdracht 5.3

Maak voor de deeltaken uit opdracht 5.2 een indeling in hoofdmodule en eventuele submodulen.

FASE 4a : Programmastructuur hoofdmodule

De algemene structuur van een hoofdmodule is:
- initiërende opdrachten
- programmabesturing
- afsluitende opdrachten

Er wordt niet gevraagd een bepaalde 'openingstekst' af te drukken, dus dat doen we niet. We volstaan met enkele commentaarregels en het op nul zetten van de cumulatieve variabelen.
De programmabesturing regelt het aanroepen van de modulen en beslist of het programma moet worden beëindigd.
De afsluitende opdracht wordt gevormd door het afdrukken van het gemiddelde.

De programmastructuur van de hoofdmodule kan als volgt puntsgewijs worden weergegeven:
- commentaar
- cumulatieve variabelen op nul zetten
- module 'invoer en controle'
- doe zolang gebruiker dat wil
  - module 'sommeer en tel' aanroepen
  - module 'invoer en controle' aanroepen
- gemiddelde afdrukken
- afsluiten

Voor de herhalingsstructuur in het programma kiezen we voor de DOE-ZOLANG-structuur, alhoewel ook een HERHAAL-TOTDAT-structuur gebruikt kan worden. De modulen 'invoer en controle' en 'sommer en tel' worden subroutines in het programma.

opdracht 5.4

Geef de structuur voor de in opdracht 5.3 gevormde hoofdmodule voor de programmeeropdracht met de e's en a's.

FASE 5a : Coderen van hoofdmodule

Uitgaande van de in fase 4a gevonden programmastructuur gaan we de hoofdmodule schrijven. We kiezen de variabelen GTAL, N en SOM voor respectievelijk het laatste ingetoetste getal, het totaal aantal ingetoetste getallen en de som van de ingetoetste getallen. We weten dat als de gebruiker het getal -1 intoetst, het programma beëindigd moet worden.

We gaan vanaf nu werken met een beeldscherm van 40 tekens (WIDTH 40 in SCREEN 0) en we maken maximaal regels van 80 tekens; twee beeldschermregels dus. Wilt u de programma's net zo op een printer kunnen afdrukken zoals in het boek gedaan is werk dan ook steeds met WIDTH 40. Werkt u met een MSX-2 dan kunt u ook de 80-kolommenstand inschakelen.

```
100 REM gemiddelde van N getallen
105 '
110 '
115 'Hoofdmodule
120 '
130 N=0 : SOM=0
140 GOSUB 1000 'invoer en controle
145 '***doe zolang niet -1 is ingetoetst
150 IF GTAL=-1 THEN 190
160   GOSUB 2000 'sommeer en tel
170   GOSUB 1000 'invoer en controle
180 GOTO 150
185 '***
190 PRINT "Het gemiddelde is:";SOM/N
200 END
```

Om deze hoofdmodule te testen nemen we voor de twee nog te ontwerpen submodulen twee lege modules op (nou ja... bijna lege):

```
1000 'Subroutine invoer en controle
1005 '
1010   'Nog niet geprogrammeerd
1015 '
1020 RETURN 'uit invoer en controle
1025 '
2000 'Subroutine sommeer en tel
2005 '
2010   'Nog niet geprogrammeerd
2015 '
2020 RETURN 'uit sommeer en tel
```

Als we het programma zouden draaien hebben de GOSUB's in de regels 140, 160 en 170 geen enkel effect, want de subroutines bestaan slechts uit één commentaarregel. Toch is het aardig om het programma zoals het nu is te draaien. Tenslotte moet de hoofdmodule worden getest:

```
RUN
```

er gebeurt niets!!

We kunnen wachten tot we een ons wegen want we zitten in een eindloze DOE-ZOLANG-lus, omdat GTAL = -1 niet waar is (GTAL heeft als waarde nul; dit is de waarde die BASIC standaard aan een variabele geeft, de default waarde). Aangezien de twee GOSUB's in de DOE-ZOLANG-lus geen uitvoer opleveren, gebeurt er dus niets.

Om de werking van de DOE-ZOLANG-lus te testen zouden we in de lege subroutines enkele 'tekenen van leven' kunnen opnemen. Om te zorgen dat er géén oneindige lus ontstaat geven we de variabele GTAL in de subroutine 'invoer en controle' de waarde -1. Daar de subroutine eerder wordt aangeroepen (regel 140) dan de lus begint (regel 150) zal de lus niet worden doorlopen, omdat de bewering GTAL = -1 dan wel waar is (GTAL is dan immers min één). Om dit alles te controleren nemen we in de subroutines de volgende opdrachten op (eerst halen we de computer uit de oneindige lus door met de CTRL/STOP-toets het programma af te breken. Nu kunnen we de nieuwe opdrachten intoetsen):

```
1000 'Subroutine invoer en controle
1005 '
1010   GTAL=-1
1020   PRINT "Ik was even in invoer en controle"
1025 '
1030 RETURN ' uit invoer en controle
2000 'Subroutine sommeer en tel
2005 '
2010   PRINT "Ik was even in sommeer en tel
2015 '
2020 RETURN 'uit sommeer en tel
```

We proberen het nog eens:

```
run
Ik was even in invoer en controle
Het gemiddelde is:
Division by zero in 190
Ok
```
← een foutmelding!
er wordt ergens door nul gedeeld

Het programma is blijkbaar één keer in de routine 'invoer en controle' geweest. Dit klopt; dat is namelijk het gevolg van regel 140. Daarna treedt er echter een fout op in regel 190. We weten dus wel dat de computer de DOE-ZOLANG-lus (150-180) overgeslagen heeft, hetgeen inderdaad juist is, want GTAL was gelijk aan -1 dus de bewering GTAL = -1 is waar en dus gaat het programma na regel 150 verder met regel 190 (THEN 190).
Maar in regel 190 gebeurt iets waardoor het programma afbreekt, namelijk delen door nul! Dit komt omdat N de waarde nul heeft; dit als gevolg van regel 130.

Nu komt iets waarvoor we moeten oppassen! Neem even aan dat we slim willen zijn (iets dat vaak heel verleidelijk is!). We veranderen regel 130 in:

```
130 N = 1 : SOM = 0
```

Als we het programma nu draaien, zien we dit:

```
run
Ik was even in invoer en controle
Het gemiddelde is: 0
Ok
```

Het lijkt erop dat we de fout hersteld hebben. Dat is ook zo, alleen wel ten koste van de logica van het programma. Immers, als N het aantal ingetoetste getallen bevat, is het erg onlogisch om nog voordat er één getal is ingetoetst de waarde van N één te maken. Dit hoeft ook niet want we laten regel 130 gewoon zoals hij was en we veranderen alleen regel 190 in:

```
190 IF N>0 THEN PRINT "Het gemiddelde is:";SOM/N
           ELSE PRINT "Er is niets opgeteld!"
```

Het kost wat meer moeite, maar het programma blijft tenminste logisch.

De tekst van de hoofdmodule wordt dus:

```
100 REM gemiddelde van N getallen
105 '
110 '
115 'Hoofdmodule
120 '
130 N=0 : SOM=0
140 GOSUB 1000 'invoer en controle
145 '***doe zolang niet -1 is ingetoetst
```

```
150 IF GTAL=-1 THEN 190
160   GOSUB 2000 'sommeer en tel
170   GOSUB 1000 'invoer en controle
180 GOTO 150
185 '***
190 IF N>0 THEN PRINT "Het gemiddelde is:";SOM/N
           ELSE PRINT "Er is niets opgeteld!"
200 END
```

In BASIC gebruiken de hoofdmodule en de submodulen vaak dezelfde variabelen. Het moet dus wel bekend zijn wat voor een bepaalde module de invoervariabelen en wat de uitvoervariabelen zijn. Dit betekent dat we precies moeten weten 'welke' variabelen 'waar' worden gebruikt.

We gaan nu de module 'invoer en controle' ontwerpen en programmeren.

opdracht 5.5

Codeer de hoofdmodule waarvan de structuur in opdracht 5.4 moest worden ontworpen.

FASE 4b : Programmastructuur module 'invoer en controle'

In deze module wordt een ingetoetst getal ingelezen en wordt de waarde ervan gecontroleerd: er mogen immers alleen positieve getallen worden ingevoerd! De complicatie hierbij is dat -1 een negatief getal is, dus verboden, maar aan de andere kant is het de waarde waarmee het programma beëindigd wordt.

De structuur is als volgt:
- lees getal in
- controleer op juiste invoer
- einde module

Anders gezegd:
- lees getal is
- doe zolang gebruiker een negatief getal ongelijk -1 intoetst
  - druk melding onjuist invoer af
  - lees getal in
- einde module

We kiezen ook hier voor een DOE-ZOLANG-structuur om de volhardende gebruiker erop te wijzen dat er positieve getallen moeten worden ingetoetst.

FASE 5b : Coderen van module 'invoer en controle'

De programmastructuur is zo eenvoudig dat het programmeren geen moeilijkheden zal opleveren.

```
1000 'Subroutine invoer en controle
1005 '
1010   INPUT "Uw getal is"; GTAL
1015   '***doe zolang negatief getal ongelijk -1 is ingetoetst
1020   IF GTAL>0 OR GTAL=-1 THEN 1060
1030     INPUT "Alleen positieve invoer"; GTAL
1040   GOTO 1020
1045   '***
1055 '
1060 RETURN 'uit invoer en controle
```

Stel dat we de computer bij de hand hebben en stel dat we de hoofdmodule ingetoetst hebben en dat ook beide bijna lege modulen van bladzijde 149 nog in het geheugen staan.

We toetsen nu de bovenstaande module in (regels 1000 t/m 1060). Hierdoor wordt de oude routine die op regel 1000 begint overschreven. De tweede subroutine staat echter nog steeds in de oude vorm in het geheugen, zo dus:

```
2000 'Subroutine sommeer en tel
2005 '
2010   PRINT "Ik was even in sommeer en tel
2015 '
2020 RETURN 'uit sommeer en tel
```

Wat nu komt is een fraaie eigenschap van BASIC en een fraaie eigenschap van **modulair programmeren**!

Door nu niet RUN maar RUN 1000 in te toetsen wordt het programma vanaf regel 1000 uitgevoerd. We kunnen op die manier de werking van de routine 'invoer en controle' testen los van de rest van het programma. De computer stopt vanzelf bij regel 1060, want daar geeft hij de foutmelding (nou ja, foutmelding) 'Return without gosub', hetgeen betekent dat hij geen RETURN-opdracht kan uitvoeren zonder dat hij de bijbehorende GOSUB gezien heeft (de bijbehorende GOSUB staat namelijk in de hoofdmodule). In ons geval is dit echter geen foutmelding, want we willen juist dat hij hier stopt! Laten we het eens proberen:

```
run 1000
Uw getal is? 5
RETURN without GOSUB in 1060
Ok
```

dit wisten we van tevoren

We hebben 5 als eerste getal ingetoetst en inderdaad wordt de lus, die bedoeld is om foutieve invoer op te sporen, overgeslagen. Immers voor GTAL = 5 is namelijk

GTAL > 0 waar èn GTAL = -1 niet waar

dus

'waar' OR 'niet waar' is 'waar'

hetgeen betekent dat voor GTAL = 5 de logische uitdrukking

GTAL > 0 OR GTAL = -1 waar is,

dus wordt de lus niet uitgevoerd. Dat de melding 'RETURN without GOSUB' zou komen wisten we van tevoren.

Voor positieve invoer doet de module het goed. Nu eens testen op negatieve invoer:

```
run 1000
Uw getal is? -5
Alleen positieve invoer? -26
Alleen positieve invoer? -1
RETURN without GOSUB in 1060
Ok
```

Voor -5 en -26 wordt de lus uitgevoerd; dit is juist want

```
- 5 < 0  èn  - 5 <> -1  èn
-26 < 0  èn  -26 <> -1  (de bewering GTAL > 0 OR GTAL = -1
                         is dus in beide gevallen niet waar!)
```

Als in een samengestelde bewering met OR één van beide beweringen waar is dan is de hele bewering waar. Als we beweren PIET ouder dan 15 OF JANNEKE jonger dan 10 dan is deze bewering waar als òf PIET ouder is dan 15 òf als JANNEKE jonger is dan 10. En ook als PIET ouder dan 15 en JANNEKE jonger dan 10 is.

Voor -1 wordt de controlelus beëindigd. Dit moet ook, want de waarde -1 moet als 'stopwaarde' (ook wel vlag genoemd) worden doorgegeven aan de hoofdmodule.

Nog even

-1 > 0 is niet waar maar -1 = -1 is wel waar,

dus voor GTAL = -1 is de logische uitdrukking

GTAL > 0 OR GTAL = -1 'waar' dus

wordt de lus beëindigd.

De module 'invoer en controle' is dus in orde.

Voordat we met de tweede module beginnen kunnen we nog de werking van hoofdmodule èn de module 'invoer en controle' samen testen. Daartoe tikken we alleen RUN in. Hierdoor wordt het programma vanaf het laagste regelnummer (regel 100) uitgevoerd:

```
run
Uw getal is? 45
Ik was even in sommeer en tel
Uw getal is? -16
Alleen positieve invoer? -1
Er is niets opgeteld!
Ok
```

We zien dat voor positieve invoer de computer 'even' in 'sommeer en tel' geweest is, hetgeen de bedoeling is. Voor negatieve invoer ongelijk -1 wordt om nieuwe invoer gevraagd; dat is ook in orde. Bij -1 wordt het programma beëindigd, maar omdat N nog steeds nul is wordt in de IF-opdracht uit regel 190 de ELSE-tak gekozen. Tot zover alles oké, nu de laatste module.

opdracht 5.6

Codeer de submodulen die u in opdracht 5.3 gevormd hebt (daarvoor moet eerst de structuur worden bepaald).
Als u niet direct ziet hoe je het aantal e's en a's in een woord bepaalt, maak daarvoor dan alsnog een module en ontwerp en codeer die module als laatste!

FASE 4c : Programmastructuur module 'sommeer en tel'

Deze structuur is heel eenvoudig, gewoon sommeren en tellen:
- tel ingetoetst getal bij som op
- verhoog aantal met één
- einde module

Een heel eenvoudige sequentiële structuur!

FASE 5c : Coderen van module 'sommeer en tel'

Rechttoe, rechtaan:

```
2000 'Subroutine sommeer en tel
2005 '
2010   SOM=SOM + GTAL
2020   N=N + 1
2025 '
2030 RETURN 'uit sommeer en tel
```

We kunnen deze module wel zonder testen in het programma opnemen! Wel is het belangrijk te weten welke variabelen moeten worden gebruikt. Dit wordt mede bepaald door eerder ontworpen modulen. Voor deze module betekent dat het gebruik van de variabelen SOM, GTAL en N. De variabelen SOM en N veranderen in deze module van waarde, de variabele GTAL echter niet.

We geven nog een keer de gehele programmatekst:

```
100 REM gemiddelde van N getallen
105 '
110 '
115 'Hoofdmodule
120 '
130 N=0 : SOM=0
140 GOSUB 1000 'invoer en controle
145 '***doe zolang niet -1 is ingetoetst
150 IF GTAL=-1 THEN 190
160   GOSUB 2000 'sommeer en tel
170   GOSUB 1000 'invoer en controle
180 GOTO 150
185 '***
190 IF N>0 THEN PRINT "Het gemiddelde is:";SOM/N
           ELSE PRINT "Er is niets opgeteld!"
200 END
1000 'Subroutine invoer en controle
1005 '
1010   INPUT "Uw getal is"; GTAL
1015   '***doe zolang negatief getal ongelijk -1 is ingetoetst
1020   IF GTAL>0 OR GTAL=-1 THEN 1060
1030     INPUT "Alleen positieve invoer"; GTAL
1040   GOTO 1020
1045   '***
1055 '
1060 RETURN 'uit invoer en controle
1065 '
2000 'Subroutine sommeer en tel
2005 '
2010   SOM=SOM + GTAL
2020   N=N + 1
2025 '
2030 RETURN 'uit sommeer en tel
```

Laten we dit programma eens draaien:

```
run
Uw getal is? 5
Uw getal is? -15
Alleen positieve invoer? 15
Uw getal is? 9
Uw getal is? 7
Uw getal is? -1
Het gemiddelde is: 9
Ok                                   Inderdaad alles Ok!
```

Ook bij Top-Down ontwerp kan er van alles fout gaan en is het eindprodukt waarschijnlijk nog lang niet volmaakt. We geven voor deze programmeeropdracht enkele suggesties:

- Zou het niet op zijn minst elegant zijn geweest de gebruiker door middel van een tekst op het scherm iets over het programma te vertellen? (Onvolledige probleembeschrijving wellicht?)
- Zijn bij de probleemanalyse de juiste verfijningen aangebracht?
- Is er een andere, efficiëntere oplossingsmethode?
- Zou de tekst op het beeldscherm niet wat overzichtelijker afgedrukt kunnen worden? (Blanco regels tussen de mededelingen!)
- Zijn er geen variabelen die alleen gehele waarden kunnen aannemen? Door integer variabelen (N%) te gebruiken sparen we geheugenruimte!

Deze lijst is vast niet compleet!

Een punt van *kritiek* zou het volgende kunnen zijn. Modulair programmeren wordt aanbevolen omdat het onderhoud en het aanpassen van het programma relatief eenvoudig en snel zijn uit te voeren. Een aanpassing van dit programma zou kunnen zijn dat het geschikt gemaakt moet worden voor zowel positieve als negatieve invoer. Bij een goed modulair programma, waarbij de invoer en de controle op de invoer in een aparte module worden geregeld, mag je verwachten dat de aanpassing zich alleen tot die module zou moeten beperken.

In ons voorbeeld is dit ook grotendeels zo, ware het niet dat de waarde -1 als 'stopwaarde' vanuit de module 'invoer en controle' wordt doorgegeven aan de hoofdmodule. Bij negatieve invoer is -1 geen goede stopwaarde, -1 zou dan een gewoon 'invoergetal' moeten kunnen zijn.

Het zou dus beter zijn geweest om in het hoofdprogramma niet één van de invoerwaarden als stopcriterium te nemen maar een stopconstructie te maken die onafhankelijk is van de invoerwaarde die eventueel als stopwaarde wordt gekozen. Dit kunnen we met behulp van een waarheids(logische)variabele realiseren.

- In plaats van coderen we
```
150 IF GTAL = -1 THEN 190
150 IF NOT VERDER THEN 190
```
- Nieuwe regel
```
130 N = 0 : SOM = 0 : VERDER = -1
```

De variabele VERDER krijgt als eerste waarde de waarde -1 (waar). In de module 'invoer en controle' krijgt deze variabele dan eventueel de waarde 'niet-waar', zodat de lus in het hoofdprogramma niet meer wordt doorlopen.

In de module 'invoer en controle' maken we, als de gebruiker de juiste stopwaarde intoetst, de variabele VERDER gelijk aan 0 (niet waar). Met deze constructie hebben we het hoofdprogramma onafhankelijk gemaakt van de stopwaarde die we op een bepaald moment kiezen; hetgeen inhoudt dat bij het veranderen van de stopwaarde alleen de module 'invoer en controle' gewijzigd hoeft te worden. Deze module zou er bij een stopwaarde -1 zo uit kunnen zien:

```
1000 'Subroutine invoer en controle
1005 '
1010   INPUT "Uw getal is"; GTAL
1015   '***doe zolang negatief getal ongelijk -1 is ingetoetst
1020   IF GTAL>0 OR GTAL=-1 THEN 1050
1030     INPUT "Alleen positieve invoer"; GTAL
1040   GOTO 1020
1045   '***
1050   IF GTAL=-1 THEN VERDER=0
1055 '
1060 RETURN 'uit invoer en controle
```

Zorg er dus voor dat de **modulen** onderling zo **onafhankelijk** mogelijk zijn. Vermijd ook het onnodig doorgeven van programmavariabelen (ook wel in- en uitvoerparameters genoemd) tussen de diverse modulen.

Behalve dit is er vast nog wel meer aan te merken op de gegeven oplossing van het 'gemiddelde probleem'. In ieder geval denken we dat dit voorbeeld aangeeft hoe je een programmeeropdracht kunt aanpakken en wat de voordelen van netjes programmeren zijn.

Het uitvoeren van de stapsgewijze verfijning van de programmeeropdracht en het formeren van deeltaken tot programmamodulen is ook een kwestie van ervaring. Na een paar keer goed oefenen zullen veel programmeurs wellicht voor kleine programmeeropdrachtjes vrij snel een gemoduleerd programma uit hun mouw schudden. Of dat bij iets ingewikkelder programma's (en in de praktijk zijn ze altijd ingewikkeld) ook het geval is valt echter te betwijfelen.

In ieder geval biedt de hier geschetste Top-Down aanpak een goede basis voor het maken van een goed programma. In de volgende paragraaf zullen we zien hoe we de structuur van een programma in schemavorm kunnen weergeven.

5.4 PROGRAMMA STRUCTUUR DIAGRAM

Een **programmastructuurdiagram**, hierna ook wel **PSD** genoemd, is een schematechniek om de structuur van een programma weer te geven.

Als we netjes programmeren, en dat doen we in dit boek, beperken we ons tot het gebruik van een drietal zogeheten logische programmastructuren, ook wel basisstructuren genoemd. Elk programma is met behulp van deze basisstructuren opgebouwd.

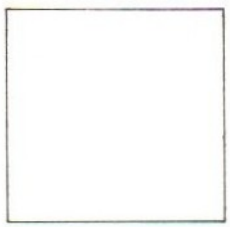
De drie basisstructuren zijn:

1. **De sequentiële structuur**
2. **De herhalingsstructuur**
3. **De keuzestructuur**

Elk van deze drie structuren kunnen we als bouwsteen in een PSD opnemen.

We zullen nu laten zien wat PSD's zijn en hoe ze worden gebruikt. We beperken ons eerst tot een programma dat uit slechts één module, het hele programma dus, bestaat.

Een Programma Structuur Diagram geeft de programmastructuur in een **rechthoek** weer:



dit is nu een PSD!

Het is de bedoeling binnen deze rechthoek de programmastructuur aan te geven.

5.5 PSD EN SEQUENTIELE PROGRAMMASTRUCTUUR

De eenvoudigste programmastructuur is een structuur waarin de opdrachten gewoon na elkaar, van de eerste tot en met de laatste opdracht, moeten worden uitgevoerd. Dit noemen we een sequentiële programmastructuur. Alle programma's uit hoofdstuk 2 hebben

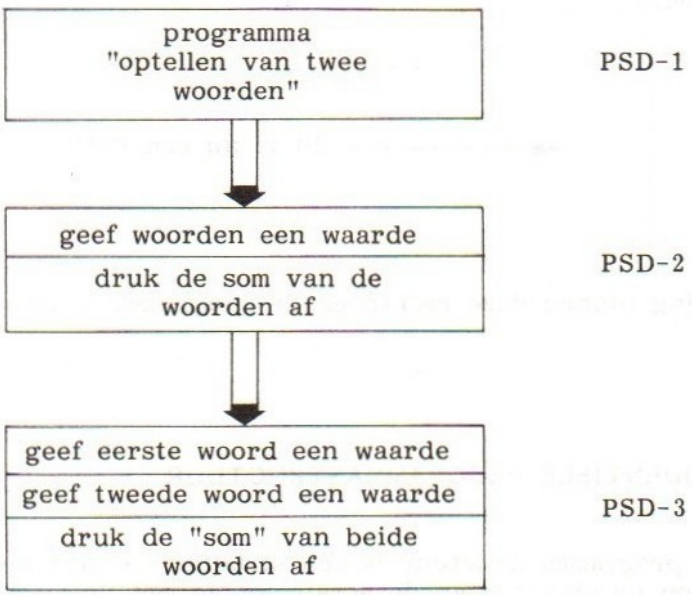
zo'n structuur. Neem nu bijvoorbeeld het programma van bladzijde 34:

```
10 W1$ = "MICRO"
20 W2$ = "COMPUTER"
30 PRINT W1$;" + ";W2$;" geeft ";W1$ + W2$
```

Een PSD voor dit programma ziet er zo uit:

geef eerste woord een waarde
geef tweede woord een waarde
druk de "som" van beide woorden af

Binnen de rechthoek staan nu drie opdrachten die achter elkaar moeten worden uitgevoerd. Dit is een PSD voor een sequentiële programmastructuur. Elke opdracht wordt binnen een PSD als een rechthoekje weergegeven. Nu is een PSD niet alleen bedoeld om de uiteindelijke programmastructuur in weer te geven, maar evenzeer als hulpmiddel om tot die programmastructuur te komen. Zo'n PSD kunnen we in elk stadium van het ontwerpproces gebruiken. Kijk hoe we het verloop van het ontwerpproces van bovenstaand programmaatje in PSD's kunnen weergeven (zie de volgende figuur).



In een **PSD** geven we de **logische structuur** van een programma weer. Een ervaren programmeur zal vanuit PSD-3 direct de BASIC-opdrachten kunnen uitvoeren. We zullen de PSD's niet gebruiken om hierin BASIC-opdrachten te zetten. De PSD's bevatten gewoon een nederlandstalige omschrijving van wat het programma zal moeten doen. Het coderen in BASIC is dan niet moeilijk meer.

We kunnen PSD's dus gebruiken tijdens het ontwerpen van een programma.

Een PSD wordt altijd van boven naar beneden gelezen

De afmetingen van de rechthoeken en van de andere nog te bespreken bouwstenen waarmee een PSD kan worden opgebouwd hebben geen enkele betekenis. De afmetingen worden alleen bepaald door de ruimte die nodig is om de tekst, die we erin willen zetten, kwijt te kunnen!

Als we het helemaal netjes willen doen geven we in een PSD ook het begin en het einde aan van een programma:

begin programma
geef eerste woord een waarde
geef tweede woord een waarde
druk de "som" van beide woorden af
einde programma

opdracht 5.7

Maak een PSD voor de sequentiële structuur van het programma op bladzijde 26. Het diagram moet wel alle afzonderlijke opdrachten laten zien, alleen in gewoon Nederlands en niet in BASIC.

Op naar de volgende bouwstenen.

5.6 PSD EN HERHALINGSSTRUCTUREN

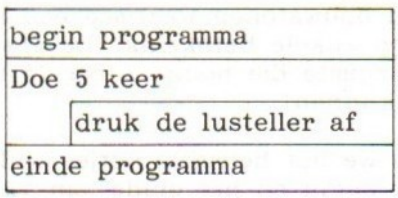
In hoofdstuk 3 hebben we drie herhalingsstructuren behandeld, en wel de

DOE-ZOVEEL-KEER lus (FOR-NEXT) , de
DOE-ZOLANG lus (DO-WHILE , en de
HERHAAL-TOTDAT lus (REPEAT-UNTIL)

Elk van deze structuren wordt op een bepaalde manier in een PSD weergegeven.

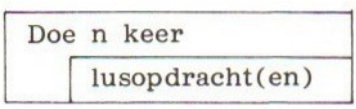
Laten we met **FOR-NEXT** beginnen. Als voorbeeld nemen we een heel eenvoudig programma:

```
10 FOR I = 1 TO 5
20 PRINT I;
30 NEXTI
40 END
```



U ziet het bijbehorende PSD naast de programmatekst weergegeven.

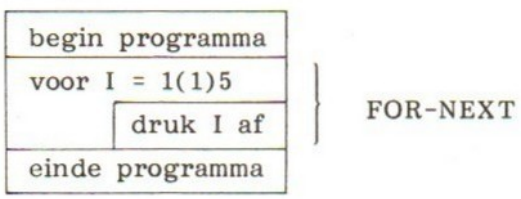
Het stukje PSD dat een FOR-NEXT-herhalingsstructuur aangeeft, ziet er zo uit:



Dat wat n keer gedaan moet worden (de opdrachten in de lus) staat in het rechthoekje binnen de grote rechthoek.

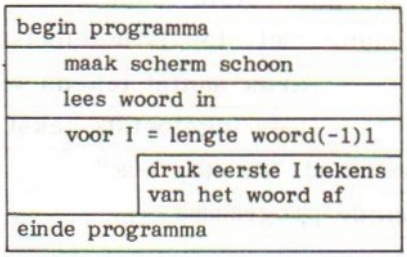
In plaats van 'Doe 5 keer' schrijven we ook wel 'voor I = 1(1)5. Dit betekent: I loopt van 1 tot en met 5 met stapjes van (1).

In gewoon Nederlands zou het PSD voor deze FOR-NEXT programmastructuur er zo uitzien:



Nog een FOR-NEXT-voorbeeld (programma van bladzijde 72 uit § 3.3:

```
10 REM negatieve stapgrootte
20 CLS
30 INPUT "Toets een woord in"; WRD$
40 PRINT
50 FOR I= LEN(WRD$) TO 1 STEP -1
60    PRINT LEFT$(WRD$,I)
70 NEXT I
80 END
```



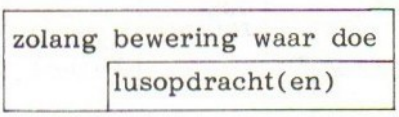
We zien in dit PSD een combinatie van de sequentiële en de herhalingsstructuur en we zien dat de lus nu twee opdrachten bevat. Dit betekent dat de 'binnenrechthoek' nu twee rechthoekjes bevat: een sequentiële structuur binnen een herhalingsstructuur.

opdracht 5.8

Kijken of u met de bouwstenen kunt omgaan. Geef een PSD voor het volgende programma:

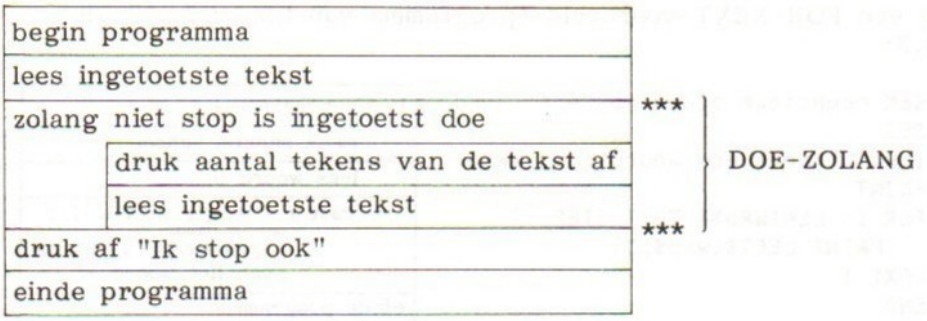
```
10 FOR I = 1 TO 10
20     FOR J = 1 TO 10
30       PRINT I + J;
40       PRINT I * J;
50     NEXT J
60 NEXT I
70 END
```

De **DOE-ZOLANG**-herhalingsstructuur geven we op dezelfde manier weer als de FOR-NEXT-structuur, namelijk zo:



Als voorbeeld nemen we het programma op bladzijde 89 uit § 3.7.

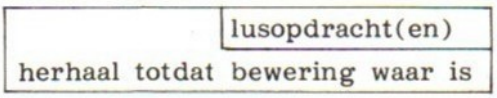
```
10 LINE INPUT "Uw zin is: "; Z$
15 '***doe zolang niet 'stop' is ingetoetst
20 IF Z$ = "stop" THEN 60
30     PRINT "'";Z$; "' is";LEN(Z$);"tekens lang"
40     LINE INPUT "Volgende zin"; Z$
50     GOTO 20
55 '***
60 PRINT "Ik stop ook"
70 END
```



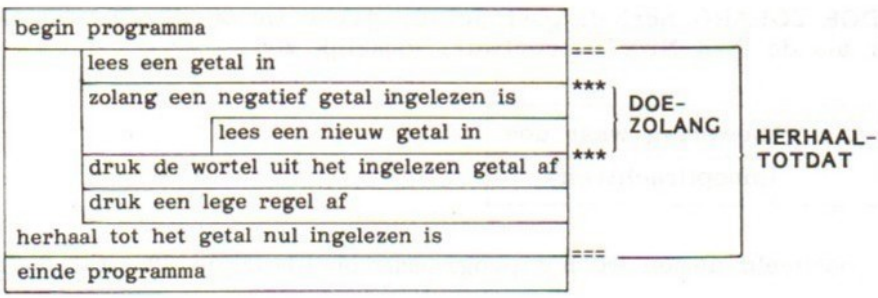
opdracht 5.9

Geef een PSD voor het programma 'kopbal' uit § 3.7 (pagina 91).

De PSD-bouwsteen voor een HERHAAL-TOTDAT-structuur ziet er zo uit:



In het volgende voorbeeld zien we een PSD voor een programma met een HERHAAL-TOTDAT-bouwsteen. Om het meteen maar moeilijk te maken nemen we een programma waar binnen in de herhaal-totdat-lus een doe-zolang-lus zit. Dit is zo bij een programma uit § 3.7 dat de wortel uit ingetoetste getallen afdrukt.



Als we voor een bepaald probleem een PSD opstellen dan zien we in dat PSD alle structuren die we bij het coderen van het programma moeten aanbrengen. In hoofdstuk drie hebben we gezien dat de HERHAAL-TOTDAT- en DOE-ZOLANG-lussen elk geprogrammeerd worden met één IF-opdracht en één GOTO-opdracht. De programmeur weet als hij (of zij) in een PSD zo'n lusstructuur

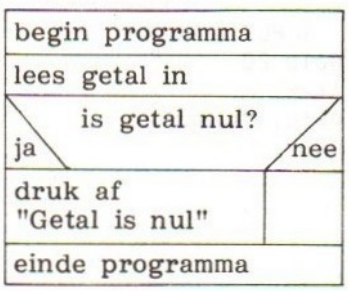
ziet dat in de programmatekst zo'n IF- en een GOTO-opdracht moeten worden opgenomen. Nu wil het geval dat in de volgende paragraaf de bouwstenen voor de IF-opdrachten besproken worden. Het is goed er nu al op te wijzen dat dit de 'echte' IF-opdrachten zijn, dus niet de IF-opdrachten die gebruikt worden voor het coderen van lusstructuren (de lus IF'en). De IF-opdracht speelt dus, zoals we weten, een dubbele rol (althans in onder andere MSX BASIC); enerzijds hebben we 'm nodig om lussen te kunnen coderen, anderzijds moeten wij 'm gebruiken als we in een programma een keuze-structuur moeten coderen. Omdat wij géén PSD's zullen gebruiken met daarin de gecodeerde BASIC-opdrachten zal deze dubbele rol van de IF-opdracht ons niet in de weg staan. Het mag ook eigenlijk geen moeilijkheden opleveren want we hebben immers geleerd in structuren te denken en niet in programma-opdrachten. Als u niet meer weet hoe u het verschil tussen een lus-IF en een echte IF kunt zien kijk dan nog even in paragraaf 4.1.

Een PSD geeft de programmastructuur weer en niet de programma-opdrachten

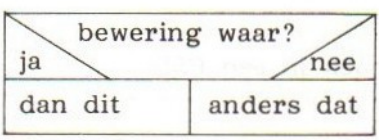
5.7 PSD EN KEUZESTRUCTUREN

Als eerste keuzestructuur nemen we de echte-IF...THEN

```
10 INPUT GTAL
20 IF GTAL = 0
     THEN PRINT "Getal is nul"
30 END
```



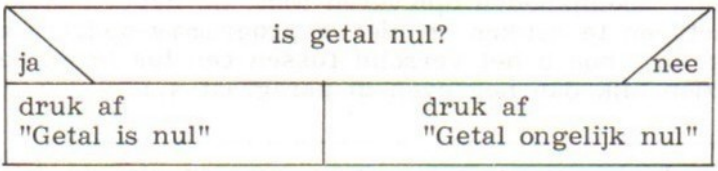
De bouwsteen voor de IF-opdracht ziet er zo uit:



Eigenlijk is dit de IF-THEN-ELSE-bouwsteen. Maar als er geen 'ELSE' is kan gewoon met de volgende programma-opdracht (de eerstvolgende opdracht na de IF-bouwsteen) worden verder gegaan. We laten in zo'n geval gewoon het hokje onder 'nee' leeg. Zouden we echter in bovenstaand programmaatje in regel 20 een IF-THEN-ELSE-constructie gebruiken:

```
IF GTAL = 0
    THEN PRINT "Getal is nul"
    ELSE PRINT "Getal ongelijk nul"
```

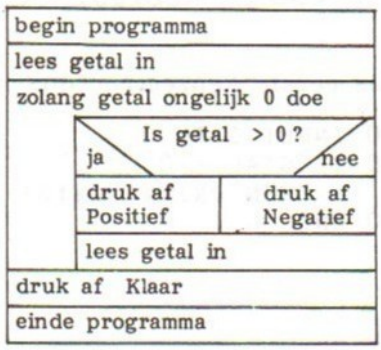
dan zou de IF-bouwsteen er zo uitzien:



In een PSD hebben we dus alleen een bouwsteen voor IF-THEN-ELSE. Voor een IF-THEN-opdracht laten we gewoon het 'neehok' leeg.

Nu een voorbeeld met een DOE-ZOLANG- en een IF-bouwsteen:

```
10 INPUT GTAL
15 '***doe zolang gtal niet nul is
20 IF GTAL=0 THEN 60
30   IF GTAL>0 THEN PRINT "Positief"
                ELSE PRINT "Negatief"
40   INPUT GTAL
50 GOTO 20
55 '***
60 PRINT "Klaar"
70 END
```

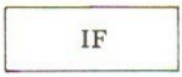


De 'ja'- en 'nee'-tak van een IF-THEN-ELSE eindigen altijd op dezelfde hoogte in een PSD, want altijd moet de volgende opdracht (in dit geval INPUT GTAL) worden uitgevoerd ongeacht of de 'ja'- of de 'nee'-tak is doorlopen.

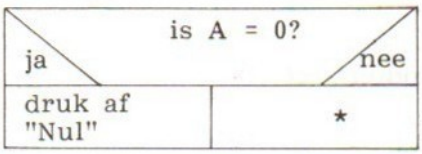
Elke structuur is een rechthoek in een PSD

```
IF A = 0 THEN PRINT "Nul"
         ELSE IF A > 0 THEN PRINT "Positief"
                       ELSE PRINT "Negatief"
```

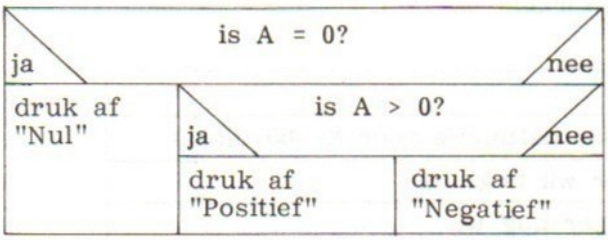
Deze geneste IF-THEN-ELSE-structuur wordt in een PSD voorgesteld als een rechthoek:



Het is een IF-THEN-ELSE dus



De nee-tak is de ELSE-tak. Die tak bevat weer een IF-THEN-ELSE dus in het rechthoekje met dat * komt weer een IF-THEN-ELSE-bouwsteentje, zo eenvoudig is dat!

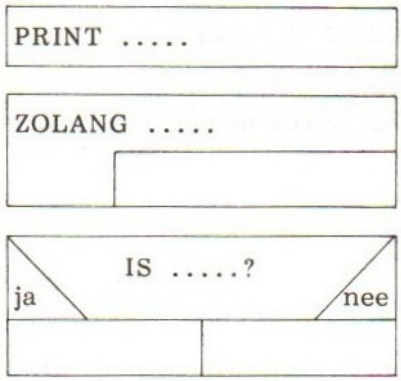


opdracht 5.10

Geef een PSD voor oefenopgave 4 uit hoofdstuk 4.

5.8 PSD EN MODULAIR ONTWERP

Elk PSD bestaat uit een aantal rechthoeken die zelf weer onderdeel kunnen zijn van andere rechthoeken. In principe is iedere structuur een rechthoek. Onder structuur verstaan we gewoon een opdracht of een lus of een keuze-opdracht.

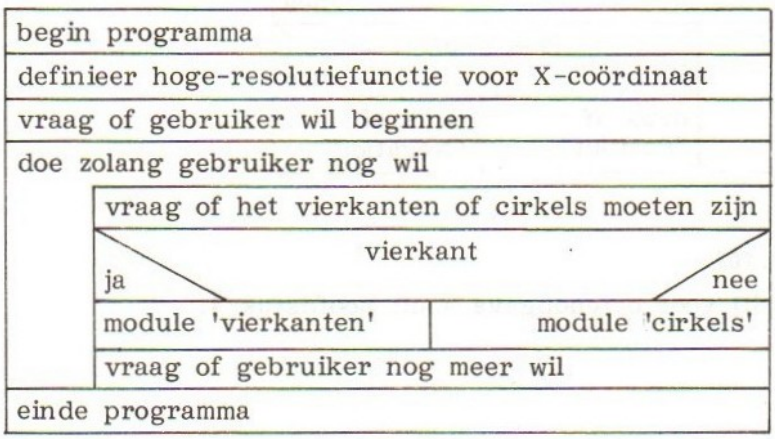


Het aanroepen van een programmamodule met

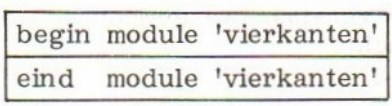
```
GOSUB regelnummer
```

is ook een 'gewone' opdracht, staat dus gewoon in een rechthoekje. Modulen zijn aparte stukjes programma en hebben dan ook aparte PSD's.

We zullen nu een PSD geven voor de hoofdmodule van het programma 'vierkanten of cirkels' uit § 4.4.



Als we per se een PSD voor de module 'vierkanten' zouden moeten geven, ziet die er zo uit (een lege module):



Nu kunnen we ons op de inhoud van de module 'vierkanten' gaan concentreren.

Elke module binnen één programma heeft dus een eigen PSD. Al deze PSD's kunnen onafhankelijk van elkaar ontworpen worden.

opdracht 5.11

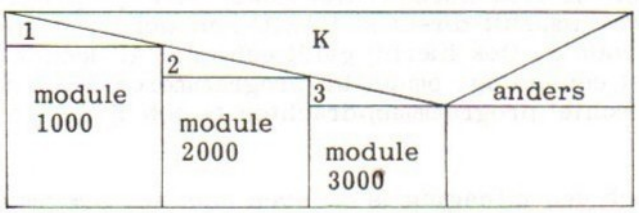
Maak voor het in het begin van dit hoofdstuk (top-down) ontworpen programma (zie bladzijde 135) voor het berekenen van het gemiddelde van N getallen drie PSD's; één voor de hoofdmodule, één voor de module 'invoer en controle' en één voor de module 'sommeer en tel'.
Geef in deze PSD's niet de BASIC-opdrachten zelf maar gewoon 'Nederlandstalige' opdrachtomschrijvingen.

5.9 PSD EN MEERKEUZESTRUCTUUR

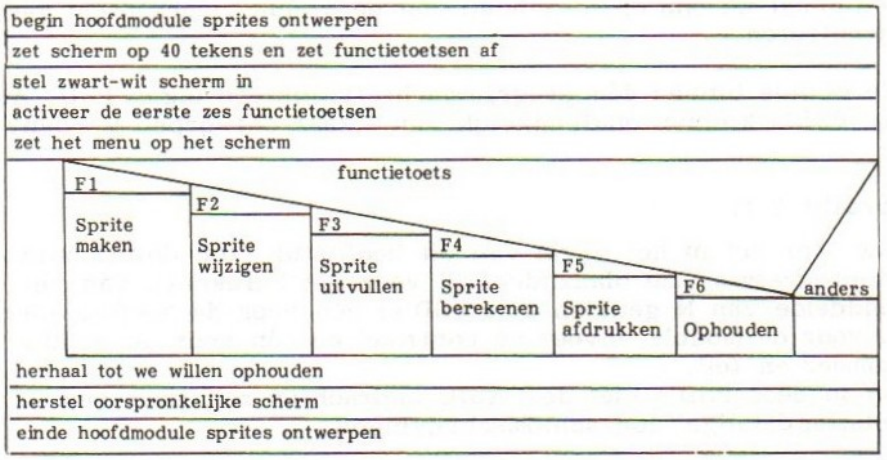
De meerkeuzestructuur is het laten uitvoeren van een bepaalde programmamodule afhankelijk van de waarde van een bepaalde variabele. Een opdracht in BASIC om deze meerkeuzestructuur in een programma te verwezenlijken is bijvoorbeeld:

```
ON K GOSUB 1000, 2000, 3000
```

Als K = 1 dan module op regel 1000 uitvoeren, is K = 2 dan die op regel 2000, tenslotte als K = 3 de module op regel 3000. Als K een andere waarde heeft dan wordt de opdracht onder 'anders' uitgevoerd. In dit geval niets, want het blokje onder 'anders' is leeg.



Een andere meerkeuze-opdracht is de ON KEY GOSUB opdracht. We zullen een PSD geven voor de hoofdmodule van het sprite-ontwerpprogramma uit hoofdstuk 4:



5.10 ENKELE SLOTOPMERKINGEN OVER NETJES PROGRAMMEREN

De PSD techniek is zeer geschikt voor het weergeven van de verschillende verfijningsstappen in een top-down ontwerpproces. Een goed uitgewerkt PSD kan de basis vormen van waaruit direct het programma kan worden gecodeerd. Als we geen echte computertaal-opdrachten in een PSD opnemen, maar de opdrachten in gewoon Nederlands omschrijven, kan vanuit een PSD in elke taal die over de eerder genoemde basisstructuren beschikt worden gecodeerd.

Programma's ontwerpen volgens de top-down methode is beslist niet eenvoudig. Voordat het echte 'programmeren begint zal enig denk- en analysewerk verricht moeten worden. Het maken van een programmastructuurdiagram van waaruit direct in BASIC kan worden gecodeerd is ook niet eenvoudig. Ook hierbij geldt echter: "Al doende leert men". Het is niet eenvoudig, omdat bij programmeren heel snel de behoefte ontstaat 'echte' programmaopdrachten te schrijven, te coderen dus!

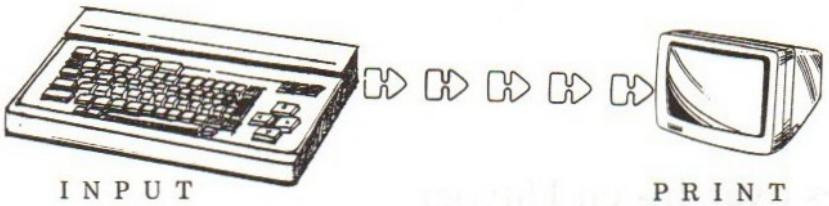
Dit komt omdat het toch een uitdaging is om even snel een programma te coderen voor een probleem dat om een oplossing vraagt. De in dit hoofdstuk aangegeven programmeermethode houdt in dat deze behoefte onderdrukt moet worden omdat volgens deze aanpak coderen juist de laatste stap in het hele ontwerpproces is. Dat het onderdrukken van een behoefte niet altijd even gemakkelijk is, weten we!

6 Alles over In- en Uitvoer

Met de kennis uit de vorige hoofdstukken moeten we nu elke programmeeropdracht aankunnen. Het wordt tijd wat meer aandacht aan MSX BASIC zelf te schenken. In dit hoofdstuk laten we, veelal aan de hand van kleine programmaatjes, zien wat er in MSX BASIC zoal aan mogelijkheden zijn voor het invoeren en afdrukken van gegevens, zowel getallen als tekst!
Aan de orde komen:

- inlezen van gegevens uit eigen programma
- de tabulatorfunctie
- afdrukken van gegevens op een printer
- invoeren van 'moeilijke' teksten
- het maken van een 'nette' uitvoer
- handige functies als hulpmiddel bij het afdrukken
- uitvoering op het hoge resolutiescherm (SCREEN 2)
- toepassing van ON INTERVAL GOSUB
- programmavoorbeeld met veel graphics en geluid.

Het gebruik van cassettebandjes en diskettes als in- en uitvoermedium (externe gegevensopslag) komt in hoofdstuk 9 ter sprake.



Tot nu toe hebben we bijna altijd de gegevens ingevoerd via het **toetsenbord** (INPUT-opdracht). De uitvoer is steeds op het **beeldscherm** afgedrukt (PRINT-opdrachten). Een andere veel gebruikte combinatie van in- en uitvoermethoden is: de invoergegevens in het programma zelf opnemen en de uitvoer op een printer afdrukken. We beginnen met het 'inlezen uit eigen programma'.

6.1 INVOERGEGEVENS ALS ONDERDEEL VAN HET PROGRAMMA

In plaats van het intoetsen van gegevens tijdens het draaien van een programma kan het programma deze gegevens ook uit een, in het programma zelf, opgenomen gegevenslijst inlezen. Hierbij worden de gegevens in een **DATA**-opdracht opgenomen, terwijl met de **READ**-opdracht een gegeven uit deze DATA-lijst kan worden ingelezen. We hebben hiervan in hoofdstuk 3 (kopbal) al een voorbeeld gezien. In onderstaand programma worden 10 getallen uit zo'n DATA-lijst ingelezen:

```
100 REM  Demonstratie READ-DATA
105 '
110 ' Dit programma bepaalt de kleinste en de grootste
115 ' waarde onder de getallen uit de DATA-lijst
120 '
130 MIN = 9999: MAX = -9999
140 CLS
150 PRINT "De Getallen zijn: ": PRINT
160 '
170 FOR I = 1 TO 10
180    READ GTAL' Nu wordt 1 waarde uit de DATA-regel gelezen
190    PRINT GTAL
200    IF GTAL> MAX THEN MAX = GTAL'Nieuw maximum gevonden
210    IF GTAL< MIN THEN MIN = GTAL'Nieuw minimum gevonden
220 NEXT I
230 '
240 PRINT: PRINT
250 PRINT "De kleinste is: "; MIN
260 PRINT
270 PRINT "De Grootste is: "; MAX
280 END
290    DATA 21.6,22.4,23.6,21.3,22.3,21.2,24.2,25.6,27.4,21.4
```

De opdracht 180 **READ GTAL** dwingt de computer een waarde uit de **DATA-lijst** (regel 290) te lezen en deze waarde toe te kennen aan de variabele GTAL. Bij elke doorloop van de lus wordt de volgende waarde uit de DATA-lijst aan GTAL toegekend. De computer gebruikt een **gegevenswijzer** (data pointer) die altijd naar het volgende in te lezen gegeven in de lijst wijst. In bovenstaand voorbeeld worden precies tien getallen ingelezen. De DATA-lijst moet dan ook tenminste tien waarden bevatten.

opdracht 6.1

Wat zou er gebeuren als de DATA-lijst minder getallen bevat dan het programma wil inlezen?

DATA-opdrachten mogen overal in het programma voorkomen. Wij zullen ze altijd na de END-opdracht, een beetje apart dus, opnemen.

Bijna nooit zal een programma worden geschreven voor het inlezen van precies tien getallen. Een programma hoort flexibel te zijn. We laten met READ-DATA hiervan een voorbeeld zien.

```
100 REM  READ-DATA met stopwaarde in DATA-lijst
105 '
110 'Dit programma geeft voor de lijst met getallen in de
115 'DATA regel een lijst met indexcyfers. Het eerste
120 'getal uit de DATA-lijst dient als basis en krijgt index
125 '100. De navolgende getallen worden als indexcyfer
130 'ten opzichte van deze basis afgedrukt
135 '
150 PRINT "Waarde   Indexcyfer": PRINT
160 READ BASIS
170 PRINT BASIS; "         100"
180 READ GTAL
185 '***doe zolang niet 99999 is gelezen
190 IF GTAL=99999  THEN 230
200       PRINT GTAL; "         "; INT(100/BASIS*GTAL + .5)
210       READ GTAL
220 GOTO 190
225 '***
230 END
240       DATA 23,26,34,45,51,59,65,77,89,99999
```

Dit programma leest een DATA-lijst totdat de waarde (99999), de 'stopvlag', gelezen wordt.

opdracht 6.2

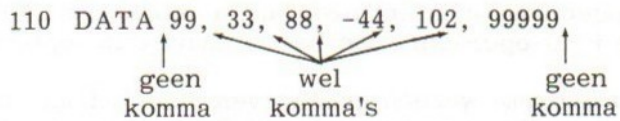
Wat gebeurt er als we het programma draaien indien deze DATA-regel is opgenomen?

```
240  DATA 99999
```

We kunnen met een READ opdracht twee (of meer) waarden tegelijk uit een DATA-lijst inlezen. De READ opdracht ziet er dan bijvoorbeeld zo uit:

```
40   READ A,B
```

Hierdoor worden steeds twee waarden tegelijk uit de data-lijst gelezen. In een data-lijst worden de gegevens door komma's van elkaar gescheiden:



DATA-opdrachten gebruiken we als we een niet-interactief programma willen maken of bij programma's met vaste invoergegevens.

Met een DATA-opdracht kunnen we ook teksten (strings) inlezen of tekst gecombineerd met getallen:

```
10 REM READ-DATA met tekst en getallen
15 '
20 READ NAAM$,LEEFTIJD%
25 '***doe zolang naam niet 'stop' is
30 IF NAAM$ = "stop" THEN 70
40       PRINT NAAM$; " is "; LEEFTIJD%; " jaar oud"
50       READ NAAM$,LEEFTIJD%
60 GOTO 30
65 '***
70 END
80      DATA "Miep", 7, "Hans", 5, "Janneke", 3, "stop", 0
```

opdracht 6.3

Doordat in bovenstaand programma steeds twee waarden tegelijk worden ingelezen moet er na "stop" toch nog een waarde voor LEEFTIJD worden opgenomen, anders krijgen we een "Out of DATA in 50" melding. Herschrijf het programma zó dat wel "stop" als laatste element in de DATA-lijst kan worden opgenomen; de uitvoer van het programma mag niet veranderen.

Pas op! Een DATA-opdracht mag geen variabelen, rekenkundige uitdrukkingen of functies bevatten. Dus dit is allemaal verboden:

```
100  DATA A, B$, 5+4, INT(6*RND(1))
```

We kunnen een DATA-lijst meer dan één keer lezen. We gebruiken daartoe de **RESTORE**-opdracht. Kijk hoe dat werkt:

```
100 REM Rekenles
105 '
110 CLS
120 VERDER=-1
125 '***doe zolang leerling onvoldoende scoort
130 IF NOT VERDER THEN 230
140       JUIST = 0
150       FOR I = 1 TO 10
160          READ A,B
170          PRINT A; " + "; B; " = ";
180          INPUT X
190          IF X = A + B THEN JUIST = JUIST + 1:
                                  PRINT "Goed "
                            ELSE PRINT "Fout"
200       NEXT I
210       IF JUIST <= 5 THEN PRINT "We doen het het nog eens !!":
                             RESTORE
                            ELSE PRINT"Het was voldoende,we stoppen":
                                 VERDER=0
220 GOTO 130
225 '***
230 END
240       DATA 5,7,10,12,15,18,19,26,38,49
250       DATA 50,67,80,91,100,110,120,115,111,125
```

In regel 210 staat achter THEN een RESTORE-opdracht. Hierdoor wordt de data pointer weer naar het begin van de DATA-lijst gebracht, zodat bij de volgende READ weer de eerste twee elementen uit de DATA-lijst worden gelezen.

6.2 DE TAB-ULATORFUNCTIE

Met de **TAB functie** kunnen we in een PRINT-opdracht regelen op welke plaats op een regel de uitvoer wordt afgedrukt. Dat kunnen we echter met **LOCATE** ook. We zullen TAB dan ook alleen gebruiken als echte tabulator, dat wil zeggen voor het afdrukken van een aantal kolommen met gegevens. Tot nu toe hebben we steeds één van de volgende mogelijkheden gebruikt:

```
FOR I = 1 TO 10
    PRINT I
NEXT I
```

10 getallen elk op een nieuwe regel

of:

```
FOR I = 1 TO 10
    PRINT I,
NEXT I
```

2 getallen per regel (standaard-tabulatie)

of:

```
FOR I = 1 TO 10
    PRINT I;
NEXT I
```

10 getallen achter elkaar op één regel

Nu willen we onze eigen tabulatorinstelling kiezen. We gaan nu vier kolommen maken. De eerste kolom bevat de getallen 1 t/m 10, de tweede kolom het kwadraat van deze getallen, de derde kolom de derde macht van de getallen en de vierde kolom....., juist: de vierde macht. De eerste kolom begint aan het begin van de eerste regel (TAB(0)); de tweede kolom op positie 9 (TAB(8)); de derde op positie 17; de vierde op positie 25.

```
10 REM Machten van 10
15 '
20 CLS
30 FOR I=1 TO 10
40     PRINT TAB(0)I;TAB(8)I^2;TAB(16)
           I^3;TAB(24)I^4
50 NEXT I
60 END
```

opdracht 6.4

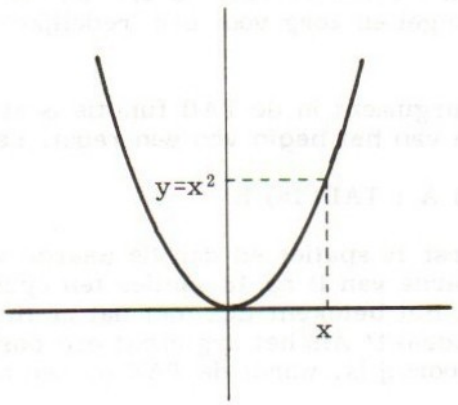
Voeg een aantal opdrachten (met TAB) toe zodat boven elke kolom een toepasselijk 'kopje' komt te staan.

Als we bovenstaand programma draaien zien we deze uitvoer:

```
 1          1          1          1
 2          4          8          16
 3          9          27         81
 4          16         64         256
 5          25         125        625
 6          36         216        1296
 7          49         343        2401
 8          64         512        4096
 9          81         729        6561
 10         100        1000       10000
 ▲          ▲          ▲          ▲
positie 1  positie 9  positie 17 positie 25
```

Elke kolom begint na het aantal spaties in TAB(0,8,16,24). Eigenlijk één positie verder, want voor elk getal wordt een spatie afgedrukt (voor het +-teken).

We mogen in TAB ook als **argument** een **rekenkundige uitdrukking** gebruiken. Zo kunnen we met TAB (op primitieve manier) grafieken van functies tekenen:



Het volgende programma maakt op het beeldscherm dezelfde grafiek, alleen 90° gedraaid.

```
100 REM grafiek van x-->x-kwadraat
105 '
110 CLS
120 PRINT "Grafiek van x-->x^2"
130 PRINT
140 FOR I=-5 TO 5
150    PRINT TAB(2) I;TAB(6+I^2) "*"
160 NEXT I
170 END
```

Hier komt de grafiek (de uitvoer van bovenstaand programma):

```
Grafiek van x-->x^2

 -5                          *
 -4                    *
 -3              *
 -2        *
 -1    *
  0   *
  1    *
  2        *
  3              *
  4                    *
  5                          *
```

Op een printer kunnen we meer punten tekenen, dat zullen we straks zien.

opdracht 6.5

Schrijf een programma voor het tekenen van de grafiek van de wortel uit x ($\sqrt{x}$) voor x = 1,2,3,...,9). Ga uit van een beeldscherm van 40 tekens per regel en zorg voor een 'redelijke' grafiek.

De waarde van het argument in de TAB functie is altijd het aantal spaties ten opzichte van het begin van een regel. Een voorbeeld:

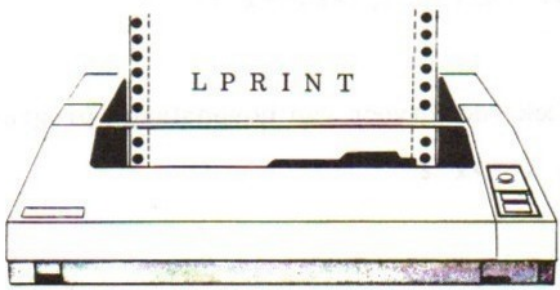
```
PRINT TAB(10) A ; TAB(18) B
```

Het effect is dat eerst 10 spaties en dan de waarde van A wordt afgedrukt en de waarde van B na 18 spaties ten opzichte van het begin van de regel. Het betekent dus niet dat er tussen A en B 18 spaties worden afgedrukt! Als het argument een positie aanwijst waar de cursor al voorbij is, wordt de TAB op een nieuwe regel uitgevoerd. In

```
PRINT TAB(10) A ; TAB(5) B
```

wordt de waarde van B op een nieuwe regel na 5 spaties afgedrukt; dus niet op dezelfde regel als de waarde van A.

6.3 HET GEBRUIK VAN EEN PRINTER



Het afdrukken van gegevens op een printer kost één letter meer in de opdracht dan het afdrukken van gegevens op een beeldscherm.

PRINT	drukt af op een beeldscherm
LPRINT	drukt af op een printer

In het sprite-ontwerpprogramma aan het einde van hoofdstuk 4 hebben we LPRINT al gebruikt.

Een van de verschillen tussen een beeldscherm en een printer is het aantal tekens dat op één regel kan worden afgedrukt. Bij beeldschermen zien we vaak 40 of 80 tekens per regel, bij printers dikwijls 80, 96 of 132. Hiermee moeten we rekening houden als we een programma dat uitvoer op een beeldscherm geeft willen aanpassen om de uitvoer naar een printer te sturen.

Stel we hebben het volgende BASIC-programma:

```
10 REM Demonstratie regelbreedte
15 '
20 FOR GTAL = 1 TO 16
30    PRINT GTAL,
40 NEXT GTAL
50 END
```

Wat zien we als we dit programma draaien op een MSX-computer met een beeldschermregelbreedte van 40 tekens?

```
run
 1             2
 3             4
 5             6
 7             8
 9             10
 11            12
 13            14
 15            16
Ok
```

Standaardtabulatie om de 14 tekens

We gaan nu de uitvoer naar een printer sturen. Hiertoe moeten we regel 30 veranderen in:

```
30  LPRINT GTAL
```

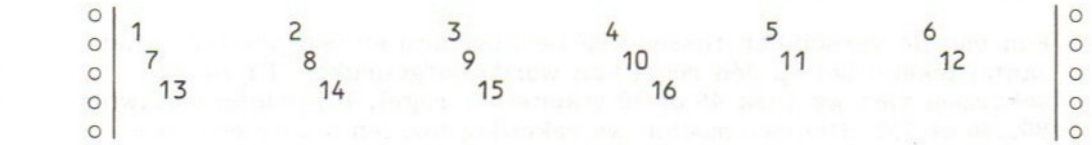
de L van Printer (sorry, Line printer)

We zorgen dat de printer aangesloten is en 'aan' staat en draaien het programma nog eens:

```
RUN
Ok
```

de uitvoer komt niet meer op het scherm!

Op het papier is nu het volgende afgedrukt:



We zien dat de uitvoer op de printer niet netjes onder elkaar wordt afgedrukt. We moeten de regelbreedte (rechtermarge) op onze printer zo instellen dat de getallen wel netjes onder elkaar worden afgedrukt. Dit hangt echter af van de printer die u gebruikt. Voor onze EPSON-printer is deze opdracht:

```
LPRINT CHR$(27); "Q"; CHR$(84)
```

CHR$(27) is een zogenaamde ESCape code waarmee we de printer vertellen dat daarachter een aantal codes volgen die ervoor zorgen dat van de standaardinstelwaarden van de printer wordt afgeweken. Bij de EPSON betekent "Q" de code voor de rechterkantlijn (right margin) ofwel de regelbreedte. Met CHR$(84) stellen we deze regelbreedte op maximaal 84 tekens. Kijk in de handleiding van uw printer hoe u de regelbreedte kunt instellen. Dat zal bijna zeker eerst

```
LPRINT CHR$(27);
```

zijn, gevolgd door één of meer codes.

Wilt u een programma, bijvoorbeeld uit een van de eerste vier hoofdstukken uit dit boek, net zo op de printer afdrukken als u het op het 37-tekens brede scherm van SCREEN 0 hebt ingetikt dan moet u de printerregelbreedte eerst op 37 tekens instellen. Dit gaat, althans voor de EPSON-printers, als volgt:

```
LPRINT CHR$(27); "Q"; CHR$(37)
```

Vervolgens geeft u LLIST en een druk op RETURN en de programmatekst zal op papier verschijnen, precies zo alsof u LIST voor een schermafdruk op het scherm zou hebben gegeven.

LIST	drukt programmatekst op het beeldscherm af
LLIST	drukt programmatekst op de printer af

Een afdruk op de printer van het programma 'Rekenles eerste klas' uit hoofdstuk 4 zou er zonder de LPRINT CHR$(27); "Q"; CHR$(37) op een EPSON-printer zo uitzien; LLIST:

```
100 REM Rekenles eerste klas
105 '
110 N=0
120 FOR SOMMETJE=1 TO 10
130     A%=INT(9*RND(1)+1)
140     B%=INT(9*RND(1)+1)
150     PRINT A%; "+"; B%; "=";:          INPUT ANTW%
160     IF ANTW%=A% + B%                    THEN PRINT "Goed!" : N=N+1
    ELSE PRINT "FOUT!!"
170     PRINT "Druk op RETURN voor de"          ;" volgende som";
180     INPUT A$
190 NEXT SOMMETJE
200 PRINT
210 PRINT "De score is"; N; "van de";        " tien goed"
220 IF N <= 5                              THEN PRINT "Dit is onvoldoende"
ELSE PRINT "Dit is voldoende"
230 END
```

Op het scherm stonden THEN en ELSE mooi onder elkaar, maar op het papier niet. We zullen in het vervolg de programma's zo intikken op de computer, dat ze op papier netjes worden afgedrukt. We hebben dit al bij de programma's in dit hoofdstuk gedaan. Het gemakkelijkste is om uw scherm bij het intikken van de programma's op 40 tekens te zetten (WIDTH 40) en om uw papier op maximaal 80 tekens in te stellen (CHR$(27); "Q"; CHR$(80)). Een printerregel komt dan precies overeen met twee beeldschermregels. Bij het intikken van uw programmatekst weet u dan dat de tekst in twee volle beeldschermregels op één volle printerregel wordt afgedrukt. Gebruikt u een MSX 2-computer en hebt u een monitor waarop 80 tekens per regel afgedrukt kunnen worden, schakel dan vanaf nu over naar de 80-kolommenstand.

opdracht 6.6

Nu eens geen programmeeropdracht, maar een stukje computerbesturing. Een programmatekst moet worden afgedrukt op een printer met 80 tekens per regel. De programmatekst is ingetoetst op een beeldscherm van 40 tekens per regel. Na het afdrukken van de programmatekst moet het programma worden gedraaid. Het programma drukt een aantal gegevens op de printer af. Deze uitvoer gebruikt regels van 64 tekens. Op het beeldscherm zien we nog net de laatste END-opdracht van het programma en de BASIC-prompt:

```
150 END
Ok
```

Geef nu aan welke vier besturingsopdrachten achtereenvolgens moeten worden ingetoetst om bovenstaande opdrachten zo uit te voeren, dat de programmatekst en de uitvoer 'netjes' op de printer worden afgedrukt. Neem de regelbreedte-instelling voor EPSON-printers (zie pagina 180).

Nu laten we het programma 'grafiek van x --> x ∧ 2' zien dat op de printer een bredere tekening kan maken. Daaronder zien we het resultaat.

```
100 REM grafiek van x-->x-kwadraat
105 '
110 CLS
120 LPRINT "Grafiek van x-->x^2"
130 LPRINT
140 FOR I=-8 TO 8
150    LPRINT TAB(2) I;STRING$(I^2+1,".");TAB(6+I^2) "*"
160 NEXT I
170 END
```

```
Grafiek van x-->x^2

 -8 ..............................................................*
 -7 ..........................................................*
 -6 ......................................................*
 -5 ..................................................*
 -4 ..............................*
 -3 ..........*
 -2 .....*
 -1 ..*
  0 .*
  1 ..*
  2 .....*
  3 ..........*
  4 ..............................*
  5 ..................................................*
  6 ......................................................*
  7 ..........................................................*
  8 ..............................................................*
```

6.4 MOEILIJKE INVOER

In deze paragraaf laten we zien dat we met de INPUT opdracht niet zomaar 'alles' kunnen invoeren. Met moeilijke invoer bedoelen we invoer die niet met de gewone INPUT opdracht ingelezen kan worden. Bekijk het volgende, heel eenvoudige, programmaatje:

```
10 REM Moeilijke invoer
15 '
20 CLS
30 '===
40  LINE  INPUT "Invoerstring=";S$
50   PRINT S$
60 IF S$ <> "stop" THEN 40
70 '===herhaal tot stop is ingetoetst
80 END
```

We gaan het draaien en het een en ander invoeren. We kunnen nu een string, al dan niet tussen aanhalingstekens, intikken. De string wordt afgedrukt en we kunnen een nieuwe string invoeren:

```
RUN
Invoerstring=? "PIET"
PIET
Invoerstring=? "AMST1-B07"
AMST1-B07
Invoerstring=? JAN EN PIET
JAN EN PIET
Invoerstring=? "JAN, DIE LIEVERD"
JAN, DIE LIEVERD
Invoerstring=? JAN, DIE LIEVERD
?Extra ignored
JAN
Invoerstring=? """"
?Redo from start
Invoerstring=?
```

Er zijn dus situaties waarin de computer een bepaalde string niet accepteert, met name strings met "-tekens en strings met komma's zonder aanhalingstekens. Als we willen dat de computer alles accepteert moeten we

LINE INPUT

gebruiken in plaats van INPUT. Dus

```
40  LINE INPUT "Invoerstring = ", S$
```

en we kunnen alles invoeren, maximaal 255 tekens per string. De string wordt afgesloten met RETURN. De computer zal nu JAN, DIE LIEVERD en "" als strings accepteren en afdrukken.

Denk erom dat als we nu "PIET" intoetsen ook "PIET" wordt afgedrukt en niet PIET. Bij LINE INPUT hebben de "-tekens geen speciale betekenis! LINE INPUT zet geen ? op het scherm!

opdracht 6.7

Schrijf een programma waarmee we de computer als tikmachine kunnen gebruiken. Het programma leest steeds een zin die we intoetsen in en drukt deze zin op de printer af. We moeten in zo'n zin alles kunnen intikken wat we willen (ook komma's). Als we op de RETURN-toets drukken wordt de ingetoetste zin ingelezen en afgedrukt. Als we # als zin (nou ja... zin?) intoetsen en op RETURN drukken, drukt de computer een blanco regel op de printer af en als we ! en RETURN indrukken wordt het programma beëindigd.

6.5 GETALLEN NETJES AFDRUKKEN

Om het afdrukken van getallen op het beeldscherm (PRINT) en op de printer (LPRINT) nog wat te verfraaien kunnen we gebruik maken van respectievelijk PRINT USING en LPRINT USING. Wat voor PRINT USING geldt, geldt ook voor LPRINT USING en andersom. In de voorbeelden zullen we dan ook maar één van de twee, PRINT of LPRINT, gebruiken.

Kijk eens naar de uitvoer van het volgende programma:

```
10 FOR I=0 TO 4
20     PRINT 10^I
30 NEXT I
40 END
RUN
 1
 10
 100
 1000
 10000
```

niet zo fraai! Wij zijn gewend de eenheden, tientallen, honderdtallen, enz. precies onder elkaar te zetten

We kunnen de getallen ook zo laten afdrukken:

```
    1
   10
  100
 1000
10000
```

Maar dan moeten we regel 20 als volgt schrijven:

```
20   PRINT USING "#####"; 10 ^ I
```

De string met hekjes (#) achter PRINT USING heet een **formatstring**. Het is een aanwijzing voor de computer hoe hij de waarden die in de PRINT opdracht staan moet afdrukken. De formatstring "#####" wil zeggen dat we de waarden als gehele getallen van maximaal vijf cijfers willen afdrukken. Elk #-je geeft een cijferpositie aan. Heeft een af te drukken waarde minder cijfers dan er hekjes in de formatstring staan dan vult de computer de eerste posities (van links af) met spaties.

opdracht 6.8

Druk een drietal getallen af. Hoe ziet de invoer eruit?

```
                                    positie  1   2   3   4
                                             ↓   ↓   ↓   ↓
a.   PRINT USING "####";5                    .   .   .   .
b.   PRINT USING "####";5897                 .   .   .   .
c.   PRINT USING "####";83.66                .   .   .   .
                        ↑                    .   .   .   .
                     pas op
```

De PRINT USING opdracht rondt de af te drukken waarde af op het laatste hekje (cijferpositie) uit de formatstring. Hier volgt nog een aantal voorbeelden.

- We kunnen de plaats aangeven waar de decimale komma (sorry, punt) moet worden afgedrukt, bijvoorbeeld twee cijfers voor en twee achter de komma:

  ```
  PRINT USING "##.##";A
  ```

- We kunnen meer dan één waarde met een PRINT USING opdracht afdrukken:

  ```
  PRINT USING "###"; A,B,C
  ```

  Maar pas op, nu staan er geen spaties meer tussen de getallen zoals bij PRINT A;B;C. De spaties moeten nu in de formatstring worden opgenomen:

  ```
  PRINT USING " ###" ; A,B,C
  ```

- Bij de 'gewone' PRINT A opdracht wordt voor A = 0.75 afgedrukt:

  ```
  .75     (zonder 0 ervoor!)
  ```

  Willen we perse wel een nul hebben dan kan het bijvoorbeeld zo: PRINT USING "##.##"; A. Dan krijgen we:

  ```
  0.75
  ```

opdracht 6.9

Geef aan hoe de uitvoer er uitziet als A = 16.25.

```
a.   LPRINT USING "##.##" ; A
b.   LPRINT USING "##.###" ; A
c.   LPRINT USING "##.#" ; A
d.   LPRINT USING "##.##  " ; A, 2*A, 3*A
                          ↑
                      2 spaties
```

6.6 MOGELIJKHEDEN MET DE FORMATSTRING IN PRINT USING

In een formatstring kunnen we nog veel meer aangeven dan alleen hekjes, punten en spaties. In deze paragraaf geven we een opsomming van mogelijkheden. We gebruiken nu eens LPRINT voor de verandering. In de voorbeelden drukken we steeds één of meer getallen af. De uitvoer van de opdracht staat direct onder de opdracht zelf.

+ teken

We kunnen voor- of achteraan de formatstring een + teken opnemen. Hierdoor wordt voor of achter het getal een + of - teken afgedrukt.

```
LPRINT USING "+##.##";-68.95, 2.4, 55.6, -.9
uitvoer: -68.95 +2.40 +55.60 -0.90
```

- teken

Een minteken achteraan de formatstring zorgt voor het afdrukken van een minteken achter een negatief getal.

```
LPRINT USING "##.##- "; -68.95, 22.449, -7.01
uitvoer: 68.95- 22.45 7.01-
```

**** dubbel sterretje**

Twee sterretjes voor de formatstring zorgen voor het afdrukken van een aantal sterretjes voor de getallen (we zien dat wel bij bedragen op bankcheques).

```
LPRINT USING "**#.## "; 2.50, 11.75, 180.55
uitvoer: **2.50 *11.75 180.55
                       ↑ geen ruimte meer voor sterretjes!
```

tekst bij de uitvoer

Soms is het handig voor een getal (of erna) wat tekst of gewoon een teken af te drukken; bijvoorbeeld het guldenteken (F) of gewoon GLD:

```
LPRINT USING "F###.##" ; 15.85
uitvoer: F 15.85
```

of

```
LPRINT USING "###.##GLD"; 15.85
uitvoer: 15.85GLD
```

Wilt u voorkomen dat met de uitvoer nog geknoeid kan worden dan:

```
LPRINT USING "F**###.##"; 15.85
uitvoer: F***15.85
```

of

```
LPRINT USING "**###.##GLD"; 15.85
uitvoer: ***15.85GLD
```

We mogen in een formatstring meer dan één formatveld opnemen. In het onderstaande programma zien we hoe we met een PRINT USING opdracht drie waarden kunnen afdrukken, elk met een eigen formatveld. Onder het programma zien we deze uitvoer.

```
100 REM          <<<< Valuta programma >>>>
110 '
120 'Dit programma berekent voor diverse bedragen in guldens
130 'het bedrag in duitse marken en in zwitserse franken
140 '
150 PRINT
160 INPUT "Koers Duitse Mark      "; DM
170 INPUT "Koers Zwitserse Frank  "; ZF
180 CLS
190 LPRINT "Een Gulden is "; DM; "Duitse Mark (DM)"
200 LPRINT "Een Gulden is "; ZF; "Zwitserse Frank  (ZwFr)"
210 LPRINT
220 FOR I = 1 TO 10
230    READ GULDENS
240    LPRINT USING "####.## Gld   ####.## DM   ####.## ZwFr";
                    GULDENS,  DM*GULDENS,  ZF*GULDENS
250 NEXT I
260 END
270     DATA 1,2.5,5,10,25,50,100,250,500,1000
```

```
Een Gulden is  .877 Duitse Mark (DM)
Een Gulden is  .743 Zwitserse Frank  (ZwFr)

   1.00 Gld      0.88 DM      0.74 ZwFr
   2.50 Gld      2.19 DM      1.86 ZwFr
   5.00 Gld      4.39 DM      3.72 ZwFr
  10.00 Gld      8.77 DM      7.43 ZwFr
  25.00 Gld     21.93 DM     18.58 ZwFr
  50.00 Gld     43.85 DM     37.15 ZwFr
 100.00 Gld     87.70 DM     74.30 ZwFr
 250.00 Gld    219.25 DM    185.75 ZwFr
 500.00 Gld    438.50 DM    371.50 ZwFr
1000.00 Gld    877.00 DM    743.00 ZwFr
```

komma

Amerikanen gebruiken een , om in een groot aantal de duizendtallen aan te geven. Wij gebruiken daarvoor juist de ., terwijl de Zwitsers een ' gebruiken (over standaards gesproken).

```
LPRINT USING "####.,##"; 5687.50
uitvoer: 5,687.50
```

`^^^^` Vier pijlpuntjes achteraan de formatstring zorgen voor het afdrukken van getallen in wetenschappelijke notatie. Zo is 123 gelijk aan $1,23 \times 10^2$ of zoals de computer het doet 1.23E+02:

```
LPRINT USING "##.##^^^^"; 234.56
uitvoer: 2.35E+02
```

of

```
LPRINT USING "+.##^^^^"; 123
uitvoer: +.12E+03
```

`%` Tot slot: het kan voorkomen dat het af te drukken getal 'langer is dan het in de formatstring opgegeven veld. De computer drukt dan een %-teken voor het getal af.

```
LPRINT USING "##"; 5434
uitvoer: %5434
```

opdracht 6.10

Wat drukt de computer af? A = 45.56.

a. PRINT USING "###.##"; A
b. LPRING USING "+###.##"; A
c. PRINT USING "**##.##"; A
d. LPRING USING "**##.#GLD"; A

Hoe ziet de formatstring er uit? B = 9.68.

e. PRINT USING "....."; B uitvoer: 9.68
f. LPRINT USING ".....";B uitvoer: F***9.68
g. PRINT USING "....."; B uitvoer: +9.680
h. PRINT USING "....."; B uitvoer: %9.68

(L)PRINT USING kunnen we ook gebruiken voor het afdrukken van tekst.

`"!"` De formatstring "!" zegt: druk alleen het eerste teken af. Stel A$ = "ABCDEFG"

```
PRINT USING "!"; A$
uitvoer: A
```

"\aantal spaties\" Als we tussen schuine strepen (\ \) in de formatstring een aantal spaties zetten, wordt van de string of stringvariabele(n) uit de PRINT opdracht een aantal tekens afgedrukt dat gelijk is aan twee plus het aantal spaties in de formatstring, dus minimaal 2. Voorbeeld:

```
                    A$ = "ABCDEFG"

geen spaties → PRINT USING "\\"; A$ ⇒ uitvoer: AB
1 spatie     → PRINT USING "\ \"; A$ ⇒
                    uitvoer: ABC
5 spaties    → PRINT USING "\      \"; A$ ⇒
                    uitvoer: ABCDEFG
```

Is de string 'korter' dan het formatveld dan wordt eerst de string afgedrukt, aangevuld met spaties.

opdracht 6.11

De variabele INIT$ bevat de voorletters van een persoon. De variabele NAAM$ bevat de achternaam van die persoon.

a. Schrijf opdrachten om alleen de eerste voorletter, gevolgd door een spatie en de achternaam af te drukken.

b. Schrijf opdrachten om alle voorletters (maximaal 5), gevolgd door een spatie en de achternaam af te drukken. Als daaronder de voorletters en achternaam van een volgend persoon worden afgedrukt (met dezelfde opdrachten) moeten de achternamen op dezelfde hoogte beginnen. Voorbeeld:

```
A    VAN KAMPEN
BJM  HUISMAN
```

6.7 HANDIGE FUNCTIES BIJ HET AFDRUKKEN

In deze paragraaf geven we nog een aantal handige functies voor het verkrijgen van een mooie 'layout' van de uitvoer.

SPACE$(X)

Deze opdracht geeft een string van X spaties. X moet liggen tussen 0 en 255. Voorbeeld:

```
10 FOR I = 1 TO 5
20     X$ = SPACE$(I)
30     PRINT X$; I
40 NEXT I
50 END
RUN
 1
  2
   3
    4
     5
Ok
_
```

SPACE$ →

SPC(I)

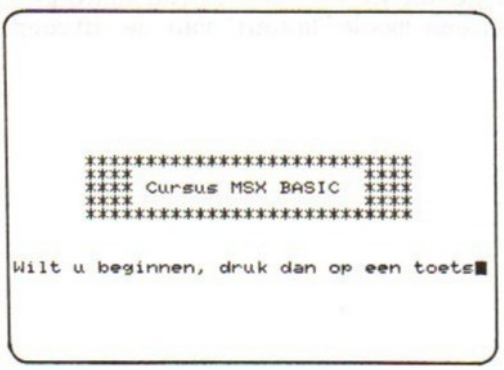
Deze functie drukt I spaties af. Deze functie mag alleen worden gebruikt bij PRINT of LPRINT. opdrachten. I moet liggen tussen 0 en 255. Voorbeeld: PRINT A$; SPC(15); B$. Dit geeft 15 spaties tussen het laatste teken uit A$ en het eerste teken uit B$.

STRING$(I,J)
of
STRING$(I,X$)

Geeft een string ter lengte I gevuld met tekens met ASCII code J of allemaal tekens gelijk aan het eerste teken uit X$. Voorbeeld:

```
10 X$ = STRING$(10,45)
                    ↑ ASCII code voor een
                      streepje (-)
20 PRINT X$; "WEEK-RAPPORT"; X$
30 END
RUN
----------WEEK-RAPPORT----------
Ok
```

Laten we deze drie hulpmiddelen eens gaan gebruiken. Als begin van een computercursus wordt het volgende op het scherm afgedrukt:



Deze aankondiging beslaat vijf beeldschermregels. Op elke regel worden eerst 20 spaties afgedrukt. De module om dit te maken zou er zo uit kunnen zien:

```
1000 ' Subroutine opening
1005 '
1010     CLS:KEY OFF:WIDTH 40
1020     LOCATE ,10
1030     X$=STRING$(27,"*")
1040     Y$=STRING$( 4,"*")
1050     TEKST$ = " Cursus MSX BASIC  "
1060     PRINT TAB(6) X$
1070     PRINT TAB(6) Y$; SPC(19); Y$
1080     PRINT TAB(6) Y$; TEKST$ ; Y$
1090     PRINT TAB(6) Y$; SPC(19); Y$
1100     PRINT TAB(6) X$
1110     PRINT:PRINT:PRINT
1120     PRINT "Wilt u beginnen, druk dan op een toets";
1130     DUMMY$=INPUT$(1)
1135 '
1140 RETURN 'uit opening
```

In regel 1130 staat de opdracht

```
DUMMY$ = INPUT$(1)
```

Het effect van INPUT$(1) is dat het programma stopt en pas verder gaat als er één willekeurige toets wordt ingedrukt. De ingedrukte toets wordt niet op het beeldscherm afgedrukt en er hoeft ook geen RETURN gegeven te worden. Zodra de toets is ingedrukt loopt het programma verder.

Voor het afdrukken van gedeelten van een string zijn ook een paar handige hulpmiddelen. Twee hiervan kennen we al.

**LEFT$** Met de functie LEFT$ kunnen we een aantal tekens uit de 'linkerkant' van een string afdrukken.
Voorbeeld:

```
PRINT LEFT$("CURSUS MSX BASIC",6)
```

uitvoer: CURSUS

Stel in een programma, dat in een internationale organisatie wordt gebruikt, wordt aan het einde van een DOE-ZOLANG lus de volgende opdracht opgenomen:

```
150        INPUT "ONCE MORE? (YES/NO)
                  ENCORE UNE FOIS? (OUI/NON)
                  NOCH EINMAL? (JA/NEIN)
                  NOG EENS? (JA/NEE)  "; A$
160 GOTO 110
```

De IF-opdracht die moet beslissen of de lus nog eens zal worden doorlopen zou er dan zo uit kunnen zien:

```
110 IF LEFT$(A$,1) = "N" THEN 170
```

Logischer (maar langer) is dit:

```
110 IF A$ <> "YES" AND A$ <> "OUI" AND A$ <> "JA" THEN 170
```

**RIGHT$** RIGHT$ haalt een aantal tekens uit de rechterkant van een string. Voorbeeld:

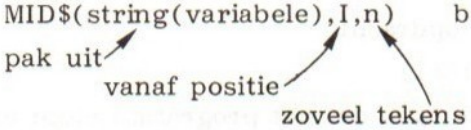
```
PRINT RIGHT$("CURSUS MSX BASIC",9)
```

uitvoer: MSX BASIC

MID$ Deze functie zijn we nog niet tegengekomen. Met deze functie kunnen we een stuk MIDden uit een string halen. Voorbeeld:

```
PRINT MID$("CURSUS MSX BASIC",8,3)
```

uitvoer: MSX



VAL Een string kan uit letters, leestekens en cijfers bestaan. Willen we met cijfers die in een string voorkomen kunnen rekenen, dan moeten deze cijfers niet als 'string' maar als getal beschouwd worden. Cijfers in een string staan opgeslagen als ASCII codes. Cijfers in een getal staan anders opgeslagen. Met de functie VAL kunnen we een string met cijfers omzetten in een getal. Dit is handig want het is soms gemakkelijker getallen als string in te lezen dan als echte getallen.

Voorbeeld:

```
TEKST$ = "25"
X = VAL(TEKST$)
PRINT 3*X
```

uitvoer: 75

Tot slot van deze paragraaf een 'praktijkvoorbeeld' met MID$ en VAL erin verwerkt.

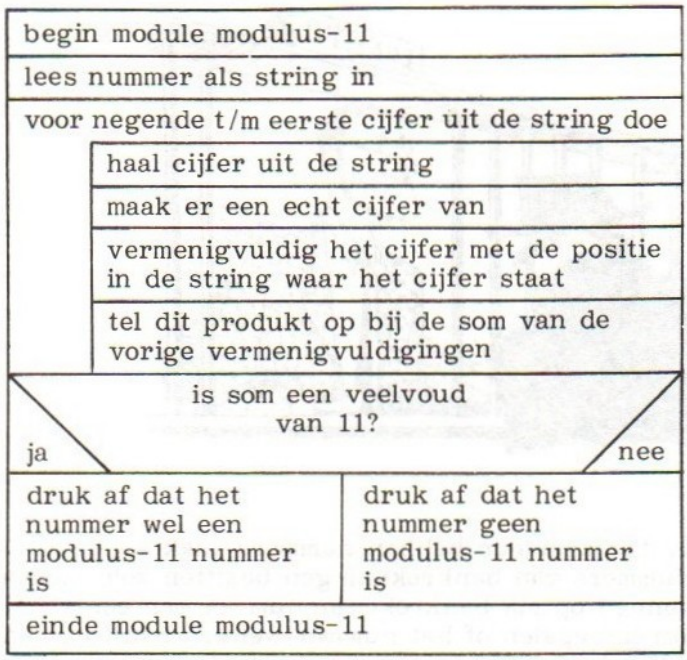
Bij administratieve toepassingen hebben nummers vaak bepaalde eigenschappen. Nummers van bankrekeningen bezitten zo'n speciale eigenschap. We kunnen op elk bankrekeningnummer een controlefoefje toepassen om te bepalen of het nummer wel een bankrekeningnummer kan zijn. Een bankrekeningnummer bestaat uit negen cijfers. Als we nu het negende cijfer (van rechts af gezien) met 9 vermenigvuldigen, het achtste met 8, ..., en het eerste (het meest rechtse cijfer) met 1 en als we al deze produkten optellen, dan is de uitkomst altijd een veelvoud van 11. Een nummer dat aan deze regel voldoet heet een modulus-11 nummer.

Hier is een bankrekeningnummer:

3	6	5	9	6	5	5	4	5	
×	×	×	×	×	×	×	×	×	
9	8	7	6	5	4	3	2	1	
27	+ 48	+ 35	+ 54	+ 30	+ 20	+ 15	+ 8	+ 5	= 242

En ... 242 is 22 × 11, dus een veelvoud van 11.

Programma's bij banken behandelen bankrekeningnummers gewoon als tekst, niet als getallen. Laten we eens een module schrijven in BASIC waarmee de bank kan controleren of een nummer een bankrekeningnummer is. De structuur van de module ziet er zo uit:



In BASIC ziet de module er zo uit:

```
1000 ' Subroutine modulus-11
1010 '
1020     SOMPROD = 0
1030     INPUT "Uw nummer "; NUMMER$
1040     FOR I = 9 TO 1 STEP -1
1050        CIJFER$ = MID$(NUMMER$,10-I,1) 'Pak i-de cijfer
1060        WAARDE = VAL(CIJFER$) ' Maak er een echt cijfer van
1070        SOMPROD = SOMPROD + WAARDE * I ' Vermenigvuldig met positie
1080     NEXT I
1090     IF SOMPROD - INT(SOMPROD/11) * 11 = 0
           THEN PRINT NUMMER$;" Is een modulus 11 nummer"
           ELSE PRINT NUMMER$;" Is geen modulus-11 nummer"
1095 '
1100 RETURN ' Modulus 11
```

wat wij het negende cijfer noemen, noemt MID$ het eerste cijfer.

we hebben steeds maar 1 cijfer nodig.

opdracht 6.12

a. Leg uit waarom de uitdrukking
SOMPROD - INT(SOMPROD/11)*11 ongelijk nul is als SOMPROD geen veelvoud is van 11.

b. Het gebruik van de variabelen WAARDE en CIJFER$ is niet nodig. Maak van de regels 1050, 1060 en 1070 één opdracht
1070 SOMPROD = SOMPROD + ...

c. Wat moet er veranderen opdat deze routine geschikt is voor getalstrings van willekeurige lengte?

Wat we allemaal nog meer met tekst kunnen doen bewaren we tot hoofdstuk 8.

6.8 UITVOER OP HET GRAFISCHE SCHERM, EEN SIMULATIEPROGRAMMA

Tot nu toe hebben we op het grafische scherm (SCREEN 2) alleen getekend. We kunnen er echter ook tekst en getallen op afdrukken, al moeten we hiervoor een kleine omweg maken. We kunnen met een PRINT-opdracht niet zo maar iets op het hoge resolutiescherm (256 bij 192 beeldpunten) afdrukken. We moeten hiervoor eerst een uitvoerbestand openen en alles wat we op het scherm willen afdrukken moeten we naar dit bestand sturen. In hoofdstuk 9 behandelen we eigenlijk pas het werken met bestanden. We geven daarom nu alleen aan hoe een uitvoerbestand voor het grafische scherm eruit ziet en hoe je in zo'n bestand iets af kunt drukken en hoe je het weer moet sluiten.

Openen van een grafisch uitvoerbestand:

```
OPEN "GRP:" FOR OUTPUT A$ #1
```

Afdrukken van gegevens in dit bestand:

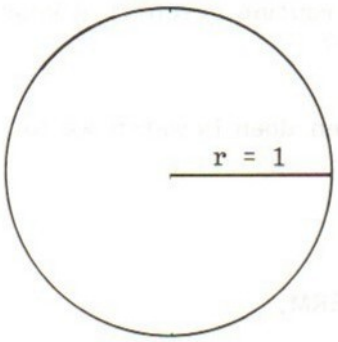
```
PRINT#1,A$,B,...
```

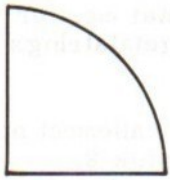
Sluiten van het grafische bestand:

```
CLOSE 1
```

Als illustratie van het afdrukken van gegevens op het grafische scherm presenteren we een 'educatief' programma waarin bijna alle graphic-opdrachten, die MSX kent, worden gebruikt. Ook de SOUND-opdracht, PRINT USING en ON INTERVAL GOSUB worden gebruikt. Eerst vertellen we iets over "wat" het programma doet en vervolgens over "hoe" het gedaan wordt.

Bijna iedereen zal op school geleerd hebben dat de oppervlakte van een cirkel gelijk is aan 'pi-maal-de-straal-van-de-cirkel-in-het-kwadraat (oppervlak cirkel met straal r is $\pi \cdot r^2$).

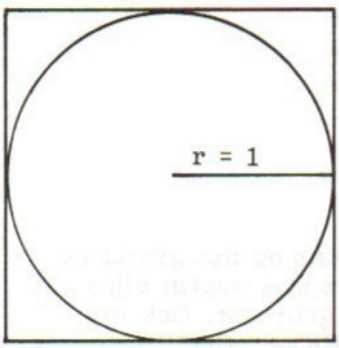


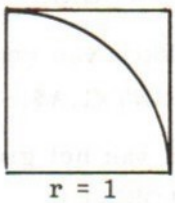


Oppervlakte = $\pi \cdot r^2$
dus $\pi \cdot 1^2$, dus π

Oppervlakte = $\frac{1}{4} \cdot \pi$

Pi is dat 'bekende' getal 3,1459... en nog meer decimalen. In het verleden hebben veel wetenschapsmensen getracht via allerlei methoden het getal pi (π) tot op vele decimalen te berekenen. Een aantal methoden berustte op het herhaald uitvoeren van een bepaald experiment, vaak duizenden malen. Wij gaan dit nu ook doen, maar laten het uitvoeren van het experiment over aan de computer. We nemen een cirkel met een straal ter lengte 1. Het oppervlak van zo'n cirkel is dan $\pi \cdot 1^2 = \pi$, precies π. Van die cirkel bekijken we een kwart deel.





Oppervlakte vierkant = $\frac{1}{4}$

De oppervlakte van de kwartcirkel is dus $\frac{1}{4} \cdot \pi$. Als we nu in staat zijn de oppervlakte van deze kwart cirkel te meten dan kunnen we daaruit de waarde voor π berekenen, immers

opp. kwart cirkel = $\frac{1}{4} \cdot \pi$, dus $\pi = 4 \times$ opp. kwart cirkel.

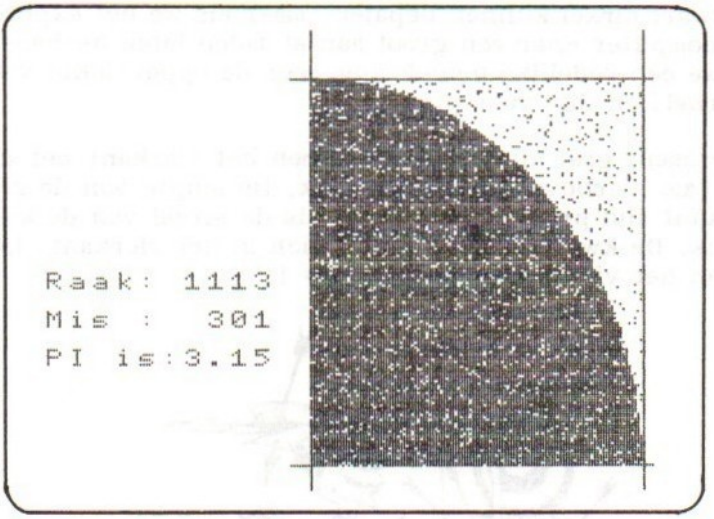
We gaan de oppervlakte van deze kwart cirkel niet meten maar via een experiment (een computersimulatie) proberen te benaderen. Op deze manier zullen we nooit de precieze waarde van de oppervlakte van de kwart cirkel kunnen bepalen, maar als we het experiment door de computer maar een groot aantal malen laten herhalen, dan krijgen we een redelijke benadering voor de oppervlakte van de kwart cirkel, en dus voor pi.

Het experiment gaat als volgt: We nemen het vierkant met zijden ter lengte 1 (zie hierboven) als schietvlak. De lengte van de zijden van het vierkant zijn precies even groot als de straal van de kwart cirkel, 1 dus. De kwart cirkel bevindt zich in het vierkant. De oppervlakte van het vierkant is dus $1 \times 1 = 1$.

We gaan op een afstandje van dit schietvlak staan en mikken met iets dat schieten kan op dit schietvlak. We schieten zo dat het vierkant altijd geraakt wordt maar we mikken niet 'echt' op een bepaald deel van het schietvlak. Met andere woorden: 'De plaats waar ons schot het vierkant treft wordt geheel door het toeval bepaald. Of we in werkelijkheid zo zouden kunnen schieten laten we even buiten beschouwing. Als we geschoten hebben dan kijken we of ons schot binnen of buiten de kwart cirkel terecht is gekomen. De cirkelrand rekenen we tot het binnengebied van de cirkel. Valt ons schot in de kwart cirkel of op de cirkelrand dan roepen we 'raak'. Op deze manier 'vuren' we een groot aantal malen op het schietvlak en onthouden hoeveel keer we geschoten hebben en hoeveel keer daarvan 'raak' is.

Als er 'echt' willekeurig geschoten wordt, dat wil zeggen als elk stukje van het vierkant even veel kans heeft om getroffen te worden dan zal de verhouding tussen het totaal aantal schoten en het aantal 'raak' overeenkomen met de verhouding tussen het oppervlak van het vierkant en de oppervlakte van de kwart cirkel, dus

aantal schoten : aantal raak = opp. vierkant : opp kwart cirkel



Noemen we N het aantal schoten, R het aantal raak, dan vinden we

$$N : R = 1 : \tfrac{1}{4}\pi$$

want de oppervlakte van het vierkant is $1 \times 1 = 1$ en van de kwart cirkel is de oppervlakte $\frac{1}{4}\pi$. Zie tekening op pagina 196.

$$\frac{N}{R} = \frac{1}{\frac{1}{4}\pi} \text{ geeft } \frac{1}{4}\pi = \frac{R}{N} \text{ ofwel } \pi = 4 \cdot \frac{R}{N}$$

Nu zal deze gelijkheid pas opgaan bij een oneindig groot aantal 'echt' willekeurige schoten. Ook met een computer zijn we niet in staat oneindig veel schoten na te bootsen (te simuleren) en bovendien kunnen we met een computer ook niet helemaal willekeurig mikken. Met het volgende programma kunnen we echter met een 'beperkt aantal schoten' al een redelijke benadering voor π berekenen. Hoe meer schoten we afvuren, des te groter de kans op een betere benadering voor π zal zijn. De functie RND geeft redelijk goede toevalsgetallen!

Als u het onderstaande programma intikt en draait dan zult u zien hoe het vierkant (LINE) en de kwart cirkel getekend worden (CIRCLE). De kwart cirkel wordt zwart gemaakt (PAINT) zodat we binnen het vierkant een zwart en een wit veld krijgen. We simuleren een 'schot' door de X- en Y-coördinaat van een punt in het vierkant

te bepalen. Om een willekeurige plaats in het vierkant te bepalen nemen we voor zowel de X- als de Y-coördinaat een toevalsgetal (RND) tussen 0 en 1. Voor een punt op de cirkelrand geldt:

$x^2 + y^2 = 1$ (cirkelvergelijking)

Voor een punt binnen de cirkel geldt:

$x^2 + y^2 < 1$

En voor een punt daarbuiten geldt:

$x^2 + y^2 > 1$

Een schot is dus raak als $x^2 + y^2 \leq 1$, waarin x en y twee toevalsgetallen tussen 0 en 1 zijn, die met de RND(1) functie worden getrokken. Als het schot de kwart cirkel treft (RAAK) dan zien we dat doordat er op de plaats van 'inslag' een wit puntje komt (PRESET). Valt het schot buiten de cirkel dan zien we een zwart puntje (PSET). Bovendien horen we als het schot het vierkant treft (SOUND). Is het schot raak dan horen we een ander geluid dan bij een schot dat mis is.

Elke halve minuut (ON INTERVAL ... GOSUB) wordt links van de tekening een benadering voor π afgedrukt, berekend als $4 \times (R/N)$, waarin R het aantal Rake schoten is en N het aaNtal afgevuurde schoten. We maken hierbij gebruik van de DRAW-opdracht om de onzichtbare grafische cursor op een bepaalde positie op het scherm te positioneren. De simulatie wordt beëindigd door op de spatiebalk te drukken (ON STRIG GOSUB). In het programma is N = R + M.

Nu volgt het programma, gevolgd door een toelichting. Let op de subroutine 'grafische uitvoer' waarin tekst en getallen via een uitvoerbestand op het grafische scherm worden afgedrukt. Het vierkant heeft een zijde van 160 beeldpunten. De straal van de kwart cirkel is dus ook 160 beeldpunten. Voor de X- en Y-coördinaat van het trefpunt van een schot berekenen we toevalsgetallen tussen 0 en 160. Een schot is dus raak als

$x^2 + y^2 \leq 160^2$

De functie FNX(X) = INT(37 + X/1.4 + .5) zorgt ervoor dat het vierkant ook 'min of meer' vierkant op het scherm getekend wordt.

```
100 REM Hoe groot is pi?
110 '   Een educatieve computersimulatie
120 DIM RG(13),MG(13) 'Geluidcodes definieren
130 FOR I=0 TO 5 :RG(I)=0:MG(I)=0:NEXT
140 RG(6)=17:RG(7)=7:RG(8)=16:RG(9)=16:RG(10)=16
150 RG(11)=1:RG(12)=5:RG(13)=1
160 MG(6)=31:MG(7)=7:MG(8)=16:MG(9)=16:MG(10)=16
170 MG(11)=0:MG(12)=60:MG(13)=0
180 '   Vierkant en kwartcirkel tekenen
190 DEF FNX(X)=INT(37+X/1.4+.5)
200 COLOR 1,15,15: SCREEN 2
210 LINE (FNX(90),0)-(FNX(90),190),1
220 LINE (FNX(250),0)-(FNX(250),190),1
230 LINE (FNX(80),180)-(FNX(255),180),1
240 LINE (FNX(80),20)-(FNX(255),20),1
250 CIRCLE (FNX(90),180),160,1,0,2*ATN(1),1.4
260 PAINT (FNX(120),100),1
270 '   Interrupts voorbereiden
280 ON STRIG GOSUB 3000
290 STRIG(0) ON
300 ON INTERVAL= 500 GOSUB 2000
310 INTERVAL ON
320 RDRST=RND(-TIME) 'Toevalsgetallengenerator starten
330 SCHIETEN=-1
340 '===
350   GOSUB 1000 ' Schiet
360 IF SCHIETEN THEN 350
370 '===herhaal totdat er niet meer geschoten wordt
380 STRIG(0) OFF : INTERVAL OFF
390 GOSUB 2000
400 A$=INPUT$(2)
410 END

1000 'Subroutine schiet
1005 '
1010   X=INT(161*RND(1))
1020   Y=INT(161*RND(1))
1030   IF X^2+Y^2<=160^2 THEN GOSUB 1500
                         ELSE GOSUB 1600
1035 '
1040 RETURN 'uit schiet
1045 '
1500 'Subroutine raak
1505 '
1510   PRESET (FNX(X+90),180-Y)
1520   R=R+1
1530   FOR I%=0 TO 13
1550      SOUND I%,RG(I%)
1560   NEXT I%
```

```
1580   FOR I=1 TO 180 : NEXT
1585 '
1590 RETURN 'uit raak
1595 '
1600 'Subroutine mis
1605 '
1610   PSET (FNX(X+90),180-Y)
1620   M=M+1
1630   FOR I%=0 TO 13
1650      SOUND I%,MG(I%)
1660   NEXT I%
1680   FOR I=1 TO 280 : NEXT
1685 '
1690 RETURN 'uit mis
1695 '
2000 'Subroutine grafische uitvoer
2005 '
2010   OPEN "grp:" AS #1
2015   COLOR 15
2020   DRAW "BM5,100":PRINT#1,STRING$(11,219):COLOR 1
2030   DRAW "BM10,100":PRINT#1, USING"Raak:#####"; R:COLOR 15
2040   DRAW "BM5,116":PRINT#1,STRING$(11,219):COLOR 1
2050   DRAW "BM10,116":PRINT#1, USING"Mis :#####"; M:COLOR 15
2060   DRAW "BM5,132":PRINT#1,STRING$(11,219):COLOR 1
2070   DRAW "BM10,132":PRINT#1, USING"PI is:#.##"; 4*R/(R+M);
2080   CLOSE#1
2085 '
2090 RETURN 'uit grafische uitvoer
2095 '
3000 'Subroutine spatiebalk
3005 '
3010   SCHIETEN=0
3015 '
3020 RETURN 'uit spatiebalk
```

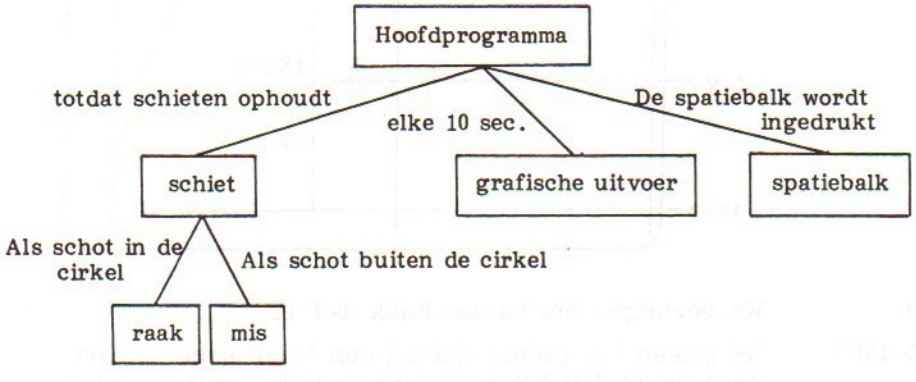
toelichting:

regel(s)	uitleg
120	We definiëren twee arrays (lijsten) voor het opslaan van de codes voor de geluidgenerator. RG is voor het RaakGeluid, MG voor het MisGeluid. De DIM-opdracht wordt aan het begin van het volgende hoofdstuk behandeld.
130-170	Hier vullen we de twee geluidcode-arrays met de codes voor de geluiden voor de SOUND-opdracht in de regels 1550 en 1650.

regel(s)	uitleg
190	De functie FNX corrigeert de horizontale (X-) coördinaat van een punt (X,Y) op het grafische scherm.
200	Instellen van zwart/wit scherm en het inschakelen van het grafische scherm (256 bij 192 beeldpunten).
210-240	Hierin tekenen we het 'vierkant' waarin straks de kwart cirkel getekend wordt. Let erop hoe de X-coördinaat via de FNX-functie berekend wordt.
250	Tekent de kwart cirkel. Het middelpunt van de cirkel is het punt (FNX(90),180); de straal is 160 (beeldpunten); de afdrukkleur is zwart (1); het cirkelsegment dat getekend moet worden loopt van de horizontale x-as (0) tot de verticale y-as (90° of $\pi/2$ ofwel 2·atn(1)); de correctiefactor voor de horizontale straal is 1,4; anders wordt het een ellips.
260	Dit kleurt de kwart cirkel zwart. Het punt (FNX(120),100) ligt in de kwart cirkel en de vulkleur is zwart (1).
280	Hier geven we aan dat als de spatiebalk (spelregelaar 0) wordt ingedrukt, de subroutine op regel 3000 moet worden uitgevoerd.
290	Hier activeren we spelregelaar 0; dit is de spatiebalk. Van nu af aan (tot STRIG(0) OFF) zal bij het indrukken van de spatiebalk het programma naar de subroutine op regel 3000 gaan.
300	Hier stellen we een tijdsinterval van 500/50 = 10 seconden in en geven we aan dat het programma na de INTERVAL ON-opdracht om de 10 seconden de subroutine op regel 2000 moet uitvoeren.
310	Vanaf nu wordt elke 10 seconden de subroutine op regel 2000 uitgevoerd.
320	Hier starten we de toevalsgetallengenerator RND door de inwendige MSX-klok (TIME) te gebruiken als startwaarde voor de generator. Laten we regel 320 weg dan genereert het programma na elke RUN dezelfde reeks toevalsgetallen (zie hoofdstuk 10).
330	We zetten de waarheidsvariabele SCHIETEN op 'waar' (true heeft de waarde -1 in MSX BASIC).
340-370	Dit is de HERHAAL-TOTDAT-lus die ervoor zorgt dat er steeds een schot wordt afgevuurd (GOSUB 1000 'schiet') totdat er niet meer geschoten wordt. Het schieten houdt op als de waarheidsvariabele SCHIETEN op 'niet-waar' (false) wordt gezet. Dit gebeurt in de subroutine 'spatiebalk' (regel 3000) als de spatiebalk wordt ingedrukt.

regel(s)	uitleg
380	Hier stoppen we 'spatiebalkdetectie' en het tijdintervalmechanisme.
390	Hier roepen we de subroutine 2000 aan die de uiteindelijke benaderde waarde van het getal pi op het scherm afdrukt.
400	De INPUT$-functie tast vanaf de start van het programma (na RUN dus) het toetsenbord af. Zouden we hier A$ = INPUT$(1) gezet hebben dan zou het programma bij regel 400 niet wachten, want we hebben immers op één toets, namelijk de spatiebalk, moeten drukken om bij regel 400 te komen. INPUT$(1) zou dit gemerkt hebben (de 1 betekent wacht tot één toets is ingedrukt) en het programma zou beëindigd worden waardoor het grafische scherm gewist zou worden en we dus niet rustig naar de simulatie-uitkomst zouden kunnen kijken. Door in regel 400 A$ = INPUT$(2) te zetten wacht het programma hier totdat twee toetsen zijn ingedrukt. De spatiebalk is, zoals gezegd, al ingedrukt (anders zou het programma nog met de HERHAAL-TOTDAT lus bezig zijn) zodat we het programma kunnen beëindigen door nog één toets in te drukken.

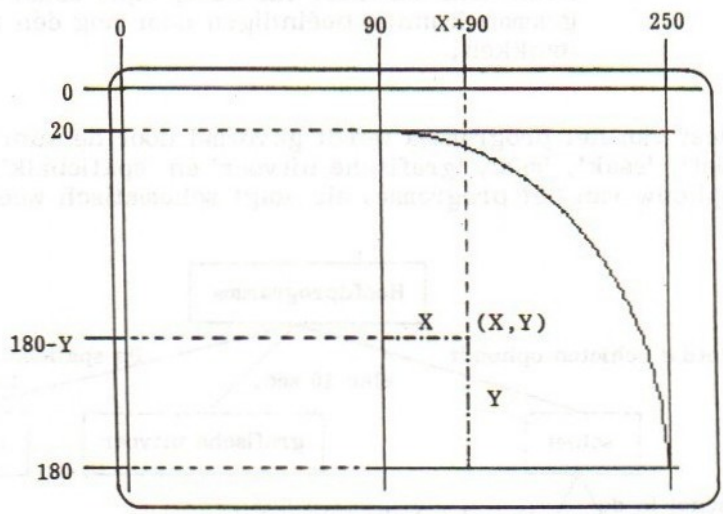
De rest van het programma wordt gevormd door de subroutines 'schiet', 'raak', 'mis', 'grafische uitvoer' en 'spatiebalk'. We kunnen de opbouw van het programma als volgt schematisch weergeven.



We geven nog een toelichting bij elk van de hierboven genoemde subroutines.

toelichting:

regel(s)	uitleg
1000-1040	Subroutine 'schiet'; hierin bepalen we de X- en Y-coördinaat. INT(161*RND(1)) levert een toevalsgetal 0,1,2,... of 160 op.
1030	Als het berekende schot in de kwart cirkel of op de rand van de kwart cirkel terechtkomt, dan wordt de subroutine 'raak' uitgevoerd, anders de subroutine 'mis'.
1500-1590	Subroutine 'raak'; hierin zien we hoe het schot de cirkel treft (wit puntje) en horen we de inslag ervan op het scherm.
1510	Met PRESET(FXN(X+90),180-Y) maken we het beeldpunt met de schermcoördinaten (FNX(X+90),180-Y) wit. In de onderstaande tekening zien we hoe de X en Y die berekend zijn in de regels 1010 en 1020 gebruikt worden om de schermcoördinaten te bepalen. De oorsprong ligt in de linkerbovenhoek.



1520	We verhogen het aantal Raak met 1.
1530-1560	We maken het geluid dat bij een 'raak schot' hoort door de 14 SOUND-registers te vullen met de juiste codes (RG(0) t/m RG(13)).
1580	Dit geeft een tijdvertraging die nodig is om het geluid en het schieten enigszins te synchroniseren.

regel(s)	uitleg
1600-1690	Subroutine 'mis'; deze heeft dezelfde opzet als de vorige subroutine. We maken het beeldpunt echter zwart (PSET) in plaats van wit (PRESET); in plaats van het aantal Raak verhogen we het aantal Mis met 1; we gebruiken een ander geluid (MG) en een grotere tijdvertraging omdat het geluid langer aanhoudt dan het 'raak geluid'.
2000-2090	Subroutine 'grafische uitvoer'; hierin drukken we aan de linkerkant van het grafische scherm het aantal rake schoten en het aantal missers af, alsmede een benadering van de waarde van pi.
2010	We openen een uitvoerbestand voor het grafische scherm. Dit bestand krijgt nummer 1 als referentienummer.
2015	We maken de afdrukken wit (code 15) anders worden spaties in regel 2020 in zwart afgedrukt.
2020	We verplaatsen de onzichtbare grafische cursor naar de beeldschermpositie (X = 5, Y = 100) en drukken daar 11 spaties af met de STRING$-functie. Code 219 is de videocode voor een spatie. Hierdoor wordt de tekst die hier stond uitgeveegd. We veranderen ten slotte de afdrukkleur weer in zwart (code 1), anders wordt de tekst in regel 2030 in wit (en dus onzichtbaar) afgedrukt.
2030	We zetten de grafische cursor op het beeldpunt (X = 10, Y = 100) en drukken vanaf daar af Raak:, gevolgd door het aantal rake schoten, waarvoor we maximaal 5 (#####) posities gereserveerd hebben. We maken de afdrukkleur weer wit (COLOR 15), anders worden de spaties uit regel 2040 in zwart afgedrukt.
2040-2070	Net zoals hierboven beschreven drukken we nog twee regels af met 'Mis: ', gevolgd door het aantal schoten mis en 'PI is: ', gevolgd door de benaderde waarde $4 \cdot \frac{R}{(R+M)}$ voor pi. R+M is het totaal aantal afgevuurde schoten. PI wordt afgedrukt met één cijfer voor de komma en twee cijfers achter de komma (sorry... punt).
2080	We sluiten de uitvoer naar het grafische bestand. De vraag rijst natuurlijk waarom we in de regels 2020, 2040 en 2060 eerst de regels moeten wissen (door het afdrukken van witte spaties) voordat we op diezelfde regel de tekst kunnen afdrukken. Op het gewone

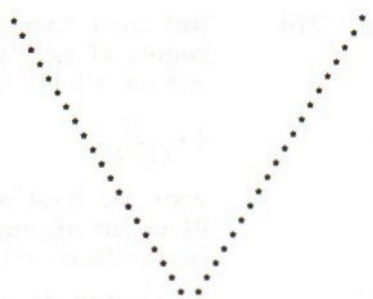
scherm (SCREEN 0 of 1) hoeft dat ook niet. Als we daar een bepaald teken over een ander teken afdrukken dan wordt het 'gehele teken' gewist en vervangen door het nieuwe teken. Nemen we even aan dat een teken uit 8 bij 8 pixels (beeldpunten) is opgebouwd, dan wordt dus steeds een blokje van 8 bij 8 beeldpunten vervangen door een nieuw blokje van 8 bij 8 beeldpunten. Op het grafische scherm (SCREEN 2) blijft bij het afdrukken van een nieuw teken over een oud teken het oude teken gewoon staan. Zouden we op het grafische scherm op dezelfde positie bijvoorbeeld een 2 afdrukken waar een 1 stond, dan krijgen we dit: **2**. We moeten dus eerst de 1 weghalen voordat we de 2 afdrukken. Dit weghalen kan door op die plaats eerst een spatie af te drukken. Die is 'groot' genoeg om elk teken dat er zou kunnen staan te wissen. Dit verklaart de regels 2020, 2040 en 2060.

3000-3020 Subroutine 'spatiebalk'; deze eenvoudige subroutine zet de waarheidsvariabele SCHIETEN op 'niet-waar'.

Hiermee eindigt de toelichting op het grafische simulatieprogramma 'Hoe groot is pi?' Hopelijk is door deze toelichting en door de duidelijke programmastructuur en de commentaarregels in het programma de werking van het programma voor iedereen duidelijk.

OEFENOPGAVEN HOOFDSTUK 6

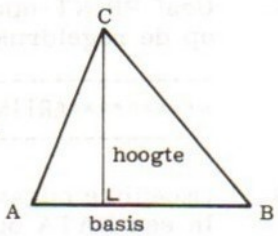
1. De uitvoer die hiernaast is afgebeeld wordt afgedrukt met een FOR-NEXT opdracht waarin een LPRINT opdracht staat met de TAB functie. Welke LPRINT opdracht is dat?
Het sterretje linksboven staat op positie 10 van een regel; het sterretje rechtsboven staat op positie 50 van die regel.



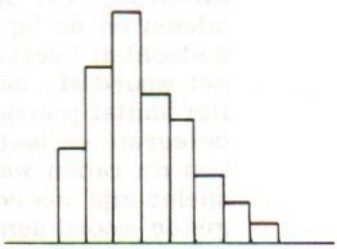
2. Uit het 'valuta-programma' in § 6.6 blijkt dat in de formatstring bij (L)PRINT USING we méér dan één formatveld mogen opgeven. De tabel hiernaast bevat vier kolommen die door een FOR-NEXT opdracht zijn afgedrukt. In die FOR-NEXT opdracht wordt één PRINT USING gebruikt met vier formatvelden.
De vier kolommen bevatten de 1-ste, 2-de, 3-de, 4-de macht van de getallen 1.0, 1.1, 1.2 t/m 2.0.
Geef de FOR-NEXT lus waarmee dit wordt afgedrukt.

1.0	1.00	1.000	1.0000
1.1	1.21	1.331	1.4641
1.2	1.44	1.728	2.0736
1.3	1.69	2.197	2.8561
1.4	1.96	2.744	3.8416
1.5	2.25	3.375	5.0625
1.6	2.56	4.096	6.5536
1.7	2.89	4.913	8.3521
1.8	3.24	5.832	10.4976
1.9	3.61	6.859	13.0321
2.0	4.00	8.000	16.0000

3. Schrijf een programma dat voor een aantal driehoeken de oppervlakte uitrekent en afdrukt. (Opp. driehoek is halve basis × hoogte.) Zowel het aantal driehoeken waarvan de oppervlakte moet worden berekend als de waarden voor basis en hoogte staan in DATA opdrachten. Geef eerst zo mogelijk een programmastructuurdiagram.



4. Onder een aantal autobezitters is een onderzoek gedaan naar het merk auto dat 'wordt gereden'. In een DATA lijst staan tien merken, elk gevolgd door een getal dat aangeeft hoeveel van de autobezitters met dat merk rijden. Schrijf een programma dat deze gegevens inleest en op een regeldrukker eerst de naam van het merk (maximaal 15 tekens) afdrukt, gevolgd door een aantal sterretjes dat overeenkomt met het aantal ondervraagden dat met dat merk rijdt; en dit voor alle tien merken. In de statistiek noemen we zo'n grafiek een histogram.



```
(Volkswagen : *******
 Opel       : ***********
              enzovoorts)
```

5. Het 'machtenprogramma' van p.176 geeft een uitvoer bestaande uit vier kolommen met getallen. Deze getallen staan in een kolom 'linksaangesloten'. Gebruik een PRINT USING opdracht om deze getallen rechtsaangesloten te krijgen. De formatstring moet gebaseerd zijn op het grootste getal dat moet worden afgedrukt.
Hoe ziet die PRINT USING opdracht(en) eruit?

6. Schrijf een programma dat twee kolommen afdrukt. In de eerste kolom staat een aantal graden Celsius. In de tweede kolom het bijbehorende aantal graden Fahrenheit. Het aantal graden Fahrenheit moet op één decimaal worden afgerond en deze kolom moet 'rechtsaangesloten' worden afgedrukt. Het aantal graden Celsius in de linkerkolom begint met een BEGINwaarde en gaat met bepaalde STAPjes naar een EINDwaarde en wordt ook rechtsaangesloten afgedrukt. Deze drie waarden (BEGIN, EIND, STAPje) worden eerst uit een DATA lijst ingelezen. De formule is F = 1.8 × C + 32. Het aantal graden Celsius ligt tussen -100 en +100 en is altijd een geheel getal. De kolom met Fahrenheit-graden moet op regelpositie 10 beginnen.

7. Schrijf een programma dat van elk ingetoetst woord de middelste letter afdrukt. Als er stop wordt ingetoetst wordt het programma beëindigd. Als een woord een even aantal letters bevat worden de middelste twee letters afgedrukt.

8. Geef PRINT-opdrachten om de volgende 'kop' boven een tabel op de regeldrukker af te drukken.

```
----------------------------------------
**********ARTIKEL***PRIJS***BTW**********
----------------------------------------
```

9. (moeilijke opgave)
In een DATA opdracht staan tien woorden, elk met een even aantal letters. Schrijf een programma dat steeds een woord inleest en de 'speler' laat raden welk woord de computer in gedachten heeft. De computer drukt eerst de eerste letter van het woord af, dan een aantal puntjes en dan de laatste letter. Het aantal puntjes is precies het aantal letters dat afgezien van de eerste en laatste letter in het woord voorkomt. De speler kan nu raden wat de verborgen letters zijn. Hiertoe toetst de speler zijn woord in. Is het woord geraden dan komt het volgende woord aan de beurt.
Is het woord niet geraden dan vraagt de computer aan de speler of hij de volgende twee letters (één vooraan, één achteraan het woord) wil zien. Zo ja, dan worden deze twee letters afgedrukt. Zo nee, dan mag de speler nog eens raden. Dit wordt herhaald tot de speler het woord heeft geraden of tot de computer zelf het woord heeft afgedrukt.
Probeer eerst structuurdiagrammen te maken.
(Hint: maak voor het 'woord raden' en voor het 'afdrukken van het geheime woord' een aparte module. Regel het inlezen van de woorden en het starten van elk spel in de hoofdmodule.)

7 Werken met Gegevenslijsten

In het 'dagelijks' leven werken we erg veel met lijsten. Om er maar een paar te noemen: boodschappenlijst, boekenlijst, ledenlijst, bestellijst, verzendlijst en ga zo maar door. Computerprogramma's die voor 'dagelijks' gebruik worden geschreven zullen dus ook met dergelijke lijsten moeten kunnen werken. Welnu, dat kan ook en we zullen zien hoe dat bij MSX BASIC in zijn werk gaat.

7.1 LIJSTSTRUCTUUR

Als we in een BASIC programma met een lijst willen gaan werken, moeten we drie dingen doen:

1. We moeten **aankondigen** dat we met een lijst willen gaan werken.
2. We moeten de lijst een **naam** geven.
3. We moeten vertellen **hoeveel elementen** de lijst maximaal zal mogen bevatten.

We doen dit aan het begin van een programma. Een voorbeeld:

```
10 REM  Demonstratie Lijststructuur
15 '
20 DIM NAAM$(11)
30 FOR I = 1 TO 11
40    READ NAAM$(I) ' Het i-de lijstelement wordt gelezen
50 NEXT I
60 GOSUB 1000 ' Naamanalyse
70 END
80     DATA "Jan", "Ellis", "Klaas", "Piet", "Wies", "Janneke"
90     DATA "Karel", "Arnold", "Irene", "Jeanette", "Joke"
```

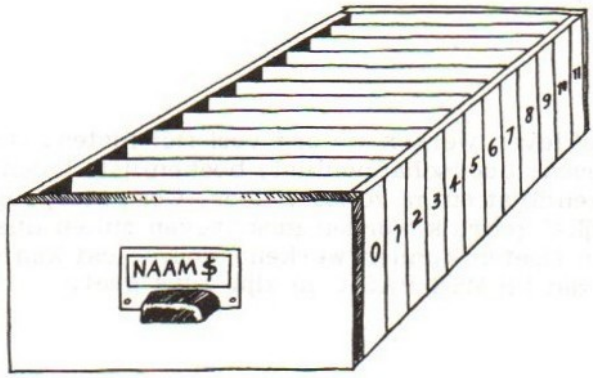
De opdracht

```
20 DIM NAAM$(11)
```

houdt in:

1. dat we een lijst willen gaan gebruiken (DIMension opdracht);
2. dat we de lijst de naam NAAM$ geven;
3. dat de lijst maximaal 12 elementen kan bevatten (elementen 0 t/m 11).

Hieronder zien we het effect van deze opdracht:



De computer heeft een 'grote' geheugenla gemaakt en deze in twaalf stukjes verdeeld. Op de la is het etiket NAAM$ geplakt. Willen we iets uit deze la hebben dan moeten we de naam van de la en het nummer van het vakje opgeven, bijvoorbeeld NAAM$(7).

De lijst NAAM$ wordt ook wel een **array** (= reeks) genoemd. NAAM$ zelf heet een **lijstvariabele** of **rijvariabele** of **geïndexeerde variabele**. In dit voorbeeld staat er een $-teken in de naam omdat we in elk element van de lijst een stukje tekst willen gaan opbergen.

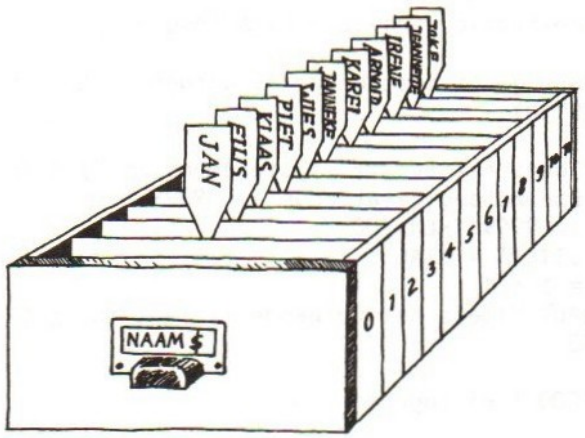
De opdracht DIM NAAM$(11) heeft tot gevolg dat de computer 12 plaatsen reserveert, genummerd van 0 tot en met 11. Aangezien wij gewend zijn vanaf 1 te tellen zullen wij dit 0-de element nauwelijks of niet gebruiken. Soms kan het handig zijn om bij het 0-de element te beginnen. In het pi-programma aan het einde van het vorige hoofdstuk zijn we inderdaad met 0 begonnen (zie de FOR-opdracht in regel 1530).

Terug naar het programma. De regels 30 tot en met 50:

```
30 FOR I = 1 TO 11
40     READ NAAM$(I) ' Het i-de lijstelement wordt gelezen
50 NEXT I
```

zorgen ervoor dat de lijst NAAM$ gevuld wordt met namen uit de DATA-lijst. De FOR-variabele I wordt tevens gebruikt als index voor het aangeven van het element in de lijst waarin de naam moet

worden opgeslagen. Na het uitvoeren van de FOR-NEXT-lus is de situatie als volgt (het 0-de element blijft leeg!):



opdracht 7.1

Wat wordt er afgedrukt als het programma van bladzijde 209 met deze routine wordt gedraaid?

```
1000 ' Subroutine Naamanalyse
1005 '
1010     FOR I = 1 TO 11
1020        IF LEFT$(NAAM$(I),3) = "Jan" THEN PRINT NAAM$(I)
1030     NEXT I
1035 '
1040 RETURN ' Naamanalyse
```

Natuurlijk zijn er ook numerieke lijsten:

```
10 REM demonstratie variabele lijstlengte
15 '
20 INPUT "Hoeveel metingen gaat u invoeren "; N
30 DIM METING(N) ' Reserveer ruimte voor N getallen
40 FOR I = 1 TO N
50    INPUT "Meetwaarde "; METING(I)
60 NEXT I
70 GOSUB 1000 ' Analyse Metingen
80 END
```

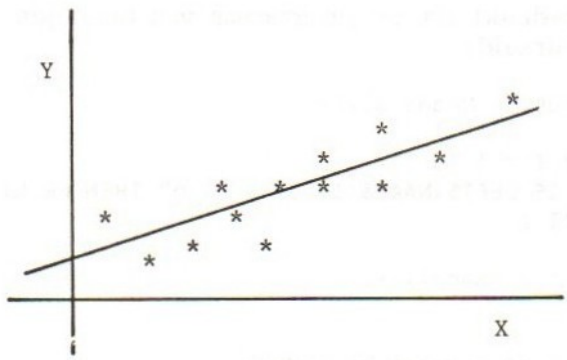
Dit programma kan worden gebruikt voor het inlezen en onthouden van een willekeurig aantal ingetoetste meetwaarden. Pas na het inlezen van de waarde van N geven we de computer te kennen dat de array METING, N elementen zal bevatten.

opdracht 7.2

Wat is uw oordeel over het volgende programmaatje:

```
100 REM Demonstratie variabele lijstlengte
105 '
110 INPUT "Hoe groot moet de array worden "; N
120 DIM LIJST(N)
130 I = 1
140 INPUT "Toets het eerste lijstelement in "; GTAL
145 REM*** Doe zolang getal ongelijk 9999
150 IF GTAL = 9999 THEN 200
160       LIJST(I) = GTAL
170       I = I + 1
180       INPUT "Toets het volgende element in "; GTAL
190 GOTO 150
195 REM***
200 GOSUB 1000 ' Histogram
210 END
```

In een DIM opdracht mogen we het gebruik van meer dan één lijst aankondigen:




```
100 REM Rechtelijnaanpassing
110 '
120 INPUT "Hoeveel punten"; N
130 DIM X(N), Y(N)
140 FOR PUNT = 1 TO N
150    INPUT "x,y"; X(PUNT), Y(PUNT)
160 NEXT PUNT
170 GOSUB 1000 'regressielijn
180 A$=INPUT$(1)
190 GOSUB 2000 'spreidingsdiagam
200 A$=INPUT$(1)
210 COLOR 15,4,4 : SCREEN 0
220 END
```

twee(of meer) waarden tegelijk inlezen, dat mag ook. Denk aan de ,

In de wiskunde wordt een punt in 'het platte vlak' vaak door (x,y) voorgesteld, waarbij x de horizontale coördinaat is en y de verticale coördinaat in een rechthoekig assenstelsel. Als er meer punten (n) zijn zien we dit: $(x_1,x_2);(x_2,y_2);.....(x_n,y_n)$.

In BASIC-programma's kunnen we deze notatie gewoon overnemen. Het punt (x_4,y_4) in het platte vlak wordt dan voorgesteld door het vierde element uit de X-rij en het vierde element uit de Y-rij, dus door X(4), Y(4).
Er kunnen echter heel andere dan wiskundige redenen zijn om meer dan één lijst te gebruiken. Hierover gaat de volgende opdracht.

opdracht 7.3

Schrijf een stukje programma waarin twee lijsten worden gebruikt. In de eerste lijst komen namen van personen te staan. In de tweede lijst komen de leeftijden van die personen te staan. Laat in het stukje programma de DIM-opdracht en het inlezen van de gegevens uit een DATA-lijst zien. Het eerste element uit de DATA-lijst is een getal dat aangeeft hoeveel naam-leeftijd combinaties ingelezen moeten worden.

Soms kan het nodig zijn een programma te gebruiken waarin berekeningen voorkomen die voor de 'leek' niet te volgen zijn. Dat hoeft geen belemmering te zijn om van het programma nuttig gebruik te maken. Wij zullen daarom de subroutines 1000 en 2000, die in het vorige programma worden aangeroepen, hier presenteren. Deze subroutines berekenen en tekenen uit de ingetoetste coördinaten X(1),Y(1),... tot en met X(N),Y(N), volgens een bepaalde methode een rechte lijn die zo 'goed' mogelijk door deze punten heenloopt.
De subroutine 1000 berekent uit de N ingetoetste coördinaatparen (x,y) volgens de 'methode van de kleinste kwadraten' de coëfficiënten a en b van de regressielijn y = ax + b. Deze regressielijn wordt in de statistiek gebruikt bij het analyseren van een mogelijk rechtlijnig verband tussen twee variabelen (x = oorzaak en y = gevolg). We gaan hier niet op deze methode in, maar geven wel de subroutine die de berekeningen uitvoert en de vergelijking van de berekende regressielijn op het scherm afdrukt.

```
1000 'subroutine regressielijn
1010 '
1020 FOR I=1 TO N
1030    XSOM=XSOM+X(I)
1040    YSOM=YSOM+Y(I)
1050    XKWSOM=XKWSOM+X(I)^2
1060    XYSOM=XYSOM+X(I)*Y(I)
1070 NEXT I
1080 XGEM=XSOM/N
1090 YGEM=YSOM/N
1100 A=(XYSOM-N*XGEM*YGEM)/(XKWSOM-N*XGEM^2)
1110 A=INT(100*A+.5)/100
1120 B=YGEM-A*XGEM
1130 B=INT(100*B+.5)/100
1140 IF B<0 THEN PM$="-" ELSE PM$="+"
1150 CLS
1160 PRINT "De regressielijn is de lijn:"
1170 PRINT
1180 PRINT "y=";:PRINT USING "##.##";A;:PRINT "*x ";
1190 IF B<>0 THEN PRINT PM$;ABS(B)
1200 '
1210 RETURN 'regressielijn
```

Elk statistisch 'pakket' bevat behalve zo'n regressielijnsubroutine ook een subroutine waarmee de ingetoetste punten (x,y) en de regressielijn in een grafiek getekend worden. Zo'n spreidingsdiagram laat de X- en Y-as zien, de N punten in het X-Y-vlak en de regressielijn. We geven hier ook een dergelijke subroutine. Deze subroutine is bedoeld voor regressie-analyses waarbij de waarden voor x en y positieve gehele getallen tussen 0 en 1000 zijn. De subroutine tekent namelijk de X-as onder aan het scherm en de Y-as links op het scherm; er is dus geen ruimte voor negatieve x- of y-waarden. Dat de x- en y-waarden positieve gehele getallen onder de 1000 moeten zijn hangt samen met de beperkte ruimte die we voor het afdrukken van waarden bij de schaalverdeling op het scherm hebben. 'Echte' statistische pakketten kennen deze beperkingen natuurlijk niet, alhoewel deze subroutine toch voor een groot aantal problemen te gebruiken is. We illustreren beide subroutines aan de hand van een voorbeeld.

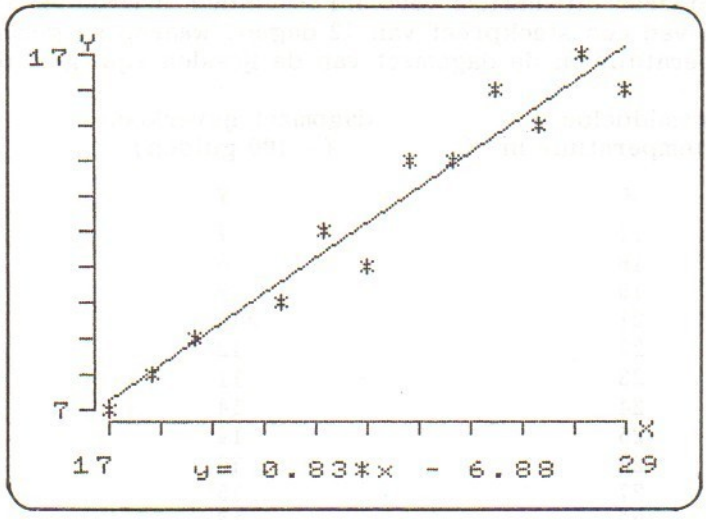
De eigenaar van een ijssalon wil graag weten wat het verband is tussen zijn dagelijkse omzet (in guldens) en de gemiddelde temperatuur van de buitenlucht. Natuurlijk weet hij uit ervaring wel hoeveel ijs hij van tevoren moet klaarmaken, maar hij is best geïnteresseerd in de meer precieze aard van dit verband. Hij weet, net als wij, dat hoe warmer het buiten is, hoe meer ijs er verkocht wordt. Wat hij niet weet is hoeveel meer ijs er verkocht zal worden als het buiten, bijvoorbeeld, twee graden warmer is. Als we even veronderstellen dat het verband tussen de buitentemperatuur (x) en de ijsomzet (y) rechtlijnig is (y = ax + b) dan kunnen we met lineaire regressie-

analyse proberen dit lineaire verband te vinden. Hieronder staan de uitkomsten van een steekproef van 12 dagen, waarop de gemiddelde buitentemperatuur en de dagomzet van de ijssalon zijn 'gemeten'.

gemiddelde buitentemperatuur in °C	dagomzet ijsverkoop (× 100 gulden)
x	y
17	7
18	8
19	9
21	10
22	12
23	11
24	14
25	14
26	16
27	15
28	17
29	16

```
run
Hoeveel punten? 12
x,y? 17,7
x,y? 18,8
x,y? 19,9
x,y? 21,10
x,y? 22,12
x,y? 23,11
x,y? 24,14
x,y? 25,14
x,y? 26,16
x,y? 27,15
x,y? 28,17
x,y? 29,16

color  auto  goto  list  run
```

We hebben deze 12 waarnemingen als 12 coördinatenparen (x,y) met bovenstaand programma ingevoerd en laten door subroutine 1000 de regressielijn (het rechtlijnig verband) berekenen. Vervolgens (door een druk op een toets) laten we subroutine 2000 (zie hieronder) een plaatje van de regressie-analyse tekenen. Op de volgende bladzijde bovenaan zien we wat subroutine 2000 op het scherm tekent.



We zien onderaan het scherm de vergelijking van de regressielijn, namelijk y = 0,83.x - 6,88. Als dit inderdaad het verband tussen de buitentemperatuur (x) en de ijsomzet (y) is dan zou dit betekenen dat één graad verhoging van de buitentemperatuur een omzetstijging van 0,83 × 100 = 83 gulden tot gevolg heeft. Op de vraag of de eigenaar van de ijssalon deze statistische analyse bij zijn 'produktie-planning' zou moeten gebruiken gaan we hier niet in. Wel kan hij zeer globaal 'becijferen' welke omzet hij had kunnen verwezenlijken indien de zomer wat minder 'hollands' geweest zou zijn.

De subroutine 2000, die het bovenstaande plaatje (spreidingsdiagram) tekent, bepaalt eerst de grootste (XG en YG) en de kleinste (XK en YK) x- en y-waarde van alle ingetoetste waarnemingen (x,y). Hiervoor gebruiken we dezelfde methode als in paragraaf 6.1. We zien dat deze kleinste en grootste x- en y-waarden respectievelijk aan het begin en einde van de X- en Y-as geplaatst worden; 17 en 29 als kleinste en grootste x-waarde, 7 en 17 als kleinste en grootste y-waarde. Zowel de X-as als de Y-as is verdeeld in precies 10 gelijke stukken. De lengte van elk stuk hangt af van de begin- en eindwaarde van de schaalverdeling. In de bovenstaande tekening komt elk stukje op de X-as overeen met 1,2° Celsius en elk stukje op de Y-as met 1 × 100 gulden.

Het tekenen van de X-as met schaalverdeling gebeurt in de regels 2150-2180; de Y-as wordt getekend in de regels 2200-2230. We hebben al eerder gezien hoe we tekst op het grafische scherm (SCREEN 2) kunnen afdrukken. Regel 2250 opent een uitvoerbestand naar het grafische scherm, terwijl de regels 2260 t/m 2310 voor de tekst langs de X- en Y-as zorgen. In de FOR-lus van de regels 2330-2390 worden

de eerder ingetoetste waarnemingen (x,y) als *-tjes in het xy-vlak getekend. De regels 2340 en 2350 zorgen ervoor dat de ingetoetste coördinaten (X(I),Y(I)) omgezet worden naar de juiste beeldscherm-coördinaten. Regel 2360 zet de cursor op de juiste plaats, terwijl regel 2370, onder het geluid van een piep (BEEP), op die plaats een *-tje afdrukt. De pauze in regel 2380 is nodig om de punten niet te snel te laten tekenen. De regels 2410 en 2420 zetten de regressielijn midden onder de grafiek. Deze regressielijn wordt vervolgens met een fraai begeleidend geluid (regels 2450 en 2460) door de punten (*-tjes) getrokken. Aangezien de kleinste x-waarde altijd een x-waarde van 50 op het scherm heeft en de grootste y-waarde een schermcoördinaat van 230, wordt de regressielijn in regel 2490 altijd getrokken tussen de punten (50,Y1) en (230,Y2), waarbij Y1 en Y2 de Y-coördinaten zijn van de punten op de regressielijn met respectievelijk als X-coördinaat de kleinste (XK) en grootste (XG) door ons ingetoetste x-waarden. Hier volgt tenslotte de subroutine 'spreidingsdiagram'.

```
2000 'subroutine spreidingsdiagram
2010 '
2020 'de grootste en kleinste x en y bepalen
2030 XG=-9999 : YG=-9999
2040 XK= 9999 : YK= 9999
2050 FOR I=1 TO N
2060    IF X(I)>XG THEN XG=X(I)
2070    IF Y(I)>YG THEN YG=Y(I)
2080    IF X(I)<XK THEN XK=X(I)
2090    IF Y(I)<YK THEN YK=Y(I)
2100 NEXT I
2110 RX=XG-XK
2120 RY=YG-YK
2130 COLOR 1,15,4 : SCREEN 2
2140 'x-as met schaalverdeling
2150 LINE (50,165)-(230,165)
2160 FOR I=0 TO 10
2170    LINE(50+I*18,165)-(50+I*18,170)
2180 NEXT I
2190 'y-as met schaalverdeling
2200 LINE(45,160)-(45,10)
2210 FOR I=0 TO 10
2220    LINE(45,160-I*15)-(40,160-I*15)
2230 NEXT I
2240 'grootste en kleinste x en y langs de assen zetten
2250 OPEN "grp:" FOR OUTPUT AS #1
2260 DRAW "BM15,8":PRINT#1,USING "###";YG
2270 DRAW "BM41,2":PRINT#1,"Y"
2280 DRAW "BM15,157":PRINT#1,USING "###";YK
2290 DRAW "BM30,180":PRINT#1,USING "###";XK
2300 DRAW "BM235,165":PRINT#1,"X"
2310 DRAW "BM220,180":PRINT#1,XG
```

```
2320 'de punten (x,y) tekenen
2330 FOR I=1 TO N
2340    X=50+(X(I)-XK)*180/RX
2350    Y=160-(Y(I)-YK)*150/RY
2360    PRESET(INT(X+.5)-2,INT(Y+.5)-3)
2370    PRINT#1,"*" : BEEP
2380    FOR PAUSE=1 TO 100 : NEXT
2390 NEXT I
2400 DRAW "BM80,182"
2410 PRINT#1, "y=";:PRINT#1, USING "##.##";A;:PRINT#1, "*x ";
2420 IF B<>0 THEN PRINT#1, PM$;ABS(B)
2430 CLOSE 1
2440 'regressielijn tekenen
2450 SOUND 7,62:SOUND 1,0:SOUND 0,254:SOUND 8,16
2460 SOUND 13,9:SOUND 12,60:SOUND 11,0
2470 Y1=160-(A*XK+B-YK)*150/RY
2480 Y2=160-(A*XG+B-YK)*150/RY
2490 LINE(50,Y1)-(230,Y2)
2500 '
2510 RETURN 'spreidingsdiagram
```

De lezer, voor wie regressie-analyse op zich niet interessant is, zal toch een aantal waardevolle dingen uit dit programma kunnen halen, zoals het bepalen van de kleinste en grootste waarde in een reeks waarden, het aanbrengen van een vaste schaalverdeling, het afdrukken van tekst op het grafische scherm en het toepassen van geluidseffecten.

De lezer die dit programma wellicht bij zijn of haar werk of studie kan gebruiken heeft hiermee een eenvoudig programma voor lineaire regressie-analyse voor in principe een zeer groot aantal waarnemingen. Aangezien bij een groot aantal waarnemingen de benodigde geheugenruimte toeneemt is het verstandig om de arrays X(N) en Y(N) als 'integer arrays' te dimensioneren. Dit gaat met DIM X%(N), Y%(N) (zie later in dit hoofdstuk). Bovendien is het dan verstandig geen *-tjes in het spreidingsdiagram af te drukken, maar bijvoorbeeld zwarte puntjes (PSET-opdracht).

7.2 LIJSTEN OM IN TE 'TELLEN'

De lijststructuur is erg geschikt als we moeten bijhouden hoe vaak iets is voorgekomen. We gebruiken dan de elementen uit een lijst om te tellen hoe vaak zich bepaalde gebeurtenissen hebben voorgedaan. In het volgende programmavoorbeeld wordt vijftig maal met een dobbelsteen gegooid. In de array OGEN houden we bij hoeveel maal een bepaald aantal ogen gegooid is:

```
100 REM  lijst om in te tellen
110 ' gooien en turven(tellen)
120 FOR I=1 TO 50
130    K%=INT(6*RND(1)+1)
140    OGEN(K%)=OGEN(K%)+1
150 NEXT I
160 ' resultaat afdrukken
170 FOR I= 1 TO 6
180    PRINT USING "Ogen(#) :";I;
190    PRINT STRING$(OGEN(I),"*")
200 NEXT I
210 END

run
Ogen(1) :***********
Ogen(2) :*******
Ogen(3) :************
Ogen(4) :******
Ogen(5) :*********
Ogen(6) :****
Ok
```

We zien hoe K%, het gegooide aantal ogen, wordt gebruikt als index voor de array OGEN. Als er bijvoorbeeld vijf ogen gegooid zijn (K% = 5), wordt het vijfde element uit OGEN met één opgehoogd (OGEN(5) = OGEN(5)+1). Zo bepalen we dus voor elk aantal ogen de frequentie, dat wil zeggen het aantal keren dat een bepaald aantal ogen is gegooid. De index van een lijstelement moet altijd een geheel getal zijn. Het is dan ook verstandig daarvoor een integervariabele (%) te nemen, dan kan er tenminste niets fout gaan! Ziet u hoe u met PRINT USING (regel 180) gewoon de plaats kunt aangeven waar I moet worden afgedrukt!

opdracht 7.4

Schrijf een aantal opdrachten (regels 191-195) om het voorgaande programma uit te breiden met een stukje programma waarin, aan de hand van de lijst OGEN, het gemiddelde van het aantal gegooide ogen wordt uitgerekend en afgedrukt. We weten dat er 50 keer is gegooid. De kunst is dus het totaal aantal gegooide ogen uit te rekenen aan de hand van de waarden OGEN(1), OGEN(2) t/m OGEN(6).

Nu nog een telvoorbeeld.
Om een indruk te krijgen van de leeftijd van het winkelende publiek zijn in een warenhuis 200 klanten die net iets hebben gekocht ondervraagd. Onder andere is hierbij naar hun leeftijd gevraagd. Van de 200 klanten wilden 174 hun leeftijd noemen. In onderstaand programma kunnen deze 174 leeftijden worden ingetoetst. Het programma

stopt elke leeftijd in één van de leeftijdsklassen 0-9; 10-19; 20-29; 30-39;..... tot en met 90-99 jaar en drukt de verdeling van de leeftijden over de tien klassen af. We noemen dit een frequentieverdeling.

```
100 REM  Tellen in klassen
110 '
120 INPUT "Leeftijd:"; N%
125 '***doe zolang niet -1 is ingetoetst
130 IF N%=-1 THEN 210
135     '***doe zolang verkeerde leeftojd is ingetoetst
140     IF N%>=1 AND N%<=99 THEN 170
150         INPUT "Foute leeftijd,opnieuw:";N%
160         GOTO 140
165     '***
170     N%=N%-10+1 'maak er een 1,2,3,...of 10 van
180     LFT(N%)=LFT(N%)+1 'verhoog klasse-aantal
190     INPUT "Leeftijd:"; N%
200     GOTO 130
205 '***
210 LPRINT CHR$(12)
220 LPRINT "Leeftijdsopbouw:":LPRINT
230 FOR I= 1 TO 10
240    LPRINT USING "Klasse ## : ## - ## :";I,10*(I-1),10*I-1;
250    LPRINT STRING$(LFT(I),"*")
260 NEXT I
270 END
```

We hebben de 174 leeftijden ingetoetst; hier is het resultaat:

```
Leeftijdsopbouw:

Klasse  1 :  0 -  9 :***********
Klasse  2 : 10 - 19 :*****************
Klasse  3 : 20 - 29 :************************
Klasse  4 : 30 - 39 :*************************
Klasse  5 : 40 - 49 :******************************
Klasse  6 : 50 - 59 :*************************
Klasse  7 : 60 - 69 :**********************
Klasse  8 : 70 - 79 :********
Klasse  9 : 80 - 89 :***
Klasse 10 : 90 - 99 :*
```

opdracht 7.5

Maak voor dit programma een structuurdiagram waarin de opdrachten in gewoon Nederlands worden omschreven.

7.3 SORTEREN VAN GEGEVENSLIJSTEN

Sorteren is één van de dingen die geknipt zijn om door een computer te laten uitvoeren. **Sorteren** kan namelijk worden uitgevoerd door **herhaald vergelijken** en **verschuiven** van gegevens in een te sorteren lijst. Voor ons 'mensen' is het sorteren van een redelijk groot aantal gegevens een saaie en langdurige bezigheid. De computer kent dit soort gevoelens niet; hij kan urenlang vergelijken en schuiven zonder daarbij ook maar één fout te maken.

Wel moeten 'wij' hem een sorteerprogramma (een **sorteeralgoritme**) geven aan de hand waarvan hij zo lang en nauwkeurig kan werken. Nu zijn er langzame en snelle sorteermethoden, zeg maar inefficiënte en efficiënte sorteeralgoritmen. Efficiënte sorteerprogramma's zijn vaak gecompliceerde programma's. Helaas behoren de twee methoden die wij nu gaan behandelen tot de minder efficiënte methoden, maar daartegenover staat dat ze voor de 'beginner' nog redelijk te volgen zijn. We gaan het proberen.

Sorteren door selectie

We gaan een willekeurige lijst met getallen sorteren van klein naar groot. Als voorbeeld nemen we een array X met vijf elementen. We beginnen met een ongesorteerde lijst X en eindigen met de gesorteerde lijst X. We sorteren dus '**in de lijst**' zelf. Er komt geen tweede lijst aan te pas.

We gaan uit van de volgende lijst:
X(1) = 5 X(2) = 4 X(3) = 8
X(4) = 7 X(5) = 3

kort gezegd: 5,4,8,7,3

Er zal een aantal elementen van plaats moeten verwisselen om tot de gesorteerde lijst

3,4,5,7,8

te komen.
We gaan het sorteren in stapjes uitvoeren. Sorteren van klein naar groot betekent dat de kleinste waarde voorop komt te staan. De eerste stappen zijn dan ook:

- zoek kleinste in de lijst
- zet kleinste voorop

De kleinste in de lijst is 3. Goed, we zetten 3 voorop, maar waar laten we dan die 5 die eerst voorop stond? Een wellicht voor de hand liggende oplossing is om die 5 op de plaats van de 3 te zetten, dus:

```
begin      : 5, 4, 8, 7, 3
na 1e stap : 3, 4, 8, 7, 5
             ↑ staat goed
```

Die 3 staat goed en hoeft daar niet meer weg. De rest van de lijst staat nog niet goed. Dat die 4 ook al op de goede plaats staat zien wij direct maar de computer niet!

We gaan nu vanaf het tweede element in de rest van de lijst de kleinste zoeken en die voorop zetten (nou ja...voorop!).

```
begin      : 5, 4, 8, 7, 3
na 1e stap : 3, 4, 8, 7, 5
na 2e stap : 3, 4, 8, 7, 5
```

We herhalen dit selecteren van de kleinste en dit voorop zetten tot de hele lijst gesorteerd is:

```
begin   : 5, 4, 8, 7, 3
1e stap : 3, 4, 8, 7, 5
2e stap : 3, 4, 8, 7, 5 ← die 4 stond toevallig al goed!
3e stap : 3, 4, 5, 7, 8
4e stap : 3, 4, 5, 7, 8 ← dat die 7 al goed stond kon de computer
                          niet weten, vandaar die laatste, in
                          onze ogen, overbodige stap.
```

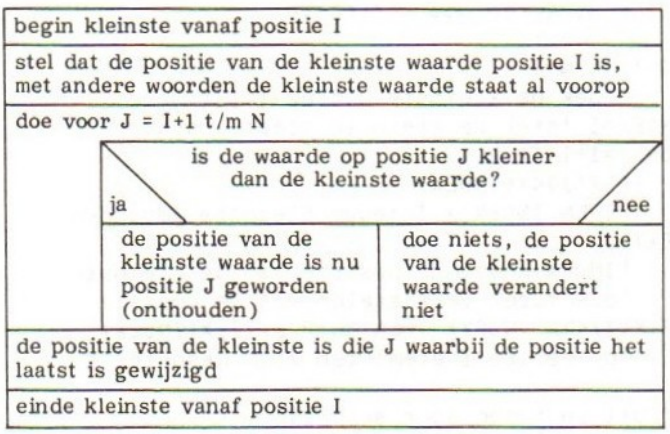
We sorteren de 5 elementen in 4 stappen. Dit komt omdat, als we het één-na-laatste element op de goede plaats hebben gezet, automatisch het laatste element ook op de goede plaats staat.

Als we een lijst hebben met N elementen, kunnen we volgens bovenstaande methode deze lijst sorteren in N-1 stappen. Een sorteerroutine die volgens deze methode te werk gaat heeft dus de volgende globale structuur:

begin sorteren door selectie
doe voor I = 1 t/m N
bepaal kleinste in de lijst vanaf positie I
verwissel eventueel het gevonden kleinste getal met de waarde op positie I, het voorste getal uit de (deel)lijst
einde sorteren door selectie

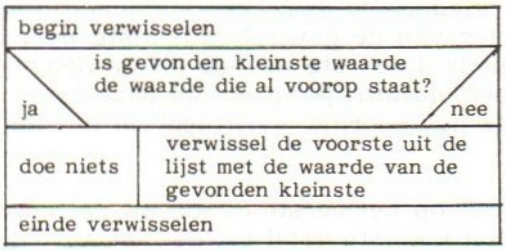
We zien dat we (N-1) keer de kleinste in een (steeds kleinere) lijst moeten bepalen. Het verwisselen van de gevonden kleinste waarde met de waarde op positie I (dit is de eerste waarde uit de lijst die nog bekeken moet worden) hoeft natuurlijk alleen als deze twee waarden ongelijk zijn, vandaar het woordje 'eventueel' in bovenstaand structuurdiagram.

Laten we ons eerst concentreren op het eerste blokje uit de 'voor-lus'. Als K de positie is van het kleinste getal uit de lijst X, dat wil zeggen de index van het kleinste element uit X, dan is X(K) de kleinste waarde in de lijst X. Het is dus voldoende om de positie van de kleinste waarde te bepalen. Laten we eens kijken hoe we in een lijst vanaf positie I de computer de positie van de kleinste waarde in die lijst kunnen laten bepalen.



De computer kiest gewoon de eerste waarde uit de te onderzoeken lijst als kleinste waarde (weet hij veel!). Vervolgens worden alle volgende waarden (posities I+1 t/m N) vergeleken met die kleinste waarde. Als hierbij een kleinere waarde wordt gevonden, wordt dit de 'nieuwe kleinste' waarde en worden alle volgende waarden met deze nieuwe kleinste vergeleken. Als we steeds onthouden op welke positie de laatste wijziging 'in de kleinste waarde' zich heeft voorgedaan, hebben we na afloop van de voor-lus de positie van de kleinste bepaald.

Als dit is gebeurd moet er worden beslist of de gevonden kleinste waarde met de waarde vooraan de lijst (positie I) moet worden verwisseld. Dit hoeft niet als de voorste de kleinste blijkt te zijn, als er dus in de rest van de lijst geen kleinere waarde is gevonden:



Als we deze laatste twee structuurdiagrammen opnemen in het structuurdiagram op bladzijde 222, komen we tot de volgende sorteerroutine:

```
1000 'subroutine sorteren door selectie
1010 '
1020    FOR I=1 TO N-1
1030          'zoek de kleinste uit de lijst
1040       INDEX=I 'stel de kleinste staat vooraan
1050       FOR J=I+1 TO N
1060          IF X(J)<X(INDEX)
                 THEN INDEX=J 'nieuwe kleinste gevonden
1070       NEXT J
1080          'INDEX bevat de positie van de kleinste
1090          'eventueel verwisselen met de voorste
1100       IF X(I)<>X(INDEX) THEN SWAP X(I),X(INDEX)
1110    NEXT I 'de eerste I elementen zijn gesorteerd
1120 '
1130 RETURN ' uit sorteren door selectie
```

Deze subroutine sorteert een lijst X met N getallen van klein naar groot. Als u dit programma zelf gaat gebruiken kunt u het wellicht voor u overbodige commentaar weglaten!

opdracht 7.6

Stel de X-lijst ziet er zo uit: 15, 7, 25, 35, 14, 15. Geef na elke doorloop van de I-lus (stap), uit bovenstaande sorteerroutine, de inhoud van de array X.

Als we in deze sorteerroutine de lijst X vervangen door de lijst X$, hebben we een routine om een lijst X$ met tekstwaarden alfabetisch te rangschikken.

SWAP A,B	verwisselt de inhoud van de variabelen A en B.

opdracht 7.7

Maak van de subroutine hierboven drie aparte modulen, één hoofdsubmodule en twee aparte submodulen, één voor het bepalen van de kleinste en één voor het verwisselen. Geef de programmatekst van alle drie de modulen. Schrijf de modulen voor het sorteren van tekst in plaats van voor het sorteren van getallen.

opdracht 7.8

Zou de IF-opdracht in regel 1100 ook zo gesteld mogen worden:
IF I <> INDEX?

Sorteren door tussenvoegen

De tweede sorteermethode is een methode waarbij de nog niet gesorteerde elementen uit een lijst op de juiste plaats in het reeds gesorteerde deel van de lijst worden tussengevoegd. Deze methode lijkt wel wat op de manier waarop sommige kaartspelers hun speelkaarten sorteren.

We kunnen ons deze manier van sorteren als volgt voorstellen. Neem een aantal speelkaarten. Schud de kaarten. Leg ze op een stapeltje op tafel. Neem nu één voor één een kaart van de stapel en voeg deze kaart direct op de juiste plaats tussen de kaarten die reeds gesorteerd in de hand worden gehouden. De eerste kaart wordt gewoon gepakt. Is de tweede kaart 'groter', dan wordt deze kaart achter de eerste gezet, anders ervoor. Is de derde kaart groter dan de tweede dan komt die gewoon achteraan, is de derde kaart kleiner dan de tweede (in de hand) maar groter dan de eerste, dan komt de derde kaart tussen de twee reeds gesorteerde kaarten in, anders komt de derde kaart helemaal vooraan. Enzovoorts! Als een kaart wordt tussengevoegd of vooraan moet worden gezet moet een aantal kaarten dat al gesorteerd is een plaatsje opschuiven.

Deze sorteermethode is dus een kwestie van **vergelijken** en **opschuiven** in plaats van **vergelijken** en **verwisselen**. We geven meteen de routine in BASIC. Zo af en toe is het een goede oefening om, met als enig houvast de programmatekst, erachter proberen te komen hoe een programma in elkaar zit!

```
2000 'subroutine sorteren door tussenvoegen
2010 '
2020    FOR I=2 TO N 'vanaf 2,want eerste staat goed
2030       VOLG=X(I)
2040       J=I-1 'vergelijk met al gesorteerde deel
2045       '***doe zolang er verschoven moet worden
2050       IF J=0 OR VOLG>=X(J) THEN 2090
2060          X(J+1)=X(J) 'schuif X(J) een plaats op
2070          J=J-1 'ga nog verder terug in de lijst
2080          GOTO 2050
2085       '***
2090       IF J+1 <> I THEN X(J+1)=VOLG 'voeg nieuwe tussen
2100    NEXT I
2110 '
2120 RETURN 'uit sorteren door tussenvoegen
```

Ook deze methode kan voor het alfabetisch rangschikken van teksten worden gebruikt. Daartoe behoeven we X slechts te vervangen door X$ en VOLG door VOLG$.

opdracht 7.9

Stel de X-lijst ziet er zo uit: 15,7,25,35,14,15. Geef na elke doorloop van de I-lus (stap) in de 'tussenvoegmethode' aan wat de inhoud van de array X is.

De twee hierboven gegeven sorteeralgoritmen zijn voor een groter aantal elementen (meer dan 30) vrij langzaam. Er zijn veel snellere sorteermethoden. We geven hiervan één voorbeeld, namelijk de 'Quicksort' volgens het algoritme van C.A.R. Hoare. We gaan deze methode niet uitleggen, maar geven alleen de subroutines in BASIC. Gebruik deze sorteermethode als u een groot aantal elementen in een array X (of X$) moet sorteren.

Quicksort

De quicksortmethode verdeelt de te sorteren array X in twee delen, zodanig dat alle array-elementen in de linkerhelft kleiner dan of gelijk zijn aan een bepaalde van tevoren bepaalde waarde en alle elementen in de rechterhelft groter dan of gelijk zijn aan die waarde. De volgorde waarin de array-elementen in het linker- en rechterdeel staan is niet van belang. Wel van belang is dat geen enkel element uit het linkerdeel nog eens met een element uit het rechterdeel vergeleken hoeft te worden.
Beide delen worden vervolgens weer in een linker- en een rechterdeel gesplitst; dit gaat door totdat alle delen nog maar één element bevatten, waarmee de array gesorteerd is.

De quicksortmethode gebruikt wel twee hulparrays die als 'stapels' dienst doen en is gemiddeld veel sneller (voor grote arrays) dan de twee eerder besproken sorteermethoden, die echter geen hulparrays nodig hebben.

```
1000 'subroutine quicksort
1010 '
1020 DIM F(100),L(100)
1030 S=1 : F(1)=1 : L(1)=N
1040 '*** doe zolang S niet nul is
1050 IF S=0 THEN 1170
1060    F1=F(S) : L1=L(S) : S=S-1
1070    '***doe zolang F1 kleiner dan L1
1080    IF F1>=L1 THEN 1140
1090        GOSUB 2000 'subroutine splits
1100        S=S+1
1110        F(S)=I : L(S)=L1 : L1=J
1120        GOTO 1080
1130    '***
1140    GOTO 1050
1150 '***
1160 '
1170 RETURN 'quicksort
1180 '
2000 'subroutine splits
2010 '
2020 I=F1 : J=L1 : X0=X(INT((F1+L1)/2))
2030 '===
2040     '*** doe zolang X(I)<X0
2050     IF X(I)>=X0 THEN 2100
2060        I=I+1
2070        GOTO 2050
2080     '***
2090     '*** doe zolang X(J)>X0
2100     IF X(J)<=X0 THEN 2140
2110         J=J-1
2120         GOTO 2100
2130     '***
2140     IF J<I THEN RETURN
2150     IF J>I THEN SWAP X(I),X(J)
2155     I=I+1:J=J-1
2160     IF J>=I THEN 2050
2170 '===herhaal tot J kleiner dan I
2180 '
2190 RETURN 'splits
```

Wilt u uitpluizen hoe de methode werkt, voeg dan tussen de regels 1060 en 1070 toe:

```
1061 FOR K = 1 TO N
1062    LPRINT USING " ### "; X(K)
1063 NEXT K
```

waardoor steeds na elke 'splits'-stap de array X op de printer wordt afgedrukt. Laat bijvoorbeeld de lijst 4,9,1,8,6,7,3,5,2 sorteren of neem 15,7,25,35,14,15 (opdrachten 7.6 en 7.9) en vergelijk quicksort met 'sorteren door selectie' en 'sorteren door tussenvoegen'. In hoofdstuk 10 kunt u lezen hoe u deze drie sorteerroutines sneller kunt maken.

7.4 AFDRUKKEN VAN VEEL GEGEVENS

Als we veel gegevens moeten afdrukken (een grote lijst), moeten we zorgen dat de **uitvoer overzichtelijk** is en dat het beeldscherm of het papier in een printer efficiënt wordt gebruikt. Stel dat we een lijst met honderd getallen willen afdrukken. Bekijk de volgende mogelijkheden:

Ⓐ
```
FOR I = 1 TO 100
   PRINT X(I)
NEXT I
```

Ⓑ
```
FOR I = 1 TO 100
   PRINT X(I),
NEXT I
```

Ⓒ
```
FOR I = 1 TO 100
   PRINT X(I);
NEXT I
```

In voorbeeld A wordt elk getal op een nieuwe regel afgedrukt. Als we achter het beeldscherm zitten zien we de eerste 76 getallen voorbijflitsen en alleen de laatste 24 blijven zichtbaar. Bij een regeldrukker krijgen we een heel lang vel dat bijna leeg is (verspilling van papier).

In voorbeeld B wordt elk getal op een standaardschermpositie (om de 14 tekens) of met LPRINT op een standaardregeldrukkerpositie afgedrukt. Op een beeldscherm met 24 regels van elk 40 tekens (WIDTH 40, 2 standaardprintposities) gaan maar 48 getallen!

In voorbeeld C worden alle getallen als één grote sliert afgedrukt, waarbij sommige getallen gedeeltelijk op de ene en gedeeltelijk op de volgende regel afgedrukt zullen worden.

We willen eigenlijk:
- meer dan één getal op een regel afdrukken;
- niet van de standaardtabulatorinstelling gebruik maken;
- ook niet alle getallen domweg achter elkaar afdrukken.

Goed, we willen dus weer eens iets anders dan hetgeen 'standaard' geboden wordt. Stel dat elk van de 100 elementen van de lijst X een getal van twee cijfers bevat. We willen 8 getallen op een regel

afdrukken met twee spaties tussen de getallen. We krijgen dus 12 regels met 8 getallen en een dertiende regel met 4 getallen. Het afdrukken van een getal van twee cijfers met twee spaties ervoor kan met PRINT USING:

```
PRINT USING "  ##"; X(I);
```

Als we even uitgaan van 100 getallen in de array X moet dus het 9e, het 17e, het 25e,..... en het 97e getal aan het begin van een nieuwe regel worden afgedrukt. Laten we hier het 1e getal maar bij doen!, dat moet tenslotte ook aan het begin van een nieuwe regel staan. Als de getallen uit de array X met een FOR-NEXT lus worden afgedrukt moet dus in de lus worden getest of de lusteller (I) gelijk is aan 1, aan 9, aan 17, aan 25, ... of aan 97. Nu valt ons meteen op (nou ja... meteen?) dat

```
 1 gelijk is aan  0×8+1      èn dat
 9 gelijk is aan  1×8+1      èn dat
17 gelijk is aan  2×8+1      èn dat
                   :
97 gelijk is aan 12×8+1
```

Als I dus gelijk is aan een geheel aantal malen acht plus één, dan moet naar een nieuwe regel gesprongen worden. Anders gezegd: als er na deling van I door 8 een rest 1 overblijft dan naar een nieuwe regel.

```
8/17\2          vergelijk dit met    8/ I  \K
 16                                    K×8
 -                                     -
  1                                   rest
```

K is het geheel aantal maal 8 dat 'in I zit'. In BASIC is K gelijk aan INT(I/8). De REST is dus gelijk aan I - INT(I/8)*8.
We kunnen ook gebruik maken van geheeltallig delen of van de MOD-functie. Geheeltallig delen geven we aan met het bewerkingsteken \ (back slash). I\8 heeft hetzelfde effect als INT(I/8). We kunnen de REST, die na deling van I door 8 overblijft, ook schrijven als

```
REST = I - (I\8)*8
```

In plaats van de INT-functie of geheeltallig delen (\) kunnen we de REST ook bepalen met de MOD-functie. We schrijven dan

```
REST = I MOD 8
```

De MOD functie heeft twee rekenkundige uitdrukkingen als operanden nodig. De waarde 'links' van MOD is het deeltal; de waarde 'rechts' van MOD is de deler. De MOD-functie trekt net zo lang de deler van het deeltal af tot er een rest overblijft die kleiner is dan de deler zelf (MOD komt van MODulo). Zonodig maakt MOD (en ook \) van de waarden die er links en rechts van staan eerst gehele getallen.

We kunnen de volgende definitie van MOD geven:

A MOD B = INT(A) - INT(A\B)*INT(B)

opdracht 7.10

a. Geef de uitkomst van de volgende geheeltallige delingen:

5\4; 25\7; 16\3,5; 21,7\2,8

b. Geef de uitkomst van de volgende MODulo-delingen; gebruik de bovenstaande definitie van MOD:

48 MOD 7; (4×8+2) MOD 3; 18 MOD 9,7; 348,76 MOD 23,45

De FOR-NEXT-lus, waarin we om de 8 getallen naar een nieuwe regel moeten springen, kunnen we dus op z'n kortst zo formuleren:

```
FOR I = 1 TO 100
   IF I MOD 8 = 1 THEN PRINT
   PRINT USING " ##"; X(I);
NEXT I
```

denk aan de puntkomma

We vinden dit terug in het onderstaande programma in de subroutine 'lijst afdrukken' (regels 3000-3050). In deze subroutine nemen K en FS$ de plaats in van respectievelijk 8 en " ##". De waarden voor K en FS$ worden eerst uit de DATA-lijst ingelezen. Het programma bevat verder een DATA-lijst met te sorteren getallen, die in de ARRAY X wordt ingelezen. Deze ongesorteerde lijst wordt vervolgens afgedrukt, gesorteerd en nogmaals afgedrukt.

```
100 REM  lijst inlezen,afdrukken,sorteren en afdrukken
110 '
120 READ N    'lengte van de datalijst
130 READ K    'aantal getallen op uivoerregel
140 READ FS$ 'formatstring voor uitvoerregel
145 DIM X(N)
150 ' Lees datalijst
160 FOR I= 1 TO N
170    READ X(I)
180 NEXT I
190 'Lijst afdrukken
200 LPRINT CHR$(12);"Ongesorteerde lijst:":LPRINT
210 GOSUB 3000 'lijst afdrukken
220 'Lijst sorteren
230 GOSUB 2000 'sorteren door tussenvoegen
240 'Lijst weer afdrukken
```

```
250 LPRINT:LPRINT:LPRINT "Gesorteerde lijst:":LPRINT
260 GOSUB 3000 'lijst afdrukken
270 END
280    DATA 34, 8, "  ##"
290    DATA 31,26,12,73,41,70,83,53,31,98,53,32,81,18,
            75,83,76,43,53,13,34,20,98,36,70,85,65,62,
            49,42,61,56,38,59
2000 'subroutine sorteren door tussenvoegen
2010 '
2020    FOR I=2 TO N 'vanaf 2,want eerste staat goed
2030       VOLG=X(I)
2040       J=I-1 'vergelijk met al gesorteerde deel
2045       '***doe zolang er verschoven moet worden
2050       IF J=0 OR VOLG>=X(J) THEN 2090
2060           X(J+1)=X(J) 'schuif X(J) een plaats op
2070           J=J-1 'ga nog verder terug in de lijst
2080           GOTO 2050
2085       '***
2090       IF J+1 <> I THEN X(J+1)=VOLG 'voeg nieuwe tussen
2100    NEXT I
2110 '
2120 RETURN 'uit sorteren door tussenvoegen
3000 'subroutine lijst afdrukken
3005 '
3010    FOR I=1 TO N
3020       IF I MOD K= 1 THEN LPRINT
3030       LPRINT USING FS$;X(I);
3040    NEXT I
3045 '
3050 RETURN 'uit lijst afdrukken
```

De uitvoer is:

```
Ongesorteerde lijst:


  31  26  12  73  41  70  83  53
  31  98  53  32  81  18  75  83
  76  43  53  13  34  20  98  36
  70  85  65  62  49  42  61  56
  38  59

Gesorteerde lijst:


  12  13  18  20  26  31  31  32
  34  36  38  41  42  43  49  53
  53  53  56  59  61  62  65  70
  70  73  75  76  81  83  83  85
  98  98
```

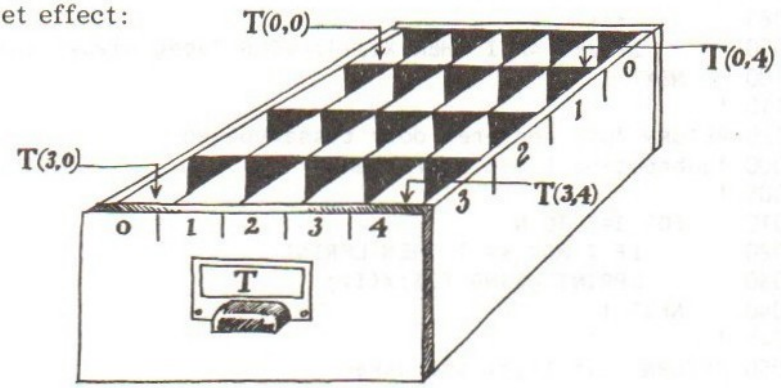
7.5 MEER-DIMENSIONALE GEGEVENSSTRUCTUUR

De tot nu toe behandelde lijststructuur zou je ook **één-dimensionale tabelstructuur** kunnen noemen. Een lijst is een tabel met maar één kolom. We kunnen in BASIC ook een 'echte' tabelstructuur, met **rijen** en **kolommen**, creëren. Dit doen we door een tabelvariabele twee indexen te geven, één om de rij aan te geven en één om de kolom aan te geven. In de onderstaande DIM opdracht vertellen we de computer dat we met een tabel met drie rijen en vier kolommen willen gaan werken, terwijl we de tabel de naam T geven:

```
10  DIM  T (3, 4)
```

naam van de tabel — 3 rijen — 4 kolommen

Dit is het effect:



Een geheugenlade met etiket 'T' en 20 vakjes.
De 20 elementen van de tabel T worden zo benoemd:

rij-0	T(0,0)	T(0,1)	T(0,2)	T(0,3)	T(0,4)
rij-1	T(1,0)	T(1,1)	T(1,2)	T(1,3)	T(1,4)
rij-2	T(2,0)	T(2,1)	T(2,2)	T(2,3)	T(2,4)
rij-3	T(3,0)	T(3,1)	T(3,2)	T(3,3)	T(3,4)
	↑	↑	↑	↑	↑
	kolom-0	kolom-1	kolom-2	kolom-3	kolom-4

Ook bij tweedimensionale arrays geldt dat de computer zonder tegenbericht (default) bij rij-0 en kolom-0 begint.

Zo'n tweedimensionale tabel wordt ook wel een **matrix** genoemd. In een bepaald onderdeel van de wiskunde, de lineaire algebra, wordt erg veel gebruik gemaakt van deze matrices. Met name voor dit 'stukje wiskunde' zijn veel computerprogramma's geschreven, waarin de meest fantastische dingen met matrices (matrixrekening)

worden uitgehaald om tot een bepaalde oplossing te komen. In dit boek gaan we hier niet verder op in. Wees gerust, ook buiten de 'lineaire algebra' kunnen we van zo'n matrix goed gebruik maken.

Stel dat we een marktonderzoek gaan houden naar de mate waarin mensen bepaalde produkten waarderen. We kiezen vier proefpersonen uit die elk drie produkten moeten keuren en waarderen met het cijfer 1,2,3,4,5,6,7,8,9 of 10. Zo zouden we ons kunnen voorstellen dat we vier wijnproevers elk drie soorten wijn laten proeven. Elk cijfer is gebaseerd op het proeven van één glaasje van de desbetreffende soort wijn.
De eerste wijnproever gaf de cijfers 5,6,5; de tweede gaf de cijfers 6,7,5; de derde de cijfers 7,6,6 en de vierde de cijfers 4,5,7.

Met deze 'steekproefuitkomst' kunnen we een aantal statistische analyses uitvoeren:
- We kunnen nagaan of de produkten onderling verschillend gewaardeerd worden (vergelijk de rijen).
- We kunnen nagaan of de proefpersonen onderling op verschillende wijze waarderen (vergelijk de kolommen).
- We kunnen nagaan of de produkten en proefpersonen elkaar op een bepaalde manier beïnvloeden (bekijk de matrix).

Als we deze analyses met de computer willen gaan uitvoeren zou het handig zijn om in het programma over de matrix met de steekproefuitkomsten te beschikken. Laten we dus deze matrix eerst inlezen:

```
100 REM Statistische analyse tweedimensionale tabel
105 '
110 DIM UITSLAG(3,4)
120 FOR RIJ = 1 TO 3
130    FOR KOLOM = 1 TO 4
140       READ UITSLAG(RIJ,KOLOM) ' Lees matrixelement
150    NEXT KOLOM
160 NEXT RIJ
300 END
310       DATA 5, 6, 7, 4
320       DATA 6, 7, 6, 5
330       DATA 5, 5, 6, 7
```

De waarderingscijfers zijn nu rij voor rij ingelezen en opgeslagen in de array (matrix) UITSLAG. We kunnen nu aan de hand van deze array allerlei dingen gaan berekenen en de eerder genoemde analyses gaan uitvoeren. De subroutines die we hiervoor gaan maken willen we algemeen houden, dat wil zeggen voor een tabel met een willekeurig aantal rijen en kolommen. Daartoe voeren we eerst twee variabelen in, te weten NR voor het aantal rijen en NK voor het aantal kolommen:

```
170 NR = 3
180 NK = 4
```

De eerste subroutine wordt een subroutine waarin het gemiddelde in elke rij wordt afgedrukt:

```
190 GOSUB 2000 'rij-gemiddelden
```

Deze routine ziet er als volgt uit:

```
2000 ' Subroutine rij-gemiddelden
2010 '
2020     FOR I = 1 TO NR
2030        SOM = 0
2040        FOR J = 1 TO NK
2050           SOM = SOM + UITSLAG(I,J)
2060        NEXT J
2070        PRINT "Het gemiddelde van rij ";I;" is "; SOM/NK
2080     NEXT I
2090 '
2100 RETURN ' uit rij-gemiddelden
```

Ziet u hoe handig geneste FOR-NEXT-lussen kunnen zijn als we met tweedimensionale tabellen werken!

opdracht 7.11

Schrijf een routine 3000 voor het afdrukken van de kolomgemiddelden, zodat we de volgende regel aan het hoofdprogramma kunnen toevoegen:

```
200 GOSUB 3000 'kolom-gemiddelden
```

Als we van de regels 2020 en 2030 de regelnummers verwisselen en ook die van de regels 2070 en 2080 en bovendien de nieuwe regel 2080 programmeren als

```
2080 PRINT "Het gemiddelde in de tabel is"; SOM/(NR*NK)
```

dan hebben we een subroutine voor het berekenen van het gemiddelde van alle tabelwaarden.

opdracht 7.12

Als we net als de uitslagen zelf ook de rijgemiddelden en de kolomgemiddelden, die in de routines 2000 en 3000 worden berekend en afgedrukt, in arrays (RIJGEM en KOLOMGEM) willen opslaan, wat moet er dan aan deze routines en aan het hoofdprogramma worden toegevoegd?

We zullen de hiervoor genoemde statistische analyses niet echt gaan programmeren. In de eerste plaats omdat daarvoor de nodige specifieke kennis is vereist en in de tweede plaats omdat het allang vele malen door anderen is gedaan. Als we aannemen dat deze subroutines reeds geprogrammeerd zijn en in een **programmabibliotheek** op een achtergrondgeheugen liggen opgeslagen, hoeven we alleen maar de gewenste routines op te zoeken en aan ons programma te 'koppelen'. Hoe dit opzoeken en aankoppelen gebeurt zien we in hoofdstuk 9.

Stel dat ook het inlezen van de matrix met uitslagen door een bibliotheekroutine kan worden uitgevoerd. Dan wordt het ontwikkelen van een analyseprogramma gereduceerd tot het schrijven van een hoofdprogramma waarin die routines worden aangeroepen en het aankoppelen van die bibliotheekroutines aan het hoofdprogramma.

```
100 REM  Statistische analyse productwaardering
105 '
110 READ NR,NK ' Lees aantal rijen en kolommen
120 DIM UITSLAG(NR,NK), RIJGEM(NR), KOLOMGEM(NK)
130 '
140 ' Roep bibliotheekroutines aan
150 '
160 GOSUB 1000 ' Invoer uitslagen
170 GOSUB 2000 ' Rij-gemiddelden
```

```
180 GOSUB 3000 ' Kolom-gemiddelden
190 GOSUB 4000 ' Rij-analyse
200 GOSUB 5000 ' Kolom-analyse
210 GOSUB 6000 ' Tabel-analyse
220 '
230 END
240       DATA 3, 7
250       DATA 3, 5, 4, 6, 7, 8, 2
260       DATA 4, 7, 5, 6, 9, 8, 1
270       DATA 1, 4, 3, 4, 5, 4, 3
```

↓ Hier komt een aantal besturingsopdrachten om de routines 1000 t/m 6000 vanuit een achtergrondgeheugen in het werkgeheugen van de computer te halen. Zie hoofdstuk 9.

Computers kunnen ook werken met drie-, vier- en nog meer-dimensionale tabellen. Stel dat we de vier wijnproevers van elke soort wijn drie flessen geven. Uit elke fles moeten zij een glaasje proeven en elk glaasje moeten ze met een cijfer waarderen. Zo krijgen we per wijnproever per soort wijn drie cijfers. Tenslotte is het heel gevaarlijk ons oordeel over de kwaliteit van een bepaalde soort wijn (èn van de wijnproevers) te baseren op maar één glas uit één fles van die soort.

We krijgen nu drie keer zoveel cijfers uit het onderzoek. Deze cijfers kunnen we in een driedimensionale tabel opslaan:

```
10 DIM UITSLAG(3, 4, 3)
```

3 soorten wijn — 4 wijnproevers — 3 flessen van elke soort

Als nu UITSLAG(3,1,2) gelijk is aan 8, betekent dit dat van wijnsoort 3 de wijnproever met nummer 1 de tweede fles met het cijfer 8 heeft gewaardeerd.

Aan de hand van deze driedimensionale UITSLAG-tabel kunnen we weer diverse analyses uitvoeren. Theoretisch kunnen we in BASIC nog meer dimensies invoeren. Het aantal dimensies wordt eigenlijk alleen beperkt door de beschikbare geheugenruimte. Hierover gaat de volgende en laatste paragraaf.

Tot slot nog een paar opmerkingen over het gebruik van arrays. Eigenlijk begint elk array niet met element 1, maar met element 0. Zo zouden we in het sorteerprogramma uit 7.4 in plaats van DIM X(N) de opdracht DIM X(N-1) kunnen gebruiken; immers het array begint met element 0, dus kunnen de N-elementen als volgt genummerd worden: 0,1,2,3,...,N-1. Dit betekent wel dat we de FOR-opdracht in regel 160 moeten schrijven als

```
160 FOR I = 0 TO N-1
```

of de READ-opdracht in regel 170 als

```
170 READ X(I-1)
```

Om dit soort 'onlogische' zaken te vermijden beginnen wij gewoon te tellen vanaf element 1; de computer reserveert dan echter toch ook ruimte voor element 0, jammer!
Deze ruimteverspilling is bij eendimensionale arrays beperkt tot één array-element. Bij meerdimensionale arrays kan dit echt tot verspilling van geheugenruimte leiden. Bekijk nog eens de DIM-opdracht uit het wijnproeversvoorbeeld hier vlak boven:

```
DIM UITSLAG(3,4,3)
```

De computer reserveert ruimte voor 4×5×4 = 80 array-elementen omdat voor elke dimensie ook een 0-element meedoet. We zouden in dit voorbeeld echter de opdracht

```
DIM UITSLAG(2,3,2)
```

hebben kunnen gebruiken, hetgeen een ruimtebeslag voor 3×4×3 = 36 array-elementen zou betekenen. Dit is dus minder dan de helft! Als we dit doen dan moeten we wel overal in de subroutines de indexen RIJ en KOLOM van de array UITSLAG vervangen door RIJ-1 en KOLOM-1. Voor wie echt met geheugenruimte moet woekeren zal dit wellicht onvermijdelijk zijn.

Nog een manier om, bij het gebruik van arrays, geheugen te sparen is te kijken naar de 'soort' getallen die in numerieke arrays moeten worden opgeslagen. In het wijnproeversvoorbeeld bevat de array UITSLAG gehele getallen tussen 0 en 10. Het zou daarom beter (uit oogpunt van geheugengebruik) geweest zijn om de array UITSLAG ook als geheeltallig (integer) array te DIMensioneren:

```
DIM UITSLAG%(3,4,3)
```

Hoeveel geheugenplaatsen scheelt dit? Bekijk de onderstaande screendump eens:

```
10 dim uitslag(3,4,3)
20 print fre(0)
run
 23398
Ok
10 dim uitslag!(3,4,3)
run
 23717
Ok
10 dim uitslag%(3,4,3)
run
 23877
Ok
10 dim uitslag%(2,3,2)
run
 23965
Ok
_


color  auto   goto   list   run
```

Bovenaan het scherm zien we een tweeregelig programma met de DIM-opdracht zoals die in het wijnproeversvoorbeeld gebruikt is. Direct daaronder zien we de PRINT FRE(0)-opdracht die de hoeveelheid vrije geheugenruimte afdrukt. Als we van de array uitslag(3,4,3) (double precision array) een array maken voor getallen met enkele nauwkeurigheid (single precision array), namelijk uitslag!(3,4,3), dan scheelt dat al 23717 - 23398 = 319 geheugenplaatsen (bytes). Maken we er een geheeltallig array van, met uitslag%(3,4,3), dan scheelt dat ten opzichte van uitslag(3,4,3) nog meer, namelijk 23877 - 23398 = 479 geheugenplaatsen. Deze waarden gelden voor DISK BASIC. Voor gewoon BASIC moet u de waarden, die PRINT FRE(0) afdrukt, met 4360 verhogen.

Volgen we de bovenstaande suggestie om van de 0-elementen gebruik te maken (uitslag%(2,3,2) dan zien we dat dit ten opzichte van uitslag(3,4,3) in totaal 23965 - 23398 = 567 geheugenplaatsen scheelt. Dit is meer dan ½K (512) geheugen. Op een veelal bij MSX-computers beschikbaar vrij geheugen van 24½K toch een respectabele winst.

Pas op met grote arrays. Als we bijvoorbeeld in bovenstaand programmaatje regel 10 als volgt intoetsen:

```
10 dim uitslag(15,15,15)
```

en het draaien dan zien we een 'out of memory in 10'-foutmelding. Blijkbaar gebruiken 15×15×15 = 3375 double precision getallen meer ruimte dan ervoor beschikbaar is. In zo'n geval moeten we wel werken met integer arrays en 0-elementen; zo zal uitslag%(14,14,14) nog 17656 bytes vrije geheugenruimte overlaten.

Een allerlaatste opmerking over arrays is dat, als we een array van maximaal 11 elementen (de elementen 0 t/m 10) in een programma willen gebruiken, we geen DIM-opdracht voor dat array hoeven te coderen. Zo zouden we in het eerste programma in dit hoofdstuk regel

```
20 DIM NAAM$(11)
```

weg kunnen laten, als we dan tenminste regel 30 als volgt veranderen:

```
30 FOR I = 0 TO 10
```

Toch is het, vooral bij de grotere programma's, raadzaam ook voor dergelijke kleine arrays aan het begin van een programma DIM-opdrachten op te nemen. Hierdoor is in één oogopslag te zien welke arrays in een programma gebruikt worden.

OEFENOPGAVEN HOOFDSTUK 7

1. Welke waarden krijgen de elementen van de lijst X in het volgende stukje programma?

```
FOR I = 1 TO 15
    X(I) = I^2
NEXT I
```

2. Gegeven zijn twee lijsten X en Y. Beide lijsten bevatten 15 elementen. Schrijf een stukje programma (met DIM opdrachten) waarin een nieuwe lijst Z wordt gemaakt waarbij elk element uit Z gelijk is aan de grootste van de twee overeenkomstige elementen uit X en Y. (Z(4) is dus gelijk aan de grootste van X(4) en Y(4)).

3. Schrijf een programma dat woorden inleest en bijhoudt uit hoeveel letters elk woord bestaat. De woorden worden via het toetsenbord ingetoetst. Het maximum aantal letters per woord is 15. Wordt er een woord met meer dan 15 letters ingetoetst,

dan moet er een foutmelding worden afgedrukt en moet om nieuwe invoer worden gevraagd. Als de gebruiker de lege string (alleen RETURN) intoetst wordt het programma beëindigd. Druk af hoeveel woorden met 1,2,3,4,5,6,..., of 15 letters er zijn ingetoetst.

4. Wat wordt er door het volgende programma afgedrukt en wat doet het programma?

```
10 READ N
15 DIM X(N)
20 FOR I = 1 TO N
25    READ X(I)
30 NEXT I
35 MAX = X(1)
40 FOR I = 2 TO N
45    IF X(I) > MAX THEN MAX = X(I)
50 NEXT I
55 PRINT MAX
60 END
65     DATA 10
70     DATA 15,7,6,8,20,28,35,4,9,21
```

5. Schrijf een subroutine die de elementen X(I) t/m X(N) uit de lijst X in omgekeerde volgorde zet.

6. Schrijf een subroutine om een willekeurig aantal elementen (N) uit een lijst X af te drukken op de regeldrukker. De waarden in de array X zijn getallen met twee cijfers voor de komma en twee erachter (PRINT USING). Tussen de getallen moeten twee spaties worden opengelaten. Er moeten negen getallen op een regel worden afgedrukt.

7. Bij statistisch onderzoek wordt vaak het rekenkundig gemiddelde van alle steekproefuitkomsten als maat voor het 'midden' van die steekproefuitkomsten genomen. Er is echter nog een belangrijke 'centrummaat' en dat is de mediaan. De mediaan is de middelste waarneming na rangschikken van alle waarnemingen van klein naar groot. Zo is bij de uitkomsten 5,7,9,4,2,1,8 de mediaan 5, want dit is na rangschikken (1,2,4,5,7,8,9) de middelste.

   Schrijf een programma dat uit een DATA-lijst eerst het aantal in te lezen getallen inleest, gevolgd door de getallen zelf. Deze getallen worden opgeslagen in de array X. De array X wordt gesorteerd (roep alleen de sorteerroutine aan) en het middelste element, de mediaan, wordt afgedrukt. Bij een even aantal elementen is de mediaan het rekenkundig gemiddelde van de middelste twee waarnemingen (na rangschikken). Een getal N is even als N MOD 2 gelijk is aan nul!

8. Schrijf een programma dat de matrix METING(4,5), vanuit een DATA-lijst, rij voor rij inleest en dat vervolgens elke rij laat sorteren met behulp van een sorteerroutine. De sorteerroutine sorteert een ééndimensionaal array X met N elementen. Zorg dus dat vóór de subroutine-aanroep de array X en N de juiste waarden krijgen en denk aan de DIM opdrachten. Druk de gesorteerde tabel af.

9. Bij een opinie-onderzoek heeft men de geënqueteerden naar leeftijd in bepaalde klassen ingedeeld. Hieronder zien we de leeftijdsverdeling van 300 ondervraagden:

klasse	frequentie	cum.freq.	rel.freq.	cum.rel.freq.
0-20	45	45	.	.
21-30	77	122	.	.
31-40	80	202	.	.
41-60	50	252	.	.
61-90	48	300	.	.

De eerste kolom geeft de klasse-indeling weer. De tweede kolom de frequentie (het aantal) per klasse. De derde kolom geeft de cumulatieve frequentie weer. Voor de klasse 31-40 is deze cumulatieve frequentie gelijk aan 45+77+80 = 202. Deze 202 is het aantal geënqueteerden met een leeftijd tot 40 jaar. De vierde kolom is de relatieve frequentie. Deze kolom geeft voor elke klasse aan welk deel (percentage) van het totaal aantal ondervraagden in die klasse valt. Ook deze kolom kunnen we cumulatief berekenen en dat geeft de laatste kolom.
Schrijf een BASIC programma dat de klasse-indeling in een array KLASSE$ inleest en dat de frequentie in een array FR% inleest. Eerst wordt het aantal klassen ingelezen, dan steeds een klasse-omschrijving (als string) en de frequentie in die klasse. Deze gegevens staan in DATA-lijsten. Als deze arrays zijn ingelezen drukt het programma de vijf kolommen af zoals dat hierboven is aangegeven. Maak de lay-out zo mooi als u zelf wilt.

10. We gaan nog even terug naar de wijnproevers.
De array UITSLAG(3,4,3) bevat 3×4×3 = 36 cijfers voor evenzoveel geproefde glazen wijn. Schrijf een subroutine waarin voor elke soort wijn (eerste index) een gemiddeld cijfer wordt berekend en afgedrukt voor alle glazen die van die soort zijn geproefd.

11. Iedereen weet dat in de woorden die wij gebruiken de letter 'e' het meest voorkomt. Maar hoe zit dat met de andere letters? Schrijf een programma waarmee een onbeperkt aantal woorden, via het toetsenbord, kan worden ingetoetst en waarin de letterfrequentie in de woorden wordt bijgehouden. Gebruik een FOR-NEXT lus met daarin de MID$ functie om letter voor letter uit

een woord te bekijken en gebruik de ASC-functie om een letter in een getal te veranderen. Ga uit van hoofdletters (zie ASCII-tabel in Appendix D).
Registreer het voorkomen van een bepaalde letter in de array L door een bepaald element van die array met één te verhogen. De elementen van die array zijn L(1) t/m L(26). De ASCII-codes van de letters A t/m Z lopen echter niet van 1 t/m 26!
Als de gebruiker het woord fini intoetst wordt op de regeldrukker een histogram afgedrukt waarin de letterfrequentie in alle ingetoetste woorden weergegeven wordt. Dit histogram ziet er zo uit:

```
A : ************************
B : **
C : **
D : ************
E : ************************************************************
F : **
G : *********
H : *****
I : **************
J : ***
K : **********
L : ***********
M : *******
N : ****************************
O : ****************************
P : ******
Q : **
R : *************************
S : ***********
T : ******************************
U : *********
V : *******
W : ****
X : *
Y :
Z :
```

(Nog mooier is niet de absolute frequentie maar de relatieve frequentie af te drukken, d.w.z. voor elke letter een percentage van het totaal aantal ingetoetste letters; dan kunnen we ook voor een heel groot aantal woorden het histogram netjes op de regeldrukker afdrukken zonder van het 'papier' te raken.)
Maak desnoods eerst via TOP-DOWN ontwerp een aantal structuurdiagrammen om een goed beeld te krijgen van 'hoe het worden moet'.

N.B. Als u dit programma draait, druk dan eerst de hoofdlettervastzettoets in. Wilt u de woorden met kleine letters intikken, vervang dan regel 1040 door:

```
1040 CODE% = ASC(L$) - 96
```

8 Tekstbewerking

In de vorige hoofdstukken zijn we al een aantal functies tegengekomen die met of op tekst (strings) werken. Laten we ze nog eens op een rijtje zetten.

ASC("+")	geeft de ASCII code van het opgegeven teken (in dit geval het + teken)
CHR$(65)	geeft een string bestaande uit één teken met de opgegeven ASCII code (in dit geval code 65)
LEN(A$)	geeft het aantal tekens in de string A$
LEFT$(A$,n)	geeft een string bestaande uit de eerste n tekens van de string A$
MID$(A$,n,k)	geeft een string bestaande uit k tekens geteld vanaf het n-de teken van de string A$
RIGHT$(A$,n)	geeft een string bestaande uit de laatste n tekens van de string A$
SPACE$(n)	geeft een string bestaande uit n spaties
STRING$(n,"*")	geeft een string bestaande uit n dezelfde tekens (in dit geval het * teken)
VAL("1234")	maakt van een string een getal, waarmee kan worden gerekend.

In dit hoofdstuk zullen we laten zien dat BASIC nog meer met tekst, zeg maar strings, kan doen. Met name gaan we kijken naar hoe je in een stuk tekst bepaalde stukjes tekst kunt opzoeken en hoe je bepaalde stukjes tekst door andere stukjes tekst kunt vervangen. Zo krijgen we een idee hoe de 'echte' tekstverwerkers werken. Ook laten we zien hoe je een woordspelletje kunt programmeren en hoe je de tekst van een programma op het beeldscherm precies zo op de printer kunt afdrukken.

8.1 HET OPZOEKEN VAN BEPAALDE STUKJES TEKST

Het opzoeken van een stukje tekst binnen een grotere tekst doen we met de **INSTR**-functie (INSTRing). Een voorbeeld:

```
10 TEKST$="cursus MSX BASIC"
20 ZOEK$="MSX"
30 PRINT INSTR(TEKST$,ZOEK$)
40 END

run
 8
Ok
```

de zoekstring begint op de 8-ste positie in de tekst

De functie INSTR geeft de plaats in de tekst (TEKST$) waar het opgegeven stukje (ZOEK$) is gevonden. Wordt het stukje tekst niet gevonden dan geeft INSTR de waarde nul.

In het volgende programma worden een viertal namen (voorletter(s) plus achternaam) in een lijst NAAM$ ingelezen. De namen staan in een DATA-lijst. Elke naam bestaat uit één of meer aan elkaar geschreven voorletters en een achternaam. Tussen voorletters en achternaam staat altijd één spatie.

```
10 READ N
20 DIM NAAM$(N
30 FOR I=1 TO N
40     READ NAAM$(I)
50 NEXT I
90 END
100    DATA 4
110    DATA "AJJ Bakker","CJMHM de Groot"
120    DATA "T Muller","KM Buitelaar"
```

We gaan een subroutine schrijven die deze namen afdrukt, maar dan eerst de achternaam en dan de voorletters. De subroutine-aanroep is eenvoudig te programmeren:

```
60  GOSUB 1000 'namen-afdrukken
```

De structuur van de subroutine is ook niet zo ingewikkeld:

begin namen-afdrukken	
doe voor I = 1 t/m N	
	neem naam uit de lijst
	splits de naam in voorletters en achternaam
	druk eerst achternaam af, dan de voorletters
einde namen afdrukken	

Het splitsen van de naam in voorletter(s) en achternaam is een kwestie van het lokaliseren van de plaats van de spatie tussen voorletter(s) en achternaam. Als deze plaats bekend is, is ook bekend hoeveel tekens voor de spatie komen en hoeveel erna, want de lengte van de hele naam is bekend. We gaan alleen dit 'splitsen' programmeren.

```
PLAATS=INSTR(NAAM$(I)," ") 'zoek de spatie
LINKS$=LEFT$(NAAM$(I),PLAATS-1) 'pak voorletters
L=LEN(NAAM$(I))
RECHTS$=RIGHT$(NAAM$(I),L-PLAATS) 'pak achternaam
```

PLAATS is de plaats (de positie) van de spatie tussen de voorletter(s) en de achternaam. Het aantal voorletters is dus PLAATS-1 en de achternaam bevat dan een aantal tekens dat gelijk is aan het totaal aantal tekens in NAAM$(I) minus PLAATS.
Met het oog op de volgende opdracht zijn we de regelnummers vergeten.

opdracht 8.1

Programmeer de hele subroutine namen-afdrukken. Gebruik de standaardtabulatorposities voor het afdrukken van achternaam en voorletter(s).

We kunnen binnen één tekst ook **vaker** naar een **bepaald stukje tekst zoeken**. Beter gezegd we kunnen bepalen hoe vaak een bepaald stukje tekst in een stuk tekst voorkomt. Laten we eens een programma ontwikkelen waarmee de computer het aantal spaties (een spatie is ook een 'stukje tekst'!) in de door de gebruiker ingetoetste zinnen telt.

begin spaties tellen	
lees zin in	
doe zolang de gebruiker dat wil	
	tel aantal spaties in de ingetoetste zin
	lees volgende zin in
einde spaties tellen	

Zonder ons druk te maken hoe dat 'spaties tellen' moet worden gedaan kunnen we deze hoofdmodule reeds coderen:

```
10 REM  spaties tellen
20 '
30 LINE INPUT "Uw zin is:"; ZIN$
35 '***doe zolang niet stop of STOP getoetst is
40 IF ZIN$="stop" OR ZIN$="STOP" THEN 80
50     GOSUB 1000 'tel spaties
60     LINE INPUT "Uw zin is:"; ZIN$
70     GOTO 40
75 '***
80 END
```

Nu wordt het tijd om over het tellen van de spaties te gaan nadenken.

begin tel-de-spaties
zet teller op nul
zoek spatie
zolang einde zin nog niet bereikt
verhoog teller met 1
zoek volgende spatie
druk het aantal spaties af
einde tel-de-spaties

Is er geen spatie gevonden, hetgeen betekent dat de INSTR functie de hele zin heeft afgezocht, dan valt er niets (meer) te tellen, dus zijn we klaar. We weten dat als de INSTR functie niet het opgegeven stukje tekst (in dit geval een spatie) vindt, de waarde van INSTR nul is. Het vinden van de volgende spatie is iets lastiger. De INSTR functie begint altijd vooraan in de opgegeven tekst te zoeken, ... tenzij wij aangeven dat we vanaf een bepaalde positie willen gaan zoeken. Dat doen we zo:

```
INSTR(n,TEKST$,ZOEK$)
```

Nu wordt binnen TEKST$ gezocht naar ZOEK$, beginnend bij het n-de teken uit TEKST$. Als we dus een spatie gevonden hebben, verhogen we de positieteller met één en we gaan verder zoeken vanaf het teken na de gevonden spatie.

Dit geeft ons voldoende aanwijzingen om de subroutine te kunnen programmeren.

```
1000 'subroutine tel-de-spaties
1010 '
1020   TELLER=0
1030   P=INSTR(ZIN$," ")
1035   '***doe zolang spatie gevonden
1040   IF P=0 THEN 1080
1050       TELLER=TELLER+1
1060       P=INSTR(P+1,ZIN$," ")
1070       GOTO 1040
1075   '***
1080   PRINT:PRINT "Deze zin bevat";TELLER;"spatie(s)"
1090 '
1100 RETURN 'uit tel-de-spaties
```

opdracht 8.2

Stel dat elk element van de array PROG$ een BASIC opdracht als tekst bevat. Er zijn 10 elementen in PROG$. PROG$ bevat dus de tekst van een 10-regelig BASIC programma. Schrijf een stukje programma dat uitrekent hoeveel maal in dat programma van de LEFT$ functie gebruik gemaakt wordt!

8.2 HET VERANDEREN VAN BEPAALDE STUKJES TEKST

Als we een BASIC-programma intoetsen doen we eigenlijk niets anders dan het invoeren van een stuk tekst. De computer slaat deze tekst gewoon in zijn geheugen op en weet dat het om een BASIC programma gaat. Als we tekst intikken maken we fouten, doodgewone tikfouten! Meestal merken we meteen dat we een verkeerde toets hebben aangeslagen, maar soms merken we dat pas als we al een aantal regels verder zijn of als we het programma draaien. In het laatste geval zal de computer ons, door middel van een foutmelding, erop wijzen waar de fout ontdekt is en om wat voor soort fout het gaat.

Als we een fout in de programmatekst ontdekken moeten we deze fout verbeteren. Dit heet 'editen'. MSX computers beschikken over een zeer handige schermeditor, waarmee we BASIC opdrachten in een programmatekst kunnen verbeteren en wijzigen. We LISTen gewoon de programmaregel met de tikfout(en) op het scherm. We gebruiken de cursor-toetsen om de cursor naar de foute positie te brengen, verbeteren de fout(en) en drukken op RETURN.

Heel iets anders dan het verbeteren of wijzigen van een programmatekst is het maken van een BASIC programma waarmee we in een bepaalde TEKSTstring een bepaalde ZOEKstring kunnen vervangen door een NIEUWEstring.

Stel dat de TEKSTstring T$ de volgende zin bevat:

T$ = "De start is om tien over tien."

En stel dat we in deze regel de tekst "tien over tien" willen vervangen door "twaalf over twaalf". We kunnen nu twee dingen doen. We kunnen de computer als ZOEKstring de string "tien" opgeven en als NIEUWEstring "twaalf", of we kunnen respectievelijk "tien over tien" en "twaalf over twaalf" opgeven. Dit laatste verdient natuurlijk de voorkeur, omdat de tekst "tien" ook nog wel eens als onderdeel van een ander woord zou kunnen voorkomen, zoals in het volgende voorbeeld:

"De tien tieners starten om tien over tien met de tienkamp."

Om te demonstreren wat er (ook in 'echte' tekstverwerkers) komt kijken bij het wijzigen van tekst zullen we toch kiezen voor de eerste mogelijkheid met "tien" als ZOEKstring. Een globaal structuurdiagram voor een programma dat dit moet gaan doen is snel opgesteld:

In het nu volgende programma is deze structuur verwezenlijkt:

```
100 REM  wijzig tekst
110 READ T$ 'lees tekst
120 READ Z$ 'lees zoekstring
130 READ N$ 'lees nieuwestring
140 Z=LEN(Z$) : N=LEN(N$)
150 P=INSTR(T$,Z$) 'zoek de zoekstring
160 '***doe zolang zoekstring gevonden
170 IF P=0 THEN 290
180     GEV=-1 'GEVonden wordt true
190     H$=T$ : T=LEN(T$)
200     L$=LEFT$(T$,P-1)
210     R$=RIGHT$(T$,T-P-Z+1)
220     T$=L$+N$+R$
230     PRINT H$
240     PRINT "***** is gewijzigd in *****"
250     PRINT T$
260     P=INSTR(P+N,T$,Z$) 'zoek de zoekstring
270     GOTO 170
280 '***
290 IF NOT GEV THEN PRINT Z$;" niet gevonden"
300 END
310    DATA"De start is om tien over tien."
320    DATA"tien"
330    DATA"twaalf"
```

De regels 100 t/m 140 doen enig voorbereidend werk. In regel 150 gaan we voor de eerste keer vanaf het begin van T$ zoeken naar de string Z$. Als Z$ niet als deelstring van T$ voorkomt dan zal de INSTR-functie in regel 150 aan P de waarde nul toekennen. Omdat in dit geval P nul is zal de DOE-ZOLANG-lus (regels 180-270) niet doorlopen worden en zal het programma verder gaan met regel 290. In regel 290 wordt als NOT GEV 'waar' is, dus als GEV 'niet waar' is, de zoekstring Z$ afgedrukt met de tekst 'niet gevonden'. GEV is een waarheidsvariabele die aan het begin van het programma (default, dat wil zeggen zonder dat wij dit in het programma hoeven te doen) de waarde 0 gekregen heeft (0 = niet waar, ook wel false genoemd). We maken zo'n waarheidsvariabele 'waar' door er de waarde -1 aan toe te kennen en zeggen dan: de variabele wordt 'true'. Dit gebeurt pas in regel 180 als inderdaad Z$ gevonden is. In dit programmavoorbeeld wordt Z$ inderdaad gevonden. Kijk wat er gebeurt als we in regel 320 "teen" in plaats van "tien" coderen

We gaan nu bespreken wat er in het programma gebeurt.
We hebben dus de volgende situatie:

```
Z$ = "tien" ⇒ Z = LEN("tien") ⇒ Z = 4
N$ = "twaalf" ⇒ N = LEN("twaalf") ⇒ N = 6
GEV = false
T$ = "De start is om tien over tien."
```

Het volgende gaat gebeuren:

regel 150 ⇒ P = 16, want "tien" begint op de 16-de positie van de tekst in T$
regel 170 ⇒ voorwaarde P = 0 is niet waar (false), want 16 is ongelijk aan nul dus
regel 180 ⇒ GEV wordt true
regel 190 ⇒ Hulp$ wordt oude T$ en T wordt 30, het aantal tekens in T$
regel 200 ⇒ L$ wordt "De start is om ", want dit zijn de eerste 15 (P-1 = 15) tekens uit T$
regel 210 ⇒ R$ wordt " over tien.", want dit zijn de laatste 11 (T-P-Z+1 = 11) tekens uit T$

regel 220 ⇒ T$ wordt "De start is om twaalf over tien."
L$ N$ R$

regels 230-250 ⇒ oude T$ (H$) en nieuwe T$ worden afgedrukt
regel 260 ⇒ P = 28, want de tweede "tien" wordt gevonden op de 28-ste positie in de 'nieuwe' tekst in T$
regel 270 ⇒ naar begin van de lus

Omdat weer niet aan de voorwaarde in regel 170 voldaan is (28 is ongelijk aan nul) wordt de lus nogmaals doorlopen. We hebben dan de volgende situatie:

```
Z$ = "tien"; Z = 4
N$ = "twaalf"; N = 6
GEV = true
T$ = "De start is om twaalf over tien."; T = 32
```

Als de lus doorlopen is bevat T$ (regel 220) de tekst: "De start is om twaalf over twaalf.". Nu zal regel 260 aan P de waarde nul toekennen want er komt geen derde "tien" voor in T$. Nu zal aan de voorwaarde P = 0 in de lus-IF-opdracht van regel 170 wel voldaan zijn en wordt het programma beëindigd.

opdracht 8.3

In regel 260 staat de opdracht:

```
P = INSTR(P+N,T$,Z$)
```

Zou het ook goed gegaan zijn als we hier hadden gecodeerd:

```
P = INSTR(1,T$,Z$)
```

We kunnen het bovenstaande programma gemakkelijk veranderen in een subroutine die voor ons de gewenste tekst Z$ in een andere tekst T$ wijzigt in N$ op alle plaatsen waar Z$ in T$ voorkomt. Stel dat we een array A$ hebben met N elementen die elk een tekst van maximaal 60 tekens bevatten (bijvoorbeeld 40 regels van een brief; een stuk tekst van 40 × 60 = 2400 tekens verdeeld in 40 stukjes van elk 60 tekens). Stel dat we een programma willen maken waarmee we in deze tekstarray bepaalde stukken tekst kunnen veranderen op elke plaats waar het betreffende stuk tekst voorkomt. Neem even aan dat deze N tekstregels in een bestand op diskette liggen opgeslagen. We geven nu een structuurdiagram voor een algemeen programma waarmee een dergelijke tekst gewijzigd zou kunnen worden.

begin tekstwijzigen
lees tekst van diskette in tekstarray
vraag of er gewijzigd moet worden
zolang er gewijzigd moet worden
vraag wat er gewijzigd moet worden
vraag waarin dit gewijzigd moet worden
wijzig tekstarray
vraag of er nog meer gewijzigd moet worden
einde tekstarray

De gecodeerde opdrachten van de DOE-ZOLANG-lus zouden er zo uit kunnen zien:

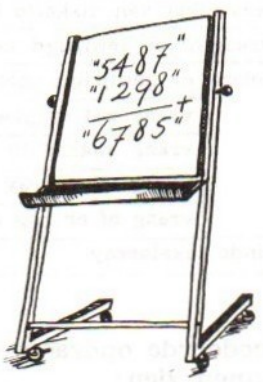
```
INPUT "Wat wilt u wijzigen", Z$
INPUT "Waarin wilt u dit wijzigen", N$
FOR I = 1 TO N
    T$ = A$(I)
    GOSUB 2000, wijzig tekstregel
    A$(I) = T$
NEXT I
INPUT "Wat wilt u nog meer wijzigen", ANTW$
```

Voor elk element uit de tekstarray A$ (een regel uit de brief) wordt gekeken of de string Z$ er in voorkomt en zo ja dan wordt Z$ vervangen door N$. Dit alles wordt uitgevoerd in een FOR-NEXT-lus, waarin subroutine 2000 wordt aangeroepen. Deze subroutine 2000 zou er dan zo uit kunnen zien:

```
2000 'subroutine wijzig tekstregel
2005 '
2010     Z=LEN(Z$) : N=LEN(N$)
2020     P=INSTR(T$,Z$)
2030     '***doe zolang zoekstring gevonden
2040     IF P=0 THEN 2100
2050         T=LEN(T$)
2060         T$=LEFT$(T$,P-1)+N$+RIGHT$(T$,T-P-Z+1)
2070     P=INSTR(P+N,T$,Z$)
2080     GOTO 2040
2090 '***
2095 '
2100 RETURN 'uit wijzig tekstregel
```

8.3 REKENEN MET "TEKST"

Onder bovenstaande kop zullen we demonstreren hoe we getallen als tekst (strings) kunnen inlezen; hoe we er vervolgens echte getallen van kunnen maken en hoe we na wat rekenwerk deze getallen weer kunnen omzetten (converteren) in tekst.



We weten al dat, als we in BASIC twee strings optellen, deze twee strings aaneengeregen worden tot één string:

```
"5487" + "1298" = "54871298"
```

Op het schoolbord zien we echter een (tekst)optelsom, waarin twee getallen, als tekst gerepresenteerd, worden opgeteld als waren het echte getallen:

```
"5487" + "1298" = "6785"
```

Als ons dit lukt, dan kunnen we heel grote getallen bij elkaar optellen. Single precision getallen werken met zeven cijferposities; double precision getallen met 14 posities. Getallen met bijvoorbeeld dertig cijfers kunnen we in BASIC niet op een nauwkeurige manier bij

elkaar optellen. Dat zal wel kunnen als we de getallen niet als getallen maar als strings (tekst) behandelen. Natuurlijk zullen we er de nodige moeite voor moeten doen, want dit soort 'rekenwerk' is de computer niet gewend!

We gaan een programma maken waarin twee getallen bij elkaar worden opgeteld. De getallen worden als string ingelezen. Het resultaat van de optelling wordt als string afgedrukt.

Om het niet al te ingewikkeld te maken gaan we uit van de volgende beperkingen:
- de twee getallen bestaan uit evenveel cijfers (zijn even lang);
- het zijn positieve gehele getallen;
- het aantal cijfers in elk van de twee getallen is maximaal 60 (dan kunnen we de uitvoer op een beeldscherm met 64 tekens per regel netjes afdrukken).

Laten we eerst eens analyseren wat wij zelf eigenlijk doen als we twee getallen bij elkaar optellen:


```
 832
 459 +
```

Wij doen het als volgt:

1. 2 + 9 = 11, dit geeft 1 en 1 onthouden
2. 3 + 5 = 8, plus die éne geeft 9 (niets onthouden)
3. 8 + 4 = 12

De uitkomst is 1291 of korter

```
 832
 459 +
1291
```

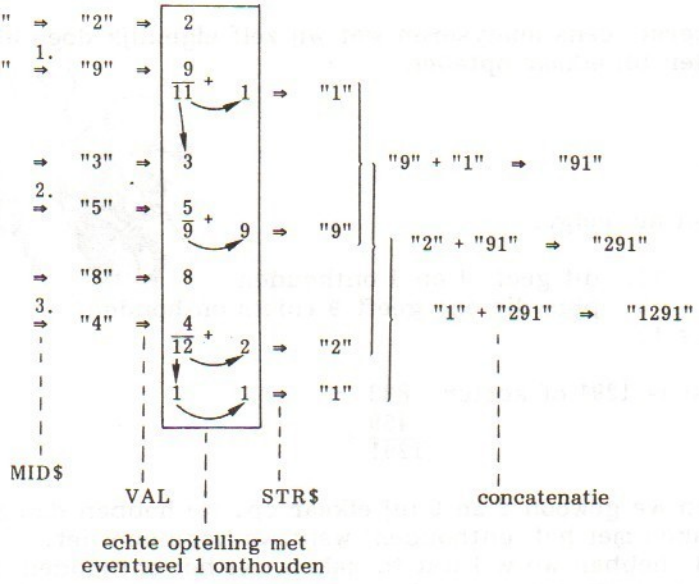
Bij 1. tellen we gewoon 2 en 9 bij elkaar op. We hebben dan nog niets te maken met het 'onthouden' want we beginnen net.
Bij 2. en 3. hebben we wel wat te maken met het onthouden. Bij 2. moeten we de 1 uit stap 1. erbij tellen. Bij 3. hoeven we dit niet want uit stap 2. volgt dat we niets hoeven te onthouden. Bij 3. is er bovendien nog iets anders aan de hand. Dit is de laatste optelling en daarom hoeven we ook niets te onthouden, we schrijven gewoon de uitkomst van deze optelling (8+4 = 12) op en zetten de 12 vóór de 9 en de 1 uit de eerdere optellingen (1. en 2.), zodat de totale uitkomst 1291 wordt.

In computerprogramma's moeten we zoveel mogelijk allerlei uitzonderingen vermijden. Dit maakt programma's langzaam, maar bovenal ondoorzichtig. In dit voorbeeld moeten we drie keer één paar cijfers bij elkaar optellen. Zou het mogelijk zijn hiervoor dezelfde programmaopdrachten te gebruiken, dat wil zeggen de eerste en de laatste

optelling net zo behandelen als de middelste? Als dat mogelijk is, kunnen we voor het herhaald optellen een FOR-NEXT lus gebruiken, want er is immers bekend hoeveel paren cijfers er moeten worden opgeteld. Dit aantal is namelijk de LENgte van de ingelezen getallen (strings). We zullen straks zien dat het inderdaad mogelijk is elk paar cijfers met dezelfde opdrachten bij elkaar op te tellen; wel is hierbij van te voren en achteraf enige actie noodzakelijk.

Voordat we het programma geven laten we eerst in een schemaatje zien hoe we te werk willen gaan. We houden ons bij het hiervoor gegeven voorbeeld:

```
"832"
"459" +
```



We zien dat bij de eerste kolom met pijlen (1., 2., 3.) de afzonderlijke cijferparen geselecteerd worden. Dit kan met behulp van de MID$ functie. De tweede kolom met pijlen geeft de conversie van stringcijfer naar echt cijfer weer. Hiervoor is de VAL functie bedoeld.
In de verticale rechthoek vindt het echte rekenwerk plaats. Soms moeten we één onthouden. Als we één moeten onthouden wordt dat aangegeven met een pijl naar beneden.
De serie pijlen na de rechthoek geeft de conversie aan van het cijferresultaat naar het stringresultaat. Deze conversie is mogelijk met de

STR$ functie

Zo is STR$(1) = " 1" en STR$(9) = " 9".

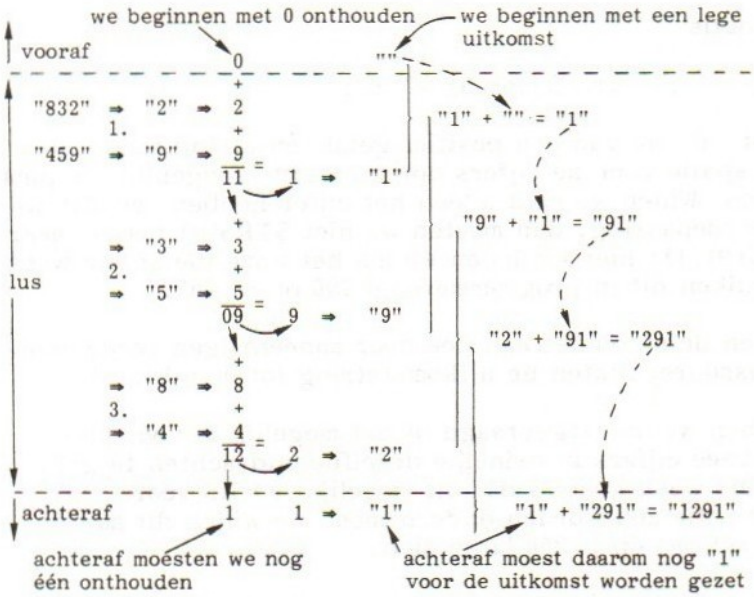
Denk erom dat, als we van een positief getal een string maken de computer een spatie voor de cijfers opneemt; dit is eigenlijk de plaats van het + teken. Willen we echt alleen het cijfer hebben, en dat willen we in deze toepassing, dan moeten we niet STR$(9) nemen maar RIGHT$(STR$(9),1); hiermee halen we als het ware die spatie weer weg. We gebruiken dit in programmaregel 290 op p. 257.

Tenslotte geven de accolades aan hoe door aaneenrijgen (concatenatie) van de tussenresultaten de uitkomststring tot stand komt.

Op p.253 hebben we ons afgevraagd of het mogelijk is voor elke optelling van twee cijfers in principe dezelfde opdrachten te gebruiken. Vervolgens werd gesteld dat dit mogelijk is mits vooraf en achteraf enige actie zou worden ondernomen. We willen dit aan de hand van het schema op p.256 laten zien.

Tussen de stippellijnen zien we nu dat voor elk cijferpaar (1., 2. en 3.) dezelfde opdrachten (lus) worden uitgevoerd, namelijk
- selecteren cijferpaar (MID$)
- conversie naar echte cijfers (VAL)
- optellen van de twee cijfers plus één onthouden (1) of plus geen onthouden
- conversie van de eenheden uit de optelling naar een string (STR$)
- toevoegen van de nieuwe cijferstring aan de uitkomststring (merk op dat in deze stringoptelling eerst de nieuwe cijferstring komt en dan pas de oude uitkomststring).

Dit is mogelijk door een actie vooraf waarin wordt begonnen met 0 onthouden en met een lege uitkomststring ("") en doordat achteraf nog een extra handeling wordt uitgevoerd omdat de derde cijferoptelling (8+4) boven de 9 uitkwam. Bij een optelling waarbij de laatste cijferoptelling beneden de tien blijft is geen actie achteraf noodzakelijk.



Nu geven we het programma dat de optelling verzorgd voor getallen met maximaal 60 cijfers. Het commentaar in het programma is, dachten wij, voldoende om samen met de hierboven gegeven schema's de werking van het programma te begrijpen.

```
100 REM  Optellen van twee getalstrings
105 '
110 INPUT "Druk op RETURN als u wilt beginnen "; A$
115 '***doe zolang alleen return is gegeven
120 IF A$<>"" THEN 390
130      GOSUB 1000 ' Invoer en controle
140 ' Actie vooraf
150      EOH = 0 'EOH betekent eenonthouden!
160      UITKOMST$ = ""
170 ' Begin lus
180      FOR I = L TO 1 STEP -1 'Van rechts naar links
190 '       Pak cijferstrings uit de getallenstrings
200         C1$ = MID$(G1$,I,1)
210         C2$ = MID$(G2$,I,1)
220 '       Conversie van een-teken-string naar cijfer
230         A = VAL(C1$)
240         B = VAL(C2$)
250 '       Optellen cijfers plus 0 of 1 onthouden
260         SOM = A + B + EOH
270         IF SOM >= 10 THEN EOH = 1 :
                              SOM = SOM - 10
                         ELSE EOH = 0
```

```
280 '       Conversie van cijfer naar stringuitkomst
290         DEELUITKOMST$ = RIGHT$(STR$(SOM),1)
300 '       concatenatie met reeds gevormde uitkomst
310         UITKOMST$ = DEELUITKOMST$ + UITKOMST$
320       NEXT I
330 ' Einde lus
340 ' Actie achteraf
350       IF EOH = 1
                THEN UITKOMST$ = "1" + UITKOMST$
                ELSE UITKOMST$ = " " + UITKOMST$
360       GOSUB 2000 ' Sommetje afdrukken
370       PRINT: INPUT "Nog een sommetje, druk dan op RETURN"; A$
380       GOTO 120
385  '***
390 END
1000 ' Subroutine invoer en controle
1010 '
1020      PRINT
1030      INPUT "Eerste getal "; G1$
1040      INPUT "Tweede getal "; G2$
1045      '***doe zolang strings ongelijk of lengte > 60
1050      IF LEN(G1$) = LEN(G2$) AND LEN(G1$) <= 60 THEN 1100
1060           IF LEN(G1$) <> LEN(G2$)
                 THEN PRINT "Getallen niet even lang ";: PRINT
                 ELSE PRINT "Getallen te lang( Max 60)": PRINT
1070           INPUT "Opnieuw, eerste getal "; G1$
1080           INPUT "Opnieuw, tweede getal "; G2$
1090           GOTO 1050
1095      '***
1100      L = LEN(G1$)
1105 '
1110 RETURN ' Invoer en controle
2000 ' Subroutine sommetje afdrukken
2005 '
2010      PRINT
2020      PRINT TAB(2); G1$
2030      PRINT TAB(2); G2$
2040      PRINT STRING$(LEN(G1$)+1,"-"); " + "
2050      PRINT UITKOMST$
2055 '
2060 RETURN ' Sommetje afdrukken
```

We gaan het programma draaien

```
Druk op RETURN als u wilt beginnen ?

Eerste getal ?2345635789654234567584
Tweede getal ?9887523106508733890297

 2345635789654234567584
 9887523106508733890297
---------------------- +
12233158896162968457881

Nog een sommetje, druk dan op RETURN? nee
Ok
```

opdracht 8.4

Verander de subroutine invoer-en-controle zodanig dat de getallen een verschillend aantal cijfers mogen bevatten en dat het kortste getal met spaties wordt opgevuld tot het even lang is als het langste getal. De spaties moeten natuurlijk voorop komen te staan.

We hebben het programma aangepast om sommetjes op de printer af te drukken. We kunnen dan nog grotere getallen optellen. Hier is zo'n som op de printer afgedrukt.

```
9898564767571716879060151641650165716587658746837638768376134665
3900767667611651746865476437873812627907610576087861640561606661
------------------------------------------------------------------ +
13799332435183368625925628079523978344495269322925500408937741326
```

In het tientallig stelsel werken we met de cijfers 0,1,2,3,4,5,6,7,8 en 9. Het hoogste cijfer (9) is één minder dan het grondtal van het talstelsel waarin we werken (10-1 = 9).

Computers werken eigenlijk in het tweetallig stelsel. Getallen in het tweetallig stelsel zijn te schrijven als een optelling van machten van twee in plaats van als een optelling van machten van 10, zoals dat in het tientallig stelsel het geval is. De cijfers in het tweetallig stelsel (binaire stelsel) zijn de cijfers 0 en 1. Het hoogste cijfer (1) is ook hier één minder dan het grondtal van het tweetallig stelsel (2-1=1).

Getallen in het tweetallig stelsel schrijven we dan ook als een combinatie van een aantal nullen en énen; een paar voorbeelden:

11011 011 11011011

Het getal 563 in het tientallig stelsel betekent

$5\times10^2 + 6\times10^1 + 3\times10^0 = 500 + 60 + 3,$

een optelling van machten van 10.

Het getal 11011 in het tweetallig stelsel betekent

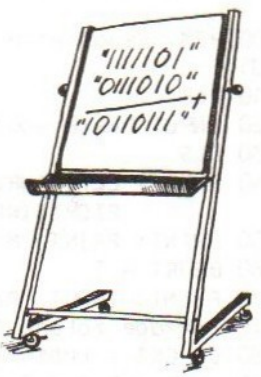
$$1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 16 + 8 + 0 + 2 + 1 = 27,$$

een optelling van machten van 2.

Een getal in het tweetallig stelsel noemen we een **binair getal**. De cijfers 0 en 1 noemen we **bits** (BInary digiTS).
Informatie wordt in een computer in **binaire code** (in bits dus) opgeslagen; niet alleen getallen maar ook tekst!

opdracht 8.5

Verander het optelprogramma zó dat we er binaire strings mee kunnen optellen. Bedenk hierbij dat alleen in regel 270 gebruik wordt gemaakt van het grondtal 10 van het tientallig stelsel. Wat verandert u?



Als u opdracht 8.5 hebt uitgevoerd kunt u het optelprogramma gebruiken om uit te vinden hoe u tweetallig moet optellen! Wellicht kunt u ook oefenen met drie-, vier-, vijf-, zes-, zeven-, acht- of negentallig optellen?

8.4 EEN WOORDSPELLETJE

In deze paragraaf programmeren we een woordspelletje. Het is een spel voor twee personen. De éne persoon tikt een bepaald woord in; de ander moet raden welk woord dit is.

De computer drukt de begin- en eindletter van het te raden woord af met daartussen evenveel puntjes als er letters horen te staan. Degene die moet raden tikt een woord in. De computer vergelijkt de letters uit het ingetoetste woord met de letters uit het geheime woord. De letters die op de goede plaats staan worden door de computer afgedrukt samen met de al eerder geraden letters. Daarna mag de speler weer raden; net zo lang tot het hele woord is geraden. De computer houdt het aantal beurten bij. Een voorbeeld:

```
Speler 1 tikt in  krant
Computer drukt af        : k...t
Speler 2 raadt  kluit
Computer drukt af        : k...t
Speler 2 raadt  kroot
Computer drukt af        : kr..t
Speler 2 raadt  krent
Computer drukt af        : kr.nt
Speler 2 raadt  krant
Computer drukt af        : krant

He..., He... Geraden in 4 Beurt(en)
```

Hier is de programmatekst:

```
100 REM  Raadt mijn woord
105 '
110 CLS
120 INPUT "Welk woord moet uw tegenspeler raden "; WRD$
130 CLS
140 HNT$ = LEFT$(WRD$,1) + STRING$(LEN(WRD$)-2,".") +
          RIGHT$(WRD$,1)
150 PRINT: PRINT "Mijn woord is: "; SPC(10); HNT$
160 BEURT = 1
170 PRINT: INPUT "Raadt maar              "; GK$
175 '***doe zolang GoK niet gelijk is aan WooRD
180 IF GK$ = WRD$ THEN 260
190       FOR I = 2 TO LEN(WRD$)-1
200          IF MID$(HNT$,I,1) = "."
              THEN IF MID$(WRD$,I,1) = MID$(GK$,I,1)
                    THEN MID$(HNT$,I,1) = MID$(GK$,I,1)
210       NEXT I
220       PRINT: PRINT "Mijn woord is: "; SPC(10); HNT$
230       BEURT = BEURT + 1
240       PRINT: INPUT "Raadt nog eens        "; GK$
250       GOTO 180
255 '***
260 PRINT: PRINT "He..., He... Geraden in ";BEURT;" Beurt(en)"
270 END
```

De FOR-NEXT lus bekijkt telkens alle letters behalve de eerste en de laatste. In regel 200 wordt getest of de letter al is geraden of niet. Is de letter nog niet geraden dan staat er nog een . op die plaats in HNT$. Als die punt er nog staat wordt vervolgens getest of de letter uit het geraden woord (GoK) gelijk is aan de letter op die plaats in het geheime WooRD. Is dit zo dan wordt de . op die plaats in HNT vervangen door de goed geraden letter. Dit vervangen gebeurt met een constructie die we nog niet eerder zijn tegengekomen:

```
THEN MID$(HNT$,I,1) = MID$(GK$,I,1)
```

We zien een MID$ functie 'links' van het = teken. Tot nu toe zijn we MID$ alleen rechts van het = teken tegengekomen. Bovenstaande MID$(HNT$,I,1) functie betekent dat het I-de teken uit HNT$ wordt vervangen door het I-de teken uit GK$ (dit staat in de MID$ rechts van het = teken). We kunnen dus MID$ behalve als functie ook als toekenningsopdracht gebruiken!

opdracht 8.6

Voeg tussen de regels 170 en 180 een DOE-ZOLANG-lus toe waardoor de tekst "SUFFERD! MIJN WOORD BESTAAT UIT ... LETTERS" wordt afgedrukt als de speler een woord intoetst dat meer of minder letters bevat dan het te raden woord.

Dat een computer veel met tekst kan doen is hopelijk in dit hoofdstuk uit de voorbeelden duidelijk geworden. Het aardige van het zelf programmeren van allerlei tekstbewerkingsaspecten is dat je wordt gedwongen jezelf te verdiepen in alles wat met tekst (en daarmee met taal) te maken heeft. Dit op zich kan soms tot verrassende ontdekkingen aanleiding zijn.

8.5 BEELDSCHERMTEKST OP DE PRINTER AFDRUKKEN

Een aardig voorbeeld van een programma voor het werken met tekst heeft te maken met het afdrukken van programma's op een printer in dezelfde vorm als waarin de programma's op het scherm zijn opgemaakt.

Stel dat we in de tekststand 'SCREEN 0' werken met een regelbreedte van 37 tekens. (Dit is de standaardinstelling als we de computer aanzetten, aangenomen dat we geen 80-kolommen mogelijkheid hebben!) Op dit 37-tekens-per-regel-scherm hebben we zojuist een programma ingetoetst, waarin een IF-THEN-ELSE-constructie gebruikt is. De ELSE en THEN hebben we op het scherm netjes onder elkaar gezet. Hieronder zien we een 'screen dump' van het programma 'Rekenles eerste klas' uit hoofdstuk vier. Wat hieronder staat hebben we precies zo op de computer (in SCREEN 0, met WIDTH 37) ingetoetst. THEN en ELSE staan netjes onder elkaar.

```
100 REM Rekenles eerste klas
105
110 N=0
120 FOR SOMMETJE=1 TO 10
130     A%=INT(9*RND(1)+1)
140     B%=INT(9*RND(1)+1)
150     PRINT A%; "+"; B%; "=";:
        INPUT ANTW%
160     IF ANTW%=A% + B%
          THEN PRINT "Goed!" : N=N+1
          ELSE PRINT "FOUT!!"
170     PRINT "Druk op RETURN voor de"
              ;" volgende som";
180     INPUT A$
190 NEXT SOMMETJE
200 PRINT
210 PRINT "De score is"; N; "van de";
          " tien goed"
220 IF N <= 5
      THEN PRINT "Dit is onvoldoende"
      ELSE PRINT "Dit is voldoende"
230 END
Ok
```

We willen dit programma graag precies zo op de printer afdrukken. Laten we het eens proberen met **LLIST**. Tot onze grote schrik wordt het programma op de printer als volgt afgedrukt:

```
100 REM Rekenles eerste klas
105
110 N=0
120 FOR SOMMETJE=1 TO 10
130    A%=INT(9*RND(1)+1)
140    B%=INT(9*RND(1)+1)
150    PRINT A%; "+"; B%; "=";:              INPUT ANTW%
160    IF ANTW%=A% + B%                       THEN PRINT "Goed!" : N=N+1
   ELSE PRINT "FOUT!!"
170    PRINT "Druk op RETURN voor de"              ;" volgende som";
180    INPUT A$
190 NEXT SOMMETJE
200 PRINT
210 PRINT "De score is"; N; "van de";          " tien goed"
220 IF N <= 5                                 THEN PRINT "Dit is onvoldoende"
ELSE PRINT "Dit is voldoende"
230 END
```

De printer trekt zich niets aan van de meerregelige opdrachten (150, 160, 170, 210 en 220) en drukt zoveel mogelijk (zover hij kan) op één printerregel af. Dit komt omdat **LLIST** pas 'gedwongen' op een nieuwe printerregel begint als óf de maximale breedte van een printerregel bereikt is, óf als er in de programmatekst een **'harde RETURN'** staat. Zo'n harde RETURN in de programmatekst ontstaat als wij bij het afsluiten van een programmaregel die we intoetsen op **RETURN** drukken. Aangezien wij, bijvoorbeeld, in regel 150 pas na INPUT ANTW% op **RETURN** gedrukt hebben en niet al na "=";: ziet **LLIST** dit als één programmaregel (terecht overigens) en drukt hem

ook op één printerregel af. Voor een dergelijk programma is **LLIST** dus niet de geschikte manier om de programmatekst, precies zoals deze op het scherm staat, op de printer af te drukken.

Wel een goede manier is om de printer in te stellen op dezelfde regelbreedte als het scherm en daarna **LLIST** te gebruiken. Alle matrixprinters hebben een instelbare rechterkantlijn (right margin), die we vanuit BASIC, via een zogenaamde **ESCAPE code**, kunnen instellen. Bij de matrixprinter die wij gebruikt hebben is dit:

```
LPRINT CHR$(27); "Q"; CHR$(37) ,
```

voor een regelbreedte van 37 tekens. Veel letterwielprinters zijn echter niet op deze manier instelbaar op een bepaalde rechterkantlijn. Een eventuele rechterkantlijn moet dan in de gebruikte software gedefinieerd worden (bijvoorbeeld een tekstverwerkingspakket). Jammer genoeg kent MSX BASIC de **WIDTH LPRINT** opdracht niet, waarmee we in BASIC een rechter printerkantlijn kunnen definiëren. Aan het begin van dit boek hebben we nogal wat programma's op een 37-tekens scherm gepresenteerd. Deze programma's zijn precies zo op een letterwielprinter afgedrukt. De twee methoden, die wij hiervoor gebruikt hebben, zullen we nu bespreken.

De eerste methode maakt gebruik van het programma COPY, dat we aan het einde van hoofdstuk 4 bij het sprite-ontwerp-programma gebruikt hebben. Dit programma kopieert een deel van het videogeheugen, waarin de beeldschermtekst die op dat moment op het scherm zichtbaar is, naar de printer. Het programma COPY ziet er als volgt uit:

```
100 REM van scherm naar printer
110 INPUT "hoeveel regels"; N
120 '***
130 IF N>=1 AND N<=24 THEN 170
140     INPUT "N tussen 1 en 24"; N
150     GOTO 130
160 '***
170 FOR I=0 TO N-1
180    FOR J=0 TO 39
190       X=VPEEK(I*40+J)
200       X$=X$+CHR$(X)
210    NEXT J
220    LPRINT X$
230 X$=""
240 NEXT I
250 END
```

Het is eigenlijk een heel eenvoudig programma, dat voor **SCREEN 0** (37 of 40 tekens per regel) de video-adressen 0 tot en met maximaal 959 (23 × 40 + 39) bekijkt (**PEEK**) en de codes die daar staan (X)

per regel (I = 0 TO 39) aan elkaar plakt tot een string (X$), die precies de tekst van een beeldschermregel bevat. Deze beeldschermtekstregel wordt vervolgens op de printer (LPRINT) afgedrukt. Aangezien we minimaal 1 en maximaal 24 beeldschermregels kunnen afdrukken moeten we zorgen dat N ook geen andere waarden kan krijgen.

Als we dit "COPY"-programma willen draaien dan zullen we daarvoor de opdracht run "COPY" (of RUN "COPY") moeten intoetsen. Als het programma draait, drukt het op het scherm (vlak onder run "COPY") de vraag af: "hoeveel regels?". Hierop moeten wij antwoorden met een getal dat aangeeft hoeveel beeldschermregels we op de printer willen afdrukken. Dit betekent dat we voor run "COPY" en 'hoeveel regels?' twee beeldschermregels nodig hebben. Deze twee beeldschermregels zullen we echter niet op de printer willen laten afdrukken. In de praktijk zullen we dus tot maximaal regel 21 afdrukken. De laatste 3 regels reserveren we dan voor het draaien van "COPY". We zullen nu een voorbeeld geven van het werken met COPY. We nemen als programma om af te drukken het programma 'vierkanten of cirkels' uit hoofdstuk 4. Dit programma is op een scherm van 37 tekens opgemaakt (THEN en ELSE onder elkaar) en het is langer dan 21 regels zodat we het in gedeelten zullen moeten afdrukken. Denk erom dat we LLIST niet kunnen gebruiken, vanwege de meerregelige opdrachten met regelnummers 105, 125 en 150. We nemen even aan dat dit programma onder de naam "vierofck" samen met "COPY" op schijf staat.

We lezen het programma van schijf (load "vierofck") en drukken het tot en met regel 1000 (list-1000) op het beeldscherm af. Met de RETURN-toets zorgen we dat regel 100 de bovenste beeldschermregel wordt. Vervolgens brengen we de cursor naar het begin van de regel vlak onder regel 1000 en toetsen in: run "COPY" en geven RETURN. Nu zal het programma "vierofck" uit het geheugen verwijderd worden en zal "COPY" erin gezet worden. COPY wordt gedraaid en drukt de tekst "hoeveel regels?" op het scherm af. Aangezien de regels 100 t/m 1000 uit vierofck samen 20 beeldschermregels beslaan toetsen we 20 in en drukken op RETURN. Deze situatie zien we op de screendump op de bladzijde hiernaast.

We willen nu graag de rest van het programma vierofck op de printer afdrukken. De printer heeft zojuist de eerste 20 regels (regels 100 t/m 1000) op het papier afgedrukt. Vlak hieronder moeten de regels 1005 t/m 2110 komen. Dit moet echter in twee stukken omdat het 27 beeldschermregels betreft. Omdat het programma vierofck niet meer in het geheugen staat moeten we het eerst weer van schijf lezen. We toetsen dus in load "vierofck" (+ return) en besluiten om eerst alle regels uit subroutine 1000 af te drukken en dan de hele subroutine 2000. We toetsen daarom in List 1005-1105 en zorgen met de return-toets dat regel 1005 de bovenste beeldschermregel wordt.

```
100 REM vierkanten of cikels?
105 'Dit programma tekent een scherm
    vol vierkanten of cirkels. Maak
    uw keuze
110 DEF FNX(X)=INT(37+X/1.4+.5)
120 INPUT "Klaar(ja/nee)"; A$
125 '***doe zolang dfe gebruiker er
      niet genoeg van heeft
130 IF LEFT$(A$,1) <> "j" THEN 180
140   INPUT "v=vierkant,o=cirkel";K$
150   IF K$="v"
        THEN GOSUB 1000
        ELSE GOSUB 2000
160   INPUT "Nog een keer(j/n)"; A$
170 GOTO 130
175 '***
180 COLOR 15,4,4 : SCREEN 0
190 KEY ON : END
195 '
1000 'Subroutine vierkanten
run"copy"
hoeveel regels? 20
color  auto   goto   list   run
```

Met de pijltoetsen zetten we de cursor ergens tussen regel 1105 en de onderste regel in (we hebben vrije regels genoeg) en toetsen in **run** "copy" (+ return). Op de vraag hoeveel regels? antwoorden we met 13, drukken op return, en de eerste dertien beeldschermregels (de programmaregels 1005 t/m 1105) worden op de printer, vlak onder de eerder afgedrukte regels 100 t/m 1000, afgedrukt.
De schermsituatie zien we op de volgende screendump.

```
1005 '
1010   KEY OFF:COLOR 1,15,15:SCREEN 2
1020   FOR VIERKANT= 1 TO 60
1030   X=FNX(212*RND(1))
1040   Y=INT(132*RND(1))
1050   D=INT((75*RND(1)+10)/SQR(2))
1060   K=INT(15*RND(1))
1070   LINE(X,Y)-(X+D/1.4,Y+D),K,BF
1080   NEXT VIERKANT
1090   A$=INPUT$(1)
1095 '
1100 RETURN 'vierkanten
1105 '

run"copy"
hoeveel regels? 13
color  auto   goto   list   run
```

Nu rest nog het afdrukken van de subroutine 2000. Omdat subroutine 2000 géén regelnummers bevat, die meer dan één beeldschermregel beslaan, kunnen we gewoon volstaan met **load** "vierofck" en **llist** 2000- Als,oefening kunt u het echter ook met copy proberen. In feite hadden we na de eerste keer "copy" **load** "vierofck" en **llist** 1005- kunnen opgeven, want ook subroutine 1000 bevat geen 'moeilijke' regelnummers. Dat we dit niet gedaan hebben komt omdat we juist de werking van "copy" wilden demonstreren.
Voor lange programma's zou het steeds in gedeelten kopiëren van de programmatekst van het scherm naar de printer veel te tijdrovend zijn. Dergelijke programma's kunnen we beter op de tweede manier, die we nu gaan bespreken, afdrukken. Deze manier is trouwens ook voor korte programma's te gebruiken.

De tweede manier om een programma net zo op de printer af te drukken als de computer het op het beeldscherm zou doen, is de programmatekst in een tekstbestand op diskette of op cassetteband op te slaan en dit 'programmabestand' door een ander programma op de printer te laten afdrukken, waarbij we net doen of de printer even 'breed' is als het beeldscherm.

Het is wellicht verstandig om eerst hoofdstuk 9 te bestuderen voordat u verder leest; u vindt daar de uitleg van hetgeen we nu kort gaan bespreken. Natuurlijk hadden we dit voorbeeld voor hoofdstuk 9 kunnen bewaren, maar het past juist zo goed in een hoofdstuk over 'tekstverwerking', vandaar dat we het toch nu al doen.

We kunnen een programma als een tekstbestand (ASCII-bestand) op cassetteband of op diskette kopiëren. We doen dit als volgt:

```
cassetteband: SAVE "programmanaam"  of
              SAVE "cas: programmanaam"
```

Gebruik niet **CSAVE** want dan wordt het programma dat in het geheugen staat in een gecomprimeerde binaire vorm op de cassetteband gekopieerd!

```
diskette: SAVE "programmanaam", A  of
          SAVE "A: programmanaam", A
```

Laten we bij Disc BASIC de , A (van ASCII) weg, dan wordt het programma als gecomprimeerd binair gecodeerd bestand op diskette gekopieerd.

Willen we een programma dus straks vanaf cassetteband of vanaf diskette, door middel van een ander programma, op de printer afdrukken, dan moeten we het als tekstbestand op het betreffende achtergrondgeheugen kopiëren. Als we dat doen, staat de programmatekst in de vorm van een reeks ASCII-codes op cassetteband of op diskette. Op de hieronder staande screendump zien we hoe we een

5-regelig programma na het intikken als ASCII-bestand, onder de naam 'tekstvb' op diskette gekopieerd wordt. In het voorbeeld wordt gebruik gemaakt van Disk BASIC en de opdracht save "tekst vb",a zorgt voor het kopiëren. Gebruikt u alleen een cassetterecorder, dan moet de opdracht gewoon save "tekstvb" luiden.

```
10 REM voorbeeld tekstbestand
20 FOR I=1 TO 10
30     PRINT I;
40 NEXT I
50 END
save"tekstvb",a
Ok
run"asciidump"
Naam van het tekstbestand? tekstvb
 49  48  32  82  69  77  32 118 111 111
114  98 101 101 108 100  32 116 101 107
115 116  98 101 115 116  97 110 100  13
 10  50  48  32  70  79  82  32  73  61
 49  32  84  79  32  49  48  13  10  51
 48  32  32  32  32  80  82  73  78  84
 32  73  59  13  10  52  48  32  78  69
 88  84  32  73  13  10  53  48  32  69
 78  68  13  10
Ok
```

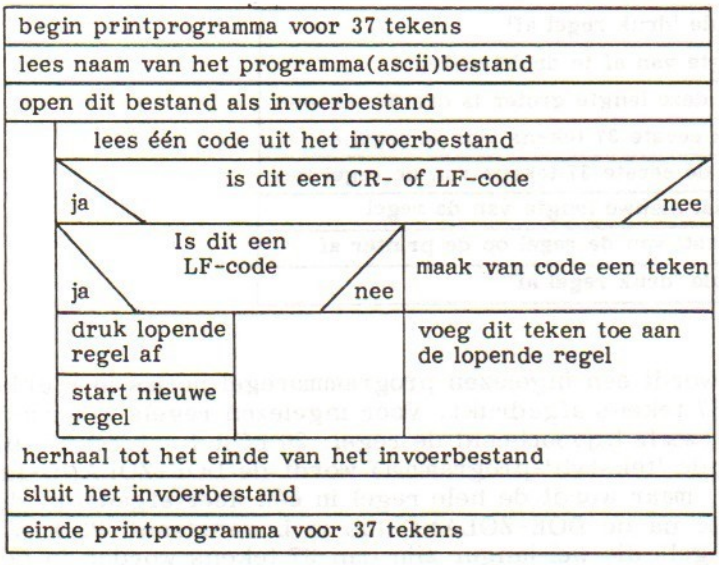
color auto goto list run

Op de schermafdruk zien we dat we direct daarna het programma 'asciidump' gedraaid hebben (run "asciidump"). Dit programma vraagt klaarblijkelijk om de naam van een (ascii) bestand en drukt daarna een hele reeks getallen op het scherm af. Wat dit programma asciidump doet is het afdrukken van de ASCII-codes van de tekens die in het gegeven bestand liggen opgeslagen. Als u de ASCII-tabel uit appendix D erbij neemt en de codes terugvertaalt naar de daarmee overeenkomende tekens dan moet u de 5-regelige programmatekst terugkrijgen. Het bijzondere is wellicht dat in de reeks codes vijf keer de combinatie 13 10 voorkomt. Als u in de ASCII-tabel van appendix D kijkt, ziet u dat ASCII-code 13 het besturingsteken Carriage Return (CR) is, dat wil zeggen cursor naar het begin van de lopende regel, en dat ASCII-code 10 het besturingsteken Line Feed (LF) is, dat wil zeggen cursor naar volgende regel. Dit is namelijk precies wat er gebeurt als we bij het intikken van de programmaregels 10 t/m 50 aan het einde van elke regel op de RETURN-toets drukken. Deze 13 en 10 zullen straks een belangrijke rol gaan spelen. Voordat we laten zien hoe het programma eruit ziet dat zo'n 'programmatekstbestand' op de printer afdrukt, geven we eerst het asciidump-programma dat op de hierboven staande schermafdruk gebruikt is en dat de inhoud van een tekstbestand als een reeks ASCII-codes op het beeldscherm afdrukt.

```
100 REM dit programma drukt de inhoud van
110 '   een tekstbestand als een reeks
120 '   asciicodes af
130 INPUT "Naam van het tekstbestand"; N$
140 WIDTH 40
150 OPEN N$ FOR INPUT AS #1
160 '===
170     A$=INPUT$(1,#1)
180     PRINT USING "### "; ASC(A$);
190 IF NOT EOF(1) THEN 170
200 '===herhaal tot einde tekstbestand
210 END
```

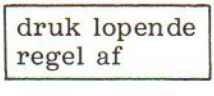
Voor de OPEN-opdracht en de EOF-functie verwijzen we naar hoofdstuk 9. Wat het programma doet is het vragen naar de naam van het tekstbestand (regel 130), het verbreden van het scherm tot 40 tekens (regel 140), het OPENen van het tekstbestand als invoerbestand (regel 150) en het herhaald lezen van één teken (regel 170) uit het tekstbestand en het afdrukken van de ASCII-code daarvan (regel 180) totdat het einde van het bestand (End-Of-File) bereikt is (regel 190).

Nu gaan we ons concentreren op het programma dat een programmatekstbestand op de printer moet afdrukken, net zoals de computer het op het beeldscherm zou doen. We gaan even uit van een bestand met een programma dat op een 37-tekens breed scherm mooi is opgemaakt. We stellen de printerbreedte niet in op 37 tekens, maar proberen zelf (in ons programma) een printer van 37 tekens per regel te 'simuleren'. Uit het bovenstaande voorbeeld zien we dat elk regelnummer, met de daarachter voorkomende BASIC-opdrachten, uit een programma, dat als een ASCII-bestand is opgeslagen, bestaat uit een reeks ASCII-codes, die afgesloten worden met de codes 13 en 10. We gaan een programma maken dat de programmatekst uit zo'n programmabestand code voor code inleest, deze codes terugvertaalt naar tekens, deze tekens netjes tot regelnummers met bijbehorende BASIC-opdrachten groepeert en de programmaregels netjes met hoogstens 37 tekens per printregel afdrukt. Het volgende PSD toont een mogelijke structuur voor een dergelijk programma.



We lezen dus code voor code in uit het invoerbestand. Zolang deze code geen CR- of LF-code is (13 of 10) zetten we de code om in een teken en 'plakken' het teken aan eerder ingelezen tekens vast. Zo vormt zich teken voor teken een programmaregel, die als een reeks ASCII-codes in het invoerbestand ligt opgeslagen. Als we een CR-code en een LF-code inlezen dan is dit het teken dat we een hele programmaregel hebben ingelezen en dat we de opgebouwde regel kunnen afdrukken. Daarna beginnen we met het inlezen en formuleren van de volgende programmaregel uit het invoerbestand. We herhalen deze werkwijze tot het einde van het invoerbestand. We weten dat zo'n ascii-programmabestand altijd met de CR- en LF-code eindigt (zie bovenstaand voorbeeld).

We gaan ons nu bezighouden met de vraag hoe we na het lezen van CR en LF (codes 13 en 10) de dan opgebouwde programmaregel, die heel weinig maar ook heel veel (maximaal 256 tekens) tekens kan bevatten, met 37 tekens per regel kunnen afdrukken. Het blokje



wordt dus een module in ons programma.

Een PSD voor deze module 'druk regel af' zou er zo uit kunnen zien:

begin module 'druk regel af'
bepaal lengte van af te drukken regel
doe zolang deze lengte groter is dan 37 tekens
druk eerste 37 tekens van de regel af
haal de eerste 37 tekens van de regel weg
bepaal nieuwe lengte van de regel
druk de 'rest' van de regel op de printer af
einde module 'druk regel af'

In deze opzet wordt een ingelezen programmaregel netjes in stukken van maximaal 37 tekens afgedrukt. Voor ingelezen regels met minder dan 37 tekens (zoals bijvoorbeeld de regel 20 FOR I = 1 TO 10 uit het bovenstaande 'tekstvb'-programma) wordt de DOE-ZOLANG-lus niet doorlopen, maar wordt de hele regel in één keer afgedrukt door de opdracht die na de DOE-ZOLANG-lus in bovenstaand PSD is opgenomen. Regels die wel langer zijn dan 37 tekens worden in de lus in stukken van 37 tekens afgedrukt totdat er minder dan 37 tekens in de regel over zijn. Deze worden dan ten slotte door de opdracht ná de lus afgedrukt. Het onderstaande printprogramma is aan de hand van de twee bovenstaande PSD's gecodeerd.

```
100 REM ascii-programma op de printer
        afdrukken met 37 tekens per
        regel
110 CLS : CLEAR 400
120 INPUT "Bestandsnaam"; B$
130 OPEN B$ FOR INPUT AS #1
140 '===
150     X=ASC(INPUT$(1,#1))
160     IF X>=32
          THEN R$=R$+CHR$(X)
          ELSE IF X=10
                 THEN GOSUB 1000:
                      R$=""
170 IF NOT EOF(1) THEN 150
180 '===herhaal tot einde bestand
190 CLOSE1
200 END
1000 'subroutine druk regel af
1010 '
1020   L=LEN(R$)
1030   '***doe zolang regel meer dan
           37 tekens bevat
```

```
1040    IF L<=37 THEN 1100
1050       LPRINT LEFT$(R$,37)
1060       R$=MID$(R$,38)
1070       L=LEN(R$)
1080       GOTO 1040
1090   '***
1100    LPRINT R$
1110 '
1120 RETURN 'uit druk regel af
```

Tik dit programma (**SCREEN 0**, 37 tekens breed) precies zo in en sla het op als ascii-programmabestand op diskette (**save** "naam",a) of op cassetteband (**save** "naam"). Test het programma door het te draaien en door het zichzelf op de printer te laten afdrukken. De printerafdruk moet precies overeenkomen met de schermafdruk, dat wil zeggen met 37 tekens per regel. Aan regel 160 is mooi te zien of het programma goed werkt. Als u dit programma draait toetst u dan op de vraag 'Bestandsnaam?' de naam in waaronder u dit programma hebt opgeslagen. Hebt u ooit een programma gezien dat zichzelf op de printer kan afdrukken?! De **CLEAR**-opdracht in regel 110 zorgt dat er genoeg stringruimte voor de lopende regel R$ gereserveerd wordt.

opdracht 8.7

We kunnen dit programma zo maken dat het maximale aantal tekens per regel, waarmee een programmabestand op de printer moet worden afgedrukt, met een **INPUT**-opdracht ingelezen wordt. Pas het programma daarvoor aan.

OEFENOPGAVEN HOOFDSTUK 8

1. We weten dat LEFT$("MICROCOMPUTER",5) = "MICRO" en dat
   RIGHT$("MICROCOMPUTER",8) = "COMPUTER".
   Beide ontledingen zijn ook mogelijk met de MID$ functie. Geef deze MID$ functie met de juiste argumenten.

2. Schrijf een programma dat bepaalt hoeveel letters "A" in een ingetoetste zin (alleen hoofdletters) voorkomen. Voer de zin in met behulp van de LINE INPUT opdracht. Geef ook het percentage dat de letter A uitmaakt van alle letters in de ingetoetste zin. Zet uw MSX-computer in de hoofdletterstand.

3. Alle namen, adressen en woonplaatsen van 250 clubleden bevinden zich in evenzoveel DATA regels in een programma dat, als het wordt gedraaid, adresetiketjes op een regeldrukker afdrukt. Om het papier efficient te gebruiken worden steeds vijf adresetiketjes naast elkaar afgedrukt. Neem aan dat elk nieuw etiket begint op een standaard printpositie van de regeldrukker. Een DATA regel ziet er bijvoorbeeld zo uit:
1000 DATA "W. V.D. WILK", "LOOLAAN 25", 2526 HH VOORBURG"
Op de regeldrukker moet dit als volgt worden afgedrukt:

W. V.D. WILK
LOOLAAN 25
2526 HH VOORBURG

Schrijf een programma dat zo steeds vijf etiketjes naast elkaar afdrukt. (Hint: gebruik drie arrays met vijf elementen voor het inlezen van steeds 5 namen, adressen en woonplaatsen.)

4. Schrijf een programma dat een zin inleest en dat deze zin afdrukt zonder spaties.

5. Schrijf een programma dat bepaalt of een ingetoetst woord een palindroom is. Dit is een woord dat van voren naar achteren hetzelfde gespeld wordt als van achteren naar voren. Voorbeelden van een palindroom zijn 'lepel', 'soos', 'mam', 'nemen', 'kajak' en een heel mooie is 'parterretrap'.

6. Bekijk het volgende programma (nou ja...programma!):

```
10 A$ = "De kat die krabt de krullen van de trap"
20 PRINT INSTR(A$,"rap")
30 END
RUN
37 <------ het woord "rap" begint op de 37e positie van A$.
```

De INSTR functie geeft de positie in de opgegeven string waar de daarachter opgegeven zoekstring begint.
Stel dat we een type BASIC gebruiken waarin de INSTR functie niet voorkomt. Schrijf nu een programma dat met behulp van de MID$ functie nagaat of het woord "RAP" in A$ voorkomt en zo ja op welke positie. Het programma moet stoppen als het woord is gevonden. Natuurlijk ook als er niets is gevonden (dan moet een nul worden afgedrukt).

7. Schrijf een programma dat een gebroken getal, bijvoorbeeld 17,65 als string inleest en afdrukt wat het gehele deel is en wat het decimale deel is. De uitvoer moet er zo uitzien:

Uw gebroken getal is? 17,65
Het gehele deel is 17
Het decimale deel is 65

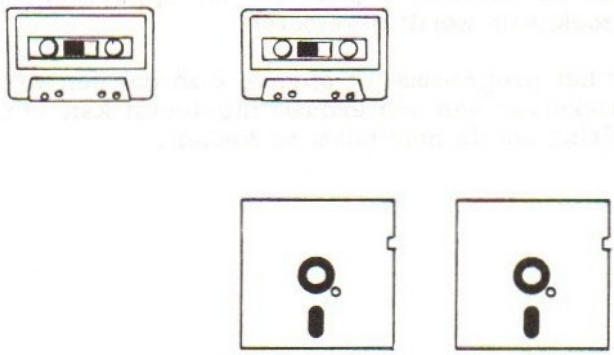
Uw gebroken getal is? 186594,8675
Het gehele deel is 186594
Het decimale deel is 8675

enzovoort.
(Het programma stopt als **stop** wordt ingetoetst.)

8. In een DATA-lijst van een programma bevinden zich 200 achternamen en telefoonnummers, beide als strings. Als het programma wordt gedraaid vraagt het programma om de eerste vier letters van een achternaam in te toetsen. Vervolgens drukt het programma alle namen met telefoonnummers af voor die namen die beginnen met de vier ingetoetste letters. Er kunnen net zo lang namen en telefoonnummers worden opgevraagd totdat stop als zoeknaam wordt ingetoetst.

9. Verander het programma uit opgave 8 zó dat een willekeurig aantal letters van een achternaam ingetoetst kan worden als aanwijzing om de hele naam te zoeken.

9 Programma's en Gegevens op Cassetteband en Diskette



In dit hoofdstuk bespreken we het opslaan en inlezen van programma's op, respectievelijk van, cassetteband en diskette. Op een (MSX-) computer kunnen we veel randapparaten aansluiten. Een randapparaat waar we niet buiten kunnen is bijvoorbeeld de monitor of het toetsenbord. Een printer is ook een heel nuttig en bijna onmisbaar randapparaat. Als we een programma of gegevens willen opslaan om op een later tijdstip weer te kunnen gebruiken, hebben we randapparaten, zoals een cassetterecorder of schijfbesturingsstation (disk drive), nodig. Met deze randapparatuur kunnen we programma's en gegevens op cassetteband en diskette, ook wel externe geheugens of achtergrondgeheugens genoemd, opslaan.

9.1 RANDAPPARATUUR EN DE OPEN OPDRACHT

Een aantal BASIC-opdrachten voor het afdrukken en inlezen van gegevens is vast verbonden met een bepaald randapparaat. Hier volgen enkele voorbeelden:

PRINT	uitvoer naar de monitor (beeldscherm)
LPRINT	uitvoer naar de printer
INPUT	invoer van het toetsenbord
LIST	programmatekst als uitvoer naar de monitor
LLIST	programmatekst als uitvoer naar de printer
STICK	invoer van de joystick

De monitor en printer zijn typische uitvoerapparaten, terwijl het toetsenbord en de joystick typische invoerapparaten zijn. Een cassetterecorder en een diskdrive zijn voorbeelden van apparaten die zowel invoer- als uitvoerapparaat zijn. Voordat we de BASIC-opdrachten, die voor deze in- en uitvoer nodig zijn, gaan behandelen, eerst iets over de 'namen' die we in MSX BASIC aan de diverse randapparaten kunnen geven en de OPEN-opdracht waarin we deze namen kunnen gebruiken. In MSX BASIC kunnen we de volgende 'logische' apparaatnamen gebruiken:

CRT of crt	monitor (tekststand, SCREEN 0 en 1)
GRP of grp	monitor (grafische stand, vanaf SCREEN 2)
LPT of lpt	printer
CAS of cas	cassetterecorder
A of a	disk drive
B of b	tweede disk drive

Als we gegevens op cassetteband of op diskette willen opslaan (of gegevens hiervan willen inlezen) dan moeten we in de computer een buffer 'aanvragen' om de uitvoer (en invoer) mogelijk te maken. Zo'n uitvoer- of invoerbuffer vragen we aan met de OPEN-opdracht. Gegevens worden in de vorm van bestanden op cassetteband en op diskette bewaard. Zo'n gegevensbestand heeft een naam, die wij zelf kunnen verzinnen. Ook programma's worden als bestand onder een bepaalde naam op deze achtergrondgeheugens opgeslagen. De geheugenbuffer, die we met een OPEN-opdracht voor een bepaald bestand aanvragen, dient als verzamelplaats voor gegevens die worden opgeslagen of ingelezen.

In hoofdstuk 6 (p. 201) hebben we al een voorbeeld van de OPEN opdracht gezien, waarin een buffer naar het grafische scherm (GRP) werd geopend. Via die buffer konden we tekst en gegevens op het grafische scherm afdrukken. We geven nu een aantal voorbeelden van de OPEN-opdracht.

```
OPEN "grp:" AS #1        uitvoerbuffer naar grafisch scherm
OPEN "lpt:" AS #1        uitvoerbuffer naar printer
OPEN "cas:data86" FOR OUTPUT AS #1
        uitvoerbuffer voor bestand 'data86' naar cassetterecorder
```

```
OPEN "cas:data86" FOR INPUT AS #1
```
invoerbuffer voor bestand 'data86' van cassetterecorder

```
OPEN "A:inkomstn" FOR INPUT AS #2
```
invoerbuffer voor bestand 'inkomstn' van diskette

OPEN	opent een in- of uitvoerbuffer op een bepaald rand-apparaat.

In een OPEN-opdracht zetten we de bestandsidentificatie van het bestand tussen "-tekens. We mogen deze identificatie met kleine letters of in hoofdletters schrijven. Zo'n identificatie bestaat uit twee delen, namelijk een apparaatnaam en een (zelf gekozen) bestandsnaam, gescheiden door een dubbele punt:

bestandsidentificatie	=	apparaatnaam	:	bestandsnaam

In sommige gevallen mogen we de apparaatnaam en de dubbele punt weglaten en ook hoeven we in bepaalde gevallen geen bestandsnaam op te geven, zoals in OPEN "grp:" AS # 1. In dit hoofdstuk zullen we ontdekken welke gevallen dit zijn.

Het 'hekje' (#) geeft het buffernummer aan. We kunnen namelijk meer dan één bestand tegelijkertijd gebruiken (OPENen). We geven de buffers voor deze bestanden verschillende nummers. Met de BASIC-functie MAXFILES kunnen we het aantal bestanden (buffers) dat we tegelijkertijd willen gebruiken, specificeren. Standaard (default) staat MAXFILES op 2.

We concentreren ons in dit hoofdstuk op de randapparaten CAS (cassetterecorder), A (hoofd-drive) en B (tweede drive). We kunnen in plaats van LPRINT ook een buffer gebruiken om gegevens op de printer af te drukken, al kost dat heel wat meer moeite, Vergelijk de twee methoden:

normaal:

```
LPRINT A$
```

via buffer:

```
OPEN "lpt:" AS #1
PRINT # 1, A$
CLOSE 1
```

Wij zullen deze 'omweg' niet gebruiken. Zoals we in hoofdstuk 6 hebben laten zien moeten we voor het afdrukken van gegevens op het grafische scherm wel een dergelijke omweg gebruiken, en wel omdat de MSX BASIC ontwerpers 'vergeten' zijn om de opdracht GPRINT (van Graphic PRINT) in MSX BASIC op te nemen!

9.2 PROGRAMMA'S OPSLAAN EN INLEZEN

U zult allang weten dat u met SAVE en LOAD programma's kunt opslaan en inlezen (LOADen). We zullen daarom de mogelijkheden kort bespreken. Sommige MSX-gebruikers zullen alleen met een cassetterecorder als achtergrondgeheugen werken, anderen met alleen een disk drive (of twee) of met beide soorten randapparaten. We geven nu een aantal voorbeelden voor het opslaan en inlezen van programma's onder diverse omstandigheden.

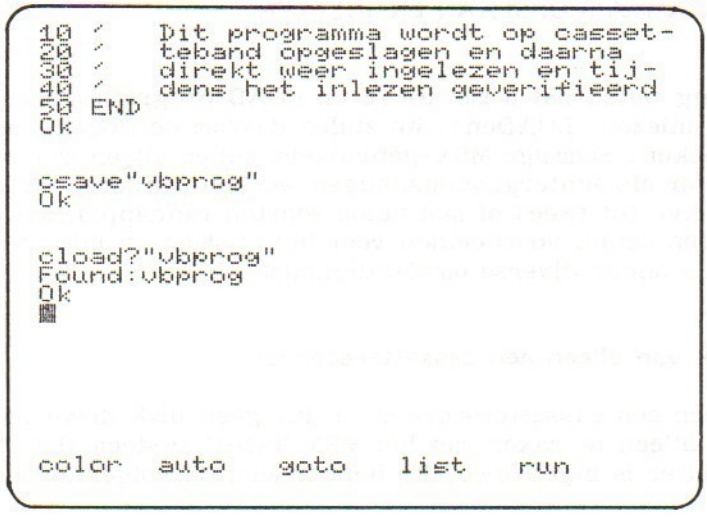
Bij gebruik van alleen een cassetterecorder

Als we alleen een cassetterecorder en dus geen disk drive gebruiken hebben we alleen te maken met het MSX BASIC systeem dat standaard in de computer is ingebouwd. We hebben dan de volgende mogelijkheden:

SAVE "naam" LOAD "naam"	Programma wordt als ascii-bestand (tekstbestand) opgeslagen en ingelezen.
CSAVE "naam" CLOAD "naam"	Programma wordt als binair bestand (gecomprimeerde vorm) opgeslagen en ingelezen.
CLOAD? "naam"	Vergelijkt het programma op de cassetteband met het programma in het computergeheugen. Bij geconstateerde verschillen wordt 'Verify error' afgedrukt. Zorg dat de cassetteband is teruggespoeld en op 'PLAY' staat.

Op de volgende 'screen-dump' zien we hoe we het programma 'vbprog' op cassetteband hebben opgenomen en hoe we de kopie vervolgens gecontroleerd (geverifieerd) hebben. De Ok onder Found geeft aan dat de kopie identiek is met het programma in het geheugen. (Screen-dump staat bovenaan bladzijde 278.)

Als u een op cassetteband opgenomen programma (of deel van een programma) later wilt kunnen toevoegen aan een programma dat al in het geheugen staat, dan moet u SAVE gebruiken (tekstprogrammabestand) en niet CSAVE (zie verderop bij MERGE).
Laat u bij SAVE (of CSAVE) de "naam" helemaal weg dan wordt het programma zonder naam op band gekopieerd. Laat u de "naam" bij LOAD (of CLOAD) weg dan wordt het eerste programma, dat op de band tegengekomen wordt, ingelezen.



Bij gebruik van alleen een disk drive

SAVE "naam" LOAD "naam"	Programma wordt als binair bestand opgeslagen en ingelezen
SAVE "naam",A LOAD "naam"	Programma wordt als ascii-bestand opgeslagen en ingelezen.

Als u alleen een disk drive gebruikt dan zal het standaard BASIC-systeen, bij het gelijktijdig aanzetten van uw computer en drive overgenomen worden door het Disk-BASIC-systeem (zie verderop).

Bij gebruik van zowel een cassetterecorder als een disk drive

SAVE "cas:naam" LOAD "cas:naam"	Programma wordt als ascii-bestand op casette-band opgeslagen, respectievelijk ingelezen.
CSAVE "naam" CLOAD "naam"	Programma wordt als binair bestand op cas-setteband opgeslagen en ingelezen
SAVE "naam" of SAVE "A:naam" LOAD "naam" of LOAD "A:naam"	Programma wordt als binair bestand op de hoofd-drive opge-slagen, respectievelijk ingelezen.

Gebruik, als u met een cassetterecorder en een disk drive tegelijkertijd werkt, het liefst altijd de apparaatnaam (cas, A of B) om misverstand over het randapparaat, waarnaar het programma gekopieerd moet worden of waarvan het ingelezen moet worden, te voorkomen.

opdracht 9.1

Geef een opdracht om het programma 'proef' op cassetteband op te slaan als u

a) alleen met een cassetterecorder als randapparaat werkt;
b) met zowel een cassetterecorder als een disk drive werkt.

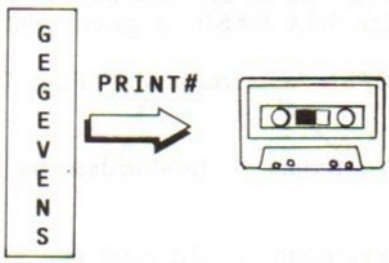
Geef een opdracht om het programma 'sorteer', dat als tekstbestand is opgeslagen in te lezen als dit bestand

c) op cassetteband is opgeslagen en geen disk drive is aangesloten;
d) op diskette is opgeslagen en zowel een disk drive als een cassetterecorder is aangesloten.

We kunnen met de MERGE-opdracht programma's (subroutines) die op cassetteband of op diskette staan 'mengen' met het programma dat op dat moment in het geheugen staat. Deze MERGE-opdracht wordt in paragraaf 9.4 besproken.

9.3 GEGEVENS VAN EN NAAR CASSETTEBAND

We gaan nu bekijken hoe we door een programma gegevens op cassetteband kunnen laten opnemen. De gegevens zullen als één lange sliert, achter elkaar op band worden opgenomen. We spreken dan van een **sequentieel gegevensbestand**. We zullen een programma gaan maken waarmee we gegevens kunnen intoetsen die in een array worden opgeslagen. Als we een bepaald 'sluitgegeven' intoetsen wordt de inhoud van de array op band opgeslagen. Een dergelijk programma heet een 'KEY-TO-TAPE' programma. Een PSD voor zo'n programma ziet er zo uit:



begin KEY-TO-TAPE
lees gegevens in een array
schrijf inhoud van het array op band
einde KEY-TO-TAPE

Een iets verder uitgewerkt PSD zou er zo uit kunnen zien:

```
begin KEY-TO-TAPE
vraag hoeveel gegevens maximaal ingevoerd worden
dimensioneer een array van de juiste lengte
vraag om bestandsnaam
vraag om eerste gegeven en zet dit in het array
doe zolang geen sluitgegeven is ingetoetst en het array niet vol is
    verhoog array-index met één
    vraag om volgende gegeven en zet dit in het array
open gegevensbestand op de cassetterecorder
voor begin tot voor zover array gevuld doe
    schrijf array-element naar gegevensbestand
sluit het bestand
einde KEY-TO-TAPE
```

Als we de inhoud van een array op band willen schrijven dan moeten we op de cassetterecorder een uitvoerbestand openen.
We zullen bij het openen van dit uitvoerbestand een naam voor het bestand moeten opgeven. Laten we het bestand data86-1 noemen.
De volgende OPEN-opdracht kan zowel in 'gewoon' MSX BASIC als in Disk MSX BASIC gegeven worden.

```
OPEN "cas:data86-1" FOR OUTPUT AS #1
```

apparaatnaam — bestandsnaam — soort bestand — buffernummer

apparaatnaam : dit moet één van de logische namen **crt**, **lpt**, **grp**, **cas**, **a** of **b** zijn.

bestandsnaam : mogen we zelf kiezen. We spreken af geen namen van meer dan 8 tekens te kiezen.

soort bestand : dit moet een uitvoerbestand (**OUTPUT**), invoerbestand (**INPUT**) of toevoegbestand (**APPEND**, zie 9.5) zijn.

buffernummer : Dit moet een geheel getal uit de reeks 0,1,2,3,... t/m 15 zijn (zie MAXFILES in 9.6).

Om het hiervoor in PSD-vorm beschreven KEY-TO-TAPE programma te kunnen coderen, moeten we alleen nog weten hoe we een gegeven op cassetteband kunnen kopiëren.

PRINT #1, A	kopieert de waarde van de variabele A in de buffer met nummer 1

De PRINT#-opdracht gebruiken we om gegevens in een buffer te kopiëren. In een OPEN-opdracht verbinden we een bepaalde buffer aan een bepaald randapparaat. Via het nummer van de buffer dat we in de PRINT#-opdracht opgeven weet de computer naar welk randapparaat het gegeven (of de gegevens) in de PRINT#-opdracht gestuurd moet worden.

We mogen in één PRINT#-opdracht meerdere waarden, ook tekstwaarden, tegelijkertijd in een buffer kopiëren. In het volgende KEY-TO-TAPE programma zien we zo'n PRINT#-opdracht waarmee echter steeds één waarde regelijk wordt gekopieerd.

```
100 REM key-to-tape programma
110 '   Dit programma leest een aantal gegevens in,
120 '   slaat ze in een array op en kopieert de
130 '   inhoud van het array als een bestand op tape
140 '
150 INPUT "Hoeveel waarden voert u maximaal in";N
160 DIM X(N)
170 PRINT
180 INPUT "Hoe wilt u het bestand noemen";B$
190 PRINT : I=1
200 INPUT "   1 Voer de eerste waarde in";X(1)
210 '***doe zolang X niet 999999 is en het array niet vol is
220 IF W=999999! OR I=N   THEN 280
230     I=I+1
240     PRINT USING "#### ";I;
250     INPUT "De volgende(999999=stop)";X(I)
260     GOTO 220
270 '***
280 REM kopieer array als bestand op tape
290 OPEN "cas:"+B$ FOR OUTPUT AS #1
300 FOR J=1 TO I
310    PRINT#1,X(J)
320 NEXT J
330 CLOSE1
340 PRINT "bestand '";B$;"' is gekopieerd"
350 END
```

Als we in dit programma in de OPEN-opdracht als naam "cas:data86-1" gezet zouden hebben, dan zullen we voor een volgend bestand deze naam in de OPEN-opdracht moeten veranderen. Het is daarom beter om de naam, waaronder het bestand gekopieerd moet worden, door de gebruiker van het programma te laten intoetsen. Zo wordt de naam waaronder de gegevens gekopieerd worden een variabele in het programma. Deze variabele ziet er dan zo uit:

```
"cas:" + B$
```

opdracht 9.2

Wijzig het programma zo dat de gebruiker kan kiezen uit cassette of diskette, zodat het programma ook als KEY-TO-DISK-programma gebruikt zou kunnen worden.

Door het programma om een 'sluitwaarde' te laten vragen hoeven we niet zoveel gegevens in te toetsen als we aan het begin (als waarde voor N) hebben opgegeven. We kunnen N 'ruim' opgeven, zodat er in ieder geval genoeg arrayruimte wordt gereserveerd. Het is aan te bevelen de gegevens eerst in een array op te nemen en deze array in één keer op de cassetteband te kopiëren. Hierdoor hebben we voor het kopiëren nog de mogelijkheid subroutines aan te roepen waarmee de gegevens bijvoorbeeld afgedrukt, gecontroleerd en eventueel gewijzigd kunnen worden, of voor het sorteren van de gegevens. Ook hebben we dan geen last met bandsynchronisatieproblemen omdat we elk gegeven niet direct na het intoetsen op de band kopiëren. Op de volgende screen-dump zien we hoe we het bestand data86-1 met 12 gegevens gevuld en op cassetteband gekopieerd hebben.

```
run
Hoeveel waarden voert u maximaal in? 12

Hoe wilt u het bestand noemen? data86-1

   1 Voer de eerste waarde in? 32
   2 De volgende(999999=stop)? 45
   3 De volgende(999999=stop)? 14
   4 De volgende(999999=stop)? 28
   5 De volgende(999999=stop)? 64
   6 De volgende(999999=stop)? 35
   7 De volgende(999999=stop)? 8
   8 De volgende(999999=stop)? 82
   9 De volgende(999999=stop)? 23
  10 De volgende(999999=stop)? 41
  11 De volgende(999999=stop)? 11
  12 De volgende(999999=stop)? 74
bestand 'data86-1' is gekopieerd
Ok
■


color    auto     goto     list     run
```

Omdat we precies zoveel gegevens intikken als we aan het begin hebben opgegeven (N = 12) hoeven we niet het sluitgegeven 999999 in te toetsen.

We mogen in een PRINT#-opdracht meer dan één variabele gebruiken en zelfs een combinatie van numerieke en tekstvariabelen. Deze variabelen worden door een ; van elkaar gescheiden. Zo zouden we een programma kunnen maken met de volgende opdrachten:

```
DIM NAAM$(100),NUMMER(100)
:
INPUT "geef naam"; NAAM$(I)
INPUT "geef nummer"; NUMMER(I)
:
OPEN "cas: telklap" FOR OUTPUT AS #1
FOR I I = 1 TO 100
   PRINT#1,NUMMER(I); NAAM$(I)
NEXT I
CLOSE1
:
```

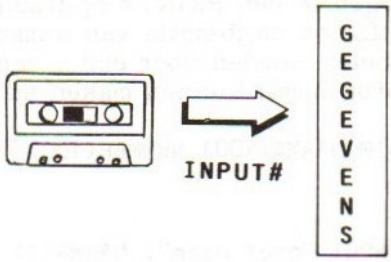
Wel moeten we voor programma's die een bestand als invoerbestand gaan gebruiken natuurlijk van dat bestand precies weten hoe het is opgebouwd. Dat wil zeggen wat voor gegevens, numeriek of alfanumeriek, erop staan en in welke volgorde.

Laten we een programma maken om te kunnen controleren of de 12 gegevens, die we zojuist als 'data86-1'-bestand hebben opgeslagen, ook goed op de band zijn gekopieerd. We lezen de gegevens uit het bestand in en drukken ze op het scherm af. Een PSD voor een dergelijk TAPE-TO-SCREEN-programma kan er zo uitzien:

begin TAPE-TO-SCREEN programma
vraag om naam bestand
open dit bestand als invoerbestand
lees gegevens uit het bestand
druk gegevens op het scherm af
herhaal tot einde van het bestand
sluit het invoerbestand
einde TAPE-TO-SCREEN programma

De vraag die meteen opkomt is: 'hoe weten we waar het einde van een bestand is?'. Welnu, dat weten wij niet, maar de computer wel. De computer sluit elk bestand af met een End-Of-File teken. Wordt

zo'n End-Of-File teken door de computer ingelezen, dan weet hij dat er geen gegevens meer behoeven te worden ingelezen. Op dat moment zet de computer de EOF-functie voor het betreffende bestand op 'waar'. Dit wil zeggen: de waarde van de EOF-functie voor dat bestand wordt -1 gemaakt. Zolang het einde van een bestand nog niet geconstateerd is, heeft EOF voor dat bestand de waarde 0 (niet waar of false). Het zal niet verwonderlijk zijn dat, als we met PRINT# gegevens naar een randapparaat kunnen kopiëren, we met INPUT# gegevens uit een randapparaat kunnen inlezen.



INPUT #1, A	kent het eerstvolgende (in dit geval numerieke) gegeven uit buffer 1 toe als waarde van de variabele A

Het is nu niet moeilijk meer het TAPE-TO-SCREEN programma te coderen. We doen het direct vanachter de machine en draaien het programma meteen. Op de volgende screen-dump zien we het gebeuren. Eerst spoelen we echter de band terug en drukken de 'PLAY'-toets van de recorder in. Vergeet dit niet want de computer maakt u er niet op attent!

```
100 REM tape-to screen progamma
110 INPUT "Bestandsnaam";B$
120 OPEN"cas:"+B$ FOR INPUT AS #1
130 '===
140     INPUT#1,W
150     PRINT USING "####";W;
160 IF NOT EOF(1) THEN 140
170 '===herhaal tot einde bestand
180 CLOSE1
190 END
Ok

run
Bestandsnaam? data86-1
  32  45  14  28  64  35   8  82  23  41
  11  74
Ok
■


color    auto     goto     list     run
```

We zien dat de 12 gegevens inderdaad goed op het bandje zijn gekopieerd.

EOF( )	is een 'logische' functie die direct na het inlezen van het laatste gegeven uit de tussen haakjes opgegeven buffer de waarde 'waar' (-1) krijgt

opdracht 9.3

Zou een doe-zolang structuur niet beter zijn geweest voor dit programma?

De CLOSE-opdracht sluit een bestand af en geeft de geheugenbuffer, die in een OPEN-opdracht gereserveerd is, vrij. In plaats van CLOSE1 mogen we ook CLOSE#1 schrijven. De opdracht CLOSE, zonder iets erachter, sluit alle op dat moment 'OPENstaande' bestanden.

Om de diverse gegevens in een bestand op cassetteband van elkaar te scheiden gebruikt MSX BASIC komma's. Als u alfanumerieke gegevens gaat opslaan (strings), zorg er dan voor dat deze geen komma's bevatten. Zorg er bovendien voor dat, als u in een PRINT#-opdracht tekstvariabelen opgeeft, deze expliciet door een scheidingskomma van elkaar gescheiden worden. Een voorbeeld:

```
PRINT#1,NAAM$; ","; ADRES$; ","; WNPL$
```

Op de band komen de komma's dan netjes op de goede plaats terecht, namelijk als afscheiding van de diverse tekstwaarden.

Denk erom dat, als u met een INPUT#-opdracht meer dan één gegeven tegelijkertijd inleest, u de variabelen niet scheidt met puntkomma's maar met komma's. De INPUT#-opdracht bij de bovenstaande PRINT#-opdracht zou zo moeten luiden:

```
INPUT#1, NAAM$, ADRES$, WNPL$
```

Als oefening kunt u nu een programma proberen te schrijven dat een gegevensbestand van cassetteband in een array inleest, vervolgens het array verder vult met nieuwe gegevens en ten slotte de aangevulde array als nieuw bestand (onder dezelfde naam, vergeet niet terug te spoelen en de PLAY- en REC-toetsen in te drukken!) weer op de cassetteband kopieert. U hebt dan een programma gemaakt voor het aanvullen van een sequentieel gegevensbestand op cassetteband.

9.4 DISK MSX BASIC

Als op een MSX-computer een disk drive is aangesloten en als deze drive aanstaat op het moment dat de computer aangezet wordt of de 'reset'-knop van de computer wordt ingedrukt, zal de interface, die tussen drive en computer is ingebouwd, het Disk BASIC systeem in werking stellen. Hierdoor krijgen we niet alleen de BASIC-opdrachten voor het werken met bestanden op diskette ter beschikking, maar ook een aantal opdrachten voor huishoudelijk werk zoals het formateren van nieuwe diskettes, het kopiëren van diskettes en van bepaalde bestanden, het verwijderen van bestanden en het veranderen van de naam van een bestand of van een groep bestanden. Voordat we hier verder op ingaan zullen we eerst de bestandsidentificatie voor diskettebestanden wat nader bekijken.

In 9.2 hebben we gezegd dat een bestandsidentificatie er zo uitziet:

apparaatnaam	:	bestandsnaam

De meeste MSX-computergebruikers zullen slechts met één drive werken. In principe kunnen deze gebruikers voor de identificatie van een diskettebestand (dit kan een programmabestand of een gegevensbestand zijn) volstaan met een bestandsnaam van maximaal 8 tekens. Overal waar de apparaatnaam wordt weggelaten kiest Disk BASIC de drive als randapparaat. Wordt er naast de drive ook nog met een cassetterecorder gewerkt dan moet voor bestanden, die daarop betrekking hebben wel de apparaatnaam vermeld worden. Laten we even terugkeren naar het gebruik van slechts één drive. Op de in deze drive te gebruiken diskettes zal een aantal verschillende bestanden staan. Er zullen BASIC-programma's op staan in de vorm van binaire en ASCII-bestanden; er zullen (zie 9.5) gegevensbestanden op staan in de vorm van sequentiële en directtoegankelijke bestanden; er zullen tekstbestanden (met een tekstverwerkingspakket gemaakt) op staan en wellicht nog andere soorten bestanden. MSX BASIC biedt de mogelijkheid een bestandsidentificatie uit te breiden met een **type-aanduiding** van maximaal 3 tekens. Deze type-aanduiding wordt achter de bestandsnaam gezet voorafgegaan door een punt:

apparaatnaam	:	bestandsnaam	.	type-aanduiding

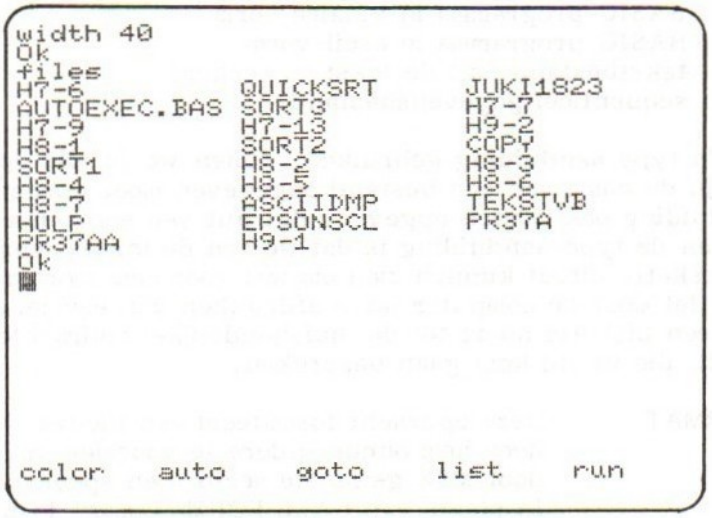
Wij, als gebruiker, kunnen elk bestand een bepaalde type-aanduiding meegeven. Zo zouden we de volgende type-aanduidingen kunnen gebruiken:

BAS	BASIC-programma in binaire vorm
ASC	BASIC-programma in ascii-vorm
TXT	tekstbestand (uit de tekstverwerker)
SEQ	sequentieel gegevensbestand.

Als we zo'n type-aanduiding gebruiken, zullen we in alle opdrachten, waarin de naam van een bestand opgegeven moet worden, deze type-aanduiding ook moeten opgeven. Het nut van een consequent gebruik van de type-aanduiding is dat we aan de inhoudsopgave van een diskette direct kunnen zien om wat voor een soort bestand het gaat. Het door de computer laten afdrukken van een inhoudsopgave van een diskette hoort tot de 'huishoudelijke' opdrachten van Disk BASIC die we nu kort gaan bespreken.

CALL FORMAT	Deze opdracht formateert een nieuwe diskette door hem onder andere te voorzien van de door MSX gewenste sector- en spoorindeling. In plaats van CALL FORMAT mag _FORMAT gegeven worden.
FILES	drukt de inhoudsopgave, dat wil zeggen de bestandsnamen met eventuele type-aanduiding, van een diskette op het scherm af.
LFILES	drukt de inhoudsopgave van een diskette op de printer af.
COPY	kopieert een hele diskette of een groep bestanden op dezelfde diskette of op een diskette in een tweede drive.
NAME	geeft een bestand of een groep bestanden een andere naam.
KILL	verwijdert een bestand of een groep bestanden uit de inhoudsopgave van een diskette.
RUN "naam"	draait een programma direct van diskette.
MERGE	voegt programmabestanden samen.

Behalve de besturingsopdrachten RUN en MERGE kunnen alle andere opdrachten ((L)FILES, COPY, NAME, KILL) betrekking hebben op meer dan één bestand. We kunnen namelijk achter deze opdrachten een groep bestanden specificeren. Hoe we met één bestandsidentificatie een hele groep bestanden kunnen specificeren zullen we nu zien. Op de screen-dump op bladzijde 288 zien we hoe de opdracht files (mag ook in hoofdletters) de inhoudsopgave (directory) van een diskette, die op dat moment in de (hoofd)drive zit, op het scherm afdrukt. Het betreft een diskette met programma's uit dit boek. In de gewone tekststand (SCREEN 0) krijgen we twee kolommen met namen; geven we echter eerst WIDTH 40 dan krijgen we drie kolommen. Met een 80-kolommenscherm krijgen we wellicht nog meer kolommen met bestandsnamen.



De programma's op bovenstaand overzicht zijn allemaal BASIC programma's. Eén heeft er al de type-aanduiding BAS. Laten we ze daarom voor alle duidelijkheid allemaal de type-aanduiding BAS (of bas) geven. We kunnen met de COPY-opdracht een bestand onder een andere naam op dezelfde diskette kopiëren. Met de volgende COPY-opdracht kopiëren we het programmabestand H7-6 onder de naam H7-6.BAS op dezelfde diskette:

```
COPY "H7-6" TO "H7-6.BAS"
```

Zo zouden we nog 24 COPY-opdrachten kunnen geven om alle andere programmabestanden te kopiëren onder de oude naam met de type-aanduiding BAS. We kunnen echter met één COPY-opdracht alle bestanden op de diskette kopiëren, waarbij aan elke naam de type-aanduiding BAS wordt toegevoegd:

```
COPY "*" TO "*.BAS"
```

Voor TO staat de bestandidentificatie van alle te kopiëren bestanden. Aan de **bestandsidentificatie "*"** voldoen alle bestanden, die een naam hebben van maximaal 8 tekens zonder type-aanduiding. Op onze voorbeelddiskette voldoen dus op één na alle namen aan deze bestandsidentificatie. Achter TO staat de nieuwe **bestandsidentificatie "*.BAS"**. De COPY-opdracht zegt dat aan elke naam die voldoet aan "*" de type-aanduiding BAS moet worden toegevoegd. Op de volgende screendump zien we deze COPY-opdracht. De FILES-opdracht drukt nu twee keer zoveel bestanden af; de oorspronkelijke 26 en de 25 kopieën met de gewijzigde naam. (AUTOEXEC had al .BAS!)

```
copy "*" to "*.bas"
Ok
files
H7-6           QUICKSRT       JUKI1823
AUTOEXEC.BAS   H7-6      .BAS  QUICKSRT.BAS
SORT3          H7-7           H7-9
H7-13          H9-2           H8-1
SORT2          COPY           SORT1
H8-2           H8-3           H8-4
H8-5           H8-6           H8-7
ASCIIDMP       TEKSTVB        HULP
EPSONSCL       PR37A          PR37AA
JUKI1823.BAS   SORT3    .BAS  H7-7     .BAS
H7-9     .BAS  H7-13    .BAS  H9-2     .BAS
H8-1     .BAS  SORT2    .BAS  COPY     .BAS
SORT1    .BAS  H8-2     .BAS  H8-3     .BAS
H8-4     .BAS  H8-5     .BAS  H8-6     .BAS
H8-7     .BAS  ASCIIDMP.BAS   TEKSTVB .BAS
HULP     .BAS  EPSONSCL.BAS   PR37A    .BAS
PR37AA   .BAS  H9-1     .BAS  H9-1
Ok

color    auto    goto    list    run
```

We gaan nu in de naam van de bestanden SORT1.BAS, SORT2.BAS en SORT3.BAS de type-aanduiding BAS vervangen door ASC omdat 'wij' weten dat deze programma's als tekst (ASCII) programmabestand geSAVEd zijn. We doen dit met de NAME-opdracht:

```
NAME "SORT?.BAS" AS "SORT?.ASC"
```

Elke naam die voldoet aan de bestandsidentificatie "SORT?.BAS" wordt veranderd in een naam die voldoet aan de bestandsidentificatie "SORT?.ASC". Welke namen voldoen nu aan "SORT?.BAS"? Welnu, elke naam die uit 5 tekens bestaat, waarvan de eerste vier SORT zijn, en waarvan de type-aanduiding BAS is. Op de navolgende screen-dump zien we het effect van de naamsverandering.

```
name "sort?.bas" as "sort?.asc"
Ok
files
H7-6           QUICKSRT       JUKI1823
AUTOEXEC.BAS   H7-6      .BAS  QUICKSRT.BAS
SORT3          H7-7           H7-9
H7-13          H9-2           H8-1
SORT2          COPY           SORT1
H8-2           H8-3           H8-4
H8-5           H8-6           H8-7
ASCIIDMP       TEKSTVB        HULP
EPSONSCL       PR37A          PR37AA
JUKI1823.BAS   SORT3    .ASC  H7-7     .BAS
H7-9     .BAS  H7-13    .BAS  H9-2     .BAS
H8-1     .BAS  SORT2    .ASC  COPY     .BAS
SORT1    .ASC  H8-2     .BAS  H8-3     .BAS
H8-4     .BAS  H8-5     .BAS  H8-6     .BAS
H8-7     .BAS  ASCIIDMP.BAS   TEKSTVB .BAS
HULP     .BAS  EPSONSCL.BAS   PR37A    .BAS
PR37AA   .BAS  H9-1     .BAS  H9-1
Ok

color    auto    goto    list    run
```

Ziet u trouwens hoe de bestanden na COPY door het besturingssysteem een andere plaats hebben gekregen? U ziet ook dat u de opdrachten COPY, NAME, FILES, enzovoorts, gewoon in kleine letters kunt intoetsen.

De * en ? die we in bestandsidentificaties kunnen gebruiken heten jokers (wildcards). Met deze jokers kunnen we met één bestandsidentificatie een hele groep bestanden aanduiden. Enkele voorbeelden:

FILES "*.BAS"	druk alle bestandsnamen af die de type-aanduiding BAS hebben.
LFILES "H*.*"	druk alle bestandsnamen af die beginnen met een H en die wel of geen type-aanduiding hebben.
COPY "H8??" TO "H8??.BAS"	kopieer elk bestand, waarvan de naam uit vier tekens bestaat en de eerste twee tekens H8 zijn, op dezelfde diskette met als nieuwe naam de oude naam plus de toevoeging .BAS.
KILL "*.ASC"	verwijder alle bestanden van het type ASC van de diskette. Maak eerst met COPY een kopie van de diskette voordat u KILL gebruikt!!

opdracht 9.4

Geef een BASIC-opdracht voor:

a) het afdrukken van de namen van bestanden die beginnen met SORT en uit precies 8 tekens bestaan;
b) het kopiëren van alle bestanden van het type ASC in bestanden zonder type-aanduiding;
c) het verwijderen van alle bestanden die geen type-aanduiding hebben;
d) het afdrukken op de printer van alle bestanden in de directory.

Werkt u met twee drives, een A-drive (de hoofddrive) en een B-drive dan kunt u met A: of B: aan het begin van een bestandsidentificatie aangeven of het bestanden op de diskette in de A-drive respectievelijk in de B-drive betreft. Hieronder volgen enkele voorbeelden:

FILES "B: *.*"	drukt alle bestandsnamen van bestanden op de diskette in drive B af.

COPY "*" TO "B: *.BAS" kopieert alle bestanden zonder type-aanduiding van de A-drive naar de B-drive. De bestanden op de B-drive worden van het type BAS.

COPY "A:" TO "B:" kopieert hele diskette in drive A naar diskette in drive B. Deze opdracht werkt ook bij één drive. U moet dan wel een aantal malen de diskette met het origineel en die waarop de kopie komt in de A-drive verwisselen. Het besturingssysteem (MSX-DOS) drukt op het scherm af wat u moet doen.

opdracht 9.5

Geef een BASIC-opdracht voor:

a) het veranderen van de naam van alle bestanden op de diskette in drive B, die nog geen type-aanduiding hebben; het type moet BAS worden;
b) het verwijderen van alle bestanden van de diskette in drive B, waarvan de naam uit vier tekens bestaat, met H als eerste teken, en waarvan het type BAS is;
c) het kopiëren van alle ASC-bestanden van drive B naar drive A.

We kunnen een MSX-computer bij het aanzetten automatisch een bepaald programma laten uitvoeren, als we dit programma onder de naam AUTOEXEC.BAS op diskette opslaan en deze diskette voor het aanzetten van de drive en de computer in de drive aanbrengen. Het hieronder gegeven programma drukt de inhoudsopgave van de diskette, waarop het AUTOEXEC.BAS programma staat, af en geeft aan hoeveel vrije ruimte er nog op de diskette aanwezig is. Als u dit programma met

```
SAVE "AUTOEXEC.BAS"
```

op een diskette kopieert, dan zal, mits de diskette voor het aanzetten van het systeem in de drive zit en u eerst de drive en dan de computer aanzet, het programma automatisch gedraaid worden.

```
100 REM zelfstartend programma
110 CLS
120 WIDTH 40
130 PRINT "Inhoudsopgave diskette"
140 PRINT STRING$(39,"-")
150 FILES : PRINT
160 PRINT STRING$(39,"-")
170 PRINT "Vrije ruimte: ";DSKF(0);"Kb"
180 PRINT STRING$(39,"-")
190 END
```

De functie DSKF(0) geeft de vrije ruimte (in kilobytes) op de diskette. Voor de al eerder gebruikte voorbeelddiskette geeft dit het volgende schermbeeld:

Tot slot van deze paragraaf nog iets over het inlezen, draaien en samenvoegen van programma's; kortom iets over **LOAD**, **RUN** en **MERGE**.

Als we in een programma, dat op diskette staat, eerst nog iets willen veranderen, alvorens het te draaien, dan moeten we het programma eerst inlezen. Als voorbeeld nemen we het indexcijferprogramma uit het begin van hoofdstuk 6 (zie pagina 173). Stel dat dit programma onder de naam INDEXCFS op diskette staat. We willen dit programma draaien met nieuwe gegevens in dataregel 240. Op de volgende screen-dump (bovenaan de volgende bladzijde) zien we hoe we het programma met **LOAD** van diskette inlezen, hoe we de nieuwe **DATA**-regel 240 intoetsen en hoe we met **RUN** het programma laten draaien.

opdracht 9.6

Kunnen we het programma met de oorspronkelijke dataregel nog draaien?

Denk erom dat, als u type-aanduidingen voor al uw programma- en gegevensbestanden gebruikt, u ook in de **LOAD** (en straks in de **RUN** en **MERGE**) opdrachten deze type-aanduiding moet vermelden; bijvoorbeeld **LOAD "INDEXCFS.BAS"**.

```
load"indexofs"
Ok
240      data 15,24,37,48,52,56,61,66,
              77,83,89,95,99999

run
 15           100
 24           160
 37           247
 48           320
 52           347
 56           373
 61           407
 66           440
 77           513
 83           553
 89           593
 95           633
Ok
_


color  auto   goto   list   run
```

Vaak willen we niets meer aan een programma veranderen, maar het gewoon draaien. We hoeven dan niet eerst LOAD "naam" en vervolgens RUN te geven maar we kunnen met één RUN-opdracht een programma inlezen (laden) en direct laten uitvoeren. Dit doen we door achter RUN de betreffende programmanaam op te geven. Een voorbeeld, dat we al vaker gegeven hebben, is het direct draaien van het COPY-programma voor het kopiëren van beeldschermtekst op de printer.
De opdracht hiervoor is:

RUN "COPY" of RUN "B:COPY" als COPY in de B-drive zit.

Als u vaak een bepaald programma moet draaien, zou het handig zijn de betreffende RUN-opdracht met de specifieke programmanaam 'onder' één van de functietoetsen te programmeren. Laten we eens zien hoe we de opdracht RUN "COPY" onder functietoets F1 kunnen programmeren. We proberen daartoe eerst het volgende:

KEY 1, "RUN "COPY"" + CHR$(13) ◁

Dit zal niet het gewenste effect hebben, want de eerste aanhalingstekens van "COPY" worden gezien als een afsluiting van de, onder de functietoets, te programmeren string. We moeten de aanhalingstekens van "COPY" dan ook anders programmeren:

KEY 1, "RUN" + CHR$(34) + "COPY" + CHR$(34) + CHR$(13) ◁

De ASCII-code voor aanhalingstekens is 34. CHR$(34) is in de functietoetsstring een echt aanhalingsteken en wordt niet gezien als het begin of het einde van de functietoetsstring. De CHR$(13) is

het besturingsteken dat overeenkomt met het aanslaan van de RETURN-toets, hetgeen bij het indrukken van de functietoets F1 tot gevolg heeft dat het programma COPY wordt uitgevoerd.

Tot slot van deze paragraaf laten we zien hoe we een **subroutine**, die als tekstbestand op cassetteband of op diskette staat, kunnen **toevoegen** aan een programma dat reeds in het geheugen van de computer aanwezig is. We doen dit aan de hand van een voorbeeld. Bekijk nog eens de screen-dump met de inhoudsopgave van de programmadiskette op pagina 292. We zien dat op deze diskette het programma H9-1 staat. Dit is het KEY-TO-TAPE programma uit het begin van dit hoofdstuk. Stel dat we dit programma willen draaien maar dat we de gegevens willen sorteren voordat ze op cassetteband gekopieerd worden. We zullen daartoe het programma moeten uitbreiden met een sorteerroutine. De programma's SORT1, SORT2 en SORT3, die op dezelfde diskette staan als het programma H9-1, bevatten de drie sorteerroutines die we in hoofdstuk 7 besproken hebben. We gaan één van deze routines, laten we SORT1 nemen, toevoegen aan het programma H9-1 en bovendien voegen we in programma H9-1 een **GOSUB 1000**-opdracht toe om de subroutine aan te roepen. Op de volgende screen-dump zien we wat we hiervoor moeten doen.

```
load"h9-1"
Ok

merge"sort1"
Ok

275 gosub 1000' sorteren door selectie

run
Hoeveel waarden voert u maximaal in? 12

Hoe wilt u het bestand noemen? data86-1

   1 Voer de eerste waarde in?
   ..............................
   ..............................
   enzovoort

color  auto  goto  list  run
```

Na **load "h9-1"** staat het programma KEY-TO-TAPE in het computergeheugen. De opdracht **merge"sort1"** voegt hieraan het programmabestand met de naam "sort1" toe. Dit is het bestand met de sorteerroutine 'sorteren door selectie' uit hoofdstuk 7. Dit bestand is een tekstbestand; dat moet, anders werkt **merge** niet. **MERGE** 'mengt' een opgegeven programmabestand met het programma dat op dat

moment in het geheugen staat. Als in het ingelezen programma (achter MERGE) en in het programma in het geheugen regels voorkomen met dezelfde regelnummers dan hebben de regels in het ingelezen programma voorrang. Zorg ervoor dat programma's die 'gemengd' worden niet dezelfde regelnummers hebben. In de praktijk is dit mengen bijna altijd toevoegen (of uitbreiden).

MERGE "bestandsidentificatie"	voegt een programmabestand van cassetteband of van diskette toe aan het programma in het geheugen.

Behalve dat we bij het MERGEn moeten zorgen dat de regelnummers in de diverse programmamodulen elkaar niet 'storen' moeten we ook eigenlijk precies weten welke variabelen in de diverse routines worden gebruikt. Als we een subroutine schrijven die aan diverse programma's 'geplakt' moet kunnen worden is het verstandig de namen voor de variabelen in die routine zo te kiezen dat ze in ieder geval niet met de namen van variabelen in andere programmamodulen overeenkomen. In de hier gebruikte sorteerroutine zouden we in plaats van de variabelen I en J wellicht beter II en JJ of I2 en J2 hebben kunnen gebruiken. Als we een subroutinebibliotheek gaan opbouwen dienen we steeds bij te houden:

a) de **naam** van de subroutine;
b) de **functie** van de subroutine;
c) de **regelnummers** die worden gebruikt;
d) de **invoervariabelen**;
e) de **lokale variabelen**;
f) de **uitvoervariabelen**;
g) eventuele andere **subroutines** die vanuit de subroutine worden aangeroepen.

Invoervariabelen voor een routine zijn programmavariabelen die elders in het programma worden gebruikt en waarvan de waarde in de subroutine wordt gebruikt; deze waarde mag alleen in de subroutine worden gewijzigd als de variabele ook een uitvoervariabele is. **Lokale variabelen** zijn variabelen die alleen binnen de subroutine worden gebruikt. Voor de in deze paragraaf gebruikte sorteerroutine zouden we het volgende moeten administreren:

a) **naam** : sorteren door selectie
b) **functie** : sorteren van een lijst getallen
c) **regelnummering** : 1000 t/m 1130
d) **invoervariabelen** : array X; de ongesorteerde lijst
N; aantal elementen in de lijst (aantal te sorteren getallen)

e) **lokale variabelen** : I ; loopt array X af
INDEX ; positie van de kleinste
J; loopt voor elke I de array X af
f) **uitvoervariabelen** : array X; de gesorteerde lijst
g) **aangeroepen subroutines** : géén

De variabele N is alleen een invoervariabele waarvan alleen de waarde in de subroutine wordt gebruikt; het is geen uitvoervariabele, dus de waarde van N mag in de routine niet veranderen.

opdracht 9.7

Neem aan dat we werken met een cassetterecorder en een drive. Geef een BASIC-opdracht om:

a) het programma "QUICKSRT" vanaf de diskette toe te voegen aan het programma dat in het geheugen staat;
b) het programma "HULP" vanaf cassetteband toe te voegen aan het programma dat in het geheugen staat.

Stel dat we ons systeem met een tweede drive, de B-drive uitbreiden.

c) We willen het programma "H7-6" laden vanaf de diskette in drive A; daaraan toevoegen het programma "SORT2" dat als ascii-bestand op de diskette in drive B staat; vervolgens toevoegen regel 280 GOSUB 2000; daarna het programma draaien; en ten slotte het gehele programma onder de naam "H7-6SORT" opslaan als tekstbestand op de diskette in drive B. Geef alle opdrachten die hiervoor moeten worden ingetoetst.
d) Met welke opdracht kunnen we dit "H7-6SORT" programma direct vanaf de diskette in drive B draaien?

9.5 GEGEVENS VAN EN NAAR DISKETTE

We kunnen het **KEY-TO-TAPE** programma uit 9.3 als voorbeeld nemen voor een **KEY-TO-DISK** programma. Het enige wat we moeten veranderen is de **OPEN**-opdracht. We veranderen gewoon

```
290  OPEN "cas:" + B$ FOR OUTPUT AS #1
```

in een van de volgende twee **OPEN**-opdrachten:

```
290  OPEN "A:" + B$ for OUTPUT AS #1
```

of

```
290  OPEN B$ FOR OUTPUT AS #1
```

Als we de apparaatnaam (cas:, A: of B:) in een OPEN-opdracht weglaten dan opent DISK BASIC automatisch (default) een buffer voor de disk drive, ook als tevens een cassetterecorder als randapparaat is aangesloten. Een gegevensbestand dat op deze manier op diskette wordt 'aangemaakt' heet een sequentieel bestand omdat het zoeken naar een bepaald gegeven in het bestand alleen kan door het bestand vanaf het begin, gegeven voor gegeven, in te lezen. Er zijn ook zogeheten 'directe bestanden' (random files) waarin bepaalde gegevens min of meer (in één record) direct uit het bestand kunnen worden gelicht zonder dat daarvoor alle voorgaande gegevens (lees: records) ingelezen hoeven te worden. In dit boek behandelen we alleen de sequentiële gegevensbestanden.

Veel mensen denken dat het toevoegen van gegevens aan een bestand heel iets anders is dan het maken van een nieuw gegevensbestand. Als je echter bedenkt dat het maken van een nieuw bestand niets anders is dan nieuwe gegevens toevoegen aan een 'leeg' bestand, dan is het duidelijk dat er eigenlijk geen verschil is. Je zou je zelfs voor kunnen stellen dat de computer bij de OPEN-opdracht:

```
OPEN "apparaatnaam:" + "bestandsnaam" FOR OUTPUT AS #1
```

eerst nagaat of er al een gegevensbestand met de gegeven bestandsnaam bestaat om vervolgens, als het inderdaad bestaat, het bestand te openen om er gegevens aan toe te kunnen voegen, en als het niet bestaat het bestand als een nieuw bestand te openen. De ontwerpers van MSX BASIC hebben echter anders besloten! Ze hebben namelijk besloten dat, als je gegevens aan een bestaand sequentieel bestand wilt toevoegen, je dat ook expliciet met behulp van het woord APPEND in de OPEN-opdracht moet aangeven. Bovendien kan je APPEND alleen gebruiken bij diskettebestanden en niet bij bestanden op cassetteband.
De volgende OPEN-opdracht opent een uitvoerbuffer voor het bestand "DATABANK" om er gegevens aan toe te kunnen voegen:

```
OPEN "DATABANK" FOR APPEND AS #1
```

Met een hele kleine aanpassing kunnen we het hierboven bedoelde KEY-TO-DISK programma zowel geschikt maken voor het maken van een nieuw bestand als voor het toevoegen van gegevens aan een bestaand bestand. Hieronder zien we wat we hiervoor moeten veranderen:

```
100 REM key-to-disk programma
110 '   Dit programma leest een aantal gegevens in,
120 '   slaat ze in een array op en kopieert de
130 '   inhoud van het array als een bestand op diskette
140 '
150 INPUT "Hoeveel waarden voert u maximaal in";N
160 DIM X(N)
170 PRINT
175 INPUT "T=toevoegen N=nieuw bestand"; A$
180 INPUT "Geef de bestandsnaam"; B$
190 PRINT : I=1
200 INPUT "   1 Voer de eerste waarde in";X(1)
210 '***doe zolang X niet 999999 is en het array niet vol is
220 IF X(I)=999999! OR I=N   THEN 280
230     I=I+1
240     PRINT USING "#### ";I;
250     INPUT "De volgende(999999=stop)";X(I)
260     GOTO 220
270 '***
280 REM kopieer array als bestand op diskette
290 IF A$="T" THEN OPEN B$ FOR APPEND AS #1
              ELSE OPEN B$ FOR OUTPUT AS #1
300 FOR J=1 TO I
310    PRINT#1,X(J)
320 NEXT J
330 CLOSE1
340 PRINT "De gegevens zijn gekopieerd"
350 END
```

De vraag is: wat doet MSX BASIC als we in een **OPEN**-opdracht voor het maken van een nieuw bestand 'per ongeluk' de naam opgeven van een reeds bestaand bestand? Het antwoord is even kort als desastreus: MSX BASIC vernietigt het oude bestand en opent een nieuw! Er wordt zelfs geen waarschuwing gegeven dat er al een bestand is met de opgegeven naam. We zullen straks zien hoe we onszelf, door een 'truc', voor zo'n situatie kunnen behoeden.

opdracht 9.8

Wat zal de computer doen als bij het uitvoeren van de opdracht OPEN "DATABANK" FOR APPEND AS #1 blijkt dat er geen bestand te vinden is met de naam "DATABANK"? Zou hij een melding geven dat het bestand niet gevonden kan worden of doet hij gewoon alsof er 'FOR OUTPUT' staat, met andere woorden opent hij gewoon een buffer voor een nieuw bestand?

Een foutafhandelingsroutine

Het goede antwoord op de vraag uit opdracht 9.8 is dat de computer een foutmelding geeft, namelijk een 'File not found'-melding. Gebeurt dit tijdens het draaien van een programma dan wordt het programma afgebroken. Omdat in het KEY-TO-DISK programma de 'OPEN B$ FOR APPEND AS #1'-opdracht pas wordt uitgevoerd nadat alle gegevens, die we aan het bestand B$ willen toevoegen, zijn ingetikt zullen ook de eventuele 'File not found'-melding en het afbreken van het programma pas dan optreden. Dit is wel erg zuur als we zojuist een paar honderd gegevens hebben zitten intikken, want die zijn we dan mooi kwijt!
Een oplossing zou zijn om de OPEN-opdracht uit regel 350 eerder in het programma, vlak na de INPUT-opdracht waarin naar de bestandsnaam gevraagd wordt, op te nemen. Dit voorkomt dan in ieder geval het 'voor niets' intikken van gegevens. Toch blijft het afbreken van het programma door een opgetreden fout een vervelende kwestie.

In MSX BASIC kunnen we echter met de ON ERROR GOTO-opdracht de computer, in plaats van het programma af te breken, dwingen een bepaalde subroutine in het programma uit te voeren. In deze subroutine kunnen wij zelf aangeven wat de computer moet doen als er een bepaalde fout is opgetreden en waar hij de uitvoering van het programma moet hervatten.

Elke fout die de MSX-vertolker kan lokaliseren heeft een nummer. Treedt ergens in een programma een fout op dan zorgt de computer ervoor dat de systeemvariabele (functie) ERR het nummer van de fout bevat en de systeemvariabele ERL het nummer van de programmaregel, die de fout veroorzaakt heeft. We gaan de werking van de ON ERROR GOTO-opdracht en zo'n foutafhandelingsroutine demonstreren aan de hand van het KEY-TO-TAPE programma, zoals dat vlak voor opdracht 9.8 is afgedrukt. In deze versie van het programma moet behalve de naam van het gegevensbestand ook een T (of t) of een N (of n) voor respectievelijk Toevoegen aan een bestaand bestand (APPEND) en Nieuw voor een nieuw te maken bestand (OUTPUT) worden ingetoetst. Als we de T voor toevoegen intikken zal de computer een foutmelding geven als we voor de bestandsnaam een naam opgeven die nog niet in de directory van de diskette, die op dat moment in de drive zit, staat. Een OPEN-opdracht met APPEND verwacht een bestandsnaam die al bestaat. Om een foutmelding te voorkomen moeten we kunnen nagaan of het genoemde bestand al bestaat. Hoe we dit kunnen nagaan zullen we straks zien. Als we de N intoetsen voor een nieuw bestand dan is het gevaarlijk als we een bestaande bestandsnaam intikken, want dan neemt de computer aan dat we het bestaande bestand met die naam willen vervangen door een nieuw bestand met dezelfde naam. Dit zou inderdaad de bedoeling kunnen zijn, maar het zou ook kunnen zijn dat we 'per ongeluk' voor het nieuwe bestand een bestaande naam intoetsen. In dat geval willen we op zijn minst een waarschuwing en de mogelijkheid om een

nieuwe naam in te toetsen. In schemavorm hebben we de volgende situatie:

bewerking \ bestand	bestaat al	bestaat niet
Toevoegen aan bestaand bestand (APPEND)	goed	geeft een foutmelding
Nieuw bestand maken (OUTPUT)	heel gevaarlijk	goed

Wat we nu in de foutafhandelingsroutine gaan doen is het onderscheppen van de foutmelding (rechtsboven in het schema) en de gebruiker attenderen op de gevaarlijk situatie (linksonder).

De aanleiding tot alle ellende is het al dan niet bestaan van een bestand waarvan de naam wordt ingetoetst (regel 240 in het programma KEY-TO-DISK). Het is dus zaak direct na het intoetsen van de bestandsnaam na te gaan of het bestand al dan niet bestaat. Hoe doen we dat?
Er is geen BASIC-functie die de waarde 'waar' of 'niet waar' aanneemt als een bepaalde naam wel of niet in de directory van een diskette voorkomt. Ook is er geen BASIC-opdracht waarmee we zelf de directory kunnen doorzoeken. We moeten dus een 'truc' verzinnen. We moeten zoeken naar een BASIC-opdracht die voor ons voor een bepaalde naam de hele directory afzoekt. Een van de opdrachten die dit doet is de NAME-opdracht, waarmee we een bestand een andere naam kunnen geven. In 9.4 hebben we hiervan een voorbeeld gezien. Nu willen we helemaal geen namen veranderen, we willen alleen dat NAME in de directory op zoek gaat naar de door ons ingetikte bestandsnaam. Bovendien willen we door de NAME-opdracht te weten komen of het bestand wel of niet in de directory staat.

Het zou prachtig zijn als NAME iets zou afdrukken als:

Bestand bestaat reeds, òf
Bestand niet gevonden

Nu wil het geval dat NAME inderdaad beide boodschappen kan afdrukken maar dan als foutmelding 'File already exists' en 'File not found'. We zullen dus met NAME zo'n foutmelding moeten forceren. De foutmelding 'File already exists' wordt afgedrukt als we met NAME een bestand de naam willen geven van een reeds bestaand bestand. Als we willen controleren of de ingetoetste bestandsnaam B$ al bestaat moeten we dus met de NAME-opdracht proberen een ander bestaand bestand de naam B$ te geven. We moeten dus de volgende NAME-opdracht gebruiken:

```
NAME "naam" AS B$
```

waarin we voor "naam" de naam van een bestaand bestand uit de directory van de, op dat moment in de drive aanwezige, diskette moeten opnemen. Wat we voor "naam" moeten nemen zullen we zo zien.

We willen ook, liefst met dezelfde NAME-opdracht, kunnen nagaan of het bestand met de naam B$ nog niet voorkomt; dan krijgen we namelijk een 'File not found'-foutmelding. Deze 'File not found'-melding krijgen we als we met de NAME-opdracht een bestand een andere naam willen geven, maar dat bestand niet op de diskette te vinden is. Het bestand B$ bestaat dus niet als de opdracht

```
NAME B$ AS "naam"
```

de foutmelding 'File not found' (lees: B$ bestaat niet) geeft. Willen we voor de beide situaties

```
NAME "naam" AS B$
```

èn

```
NAME B$ AS "naam"
```

dezelfde NAME-opdracht gebruiken, dan kan dit dus niets anders zijn dan de opdracht

```
NAME B$ AS B$
```

Inderdaad is, om uit te zoeken of het bestand B$ wel of niet op een diskette voorkomt, dit de meest praktische en logische oplossing, want:

1. Als B$ bestaat, zorgt de 'tweede' B$ in de NAME-opdracht voor een 'File already exists'-foutmelding.

2. Als B$ niet bestaat, zorgt de 'eerste' B$ in de NAME-opdracht voor een 'File not found'-foutmelding.

3. Niemand neemt zomaar een NAME B$ AS B$-opdracht in een programma op. Het is dan meteen duidelijk dat het niet om de NAME-opdracht zelf kan gaan, want wie verandert nu de naam van een bestand in dezelfde naam?!

Als we in ons KEY-TO-DISK programma deze NAME-opdracht na de INPUT-opdracht van regel 150 opnemen dan krijgen we, ongeacht of B$ wel of niet bestaat, altijd een foutmelding.

Door nu in de foutafhandelingsroutine, aan de hand van het nummer van de fout (systeemvariabele ERR), bepaalde acties te ondernemen, voorkomen we het eventuele afbreken van het programma. De acties die we moeten ondernemen halen we uit het eerder gegeven schema:

1. Als een T (of t) is ingetoetst op de vraag
   'Toevoegen of Nieuw bestand'
   èn
   Als de NAME-opdracht een 'File not found'-melding geeft
   Dan drukken we op het scherm af dat het bestand niet bestaat en vragen opnieuw om het intoetsen van een bestandsnaam

2. Als een N (of n) is ingetoetst op de vraag
   'Toevoegen of Nieuw bestand'
   èn
   Als de NAME-opdracht een 'File already exist'-melding geeft
   Dan drukken we op het scherm een waarshuwing af dat het nieuwe bestand het oude zal vervangen en vragen of dit akkoord is. Zo ja, dan gaat het programma gewoon verder, zo nee, dan vragen we opnieuw om een bestandsnaam in te toetsen.

Het KEY-TO-DISK programma met de foutafhandelingsroutine ziet er als volgt uit:

```
100 REM key-to-disk programma met foutafhandelingsroutine
110 '   Dit programma leest een aantal gegevens in,
120 '   slaat ze in een array op en kopieert de
130 '   inhoud van het array als een bestand op diskette
140 ON ERROR GOTO 1000 ' foutafhandelingsroutine
150 INPUT "Hoeveel waarden voert u maximaal in";N
160 DIM X(N)
170 PRINT
180 INPUT "Toevoegen(T,t) of Nieuw bestand(N,n)"; A$
190 '***doe zolang niet T of t of N of n is ingetoetst
```

```
200 IF A$="T" OR A$="t" OR A$="N" OR A$ ="n" THEN 240
210    INPUT "Foutieve invoer,probeer opnieuw,T of t  of N of n";A$
220    GOTO 200
230 '***
240 IF A$="T" OR A$="t" THEN T=-1
250 INPUT "Geef bestandsnaam"; B$
260 NAME B$ AS B$ ' dit veroorzaakt altijd een foutmelding
270 PRINT : I=1
280 INPUT "   1 Voer de eerste waarde in";X(1)
290 '***doe zolang X niet 999999 is en het array niet vol is
300 IF X(I)=999999! OR I=N   THEN 370
310     I=I+1
320     PRINT USING "#### ";I;
330     INPUT "De volgende(999999=stop)";X(I)
340     GOTO 300
350 '***
360 REM kopieer array als bestand op diskette
370 IF T THEN OPEN B$ FOR APPEND AS #1
        ELSE OPEN B$ FOR OUTPUT AS #1
380 FOR J=1 TO I
390    PRINT#1,X(J)
400 NEXT J
410 CLOSE1
420 IF T THEN PRINT "Gegevens toegevoegd aan '";B$; "'"
        ELSE PRINT "bestand '";B$;"' is gekopieerd"
430 CLEAR : END
445 '
1000 ' foutafhandelingsroutine
1005 '
1010 '
1020 ' deze routine reageert alleen op de "File not found"(ERR=53)
      en de "File already exists"(ERR=65) meldingen
1030 IF T AND ERR=53 THEN PRINT "Dit bestand bestaat niet":
                          PRINT : RESUME 250
1040 IF NOT T AND ERR=65 THEN PRINT:PRINT "PAS OP!!! naam bestaat al":
      PRINT "RETURN=doorgaan, andere toets=opnieuw":G$=INPUT$(1):
      IF ASC(G$)=13 THEN RESUME 270 ELSE RESUME 250
1050 RESUME 270
1060 '
1070 'terug uit foutafhandelingsroutine
```

In regel 140 zien we de 'ON ERROR GOTO'-opdracht. Mocht ergens een fout optreden dan zal het programma de subroutine op regel 1000 uitvoeren. Dit geldt voor alle fouten (zie Appendix F), behalve de 'Syntax error'-fout.
In regel 240 zien we het gebruik van de waarheidsvariabele T die 'true' (-1) gemaakt wordt als een T of een t is ingetoetst. Is een N of n ingetoetst dan blijft T 'false' (0), de systeemgekozen (default) waarde voor een waarheidsvariabele. Door dit te doen kunnen we in IF-opdrachten de nogal lange bewering A$ = "T" OR A$ = "t" gewoon vervangen door T (zie regels 370 en 420)

In de foutafhandelingsroutine zien we in regel 1030 wat er gebeurt als bij het toevoegen (T = 'true') een 'File not found'-melding (ERR=53) is opgetreden. Op het scherm wordt de melding 'Dit bestand bestaat niet' afgedrukt en het programma wordt vervolgens opgedragen verder te gaan met regel 250, waardoor we een andere bestandsnaam kunnen intoetsen.

RESUME regelnummer	geeft in een foutafhandelingsroutine aan dat het programma vervolgd moet worden met het uitvoeren van de regel met het opgegeven regelnummer

Is een N of n ingetoetst, dus geen T of t (NOT T = 'true') en is een 'File already exists'-foutmelding (ERR = 65) opgetreden, dan wordt een waarschuwing afgedrukt dat het bestand al bestaat. Vervolgens wordt gevraagd op RETURN te drukken als dit bestand door een nieuw vervangen mag worden of op een andere dan de RETURN-toets te drukken als we de naam voor het nieuwe bestand opnieuw willen intoetsen. De IF-opdracht in regel 1030 zorgt ervoor dat het programma na het indrukken van RETURN (ASC(G$) = 13) gewoon zijn weg vervolgt (RESUME 270) en anders teruggaat naar regel 250 (RESUME 250).

Is er een andere fout dan 'File not found' of 'File already exists' opgetreden dan gebeurt er niets. Het programma negeert de fout en gaat gewoon door (RESUME 270 in regel 1050) of het programma wordt door de fout afgebroken. Het programma kan echter niet meer afgebroken worden doordat een verkeerde bestandsnaam is ingetoetst; daar hebben wij nu zelf voor gezorgd. De CLEAR-opdracht in regel 430 zorgt ervoor dat MSX BASIC na afloop van het programma weer 'normaal' op fouten reageert.

In een groter programma kan natuurlijk op meer plaatsen een 'File not found' of een 'File already exists' foutmelding optreden. Afhankelijk van het nummer van de regel waarin zo'n fout is opgetreden moeten we actie ondernemen. Het is heel goed mogelijk dat we voor verschillende regelnummers verschillende boodschappen op het scherm willen afdrukken. Ook zal de uitvoering van het programma zeer waarschijnlijk op een ander punt (RESUME) hervat moeten worden. We kunnen dit alles realiseren door ook na te gaan in welke programmaregel de fout is opgetreden. We doen dit met de ERL-functie (ERror Line), waarin het systeem het regelnummer bewaart van de laatst opgetreden fout. In ons voorbeeldprogramma zouden we de twee IF-opdrachten uit de regels 1030 en 1040, te weten IF T AND ERR = 53 en IF NOT T AND ERR = 65 kunnen vervangen door:

```
IF T AND ERR = 53 AND ERL = 260
```

èn

```
IF NOT T AND ERR = 65 AND ERL = 260
```

De foutafhandelingsroutine reageert daardoor alleen op beide genoemde fouten indien deze in regel 260 (NAME B$ AS B$) zijn opgetreden.

opdracht 9.9

Verander het programma door na het optreden van een van de twee genoemde fouten eerst de inhoudsopgave van de diskette, waar op dat moment mee gewerkt wordt, af te drukken.

9.6 MEER BESTANDEN IN ÉÉN PROGRAMMA

We kunnen in één programma meerdere bestanden tegelijkertijd gebruiken. MSX BASIC neemt aan dat we standaard niet meer dan twee bestanden tegelijkertijd gebruiken. Willen we, bijvoorbeeld, drie bestanden tegelijkertijd gebruiken dan moeten we dat met behulp van de opdracht MAXFILES aan MSX BASIC vertellen. In het volgende programmafragment zien we hoe dat gaat:

```
150 MAXFILES = 3
160 OPEN "cas:DATA" FOR INPUT AS #1
170 OPEN "A:DATA" FOR OUTPUT AS #2
180 OPEN "A:CUMDATA" FOR APPEND AS #3
```

We kunnen maximaal 15 bestanden tegelijkertijd gebruiken. Denk erom: dit kost ook 15 geheugenbuffers. In de praktijk zal zich dit nooit voordoen! Ook bestanden die we openen voor bijvoorbeeld uitvoer naar het grafische scherm of naar de printer tellen voor MAXFILES mee. Gebruikt u helemaal geen bestanden in een programma, zet dan MAXFILES op nul; dat geeft iets meer vrije ruimte in het RAM-geheugen. Op de cassetterecorder kunnen we altijd maar één bestand tegelijkertijd gebruiken.

We geven nu twee voorbeelden van een programma waarin met meer dan één bestand gewerkt wordt. In het eerste programma gebruiken we twee bestanden, maar niet tegelijkertijd zodat we met MAXFILES = 1 kunnen werken. In het tweede voorbeeld werken we met twee bestanden tegelijkertijd (MAXFILES = 2).

Het eerste voorbeeld: Jeannette, een amateurmeteoroloog, tevens microcomputer-hobbyist, houdt op haar microcomputer een bestand

TEMP86 bij waarin zij elke dag de door haarzelf gemeten maximum temperatuur in bijschrijft. Op 31 december, 's avonds om tien uur, toetst zij de 365-ste temperatuurwaarde in; dit in de vaste overtuiging dat de door haar min/max-thermometer geregistreerde maximum temperatuur tussen tien en twaalf uur niet meer zal veranderen!
Ze wil nog graag die dag (dat wil zeggen voor twaalf uur 's nachts) met het TEMP86-bestand enkele berekeningen uitvoeren. Om het temperatuurverloop gedurende het jaar in beeld te brengen is het belangrijk de volgorde, waarin de temperaturen in het TEMP86-bestand zijn geregistreerd, niet te veranderen. Aan de andere kant zou het wenselijk zijn om naast dit TEMP86-bestand ook over een bestand te beschikken waarin de 365 temperatuurregistraties van 'klein' naar 'groot' gesorteerd staan; dit met het oog op een aantal statistische analyses. Jeannette besluit 'even snel' een programma te schrijven dat zo'n gesorteerd bestand maakt.

Volgens haar zal het programma achtereenvolgens de volgende handelingen moeten verrichten:
- openen van TEMP86 als invoerbestand
- inlezen van 365 gegevens in een array
- sluiten van TEMP86
- sorteren van de array
- openen van een nieuw bestand als uitvoerbestand
- kopiëren van de inhoud van de array op dit nieuwe bestand
- sluiten van het nieuwe bestand

Om straks onderscheid te kunnen maken tussen het **gewone** en het **gesorteerde bestand** besluit ze dezelfde bestandsnaam (TEMP86) te gebruiken maar voorzien van de typeaanduiding .SRT.
TEMP86.SRT is dus het geSoRTeerde bestand en **TEMP86** het ongesorteerde. Jeannette schrijft het hierna gegeven programma.

Vervolgens haalt ze met MERGE de routines 1000 en 7000 uit haar programmabibliotheek, tikt RUN in en maakt zo het TEMP86.SRT bestand. In onderstaand programma zien we dat het tweede bestand (TEMP86.SRT) pas wordt geopend nadat het eerste bestand gesloten is. Vandaar dat we in beide OPEN opdrachten (regels 100 en 160) hetzelfde buffernummer (#1) kunnen gebruiken. Beide bestanden gebruiken dus dezelfde geheugenbuffer voor het gegevenstransport

tussen programma en diskette. In het tweede programmavoorbeeld zien we een situatie waarin twee bestanden verschillende buffers gebruiken.

```
100 OPEN "TEMP86" FOR INPUT AS #1
105 DIM X(365)
110 FOR I = 1 TO 365
120    INPUT #1, X(I)
130 NEXT I
140 CLOSE
141 PRINT "Ongesorteerde temperaturen": PRINT
142 GOSUB 7000 ' Afdrukken gegevens
150 GOSUB 1000 ' Sorteren door selectie
151 PRINT "Gesorteerde temperaturen ": PRINT
153 GOSUB 7000 ' Afdrukken gegevens
160 OPEN "TEMP86.SRT" FOR OUTPUT AS #1
170 FOR I = 1 TO 365
180    PRINT #1, X(I)
190 NEXT I
200 CLOSE
210 END
```

Het tweede voorbeeld: Het bestand DATAMRT is een bestand met ik weet niet hoeveel gegevens! Het enige dat we weten is dat de gegevens in dit bestand gebroken getallen zijn en dat het bestand is afgesloten met een End-of-File teken. Het bestand moet, in verband met later uit te voeren analyses, wel worden bewaard.
In deze analyses zal echter alleen worden gewerkt met afgeronde gehele getallen. Daar deze gehele getallen minder plaats innemen dan de gebroken getallen en omdat het om nogal veel gegevens gaat wordt besloten een programma te schrijven dat het bestand inleest, de ingelezen getallen afrondt en ze weer op diskette kopieert. De naam van dit 'afgeronde' bestand moet weer DATAMRT worden.

Als we zouden weten om hoeveel getallen het gaat zouden we wellicht weer met een array kunnen gaan werken. Het gaat echter ook om veel getallen. Als er nu eens een paar duizend zijn? Dan zou het te declareren array wel erg groot worden (wellicht X(2000)?) en dat zou wel flink beslag leggen op het geheugen. De vraag is of we in dit geval wel een array nodig hebben (zo niet, dan komt dat goed uit). Het antwoord is nee, we hebben helemaal geen array nodig. We hebben zo'n array niet nodig omdat we in een programma tegelijkertijd met twee bestanden kunnen werken en wel door deze twee bestanden elk aan een eigen buffer te koppelen. Als beginners doen we al snel dit:

```
OPEN "DATAMRT" FOR INPUT AS #1
OPEN "DATAMRT" FOR OUTPUT AS #2
```

Fout! Een gegevensbestand mag nooit aan meer dan één buffer tegelijk worden toegewezen. Dit is jammer want het bestand met de afgeronde getallen (uitvoerbestand) moet wel dezelfde naam houden als het bestand met de gebroken getallen (het invoerbestand). Er zit dan niets anders op dan voor het nieuwe bestand zolang maar een andere naam te gebruiken:

```
OPEN "DATAMRT" FOR INPUT AS #1
OPEN "HULP" FOR OUTPUT AS #2
```

Als dit HULPbestand is gemaakt dopen we het gewoon om in DATAMRT; ten slotte weten we dat hiervoor de NAME opdracht beschikbaar is. Dit 'omdopen' wordt echter alleen door de computer uitgevoerd als er geen bestand is met de nieuwe naam; en dit bestand is er wel, want dat is het oude DATAMRT bestand. Voordat we dus het bestand HULP de naam DATAMRT kunnen geven moeten we zorgen dat er geen bestand is met die naam. Dit houdt in dat we voor de NAME opdracht een KILL opdracht zullen moeten geven voor het oude DATAMRT bestand.

We geven eerst een programmastructuurdiagram:

begin	
open DATAMRT als invoerbestand	
open HULP als uitvoerbestand	
	lees getal uit DATAMRT
	rond het getal af
	kopieer dit in HULP
herhaal tot einde-bestand DATAMRT is gelezen	
sluit de bestanden	
verwijder oude DATAMRT bestand	
geef HULP-bestand de naam DATAMRT	
einde	

Een sequentiële programmastructuur met één HERHAAL-TOTDAT lus; niet moeilijk te coderen:

```
100 OPEN "DATAMRT" FOR INPUT AS #1
110 OPEN "HULP" FOR OUTPUT AS #2
120 '===
130     INPUT #1, GEGEVEN
140     AFGGEG = INT(GEGEVEN + .5)
150     PRINT #2, AFGGEG
160 IF NOT EOF(1) THEN 130
170 '===HERHAAL TOT Einde bestand DATAMRT
190 CLOSE 1,2
200 KILL "DATAMRT"
210 NAME "HULP" AS "DATAMRT"
220 END
```

Bij de oefenopgaven vindt u een voorbeeld van een programma dat met drie bestanden tegelijkertijd werkt.

Behalve sequentiële bestanden kunnen we in MSX BASIC ook met 'direct georganiseerde' bestanden (random files) werken. Dit zijn gegevensbestanden die zijn opgebouwd uit een aantal 'records', die elk afzonderlijk 'direct' toegankelijk zijn, hetgeen betekent dat we om een bepaald record in te lezen alleen het recordnummer hoeven op te geven en niet het hele bestand hoeven af te zoeken tot we het gewenste record bereikt hebben.

Het programmeren zit er nu op. In het volgende hoofdstuk vindt u nog het een en ander over snellere programma's, variabelen en wiskundige functies.

OEFENOPGAVEN HOOFDSTUK 9

1. Op cassetteband staat een sequentieel gegevensbestand met de naam METING2. Schrijf een programma dat de gegevens uit dit bestand inleest en de kleinste, de grootste en de gemiddelde meetwaarde van alle in dit bestand geregistreerde metingen bepaalt. METING2 is het enige bestand dat op het tapeje staat.

2. Geef een opdracht uit DISK BASIC om
   a. het bestand PROG5 op de diskette in drive A te kopiëren en daarbij de kopie de naam PROG5.BAS te geven;
   b. alle bestanden van het type DAT de typeaanduiding GEG te geven. Er is één drive aangesloten;
   c. alle bestanden waarvan de naam begint met SAL en die geen typeaanduiding hebben te kopiëren van de diskette in drive A naar de diskette in de B drive.

3. Welke 'DISK BASIC'-opdracht doet hetzelfde als de twee vlak achter elkaar uitgevoerde opdrachten:

```
LOAD "B:PROEF.TXT"
SAVE "A:PROEF.TXT"
```

4. In het voorbeeld over Jeannette's temperatuurregistratieprogramma zeiden we dat Jeannette regelmatig de dagelijke gemeten maximum temperatuur in het programma TEMP86 bijschrijft. Schrijf een programma waarmee zij dat kan doen.

5. Op cassetteband staat het gegevensbestand STATOP86 met gegevens uit een statistisch onderzoek. Schrijf een programma dat alle gegevens uit dit bestand in een array inleest, het array sorteert, de mediaan bepaalt (uitleg volgt) en het array kopieert op diskette onder dezelfde naam. Het eerste gegeven uit het STATOP86 bestand is het getal dat aangeeft hoeveel gegevens het STATOP86 bestand bevat. Bij een statistisch onderzoek wordt vaak de 'middelste' waarneming gebruikt om een soort 'gemiddelde' waarneming te berekenen; zo'n middelste waarneming heet de mediaan. De mediaan is de middelste waarneming indien alle waarnemingen van klein naar groot gesorteerd zijn. (In de reeks 16,4,7,12,9 is de mediaan 9 want na rangschikking 4,7,9,12,16 is dit de middelste.) Bij een even aantal waarnemingen is de mediaan het gemiddelde van de 'middelste 2'!
Roep alleen de sorteerroutine aan, programmeer hem niet!

6. Stel dat u werkt met maar één drive. Op een diskette staat het programma PROG5-4. Dit programma moet samen met het programma ROUTINE7, dat ook op deze diskette staat, tot één programma worden samengesmolten. Dit nieuwe programma moet onder de naam PR5-4R7 op die diskette worden opgeslagen. Vervolgens moeten PROG5-4 en ROUTINE7 van de diskette worden verwijderd. ROUTINE7 is een ASCII bestand, PROG5-4 niet. Welke reeks besturingsopdrachten is hiervoor nodig?

7. Schrijf een programma dat alle getallen uit het bestand RUWEGEG inleest, alle positieve getallen kopieert in het bestand POSGEG en dat alle negatieve getallen kopieert in het bestand NEGGEG. Het bestand RUWEGEG staat op cassetteband, POSGEG en NEGGEG zijn diskettebestanden.

8. Op diskette staat een bestand KLAPPER waarin een aantal combinaties 'naam,telefoonnummer' staat opgeslagen. De namen en telefoonnummers staan als strings in paren achter elkaar (sequentieel) in het bestand.
Schrijf een programma dat aan de hand van een ingetoetste naam het bijbehorende telefoonnummer opzoekt en op het scherm afdrukt. Ga uit van bijvoorbeeld 70 paren naam,telefoonnummer.

Als dit is gelukt, verander het programma dan zo dat niet de hele naam behoeft te worden ingetoetst maar dat het programma de telefoonnummers opzoekt aan de hand van bijvoorbeeld 'de eerste zoveel letters' van de achternaam.

10 Nog Iets over Variabelen, Snelheid en (Rekenkundige) Functies

In dit laatste hoofdstuk gaan we wat dieper in op het gebruik van de diverse soorten variabelen. Tot nu toe hebben we alleen gelet op het netjes programmeren en niet op de 'snelheid' van een programma. Als je een programma echt snel wilt maken, moet je zoveel mogelijk geheeltallige (integers) en single precision variabelen gebruiken; zoveel mogelijk opdrachten op één regel zetten; weinig GOTO's gebruiken en geen GOSUB's. U zult begrijpen dat we, vooral dit laatste, niet zullen propageren. We zullen, binnen het kader van netjes programmeren, enkele suggesties geven om een programma sneller te maken.

10.1 PROGRAMMA'S 'SNELLER' MAKEN

Als voorbeeld nemen we het sorteerprogramma uit hoofdstuk 7 (pagina's 230 en 231). We laten dit programma op het scherm afdrukken hoe 'lang' het erover gedaan heeft. Hiertoe voegen we de volgende regels aan het programma toe:

```
110 TIME=0
265 X=TIME
266 PRINT:PRINT "Ik deed er";(X/50)\60;"minu(u)t(en) en";
                    (X/50) MOD 60;"seconde(n) over"
```

Bovendien breiden we het aantal te sorteren getallen tot 100 uit door er nog 66 aan toe te voegen. We nemen hiervoor de getallen 1 t/m 66 maar dan in omgekeerde volgorde:

```
280     DATA 100, 8, "  ##"
290     DATA 31,26,12,73,41,70,83,53,31,98,53,32,81,18,
             75,83,76,43,53,13,34,20,98,36,70,85,65,62,
             49,42,61,56,38,59
300     DATA 66,65,64,63,62,61,60,59,58,57,56,55,54,53,52,51,50,
             49,48,47,46,45,44,43,42,41,40,39,38,37,36,35,34,33,
             32,31,30,29,28,27,26,25,24,23,22,21,20,29,28,17,16
310     DATA 15,14,13,12,11,10,9,8,7,6,5,4,3,2,1
```

Om de invloed van de 'langzame printer' uit te schakelen veranderen we alle **LPRINT**'en in **PRINT**'en.
Met deze wijzigingen laten we het programma van de pagina's 230 en 231 draaien en kijken hoe lang het sorteren van 100 getallen duurt:

1 minuut en 30 seconden

Als we zoveel mogelijk geheeltallige variabelen gebruiken dan boeken we heel wat tijdwinst. Voor N, K, X( ), I, J en VOLG nemen we %-variabelen. Het sorteren duurt dan:

1 minuut en 11 seconden over

We zouden ook wat commentaar weg kunnen halen en wat meer opdrachten op één regel kunnen zetten, zoiets als:

```
100 REM  lijst inlezen,afdrukken,sorteren en afdrukken
110 TIME=0
120 READ N%,K%,FS$:DIM X%(N%)
160 FOR I%= 1 TO N% : READ X%(I%) : NEXT I%
200 CLS : PRINT "Ongesorteerde lijst:":PRINT
210 GOSUB 3020 : GOSUB 2020
250 PRINT:PRINT:PRINT "Gesorteerde lijst:":PRINT
260 GOSUB 3020 : X=TIME
266 PRINT:PRINT "Ik deed er";(X/50)\60;"minu(u)t(en) en";
                    (X/50) MOD 60;"seconde(n) over"
270 END
290 DATA 100,8,"  ##", 31,26,12,73,41,70,83,53,31,98,53,32,81,18,
                       75,83,76,43,53,13,34,20,98,36,70,85,65,62,
                       49,42,61,56,38,59
300     DATA 66,65,64,63,62,61,60,59,58,57,56,55,54,53,52,51,50,
             49,48,47,46,45,44,43,42,41,40,39,38,37,36,35,34,33,
             32,31,30,29,28,27,26,25,24,23,22,21,20,29,28,17,16
310     DATA 15,14,13,12,11,10,9,8,7,6,5,4,3,2,1
2020     FOR I%=2 TO N% : VOLG%=X%(I%) : J%=I%-1
2050        IF J%=0 OR VOLG%>=X%(J%) THEN 2090
2060            X%(J%+1)=X%(J%) : J%=J%-1 : GOTO 2050
2090        IF J%+1 <> I% THEN X%(J%+1)=VOLG%
2100     NEXT I%
2120 RETURN
3020     FOR I%=1 TO N% : IF I% MOD K%= 1 THEN PRINT
3030        PRINT USING FS$;X%(I%); : NEXT I%
3050 RETURN
```

Het duurt dan nog maar:

55 seconden

Hebt u deze tijdwinst over voor een minder leesbaar programma?

We kunnen ook alle overbodige spaties weghalen en bovendien subroutine 3000 in het hoofdprogramma opnemen. Dit onleesbare geheel afgedrukt op 40 tekens per regel ziet er dan zo uit:

```
100 TIME=0:READ N%,K%,FS$:DIM X%(N%)
110 FOR I%= 1 TO N%:READ X%(I%):NEXT I%
120 CLS:PRINT "Ongesorteerde lijst:":PRI
NT
140 FOR I%=1 TO N%:IF I% MOD K%= 1 THEN
PRINT
150 PRINT USING FS$;X%(I%);:NEXT I%
160 PRINT:PRINT:PRINT "Gesorteerde lijst
:":PRINT:GOSUB 2000
170 FOR I%=1 TO N%:IF I% MOD K%= 1 THEN
PRINT
180 PRINT USING FS$;X%(I%);:NEXT I%
190 X=TIME:PRINT:PRINT"Ik deed er";(X/50
)\60;"minu(u)t(en) en";(X/50) MOD 60;"se
conde(n) over":END
200 DATA 100,8,"  ##", 31,26,12,73,41,70
,83,53,31,98,53,32,81,18,75,83,76,43,53,
13,34,20,98,36,70,85,65,62,49,42,61,56,3
8,59,66,65,64,63,62,61,60,59,58,57,56,55
,54,53,52,51,50,49,48,47,46,45,44,43,42,
41,40,39,38,37,36,35,34,33,32,31,30,29
210 DATA 28,27,26,25,24,23,22,21,20,29,2
8,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,
2,1
2000 FOR I%=2 TO N%:VOLG%=X%(I%):J%=I%-1
:IF J%=0 OR VOLG%>X%(I%) THEN 2030
2010 IF J%=0 OR VOLG%>X%(I%) THEN 2030
2020 X%(J%+1)=X%(J%):J%=J%-1:GOTO 2010
2030 IF J%+1 <> I% THEN X%(J%+1)=VOLG%:N
EXT I%:RETURN
```

Maar het programma doet er nu nog maar 38 seconden over!

Ten opzichte van de oorspronkelijke 90 seconden is dit een hele verbetering! Als u een programma nooit meer hoeft aan te passen of aan anderen hoeft te laten zien dan kunt u gerust een snelle versie ervan bewaren, tenminste als snelheid echt belangrijk is. Ontwerp een programma toch altijd op de in dit boek voorgestelde manier. Bewaar dan altijd naast de snelle versie van een programma ook de goed gestructureerde langzamere versie. U zult deze laatste beslist nodig hebben als een programma veranderd moet worden.

10.2 VARIABELEN

In BASIC kunnen vier soorten variabelen worden gebruikt:

1. Integer variabele : voorbeeld TELLER%
2. Single Precision variabele : voorbeeld SOM!
3. Double Precision variabele : voorbeeld PRODUCT
4. String variabele : voorbeeld ANTW$

In BASIC geven we dus met een teken achter de naam van de variabele aan wat het type is van die variabele.

%	= integer
!	= single precision
niets	= double precision
$	= string

Als we 'niets' achter de naam zetten, kiest de computer automatisch (default) voor een double precision variabele. Er is wel een teken voor zo'n variabele, namelijk een hekje #. We zouden in het bovenstaande voorbeeld ook PRODUCT# in plaats van PRODUCT hebben mogen schrijven. We zullen nu de drie typen numerieke variabelen nog even de revue laten passeren en de conversie tussen de verschillende typen bekijken. Over stringvariabelen valt niet veel nieuws meer te vertellen.

Integer variabele

Een Integer variabele zoals TELLER% kan alleen geheeltallige waarden aannemen tussen -32769 en +32768. Als een integer variabele groter wordt dan +32767 of kleiner dan -32768 krijgt u een foutmelding. De waarde van een integer variabele neemt 2 bytes in beslag. Voorbeeld:

```
10 FOR I%=1 TO 10
20     TEST%=32760+I%
30     PRINT USING "  ######";TEST%;
40 NEXT I%
50 END

run
   32761    32762    32763    32764    32765
   32766    32767  32767
Overflow in 20
Ok
```

Single precision variabele

Van de waarde van een single precision variabele zijn alleen de eerste zes cijfers (van links gezien) nauwkeurig. Wordt de waarde van een single precision variabele zo klein of groot dat meer dan 14 cijfers nodig zijn om de waarde weer te kunnen geven, gaat MSX BASIC over op de wetenschappelijke notatie:

```
10 TEST!=123.4567!
20 PRINT TEST!
30 TEST!=999999!*8!
40 PRINT TEST!
50 TEST!=9999!^6!
60 PRINT TEST!
70 END
Ok
list
10 TEST!=123.457
20 PRINT TEST!
30 TEST!=999999!*8!
40 PRINT TEST!
50 TEST!=9999!^6!
60 PRINT TEST!
70 END

run
 123.457
 7999990
 9.994E+23
Ok
```

Aan dit voorbeeld kunnen we een aantal dingen zien. Als we aan een single precision variabele als waarde een single precision constante (123.4567! in regel 10), met een aantal cijfers achter de komma, toekennen, dan rondt MSX BASIC deze waarde af op 6 cijfers (zie regel 10 na list). Als we aan een single precision variabele een waarde toekennen die uit meer dan 6 cijfers bestaat (999999 × 8) dan zijn alleen de eerste 6 cijfers betrouwbaar (999999 × 8 = 7999992 en niet 7999990). Als we aan een single precision variabele een waarde toekennen die uit meer dan 14 cijfers bestaat (9999 tot de macht 6) dan gebruikt MSX BASIC de wetenschappelijke notatie; 9.994E+23 betekent 9,994 maal 10-tot-de-macht+23. (E is van exponent.) De waarde van een single precision variabele neemt **4 bytes** (geheugenplaatsen) in beslag.

Double precision variabele

De waarde van een double precision variabele wordt als een getal van maximaal 14 cijfers afgedrukt. Afgerond wordt op het meest rechtse (14e) cijfer.
Zouden er meer dan 14 cijfers moeten worden afgedrukt, gaat de computer automatisch over op de wetenschappelijke notatie. MSX-computers rekenen standaard met double precision waarden.

In onderstaand voorbeeld beginnen we met een grote double precision waarde en maken deze waarde steeds 100 maal zo groot. Let op wat er gebeurt bij het afdrukken. De #'jes heeft de computer er zelf bij gezet. Voorbeeld:

```
10 TEST=1819161024#
20 PRINT 100*TEST
30 PRINT 10000*TEST
40 PRINT 1000000#*TEST
50 END
Ok
run
 181916102400
 18191610240000
 1.819161024E+15
```

De waarde van een double precision variabele neemt 8 bytes in het geheugen in beslag.

opdracht 10.1

Wat wordt er door het volgende programma afgedrukt?

```
10  A! = .1
20  PRINT A! * 10^-7
```

10.3 CONVERSIE VAN NUMERIEKE VARIABELEN

Zonodig converteert BASIC waarden van numerieke variabelen van het ene type in het andere. Daarbij moeten de volgende punten in het oog worden gehouden.

1. Als aan een variabele een waarde wordt toegekend, wordt deze waarde eventueel geconverteerd naar het type van de gebruikte variabele.

Voorbeeld:

```
10  A% = 23.42
20  PRINT A%
run
 23
```

Als we een string als waarde aan een numerieke variabele willen toekennen geeft de computer een 'Type mismatch'-foutmelding.

2. Bij het berekenen van de waarde van rekenkundige uitdrukkingen converteert BASIC alle in de berekening gebruikte constanten en variabelen tot het type dat overeenkomt met de 'meest nauwkeurige' constante of variabele die in de berekening wordt gebruikt.
Voorbeeld:

```
10  PRODUCT! = 2.5*SOM*TELLER%+D!
```

rekenkundige uitdrukking

2.5 is een double precision constante
SOM is een double precision variabele
TELLER% is een integer variabele
D! is een single precision variabele

In de uitdrukking 2.5*SOM*TELLER%+D! is SOM de variabele met de hoogste nauwkeurigheid. Daarom zal de waarde van deze rekenkundige uitdrukking als double precision waarde worden uitgerekend (zie regel 40 in onderstaand voorbeeld). Als we deze double precision waarde aan een single precision variabele toekennen (zie regel 50) wordt de waarde afgerond op het zesde cijfer (van linksaf gezien).
Voorbeeld:

```
10 SOM=1234567#
20 TELLER%=3
30 D!=23455
40 PRINT 2.5*SOM*TELLER%+D!
50 PRODUCT!=2.5*SOM*TELLER%+D!
60 PRINT PRODUCT!
70 END

run
 9282707.5
 9282710
Ok
```

opdracht 10.2

TELLER% = 10, NOEMER% = 5, GTAL = 5.55555

Wat wordt er afgedrukt door de volgende regel 10?

```
10  PRINT TELLER% + TELLER%/NOEMER%*GTAL
```

3. Als een rekenkundige uitdrukking (rechts van het =-teken) een single precision waarde oplevert, die wordt toegekend aan een double precision variabele (links van het =-teken), dan wordt de waarde van de rekenkundige uitdrukking toch als double precision waarde berekend.

```
10 SOM!=2.3999
20 SOM#=SOM!*SOM!
30 PRINT SOM!;SOM#
40 END

run
 2.3999   5.75952001
```

Zelf kunnen we ook deze conversies uitvoeren:

CDBL(X) Converteert X in een DouBLe precision waarde.

Voorbeeld:

```
10 A!=.4555
15 B!=A!^2!
16 D#=CDBL(A!^2)
20 PRINT B!;D#
30 END

run
 .20748   .20748025
```

CINT(X) Converteert X in een INTeger waarde door het decimale stuk weg te laten. Als X groter dan of gelijk is aan 32768.0, of kleiner dan of gelijk is aan -32769, krijgt u een 'Overflow'-foutmelding.

Voorbeeld:

```
10 PRINT CINT(45.67)

run
 45
```

CSNG(X) Converteert X in een SiNGLe precision waarde.

Voorbeeld:

```
10 TEST#=123.456789#
20 PRINT TEST#; CSNG(TEST#)
30 END

run
 123.456789   123.457
```

opdracht 10.3

Wat wordt er afgedrukt?

```
10  GTAL = 567.85
20  PRINT CINT(GTAL),GTAL^3
```

Wat voor numerieke variabelen geldt, geldt ook voor numerieke arrays! Ook bij arrays kennen we vier verschillende typen:

```
RESULTAAT%(10,5) : Integer array
METING!(4,15)    : Precision array
LAB#(500)        : Double precision array
TEKST$(100)      : String array
```

Vooral bij arrays die al snel groot worden is het verstandig erop te letten dat niet een te nauwkeurig type wordt gebruikt. Zijn het alleen gehele getallen die in de array-elementen worden opgeslagen, gebruik dan %!

10.4 NAMEN VAN VARIABELEN

We kunnen de computer aan het begin van een programma opgeven dat bepaalde variabelen van een bepaald type zijn; dat gaat zo:

10	DEFSNG L-P	Alle variabelen beginnend met een L, M, N O of P zijn dan single precision variabelen.
10	DEFSTR A	Alle variabelen, beginnend met een A, zijn stringvariabelen.
10	DEFINT I-N,W-Z	Alle variabelen, beginnend met een I, J, K, L, M of N, of met een W, X, Y of Z, zijn integer variabelen

Vooral DEFINT is handig omdat we dan niet steeds al die %-tekens bij de variabelen hoeven te zetten.

opdracht 10.4

Geef een BASIC-opdracht om de computer te vertellen dat alle variabelen, die u in uw programma gaat gebruiken, van het type integer zijn.

De naam van een variabele mag erg lang zijn. Alleen de eerste 2 tekens uit een naam zijn significant, dat wil zeggen twee namen waarvan de eerste 2 tekens identiek zijn, worden door de computer als dezelfde naam gezien.

Een naam **moet beginnen met een letter** en mag verder uit letters en cijfers bestaan. Dit zijn dus goede BASIC-namen:

SOM, P15, I, R556

opdracht 10.5

Welke van de volgende namen van variabelen zijn verkeerd gespeld en waarom?

a. RENTEAFTREK82%
b. X-10
c. NAAM10$
d. 5ARTIKEL

10.5 WISKUNDIGE FUNCTIES IN BASIC

We zullen nu een overzicht geven van een aantal wiskundige functies die in BASIC kunnen worden gebruikt. Het betreft allemaal functies die werken met getallen of numerieke variabelen, geen stringfuncties dus. We geven bij elke functie aan wat de functie doet met daarbij een voorbeeld van het gebruik ervan. Hier komen ze in alfabetische volgorde.

1. ABS(X) geeft de absolute waarde van X. X kan een getal, een numerieke variabele of een rekenkundige uitdrukking zijn. Populair gezegd is de absolute waarde van een rekenkundige uitdrukking de getalwaarde van die uitdrukking zonder plus- of minteken. Voorbeeld:

```
ABS(5) = 5
ABS(7*(-5)) = 35
ABS(X1-X2)
```

In computerprogramma's komt het wel eens voor dat er moet worden getest of twee numerieke variabelen dezelfde waarde hebben, terwijl die waarden nooit exact aan elkaar gelijk zijn. We kunnen dit dan als volgt testen:

```
IF ABS(X1-X2) < 1.0E-06 THEN .....
```

In wiskundige notatie:

Als $|x_1-x_2| < 10^{-6}$ dan .....

opdracht 10.6

Geef een IF opdracht waarin u test of de variabele MAX minstens 10 groter of kleiner is dan de variabele MIN. Is dit zo, dan moet de routine op regel 2000 worden uitgevoerd.

2. EXP(X) geeft e-tot-de-macht X, dus e^X. X moet kleiner zijn dan 87.3365; e is het grondtal van de natuurlijke logaritme.
Voorbeeld:

```
10 FOR I=1 TO 5
20     PRINT EXP(I-1)
30 NEXT I
40 END

run
 1
 2.7182818284588
 7.38905609893
 20.085536923184
 54.59815003314
Ok
```

3. FIX(X) geeft het gehele deel van de getalwaarde van X. Het verschil met INT is dat bij negatieve waarden voor X FIX gewoon de cijfers voor de komma geeft, terwijl INT zo'n negatief getal afrondt tot het daaronder liggende negatieve getal.
Voorbeeld:

```
10  PRINT FIX(58.75);FIX(-58.75)

run
  58 -58  ← dit is het verschil met INT
```

4. INT(X) geeft het grootste gehele getal kleiner dan of gelijk aan X.
Voorbeeld:

```
10  PRINT INT(58.75);INT(-58.75)

run
  58 -59  ← dit is het verschil met FIX
```

5. LOG(X) geeft de natuurlijke logaritme van X, ln(x) dus.
Voorbeeld:

```
10 PRINT LOG(45/7)

run
 1.860752340715
Ok
```

LOG(X) is de inverse functie van EXP(X), met andere woorden: LOG(EXP(X)) = X.

6. RND(X) Het argument X bepaalt de manier waarop het toevalsgetal bepaald wordt. Als X positief is (we hebben tot nu toe altijd X = 1 gebruikt) dan wordt steeds bij het draaien van het programma dezelfde reeks toevalsgetallen getrokken.
Als er voor X een negatief getal nemen, wordt een andere reeks getrokken; voor elke andere negatieve waarde van X een andere reeks.
Als we voor X nul nemen, wordt het laatst getrokken toevalsgetal herhaald. De onderstaande voorbeelden laten dit zien:

```
10 '
20 FOR I=1 TO 10
30     PRINT USING "  ##";INT(100*RND(1)+.5);
40 NEXT I
50 END

run
  60   11   77   58   73   18   37   95   64   47
Ok



10 X=RND(-4)
20 FOR I=1 TO 10
30     PRINT USING "  ##";INT(100*RND(1)+.5);
40 NEXT I
50 END

run
  40   91   63   18   64   83   40    5   14   34
Ok



10 '
20 FOR I=1 TO 5
30     PRINT USING "  ##";INT(100*RND(1)+.5);
40     PRINT USING "  ##";INT(100*RND(0)+.5);
50 NEXT I
60 END

run
  60   60   11   11   77   77   58   58   73   73
Ok
```

Met **RND(-TIME)** kunnen we echt met een willekeurige startwaarde beginnen.

7. SGN(X)

Als X>0 dan SGN(X) = 1; als X<0 dan SGN(X) = -1 en als X=0 dan SGN(X) = 0.
Voorbeeld:

```
ON SGN(OPBRENGST-KOSTEN)+2 GOSUB 100,200,300
```

Als OPBRENGST-KOSTEN negatief, dan SGN = -1 en SGN+2 = +1 dan naar subroutine 100.
Als OPBRENGST = KOSTEN dan SGN = 0 en SGN+2 = 2 dan naar subroutine 200 en als OPBRENGST groter dan KOSTEN dan SGN = +1 en SGN+2 = +3 dan naar subroutine 300.

Wist u dat FIX(X) = SGN(X)*INT(ABS(X))!!!!
Zou de computer FIX(X) ook op deze wijze berekenen?

8. SQR(X)

geeft de vierkantswortel van de getalwaarde van X.
X moet groter dan of gelijk aan nul zijn.
Onderstaand voorbeeld geeft de oplossingen voor de vierkantsvergelijking $AX^2 + BX + C = 0$.
Als voorbeeld hebben we de vergelijking $x^2 - 3x + 2$ gekozen.

```
10 INPUT "Coefficienten:"; A,B,C
20 PRINT "X1 = ";(-B+SQR(B^2-4*A*C))/(2*A)
30 PRINT "X2 = ";(-B-SQR(B^2-4*A*C))/(2*A)
40 END

run
Coefficienten:? 1,-3,2
X1 =  2
X2 =  1
Ok
```

9. SIN(X), COS(X), TAN(X)

Deze drie functies berekenen respectievelijk de SINus, COSinus en TANgens van het argument X.
X moet in radialen, niet in graden worden opgegeven.
Een radiaal is een bepaalde meeteenheid waarin hoeken kunnen worden uitgedrukt. Een hoek van 360° komt overeen met 2π (twee-pi) radialen:

360° = 2π radialen.

Nu is π = 3,14159....., dus

360° = 6,28318 radialen.

Dus: 1 radiaal = 57,2958 graden èn
1 graad = 0,0174533 radiaal

In het volgende voorbeeld maken we een lijstje van de SINus en COSinus voor de hoeken tussen 0° en 90°.

```
10 PRINT "Hoek      SIN       COS" : PRINT
20 FOR HOEK=0 TO 90 STEP 10
30    H=.0174533*HOEK 'maak er radialen van
40    PRINT USING "####    +#.####   +#.####";
                  HOEK,SIN(H),COS(H)
50 NEXT HOEK
60 END

run
Hoek      SIN       COS

   0    +0.0000   +1.0000
  10    +0.1736   +0.9848
  20    +0.3420   +0.9397
  30    +0.5000   +0.8660
  40    +0.6428   +0.7660
  50    +0.7660   +0.6428
  60    +0.8660   +0.5000
  70    +0.9397   +0.3420
  80    +0.9848   +0.1736
  90    +1.0000   -0.0000
Ok
```

10.6 ZELFGEMAAKTE FUNCTIES

We hebben al veel eerder in dit boek kennis gemaakt met de **DEF FN**-opdracht voor het zelf definiëren van functies. Onder andere hebben we de functie DEF FNX = INT(1.4*X+37) gebruikt voor het 'corrigeren' van tekeningen op het grafische scherm.

In het volgende voorbeeld hebben we twee functies FNW1 en FNW2 gedefinieerd voor het uitrekenen van de twee Wortels van een vierkantsvergelijking:

```
10 DEF FNW1(A,B,C)=(-B+SQR(B^2-4*A*C))/(2*A)
20 DEF FNW2(A,B,C)=(-B-SQR(B^2-4*A*C))/(2*A)
30 INPUT "Coefficienten"; A,B,C
40 PRINT "X1 = "; FNW1(A,B,C)
50 PRINT "X2 = "; FNW2(A,B,C)
60 END
```

```
run
Coefficienten? 1,-3,2
X1 =  2
X2 =  1
Ok
```

In dit voorbeeld zien we dat we een functie meer dan één argument mogen geven en dat de variabelen die we voor de argumenten kiezen dezelfde mogen zijn als waarmee de functie wordt aangeroepen.
We kunnen ook een stringfunctie maken. Hieronder zien we een voorbeeld van een functie waarmee we een balk, bestaande uit N cursortekens, kunnen afdrukken.

```
10 DEF FNBALK$(N)=STRING$(N,219)
20 INPUT "Hoe lang moet de balk worden";N
30 PRINT FNBALK$(N)
40 END
```

De ASCII-code voor het cursorteken is 219. Willen we voor het tekenen van de balk het scherm schoon laten maken dan veranderen we regel 10 in:

```
10  DEF FNBALK$(N) = CHR$(12) + STRING$(N,219)
```

Met behulp van de CHR$-functie en andere tekstfuncties kunt u uw eigen besturingsfunctie ontwerpen. Zo zal de functie

```
DEF FNP$(X,Y) = CHR$(12) + STRING$(Y-1,31) +
                STRING$(X-1,28)
```

de cursor op kolom x en rij y zetten.
We hebben nu zelf een LOCATE-functie gedefinieerd!; 31 is de ASCII-code voor 'cursor omlaag'; 28 is de code voor 'cursor naar rechts'; 12 is de code voor een schoon scherm en de cursor links bovenin, In plaats van LOCATE X,Y: PRINT "*"; kunnen we nu schrijven: PRINT FNP$(X,Y); "*";.

Hiermee eindigt de cursus. Ik hoop dat u een idee hebt gekregen wat netjes programmeren is en dat u nu zelf volgens de geschetste methode grotere programma's kunt ontwerpen en coderen. Daarnaast hoop ik dat u bijna alles te weten bent gekomen over MSX BASIC.

Appendix A

ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .1

1.1 a. 11 c. 11 e. 1
b. 2 d. 27 f. 8

2.1
```
10 A = 15
20 B = 25
30 PRINT A; "MAAL"; B; "IS"; A*B
```

3.1 Een h.

4.1 a. Als de inhoud van de tekstvariabele A$ niet de tekst 'STOP' is.
b. Als B kleiner is dan A.
c. Als de tekstvariabele WRD$ meer dan vijf tekens bevat.
d. Als GTAL een geheel getal is.

5.1 Het doel van het programma is het bepalen van het percentage e's en a's onder alle letters van een aantal ingetoetste woorden.
- Het programma moet interactief zijn.
- Als een woord is ingetoetst moet het aantal e's en a's in dat woord worden afgedrukt.
- Als een stopwoord wordt ingetoetst wordt het programma beëindigd nadat de percentages e's en a's onder alle ingetoetste letters zijn afgedrukt.

(We nemen aan dat inderdaad echte 'woorden' worden ingetoetst.)

6.1 Dan wordt een foutmelding afgedrukt en wel de 'Out of DATA in x'; x is het regelnummer van de READ-opdracht die de foutmelding tot gevolg heeft.

7.1 Jan, Janneke

8.1
```
1000 ' subroutine namen afdrukken
1005 '
1010    FOR I=1 TO N
1020       PLAATS=INSTR$(NAAM$(I)," ") 'zoek de spatie
1030       LINKS$=LEFT(NAAM$(I),PLAATS-1) 'pak voorletters
1040       L=LEN(NAAM$(I))
1050       RECHTS$=RIGHT$(NAAM$(I),L-PLAATS) 'pak achternaam
1060       PRINT LINKS$,RECHTS$
1070    NEXT I
1075 '
1080 RETURN 'uit namen afdrukken
```

9.1 a. CSAVE "proef"
b. CSAVE "proef" of SAVE "cas:proef"
c. CLOAD "sorteer"
d. LOAD "sorteer" of LOAD "A:sorteer"

10.1 1E-08

ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .2

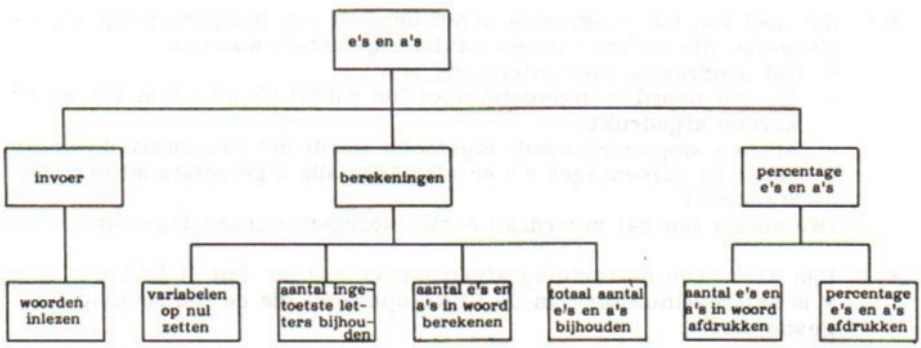
1.2 a. PRINT 5*3+2^4-5
b. PRINT 4*(3+1)-5/6
c. PRINT 2^3*3^2-8/(5+1)
d. PRINT 3*(5^2-3)^2
e. PRINT 4/5*8+(1-2*3)^2
f. PRINT 3/(4-3*2)*10

2.2 20: XA% = 5; 30: XA% = 6; 40: XA% = 5; 50: XA% = -1

3.2 a. Door een spatie in te tikken als antwoord op de vraag "Is dit een mooie kleur?"
b. In de kleur, die het laatst is gekozen (k in regel 40).
c. Ja, maar het zou ook een doe-zolang structuur kunnen zijn.

4.2 Dan gaat er in regel 70 iets fout, want daar delen we dan door nul (TELLER = 0!).

5.2



6.2 Dan gebeurt er niets; de lus 185-225 wordt niet uitgevoerd.

7.2 Er wordt niet getest of de lusvariabele I niet groter wordt dan N, de maximaal toegestane indexwaarde.

8.2
```
100 DIM PROG$(10)
110 FOR I = 1 TO 10
120    LINE INPUT PROG$(I)
130 NEXT I
140 TELLER = 0
150 FOR I = 1 TO 10
160    P = INSTR(PROG$(I),"LEFT$") ' Zoek eerste LEFT$
165    '***doe zolang P ongelijk nul
170    IF P=0 THEN 210
180         TELLER = TELLER + 1
190         P = INSTR(P+6,PROG$(I),"LEFT$") ' Zoek volgende LEFT$
200         GOTO 170
205    '***
210 NEXT I
220 PRINT "De LEFT$-functie wordt ";TELLER;" maal gebruikt"
230 END
```

9.2
```
185 INPUT "DISK OF CASSETTE (D/C); R$
186 IF R$ = "D" THEN R$ = "A: "
                 ELSE R$ = "CAS: "
290 OPEN R$ + B$ FOR OUTPUT AS #1
```

10.2 21.1111

ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .3

1.3 a. PRINT SQR(7.5) (denk aan de punt in plaats van een komma)
b. PRINT SQR(4^2+5^2)
c. PRINT SQR((4+5)^3) en niet SQR(4+5)^3!!
d. PRINT SQR(7) + SQR(8)
e. PRINT SQR(SQR(5)+7)

2.3 a. 100 : 12 = 8 REST 4

b. Tik in:
```
10 T% = 35
20 N% = 8
RUN
```

3.3 In dit geval wel. Als in een programma een numerieke variabele voor het eerst wordt gebruikt, krijgt zo'n variabele een standaard (default) waarde en wel de waarde nul. Toch is het aan te bevelen een cumulatieve variabele als SOM altijd zelf op nul te zetten, voordat je 'in' zo'n variabele begint op te tellen. Vindt een dergelijke optelling niet direct aan het begin van een programma plaats dan is het zeker verstandig; wellicht heeft dezelfde variabele in het voorgaande stuk van het programma al een waarde gekregen?!

4.3
```
34 IF GTAL < 0 THEN PRINT "MET NEGATIEVE GETALLEN DOE IK NIETS!"
```

5.3 hoofdmodule: variabelen op nul zetten
woorden inlezen
percentages afdrukken

submodule Klinker: totaal aantal ingetoetste letters bijhouden
aantal e's en a's in woord berekenen en afdrukken
totaal aantal e's en a's berekenen

6.3
```
20 READ NAAM$
35      READ LEEFTIJD%
50      READ NAAM$
```

7.3
```
10 READ N
20 DIM NAAM$(N),LEEFTIJD(N)
30 FOR I = 1 TO N
40     READ NAAM$(I),LEEFTIJD(I)
50 NEXT I
     .
     .
     .
```

8.3 Als een deel van de nieuw ingevoegde tekst gelijk is aan de zoekstring, zou dit deel weer door de nieuwe tekst vervangen worden. Stel dat Z$ = "land" en dat N$ = "waterland". Wordt er vervolgens vanaf het begin weer naar "land" gezocht, dan wordt "waterland" veranderd in "waterwaterland" en dit gaat zo door. Daarom moeten we verder zoeken vanaf de positie direct volgend op de zojuist ingevoerde tekst 'waterland'.

9.3 Als het bestand helemaal geen gegevens bevat dan treedt in regel 140 een foutmelding op omdat de INPUT #1-opdracht geen gegeven kan inlezen. Met een doe-zolang-structuur kan dit vermeden worden; maar wie houdt er nu een leeg bestand op na?

10.3 567 183105289.53663

ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .4

1.4
```
a. PRINT INT(5*2.7+6.3^2)
b. PRINT INT(X+0.5)
c. PRINT INT(SQR(520))
d. PRINT INT(SQR(433)+0.5)
e. PRINT INT(5*RND(1))-2 ;  5*RND(1) is 0, 1, 2, 3 of 4
```

2.4
```
raad het woord
het zijn 9 letters
de eerste is een p
de laatste is een a
```

3.4
```
10 REM Dit programma berekent het gemid-
   delde van de gehele getallen 1,2,3,
   4,...t/m N,het gehele getal
   dat wij intikken

80 PRINT "Het gemiddelde van 1 t/m";N;"is";SOM/N
```

4.4 110: ATW$; 120: = ; 140: 0 ; 170: GK ; 180: 1000 ; 220: TELLER ;
230: ANTW$; 1030: <, "hoger"

5.4 structuur hoofdmodule:
- commentaar
- variabelen op nul zetten
- woord inlezen
- zolang gebruiker dat wil
  - module Klinker aanroepen
  - woord inlezen
- percentages afdrukken
- afsluiten

6.4
```
25 PRINT TAB(0)"I"; TAB(7); "I^2"; TAB(15)"I^3"; TAB(23)"I^4"
26 PRINT
```

7.4
```
201 SOM = 0
202 FOR I% = 1 TO 6
203     SOM = SOM + I% * OGEN(I%)
204 NEXT I%
205 PRINT "Het gemiddelde aantal ogen per worp is"; SOM/50
```

8.4
```
1050 IF LEN(G1$) <= 60 AND LEN(G2$) <= 60 THEN 1096
1060       PRINT: PRINT "String(s) te lang (Max 60)": PRINT
1070       INPUT "Opnieuw, eerste getal "; G1$
1080       INPUT "Opnieuw, tweede getal "; G2$
1090       GOTO 1050
1096 IF LEN(G1$) <> LEN(G2$)
       THEN IF LEN(G1$) < LEN(G2$)
              THEN G1$=STRING$(LEN(G2$)-LEN(G1$)," ")+G1$
              ELSE G2$=STRING$(LEN(G1$)-LEN(G2$)," ")+G2$
1100 L = LEN(G1$)
```

9.4 a. FILES "SORT????"
b. COPY "*.ASC" TO "*" of COPY "A:*.ASC" TO "A:*"
c. KILL "*" of KILL "A:*"
d. LFILES "*.*" of LFILES "A:*.*"

10.4 DEFINT A-Z

ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .5

1.5 Een aanhalingsteken ("). Dit dient immers om het begin en het einde van een string aan te geven.

2.5 Waarschijnlijk een foutmelding omdat het programma een getal verwacht en geen string.

3.5
```
10 REM Ascii-code teken tabel
15 '
20 CLS
30 INPUT "Begin code "; BC
40 INPUT "Eind code "; EC
50 PRINT
60 FOR AC = BC TO EC
70    PRINT AC; " = "; CHR$(AC)
80 NEXT AC
90 END
```

4.5
```
55 IF A < B THEN PRINT A; "is kleiner dan"; B
              ELSE PRINT A; "is groter dan of gelijk aan"; B
```

5.5
```
100 REM FREQUENTIE van e's en a's in tekst
105 '
110 '                    ****************
115 '                    **              **
120 '                    ** Hoofdmodule **
125 '                    **              **
130 '                    ****************
135 '
140 SOM = 0: SOMA = 0: SOME = 0
150 INPUT "Uw woord "; WRD$
160 IF WRD$="STOP" THEN 200
170     GOSUB 1000 ' Klinker
180     INPUT "Uw woord "; WRD$
190     GOTO 160
200 IF SOM <> 0
    THEN PRINT"Percentage a's: ";INT(100*(SOMA/SOM+.005));" %"
         PRINT"Percentage e's: ";INT(100*(SOME/SOM+.005));" %"
    ELSE PRINT "Er is helemaal niets ingetoetst
210 END ' Hoofdmodule
```

6.5
```
10 REM Grafiek van x -> wortel(x)
15 '
20 CLS
30 PRINT "Grafiek van x naar wortel(x), voor x = 1 t/m 9 "
40 PRINT
50 FOR I = 1 TO 9
60    K% = SQR(I)*10
70    PRINT TAB(2) I; TAB(6) ":"; STRING$(K%,"."); TAB(8+K%)"*"
80 NEXT I
90 END
```

7.5

begin frequentieverdeling
lees leeftijd in
zolang gebruiker dat wil
bepaal de klasse waarin de leeftijd valt
tel 1 bij het aantal in deze klasse
lees leeftijd in
druk tabelkop af
voor klasse 1 t/m klasse 10 doe
druk frequentie in de klasse af
einde frequentieverdeling

8.5
```
270    IF SOM >= 2 THEN EOH = 1:
                        SOM = SOM - 2
                   ELSE EOH = 0
```

9.5 a. NAME "B:*" TO "B:*.BAS"
b. KILL "B:H???.BAS"
c. COPY "B:*.ASC" TO "A:" of COPY "B:*.ASC" TO "A:*.ASC"

10.5 b. fout, het streepje wordt als minteken gezien
d. fout, naam moet met een letter beginnen.

ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .6

1.6 PRINT LEFT$("PIANOKRUK",5) ; pak van vooraf de eerste vijf tekens uit PIANOKRUK

2.6 Gewoon RUN intikken en op RETURN drukken.

3.6
```
10 PRINT
20 FOR I=15 TO 45 STEP 10
30     PRINT I;"*";I;"=";I*I
40 NEXT I
50 END
```

4.6
```
130  A% = INT(10*RND(1)+10)
140  B% = INT(10*RND(1)+10)
```

5.6
```
1000 '
1005 '                        ******************
1010 '                        **                **
1015 '                        **  Subroutine    **
1020 '                        **    Klinker     **
1025 '                        **                **
1030 '                        ******************
1035 '
1040      SOM = SOM + LEN(WRD$)
1050      GOSUB 2000 ' Woordanalyse
1060      SOMA = SOMA + A
1070      SOME = SOME + E
1080      PRINT:  PRINT "In "; WRD$; " zitten"; A;
                        " a's en "; E; "e's"
1085 '
1090 RETURN ' Klinker
```

Het aantal e's en a's berekenen in een woord kan door de afzonderlijke letters uit dat woord te bekijken. We moeten dus steeds één letter van het woord vergelijken met een "A" en een "E". Als we nu eerst de laatste letter pakken, dan de laatste twee, dan de laatste drie, tot en met het hele woord en als we daarvan steeds de eerste letter pakken, dan lopen we alle letters van achteren naar voren af. We maken dus nog een verfijning van de deeltaak 'aantal e's en a's in woord berekenen' (de module Woordanalyse).


```
2000 '
2005 '                        ******************
2010 '                        **                **
2020 '                        **  Subroutine    **
2025 '                        ** Woordanalyse   **
2030 '                        **                **
2035 '                        ******************
2040 '
2050      A = 0: E = 0
2060      FOR I = 1 TO LEN(WRD$)
2070         L$ = LEFT$(RIGHT$(WRD$,I),1)
2080         IF L$ = "A"
              THEN A = A + 1
              ELSE IF L$ = "E" THEN E = E + 1
2090      NEXT I
2095 '
2100 RETURN ' Woordanalyse
```

```
6.6  LPRINT CHR$(27); "Q"; CHR$(80)     ← printer op 80-tekens
     LLIST                              ← druk programma op printer af

     LPRINT CHR$(27); "Q"; CHR$(64)     ← printer op 64-tekens
     RUN                                ← draai het programma
                                          (uitvoer komt op de printer)

7.6  begin    :  15,7,25,35,14,15
     1e stap  :  7,15,25,35,14,15
     2e stap  :  7,14,25,35,15,15
     3e stap  :  7,14,15,25,35,15
     4e stap  :  7,14,15,15,25,35
     5e stap  :  7,14,15,15,25,35  ← 25 stond toevallig al goed

8.6  171 '***doe zolang GoK niet evenveel letters
              bevat als WooRD
     172 IF LEN(GK$)=LEN(WRD$) THEN 180
     173      PRINT "SUFFERD! MIJN WOORD BESTAAT UIT";
                    LEN(WRD$);"LETTERS"
     174      INPUT "RAAD NOG MAAR EENS"; GK$
     175      GOTO 172
     176 '***
```

9.6 Ja, gewoon LOAD "INDEXCFS" en RUN. De DATA-regel is namelijk niet op de diskette veranderd, alleen in het interne geheugen.

```
10.6 IF ABS(MAX-MIN) >= 10 THEN GOSUB 2000
```

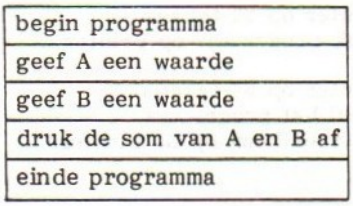
ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .7

```
1.7  PRINT "DE WORTEL UIT NEGEN IS "; SQR(9)

2.7  10 INPUT "X1=";X1
     20 INPUT "y1=";Y1
     30 INPUT "x2=";X2
     40 INPUT "y2=";Y2
     50 SCREEN2
     60 LINE (X1,Y1)-(X2,Y2)
     70 X$=INPUT$(1)
     80 SCREEN0
     90 END
```

Door regel 70 kunt u rustig naar de lijn blijven kijken. Zonder die opdracht is de lijn direct weer verdwenen!

```
3.7  111 INPUT "Hoeveel keer"; K
     112 INPUT "Hoeveel worpen"; W
     120 FOR KEER=1 TO K
     140     FOR WRP=1 TO W
     170     PRINT "Het gemiddelde van";W;
                   "worpen:";INT(10*SOM/W+.5)/10

4.7  1065 BEEP          (In plaats van BEEP kunnen we ook
     2065 BEEP           PRINT CHR$(7) nemen.)
```

5.7



6.7
```
10 REM Computer als tikmachine
20 '
30 INPUT ZIN$
35 '***doe zolang geen ! is getoetst
40 IF ZIN$="!" THEN 80
50     IF ZIN$ = "#"
         THEN LPRINT
         ELSE LPRINT ZIN$ 'Regel openlaten, anders zin afdrukken
60     INPUT ZIN$
70     GOTO 40
75 '***
80 END
```

7.7
```
1000 ' Subroutine Sorteren door selectie
1005 '
1010     FOR I = 1 TO N - 1
1020        WISSEL$ = "Nee"
1030        GOSUB 2000 'Zoek kleinste
1040        IF WISSEL$ = "Ja" THEN GOSUB 3000 'Wissel
1050     NEXT I
1055 '
1060 RETURN ' Sorteren door selectie
2000 ' Subroutine Zoek kleinste
2005 '
2010     INDEX = I
2020     FOR J = I + 1 TO N
2030        IF A$(I) < A$(INDEX) THEN INDEX = J
2040     NEXT J
2050     IF I <> INDEX THEN WISSEL$ = "Ja"
2055 '
2060 RETURN ' Zoek kleinste
3000 ' Subrouteine Verwissel
3005 '
3010     SWAP A$(INDEX),A$(I)
3015 '
3020 RETURN ' Verwissel
```

8.7
```
125 INPUT "Hoeveel tekens op een regel"; TPR
1030 '***doe zolang regel meer dan TPR tekens bevat
1040 IF L<=TPR THEN 1100
1050    LPRINT LEFT$(R$,TPR)
1060    R$=MID$(R$TPR+1)
```

9.7 a. MERGE "QUICKSORT" of MERGE "A:QUICKSORT"
b. MERGE "CAS:HULP"
c. LOAD "A:H7-6" of LOAD "H7-6"
MERGE "B:SORT2"
280 GOSUB 2000
RUN
SAVE "B:H7-6SORT", A
d. RUN "B:H7-6SORT"

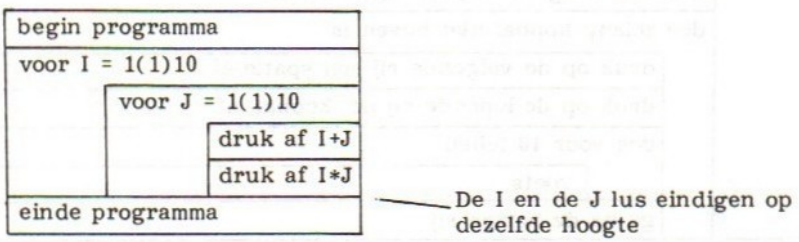
ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .8

2.8 Het opnemen van spaties in de tekstvariabelen A$, B$ en C$ betekent dat de geheugenruimte die nodig is voor het opslaan van de waarden van deze variabelen groter wordt, immers nu moet een geheugenla worden gemaakt voor het opslaan van "RIJ " in plaats van een la voor "RIJ". Als de spaties in de PRINT-opdracht (regel 50) worden opgenomen kost dat meer ruimte om de programmatekst (deze staat ook in het geheugen) op te slaan. Je zou het precies moeten bepalen (met FRE(0)) om het antwoord op deze vraag te kunnen geven.

3.8
```
115 INPUT "Hoeveel punten"; NP
116 INPUT "Voorgrondkleur"; VK
117 INPUT "Achtergrondkleur"; AK
118 INPUT "Randkleur"; RK
120 COLOR VK,AK,RK : SCREEN 2
140 FOR X1= 16 TO 240 STEP INT(224/(NP-1))
150       FOR X2= 16 TO 240 STEP INT(224/(NP-1))
```

4.8 a. Als SOM positief is.
b. Als het verschil van A en B hoogstens 5 is.
c. Als A tenminste 3 en B negatief is.
d. Als RENTE kleiner is dan 7 en BEDRAG tenminste 1000.
e. Als SOM kleiner is dan 25 en SOM tenminste 32; dus nooit!
f. Als TELLER minstens één of hoogstens 5 is; dus altijd!
g. Als de waarde van TELLER onder de 10 of boven de 20 ligt.
h. Als de inhoud van ANTW$ niet de tekst STOP is als TELLER hoogstens 50 is.
i. Als ANTW$ de string "JA" bevat of KEUS$ niet de string "STOP".

5.8



6.8

1	2	3	4	← positie
			5	
5	8	9	7	
		8	4	← wordt afgerond!

7.8 Ja, want I is de positie van de voorste en INDEX is de positie van de kleinste. Als X(I) <> X(INDEX) kan dat alleen als I<> INDEX.

9.8 Hij geeft een 'File not found'-melding.

ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .9

2.9 Nee, want in al deze waarden zit het BASIC-woord MID.

3.9
```
80 PRINT STRING$(AANTAL,219); AANTAL
```

4.9 a. Als A 'waar' is, dat wil zeggen als de waarde van A een getal ongelijk nul is (bijna altijd is dit -1).
b. Als RENTE-4 'waar' is, dus als RENTE niet gelijk is aan 4.
c. Als RENTE niet nul is en SALDO niet nul is.
d. Als NOT VERDER waar is, dus als VERDER 'niet waar' (false) is, dus nul.
e. Als NOT (A AND B) waar (true) is, dus als (A AND B) false is. A AND B is false als of A false is, of B, of allebei. (A AND B is waar, alleen als A èn B waar zijn!)

5.9

```
begin stuiterende kopbal
stel kleuren en scherm in
maak scherm schoon, zet functietoetsen af, kies basisvideogeheugen
kies kolom waarin de bal valt
druk 'kopbal' af op de beginpositie
    doe voor de eerste tot en met de een-na-laatste rij in de kolom waarin de bal valt
        druk een spatie af op de lopende rij
        druk de 'kopbal' af op de volgende rij
        doe voor 10 tellen
            niets
    lees hoe hoog de 'kopbal' moet komen
    doe zolang kopbal niet boven is
        druk op de volgende rij een spatie af
        druk op de lopende rij de 'kopbal' af
        doe voor 10 tellen
            niets
        ga na de vorige rij
herhaal tot de bal is uitgestuiterd
einde stuiterende kopbal
```

6.9 a. 16.25
b. 16.250
c. 16.3 (afronden)
d. 16.25 32.50 48.75
↑ ↑
2 spaties

7.9 begin : 15,7,25,35,14,15
1e stap : 7,15,25,35,14,15
2e stap : 7,15,25,35,14,15 ← 25 stond al goed
3e stap : 7,15,25,35,14,15 ← 35 stond ook al goed
4e stap : 7,14,15,25,35,15
5e stap : 7,14,15,15,25,35

Vergelijk deze vijf stappen met de vijf stappen uit opdracht 7.6.

9.9 Voeg in de regels 1030 en 1040 achter THEN de opdracht FILES toe.

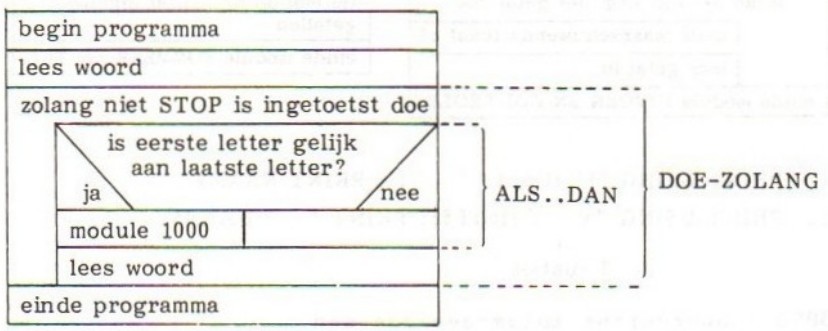
ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .10

2.10 Gebruik de PAINT-opdracht. Kies een punt binnen het vierkant en geef de coördinaten van dat punt op in de PAINT-opdracht. Neem bijvoorbeeld het middelpunt van het vierkant, namelijk X = 37 + INT(128/1.4+.5) en Y = 96. Dit geeft:

```
PAINT(37 + INT(128/1.4+.5),96)
```

3.10
```
10 REM 40 *'tjes op regel 2
20 WIDTH 40
30 CLS
40 FOR I=1 TO 40
50     VPOKE 39+I,42
60 NEXT I
```

5.10



6.10 a. 45.66 ; b. +45.56 ; c. **45.56 ; d. **45.6GLD ;
↑spatie ↑spatie
e. "#.##" ; f. "F**##.##" ; g. "+#.###" ; h. ".##" .

7.10 a. 1 ; 3 ; 5 ; 10
b. 6 ; 1 ; 0 ; 3

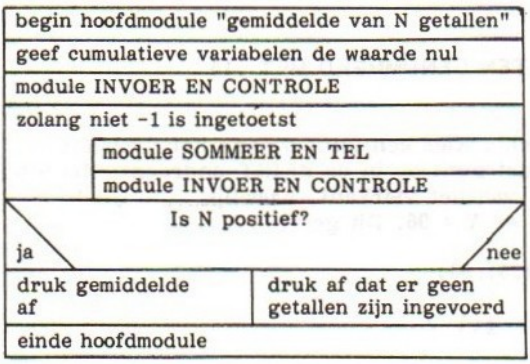
ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .11

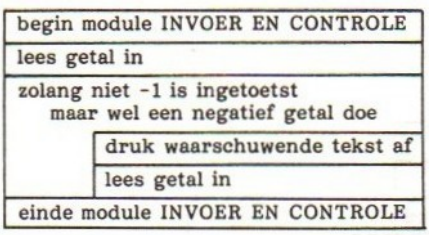
2.11
```
DEF FND = INT(6*RND(1)+1)
```

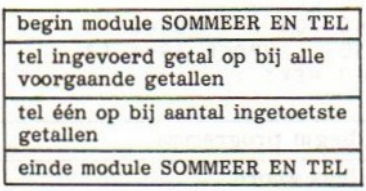
3.11
```
15 '===
20      LINE INPUT "Uw zin is: "; Z$
30      PRINT "'";Z$; "' is";LEN(Z$);
              "tekens lang"
40 IF Z$ <> "stop" THEN 20
55 '===herhaal tot 'stop' getoetst is
60 PRINT "Ik stop ook"
70 END
```

Het verschil is dat nu ook afgedrukt wordt hoe lang de zin "stop" is.

5.11







6.11 a.
```
PRINT USING "!" INIT$; "   "; : PRINT NAAM$
```

b.
```
PRINT USING "\   \";INIT$;: PRINT "   " NAAM$
            ↑
        3 spaties
```

7.11
```
3000 ' Subroutine kolom-gemiddelden
3010 '
3020     FOR J = 1 TO NK
3030        SOM = 0
3040        FOR I = 1 TO NR
3050           SOM = SOM + UITSLAG(I,J)
3060        NEXT I
3070        PRINT "Het gemiddelde van kolom "; J;
                  " is "; SOM/NR
3080     NEXT J
3090 '
3100 RETURN ' Kolom-gemiddelden
```

ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .12

3.12
```
140 PRINT CHR$(INT(224*RND(1)+32);
```

INT(224*RND(1)+32) is een geheel getal groter dan of gelijk aan 32 en kleiner dan of gelijk aan 255.

6.12 a. Als SOMPROD geen veelvoud van 11 is, dan is SOMPROD/11 een gebroken getal. Als we daarvan het gehele deel nemen (INT) en dit weer met 11 vermenigvuldigen komen we lager uit dan het getal waarmee we zijn begonnen.
Dan zal dus SOMPROD - 11*(gehele deel van SOMPROD/11) ongelijk nul zijn, en wel positief.

b.
```
1070 SOMPROD = SOMPROD + VAL(MID$(NUMMER$,11-I,1))*I
```

c.
```
1035 L = LEN(NUMMER$)
1040 FOR I = L TO 1 STEP -1
        CIJFER$ = MID$(NUMMER$,L+1-I,1)
```

7.11 In het hoofdprogramma toevoegen:

```
115 DIM RIJGEM(3), KOLOMGEM(4)
```

In subroutine 2000:

```
2065 RIJGEM(I) = SOM/NK
```

In subroutine 3000:

```
3065 KOLOMGEM(J) = SOM/NR
```

ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .13

3.13 a.
```
296 FOR I=13 TO 26
297     LOCATE I,23 : PRINT CHR$(32);
298     LOCATE I+1,23 : PRINT CHR$(1);CHR$(66);
299     FOR TEL=1 TO 30 : NEXT TEL
300 NEXT I
301 A$=INKEY$:IF A$="" THEN 301
302 END
```

b. Door in de wachtlus in regel 299 in plaats van 30 een waarde te kiezen die, naarmate de bal naar rechts rolt, steeds groter wordt, bijvoorbeeld

```
299 FOR TEL = 1 TO (I-10)^2: NEXT TEL
```

Voor I = 13 is de pauze 3^2 = 9 tellen; voor I = 26 is dit 16^2 = 256 tellen.

ANTWOORDEN VAN DE OPDRACHTEN GENUMMERD MET .14

3.14 SOM <> 40 ; TELLER% > 100 ; WRD$ = "STOP" ;
LEN(A$) <= 15 ; SQR(X) - 4*Y >= 0

ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .15

3.15 a. 4^2 < 18 : zestien "is kleiner dan" 18
b. "JAN" >"JACOB" : de ASCII code van de letter N "is groter dan" de ASCII code van de letter C
c. "A1A" < "A1B" : want "A" < "B"
d. "IK LOOP" < "IK ZING" : want "L" < "Z"
e. "JAN DE_NOOIJE" <"JAN DEN OOIJE" :
want de spatie heeft een ASCII code die kleiner is dan die van de letter N, korter gezegd: " " <"N" (32 < 78)

ANTWOORDEN VAN OPDRACHTEN GENUMMERD MET .16

3.16 a. 45 I = I+1

b. Dan komen we ook in een eindeloze lus, want I is altijd groter dan nul, omdat I in regel 20 één gemaakt wordt.

Appendix B

ANTWOORDEN VAN OEFENINGEN HOOFDSTUK 1

1. a. PRINT 4.5∧3
   b. PRINT 4*7∧2/3
   c. PRINT 6*(15.5-8.3)
   d. PRINT 3/(5+14)
   e. PRINT SQR(15)
   f. PRINT SQR(11+4/(2.5∧3*3.5))

2. a. PRINT 6.5*95
   b. PRINT 175/95*60

3. PRINT "TEMPERATUUR IN GRADEN CELSIUS: "; 5*(95-32)/9

4. PRINT "TEMPERATUUR IN GRADEN FAHRENHEIT: "; (9*36.8+160)/5

5. PRINT "EINDKAPITAAL: "; INT(75*(1+8/100) ^ 10+0.5"; "GULDEN"

6. PRINT 2*INT(6*RND(1)); 6*RND(1) is 0, 1, 2, 3, 4 of 5.

7. PRINT INT(10*8.57+0.5)/10

8. De uitkomst is 4, want

   RIGHT$("KLAVERTJEDRIE",4) = "DRIE" en LEN("DRIE") = 4

9. 5 + 49
   5 + 4 + 9
   5 + 4 = 9
   5 + 4 = 10 - 1

10. Zet het toetsenbord in de 'kleine letter stand'. Dit is de stand na het aanzetten van de computer. Tik nu in:

```
10 key off
20 key1,"print"
30 key2,"input"
40 key3,"end"
50 key on
```

Tik nu in LIST◁ en u ziet:

```
10 KEY OFF
20 KEY1,"print"
30 KEY2,"input"
40 KEY3,"end"
50 KEY ON
```

MSX BASIC maakt zelf hoofdletters van alle BASIC-opdrachten. De tekst tussen "-tekens blijft echter in kleine letters staan. Maak er een gewoonte van alles met kleine letters in te tikken. We doen dit vanaf hoofdstuk 3 ook!

ANTWOORDEN VAN OEFENOPGAVEN HOOFDSTUK 2

1. a. double precision
   b. single precision
   c. fout, streepje mag niet
   d. fout, OR is een BASIC woord
   e. geheeltallige variabele
   f. tekstvariabele
   g. double precision
   h. double precision
   i. fout, moet met een letter beginnen
   j. tekstvariabele
   k single precision
   l. fout, STOP is een BASIC woord

2. a.
```
10 INPUT "Geef een getal"; GTAL
20 PRINT "Op helen afgerond is dit";
              INT(GTAL+.5)
30 END
```

   b.
```
10 DEF FNG(X)=INT(X+.5)
20 INPUT "Geef een getal"; GTAL
30 PRINT "Op helen afgerond is dit";
              FNG(GTAL)
40 END
```

3.
```
14 -2.5  3
10000        19
```

4. a.
```
100 REM vierkant in het midden van
          het hoge resolutie scherm
110 CLS
120 INPUT "Geef een getal 0-95"; N
130 COLOR 1,10,13 : SCREEN 2
140 XA=128-N:YA = 96-N 'linksboven
150 XD=128+N:YD = 96+N 'rechtsonder
160 LINE (XA,YA)-(XD,YD),1,B
170 A$=INPUT$(1)
180 COLOR 15,4,4 : SCREEN 0
190 END
```

   Maar dit is géén vierkant. Voeg toe

```
135 DEF FNX(X) = 37 + INT(X/1.4+.5)
```

   en verander regel 160 in

```
160 LINE (FNX(XA),YA)-(FNX(XD),YD),1,B
```

   en het is wel een vierkant.

   b. Voeg achter de B in regel 160 een F toe, of voeg toe

```
165 PAINT (128,96)
```

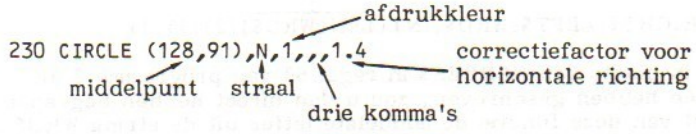
   (de F gaat sneller!).

c. Verander in regel 160 de ,1, in ,13, en voeg een F toe. Hierdoor wordt de rand van het vierkant echter ook paars.

d. Als u in uw MSX gebruikershandleiding de CIRCLE opdracht opzoekt zult u ontdekken dat de volgende opdracht een cirkel tekent met als middelpunt het midden van het beeldscherm en met een straal gelijk aan het ingetoetste getal:

```
230 CIRCLE (128,91),N,1
```

Dit levert echter een ellips op. Corrigeer dit met de X-Y verhouding; dat gaat zo:



5. a.
```
10 REM random getal tussen 10 en 20
20 PRINT INT(11*RND(1)+10)
30 END
```

b.
```
10 REM random getal tussen A en B
20 INPUT "Linkergrens";A
30 INPUT "Rechtergrens";B
40 PRINT INT((B-A+1)*RND(1)+A)
50 END
```

6.
```
10 PI=3.141592654#
20 INPUT "Uw wiel is een hoeveel inch wel"; DIAM
30 PRINT
40 PRINT "Uw wiel legt in een omwenteling"
50 PRINT "een afstand af van";INT(PI*DIAM*2.54)/100;"meter"
60 END
```

7.
```
10 INPUT "Uw telefoonnummer is";TELNUM$
20 PRINT "Dan is uw netnummer ";LEFT$(TELNUM$,3)
30 PRINT "En uw abonneenummer ";RIGHT$(TELNUM$,6)
40 END
```

Denk erom, TELNUM$ is een tekstvariabele, anders kunnen we geen streepje inlezen tussen het netnummer en abonneenummer.

8.
```
100 REM samengesteld interest
110 INPUT "Beginbedrag"; B
120 INPUT "Rentepercentage"; R
130 INPUT "Aantal jaar"; N
140 ' berekenen van het eindkapitaal
150 K=B*(1+R/100)^N
160 ' afronden op centen          dit doet de computer zelf na LIST.
170 K=INT(100*(K+5E-03))/100      Wij hebben .005 ingetoetst!
180 PRINT
190 PRINT "Eindkapitaal na";N;"jaar is f";K
200 END
```

9. Vervang in de PRINT-opdracht in regel 60 de variabele HEBBES$ door RIGHT$(LHELFT$,1). Vervang hierin LHELFT$ door LEFT$(WRD$,M%). Vervang hierin M% door INT(N%/2)+1. Vervang hierin N% door LEN(WRD$), dan krijg je dit:

```
                    HEBBES$
                       ↓
          RIGHT$(LHELFT$,1)
                       ↓
       RIGHT$(LEFT$(WRD$,M%),1)
                       ↓
    RIGHT$(LEFT$(WRD$,INT(N%/2)+1),1)
                       ↓
RIGHT$(LEFT$(WRD$,INT(LEN(WRD$)/2)+1),1)
```

Als we dit in plaats van HEBBES$ in regel 60 van programma 2 uit § 2.6 zouden hebben geschreven, zou u dan direct hebben begrepen dat de uitkomst van deze functie de middelste letter uit de string WRD$ is? Toch is het zo!

ANTWOORDEN VAN OEFENOPGAVEN HOOFDSTUK 3

1. a. A: 1,2,3,4,5
   b. B: 0,2,4,6
   c. TELLER : 10,25,40 ← (55 is groter dan 50!)
   d. TELLER : -10,0,10
   e. X: 10,9,8,7,6,5,4,3,2,1,0
   f. H: 5 (de lus wordt toch één keer doorlopen, zie pagina 72)
   g. I : 1,1,1,1,1 ..... eeuwig ← (gebruik nooit step 0)

2. `FOR GTAL = 1 TO N STEP 2`

3. ```
   10 CLS
   20 WIDTH 40
   30 FOR I=0 TO 39
   40    VPOKE I,42
   50 NEXT I
   60 END
   ```

   Vervang regel 40 door: `40 VPOKE 360+I,42`

4. ```
   100 REM opvolgende letters
   110 INPUT "Toets een a,b,c,d,e,f....of m in"; L$
   120 BC=ASC(L$)
   130 FOR I= 1 TO 10
   140    VL$=VL$+CHR$(BC+I)
   150 NEXT I
   160 PRINT "De volgende 10 letters zijn: ";VL$
   170 END
   ```

5.
```
100 INPUT "Tot hoeveel moet ik tellen"; N
110 '***doe zolang niet nul is getoetst
120 IF N=0 THEN 240
130   TIME=0
140   FOR I=1 TO N : NEXT I
150   TIJD=TIME : MIN=INT((TIJD/50)/60)
160   SEC=INT(TIJD/50)-MIN*60
170   PRINT
180   PRINT "IK HEB TOT";N;"GETELD"
190   PRINT "DIT DUURDE";MIN;"MINUTEN"
200   PRINT "          ";SEC;"SECONDEN"
210   INPUT "Tot hoeveel moet ik tellen"; N
220   GOTO 120
230 '***
240 END
```

6.
```
100 REM stuiterende kopbal
110 COLOR 12,15,4 : SCREEN1
120 CLS :KEY OFF : B=BASE(5)
130 FOR I=1 TO 27 STEP 2
140    VPOKE B+I,2
145 NEXT I
146 FOR CEL=B TO B+704 STEP 32
150    FOR I=1 TO 27 STEP 2
160      VPOKE CEL+I,32
170      VPOKE CEL+I+32,2
190  NEXT I
200 NEXT CEL
210 FOR CEL=B+704 TO B STEP -32
220    FOR I=1 TO 27 STEP 2
230      VPOKE CEL+I+32,32
240      VPOKE CEL+I,2
250    NEXT I
260 NEXT CEL
270 GOTO 146
280 END
```

7. De lusvariabelen zijn in de regels 50 en 60 verwisseld.

8.
```
10 WIDTH 40 : CLS
20 FOR I=0 TO 23
30    FOR J= 0 TO 39
40       VPOKE I*40+J,65+I
50    NEXT J
60 NEXT I
70 A$=INKEY$ : IF A$="" THEN 70
80 CLS: END
```

9.
```
a. SOM <= 525
b. VERSCHIL < 0
c. WRD$ = "GADOOR"
d. A-B >= 0
e. SQR(B^2 - 4*A*C) > 0
f. INT(6*RND(1)+1) >= 2
```

10.
```
100 REM opvolgende letters
110 INPUT "Toets een a,b,c,d,e,f....of m in"; L$
120 '***doe zolang een letter boven de m is getoetst
130 IF ASC(L$)<=109 THEN 170
140     INPUT "Niet boven een m!"; L$
150     GOTO 130
160 '***
170 BC=ASC(L$)
180 FOR I= 1 TO 10
190    VL$=VL$+CHR$(BC+I)
200 NEXT I
210 PRINT "De volgende 10 letters zijn: ";VL$
220 END
```

11.
```
100 REM opvolgende letters
120 INPUT "Beginnen? (ja/nee)"; A$
130 '***doe zolang ja is getoetst
140 IF A$<>"ja" THEN 270
145    INPUT "Wat is uw letter "; L$
150    '***doe zolang een letter boven de m is getoetst
160    IF ASC(L$)<=109 THEN 200
170        INPUT "Niet boven een m!"; L$
180        GOTO 160
190    '***
200    BC=ASC(L$)
210    FOR I= 1 TO 10
220       VL$=VL$+CHR$(BC+I)
230    NEXT I
235    PRINT "De volgende 10 letters:", VL$
240    INPUT "Nog een keer"; A$
250    GOTO 140
260 '***
270 PRINT "Einde programma"
280 END
```

12. DOE-ZOLANG:

```
10 I = 1
20 IF I > 10 THEN 60
30   PRINT I;
40   I = I+1
50   GOTO 20
60 END
```

HERHAAL-TOTDAT:

```
10   I = 0
20   I = I+1
30   PRINT I;
40 IF I < 10 THEN 20
50 END
```

13. Als in regel 100 voor W$ iets anders wordt ingetikt dan "stop", komt het programma in een eindeloze lus.

14.
```
100 REM btw tabel
110 CLS
120 INPUT "Beginnen (j/n)"; A$
130 '***doe zolang ja is getoetst
140 IF LEFT$(A$,1)<>"j" THEN 290
150 PRINT: INPUT "Beginbedrag"; B
160 PRINT: INPUT "Eindbedrag "; E
170 PRINT: INPUT "Stapgrootte"; S
180 PRINT:PRINT
190 PRINT "bedrag  netto bedrag  19%btw
200 PRINT
210 FOR BEDRAG=B TO E STEP S
220    NET=INT(100*(BEDRAG/119*100+5E-03))/100
230    BTW=INT(100*(BEDRAG-NET+5E-03))/100
240 PRINT BEDRAG;"       ";NET;"      ";BTW
250 NEXT BEDRAG
260 PRINT: INPUT "Nog een tabel"; A$
270 GOTO 140
280 '***
290 END
```

15.
```
100 CLS:SCREEN0
110 REM vierkant met random vulling
130 FOR I=0 TO 19
140    VPOKE 2*40+5+I,219
150    VPOKE 21*40+5+I,219
160 NEXT I
170 FOR I=0 TO 19
180    VPOKE 2*40+5+I*40,219
190    VPOKE 2*40+24+I*40,219
200 NEXT I
210 '===
220     X=INT(18*RND(1)+1)
230     Y=INT(18*RND(1)+1)
240     VPOKE (2+Y)*40+5+X,INT(224*RND(1)+32)
250     SPIEK=VPEEK(2*40+1)
260 IF SPIEK<>63 THEN 220
270 '===herhaal tot vraagteken in linkerbovenhoek
280 END
```

ANTWOORDEN OEFENOPGAVEN HOOFDSTUK 4

1.
```
10 INPUT "a ="; A
20 INPUT "b ="; B
30 IF A>B THEN PRINT A;"is groter dan";B
40 IF B>A THEN PRINT B;"is groter dan";A
50 END
```

2.
```
100 INPUT "telefoonnummer:";TN$
110 '***doe zolang geen STOP of stop
120 IF TN$="STOP" OR TN$="stop" THEN 170
130     IF LEFT$(TN$,3)="070" THEN T=T+1
140     INPUT "telefoonnummer:"; TN$
150     GOTO 120
160 '***
170 PRINT
180 PRINT "Ingetoetst zijn";T;"Haagsche nummers"
190 END
```

3.
```
10 FOR KEER%=1 TO 100
20    OGEN%=INT(6*RND(1)+1)
30    IF OGEN%>3 THEN T%=T%+1
40 NEXT KEER%
50 PRINT "Aantal ogen > 3"; T%
60 PRINT "Aantal ogen <=3"; 100-T%
70 END
```

(Allemaal integer variabelen; dat scheelt geheugenruimte!)

4.
```
10 INPUT "Uw woord"; W$
20 '***doe zolang niet STOP
30 IF W$="STOP" THEN 80
40     IF LEFT$(W$,1)=RIGHT$(W$,1) THEN GOSUB 1000
50     INPUT "Uw woord"; W$
60     GOTO 30
70 '***
80 END
```

5. 1 2 3 4 6 5 4 3 4 5

6. De regels 30 en 40 verdwijnen; daarvoor in de plaats komt:

```
35 IF A > B
     THEN PRINT A; "is groter dan"; B
     ELSE PRINT B; "is groter dan"; A
```

We nemen hierbij aan dat A niet gelijk is aan B.

7.
```
10 REM N-faculteit
20 INPUT "N="; N
30 IF N>25 THEN PRINT "N mag maximaal 25 zijn"
           ELSE GOSUB 1000 'N-faculteit
40 END
1000 'subroutine N-faculteit
1010    NFAC=1
1020    FOR I=1 TO N
1030       NFAC=NFAC*I
1040    NEXT I
1050    PRINT N;"-faculteit is:"; NFAC
1060 RETURN 'uit N-faculteit
```

8.
```
IF N >= 1 AND N <= 25
```

9.
```
10 REM salaris berekenen
20 INPUT "Salaris"; SAL
30 IF SAL<7500 THEN PCT=0
                ELSE IF SAL<25000 THEN PCT=.1
                                  ELSE PCT=.25
40 PRINT "Te betalen belasting:";INT(PCT*SAL+.5)
50 END
```

10. a,
```
IF K = 1  THEN GOSUB 1000
IF K = 2  THEN GOSUB 2000
IF K = 3  THEN GOSUB 3000
```

b.
```
IF K = 1  THEN GOSUB 1000
          ELSE IF K = 2  THEN GOSUB 2000
                         ELSE IF K = 3  THEN GOSUB 3000
```

11. a. ON K/2 GOSUB
b. ON K+2 GOSUB
c. ON INT(K) GOSUB
f. ON SQR(K) GOSUB

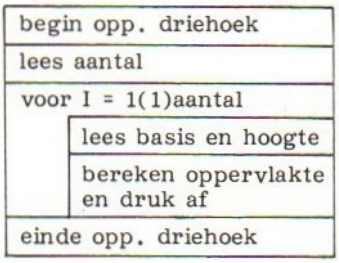
12. a.
```
SOM > 0 AND SOM < 100
VERSCHIL >= 7 AND VERSCHIL <= 12
LEEFTIJD < 18 OR LEEFTIJD > 65
ANTW$ = "JA" OR GEVDEN$ = "NEE"
GTAL > 10 AND SQR(GTAL) < 200
A+B = 10 AND A-B = 2
TELLER = 1 OR TELLER = 2 OR TELLER = 3
A > 0 AND B > 0 AND A <> B
```

ANTWOORDEN VAN OEFENOPGAVEN HOOFDSTUK 6

1.
```
10 FOR I=1 TO 29
20    LPRINT TAB(I)"*";TAB(60-I)"*"
30 NEXT I
```

2.
```
10 FOR I=1 TO 2 STEP .1
20    LPRINT USING "#.# ##.## ##.### ##.####";I,I^2,I^3,I^4
30 NEXT I
```

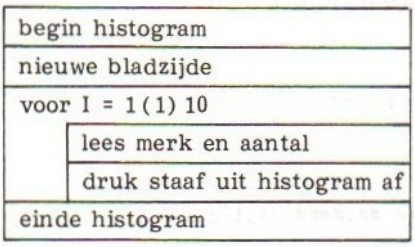
3.




```
100 REM oppervlak van een driehoek
110 READ AANTAL%
120 FOR DH%=1 TO AANTAL%
130    READ BASIS,HOOGTE
140    PRINT "Basis:"; BASIS
150    PRINT "Hoogte:"; HOOGTE
160    PRINT "Oppervlak:"; .5*BASIS*HOOGTE
170    PRINT:INPUT "Geef RETURN voor volgende driehoek"; D$
180 NEXT DH%
190 END
200    DATA 4
210    DATA 2,.5,4,1.5,3,.8,,3
```

4.

```
100 REM histogram
110 FOR I= 1 TO 10
120    READ MERK$,AANTAL%
130    LPRINT MERK$,TAB(16)":";TAB(17) STRING$(AANTAL%,"*")
140 NEXT I
150 END
160    DATA volkswagen,20,opel,30,citoen,15,ford,23
170    DATA renault,25,volvo,14,peugeot,16,mazda,18,toyota,24,bmw,16
```



5.

```
PRINT USING "##### ##### ##### #####"; I%,I%^2,I%^3,I%^4
```

(TAB mag niet worden gebruikt in PRINT USING!!)

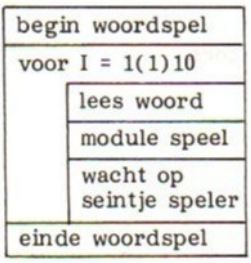
6.

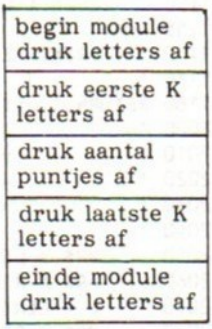
```
10 REM celsius en fahrenheit
20 READ BG,ED,ST
30 FOR GRADEN=BG TO ED STEP ST
40    LPRINT USING "+### C   =   +###.# F"; GRADEN,1.8*GRADEN+32
50 NEXT GRADEN
60 END
70    DATA -100,100,8
```

7.
```
100 REM de middelste letter
110 '===                          dit is waar als K even is
120     INPUT "Uw woord"; W$
130     K=LEN(W$)                                middelste 2 letters
140     IF W$<>"" THEN IF K-(K-2)*2=0
                  THEN PRINT "Middelste letters ";MID$(W$,K/2,2)
                  ELSE PRINT "Middelste letter ";MID$(W$,INT(K/2+1),1)
150 IF W$<>"stop" THEN 120
160 '===herhaal tot stop                          middelste letter
170 END
```

8.
```
1000 REM 'subroutine kop
1010  STREEP$=STRING$(41,"-")
1020  S1$=STRING$(10,"*")
1030  S2$="***"
1040  LPRINT STREEP$
1050  LPRINT S1$;"ARTIKEL";S2$;"PRIJS";S2$;"BTW";S1$
1060  LPRINT STREEP$
1070 RETURN ' uit kop
```

9.






```
100 REM Woordspel
105 '
110 FOR I = 1 TO 10
120    CLS
130    READ WRD$
140    GOSUB 1000 ' Speel
150    PRINT:
       INPUT "Druk op RETURN voor volgende spel "; DUMMY$
160 NEXT I
170 PRINT: PRINT "Het spel is afgelopen, tot ziens."
180 END
190    DATA MENS, HOED, KARPER, HELPER, KRANIG
200    DATA KAPPER, OPDRACHT, COMPUTER, VLOERKLEED, PARTERRETRAP
```

```
1000 '
1010 '          ******************
1020 '          *Subroutine Speel*
1030 '          ******************
1040 '
1050      K = 1 ' Druk de eerste en de laatste letter af
1060      PRINT "We beginnen": PRINT
1070      GOSUB 2000 ' Druk letters af
1080      PRINT
1090      INPUT "Welk woord heb ik in gedachten "; GOK$
1095      '***doe zolang woord niet geraden
1100      IF GOK$=WRD$ THEN 1160
1110       PRINT
1120       INPUT "Mis, wilt u de volgende twee letters zien "; A$
1130       IF A$ = "JA" THEN K = K + 1:
                             GOSUB 2000 ' Druk letters af
1140       IF K = .5*LEN(WRD$)
             THEN GOK$ = WRD$
             ELSE PRINT:
                  INPUT "Welk woord heb ik in gedachten"; GOK$
1150      GOTO 1100
1155 '***
1160      PRINT "Eindelijk ! Het woord is inderdaad "; GOK$
1170 '
1180 RETURN ' uit speel
2000 '
2010 '          ****************************
2020 '          *Subroutine druk letters af*
2030 '          ****************************
2040 '
2050      VR$ = LEFT$(WRD$,K)    'Eerste K letters
2060      AR$ = RIGHT$(WRD$,K)'Achterste K Letters
2070      L = LEN(WRD$) - 2*K      'Aantal nog te raden letters
2080      MN$ = STRING$(L,".")  'De geheime letters
2090      PRINT: PRINT TAB(20);VR$+MN$+AR$
2100 '
2110 RETURN ' uit druk letters af
```

ANTWOORDEN VAN OEFENOPGAVEN HOOFDSTUK 7

1. De kwadraten van de gehele getallen 1 t/m 15.

2.
```
10 DIM X(15), Y(15), Z(15)
20 FOR I = 1 TO 15
30     IF X(I) >= Y(I)
         THEN Z(I) = X(I)
         ELSE Z(I) = Y(I)
40 NEXT I
50 END
```

3.
```
100 REM  Frequentie van woordlengte
105 '
110 DIM AANTAL%(15)
120 INPUT "Uw woord ";WRD$
125 '***doe zolang meer dan alleen RETURN
130 IF WRD$="" THEN 210
135       '***doe zolang WooRD meer dan 15 tekens bevat
140       IF LEN(WRD$)<=15 THEN 170
150            INPUT "Uw woord is te lang (<= 15) "; WRD$
160       GOTO 140
165       '***
170       K% = LEN(WRD$)
180       AANTAL%(K%) = AANTAL%(K%) + 1
190       INPUT "Uw woord "; WRD$
200 GOTO 130
205 '***
210 CLS
220 FOR I = 1 TO 15
230    PRINT USING "## Letters : "; I;
240    PRINT STRING$(AANTAL%(I),"*")
250 NEXT I
260 END
```

4. De waarde 35; namelijk de grootste waarde uit de DATA-lijst.

5.
```
1000 ' Subroutine array omkeren
1010 '
1020      FOR I = 1 TO N/2
1030         SWAP X(I), X(N+1-I)
1040      NEXT I
1050 '
1060 RETURN ' uit array omkeren
```

6.
```
1000 ' Subroutine Lijstuitvoer
1010 '
1020      FOR I = 1 TO N
1030         IF I MOD 9 = 1 THEN PRINT
1040         PRINT USING "  ##.##"; X(I)
1050      NEXT I
1060 '
1070 RETURN ' uit lijstuitvoer
```

7.
```
100 REM  Mediaan bepalen
105 '
110 READ N
120 DIM X(N)
130 FOR I = 1 TO N
140    READ X(I)
150 NEXT I
160 GOSUB 1000 ' Sorteren door selectie
170 IF N MOD 2 = 0
         THEN PRINT "Mediaan is "; (X(N/2) + X(N/2 + 1))/2
         ELSE PRINT "Mediaan is "; X(INT(N/2)+1)
180 END
190      DATA 9
200      DATA 3, 5, 7, 12, 3, 6, 4, 9, 3
```

8.
```
100 DIM METING(4,5), X(5)
105 N = 5
110 FOR I1 = 1 TO 4
120    FOR J1 = 1 TO 5
130       READ METING(I1,J1)
132       X(J1) = METING(I1,J1)
140    NEXT J1
145    GOSUB 1000 ' Sorteren door selectie
146    FOR J1 = 1 TO 5
147       METING(I1,J1) = X(J1)
148    NEXT J1
150 NEXT I1
151 FOR I1 = 1 TO 4
152    FOR J1 = 1 TO 5
153       PRINT USING "### "; METING(I1,J1);
154    NEXT J1
155    PRINT
156 NEXT I1
160 END
170      DATA 11, 14, 13, 16, 14
180      DATA 9, 10, 3, 6, 11
190      DATA 22, 11, 33, 55, 44
200      DATA 5, 4, 3, 2, 1
```

9.
```
100 REM Klassen en frequenties
110 READ N ' N is het aantal waarnemingen
120 DIM KLASSE$(N), FR%(N)
130 SOM = 0: SC = 0: SR = 0
140 FOR I = 1 TO N
150    READ KLASSE$(I), FR%(I)
160    SOM = SOM + FR%(I)
170 NEXT I
180 PRINT "Klasse  Freq.  Cum.freq.  Rel.freq  Cum.relfreq"
190 PRINT
200 FOR I = 1 TO N
210    RELFREQ = FR%(I)/SOM ' Bereken relatieve klassefreq.
220    SC = SC + FR%(I) ' Cummuleer frequentie
230    SR = SR + RELFREQ ' Cumuleer rel.freq.
240    LPRINT KLASSE$(I); SPC(4);
250    LPRINT USING "###      ####       #.###       #.###";
                                    FR%(I),SC,RELFREQ,SR
260 NEXT I
270 END
280      DATA 5
290      DATA " 0-20", 45, "21-30", 77, "31-40", 80, "41-60", 50
300      DATA "61-90", 48
```

10.
```
7000 FOR I = 1 TO 3
7010    SOM = 0
7020    FOR J = 1 TO 4
7030       FOR K = 1 TO 3
7040          SOM = SOM + UITSLAG(I,J,K)
7050       NEXT K
7060    NEXT J
7070    PRINT "Gemiddeld cyfer voor wijnsoort ";I;" is "; SOM/12
7080 NEXT I
```

11.
```
100 REM  Letterfrquentie
105 '
110 DIM L(26)
120 INPUT "Uw woord "; WRD$
125 '***doe zolang geen fini is getoetst
130 IF WRD$="fini" THEN 170
140       GOSUB 1000 ' Woordanalyse
150       INPUT "Uw woord ";WRD$
160 GOTO 130
165 '***
170 GOSUB 2000 ' Letterhistogram
180 END
1000 ' Subroutine Woordanalyse
1010 '
1020      FOR I = 1 TO LEN(WRD$)
1030         L$ = MID$(WRD$,I,1)
1040         CODE% = ASC(L$) - 96
1050         L(CODE%) = L(CODE%) + 1
1060      NEXT I
1070 '
1080 RETURN ' uit Woordanalyse
2000 ' Subroutine Letterhistogram
2010 '
2020      CLS
2030      FOR I = 1 TO 26
2040         LPRINT CHR$(I+64);" : "; STRING$(L(I),"*")
2050      NEXT I
2060 '
2070 RETURN ' uit Letterhistogram
```

ASC(L$) is minimaal 65 (letter A) en maximaal 90 (letter Z). Dus ASC(L$)-64 is minimaal 1 en maximaal 26. De A's worden dus geturfd in L(1), de B's in L(2),....., en de Z's in L(26).
In regel 2040 moeten we bij het afdrukken erop letten dat we niet CHR$(I) maar CHR$(I+64) coderen, anders worden er geen hoofdletters afgedrukt!

ANTWOORDEN VAN OEFENOPGAVEN HOOFDSTUK 8

1. MID$("MICROCOMPUTER",1,5) = "MICRO" en
   MID$("MICROCOMPUTER",6,8) = "COMPUTER"

2
```
100 LINE INPUT ZIN$
110 T = 0
120 P = INSTR(ZIN$,"A")
130 '***doe zolang A gevonden
140 IF P=0 THEN 190
150       T = T + 1
160       P = INSTR(P+1,ZIN$,"A")
170 GOTO 140
180 '***
190 PRINT "In de zin zitten "; T; " A's"
200 PRINT "Dit is";INT(100*T/LEN(ZIN$)+.5);"procent."
210 END
```

3.
```
100 REM Adresetiketjes afdrukken
110 DIM NAAM$(5), ADRES$(5), WNPL$(5)
120 FOR I = 1 TO 50
130    FOR J = 1 TO 5
140       READ NAAM$(J), ADRES$(J), WNPL$(J)
150    NEXT J
160    FOR J = 1 TO 5
170       LPRINT NAAM$(J)
180    NEXT J
190    FOR J = 1 TO 5
200       LPRINT ADRES$(J)
210    NEXT J
220    FOR J = 1 TO 5
230       LPRINT WNPL$(J)
240    NEXT J
250 NEXT I
260 END
270     DATA .....................................
```

4.
```
100 REM Verwijderen van spaties
110 LINE INPUT "Uw zin is "; ZIN$
120 P = INSTR(ZIN$," ")
130 '***doe zolang een spatie gevonden
140 IF P=0 THEN 190
150      ZIN$=LEFT$(ZIN$,P-1)+RIGHT$(ZIN$,LEN(ZIN$)-P)
160      P = INSTR(ZIN$," ")
170 GOTO 140
180 '***
190 PRINT "Zin zonder spaties: "
200 PRINT ZIN$
210 END
```

5.
```
100 REM Palindromen
110 INPUT "Uw woord "; WRD$
120 L = LEN(WRD$)
130 N = INT(L/2)
140 PLDM = -1
150 I = 1
155 '***doe zolang PaLinDrooM niet gevonden
160 IF I>N OR NOT PLDM THEN 200
170      IF MID$(WRD$,I,1) <> MID$(WRD$,L-I+1,1)
             THEN PLDM = 0
180      I = I + 1
190 GOTO 160
195 '***
200 IF PLDM
        THEN PRINT WRD$; " is een Palindroom"
        ELSE PRINT WRD$; " is geen Palindroom"
210 END
```

6.
```
100 A$ = "De kat die krabt de krullen van de trap"
110 I = 1
120 DEEL$ = MID$(A$,I,1)
125 '***doe zolang 'rap' niet gevonden
130 IF DEEL$="rap" OR I>LEN(A$)-2 THEN 170
140       I = I + 1
150       DEEL$ = MID$(A$,I,3)
160       GOTO 130
165 '***
170 IF DEEL$ = "rap"
       THEN PRINT "Positie: "; I
       ELSE PRINT "Positie: "; 0
180 END
```

7.
```
100 LINE INPUT "Uw gebroken getal is "; GTAL$
105 '***doe zolang niet stop is getoetst
110 IF GTAL$="stop" THEN  190
120       P = INSTR(GTAL$,",")
130       GEHEEL$ = LEFT$(GTAL$,P-1)
140       DECIMAAL$ = RIGHT$(GTAL$,LEN(GTAL$)-P)
150       PRINT "Het gehele deel is: ";GEHEEL$
160       PRINT "Het decimale deel is: "; DECIMAAL$
170       LINE INPUT "Uw gebroken getal is "; GTAL$
180 GOTO 110
185 '***
190 END
```

8.
```
100 REM Telefoonklapper
110 INPUT "Eerste vier letters van achternaam "; NAAM$
115 '***doe zolang naam ongelijk stop
120 IF NAAM$="stop" THEN  200
130       FOR I = 1 TO 200
140          READ HNAAM$, TELNO$
150          IF LEN(HNAAM$) >= 4
             THEN IF LEFT$(HNAAM$,4) = NAAM$
                THEN PRINT "Naam: ";HNAAM$,"Tel: ";TELNO$
160       NEXT I
170       RESTORE ' Die was u vergeten, of niet !?
180       INPUT "Eerste vier letters van achternaam "; NAAM$
190 GOTO 120
195 '***
200 END
210       DATA .....
```

9.
```
150          IF LEN(HNAAM$) >= LEN(NAAM$)
             THEN IF LEFT$(HNAAM$,LEN(NAAM$)) = NAAM$
                THEN PRINT "Naam: ";HNAAM$," Telno: ";TELNO$
160       NEXT I
170       RESTORE
180       INPUT "Zoeknaam "; NAAM$
```

ANTWOORDEN VAN OEFENOPGAVEN HOOFDSTUK 9

1.
```
100 REM gegevens inlezen en analyseren
110 OPEN "cas:METING2" FOR INPUT AS#1
120 KLS=999999!
130 GRS=-999999!
140 SOM=0:N=0
150 '===
160     INPUT#1,X
170     N=N+1
180     IF X>GRS THEN GRS=X
190     IF X<KLS THEN KLS=X
200     SOM=SOM+X
210 IF NOT EOF(1) THEN 160
220 '===herhaal tot einde bestand
230 PRINT "Gemiddelde waarneming:";SOM/N
240 PRINT "Grootste waarneming  :";GRS
250 PRINT "Kleinste waarneming  :";KLS
260 CLOSE1
270 END
```

2. a. COPY "PROG5" TO "PROG5.BAS" of
COPY "A:PROG5" TO "A:PROG5.BAS"

   b. NAME "*.DAT" AS "*.GEG"

   c. COPY "A:SAL*" TO "B:" of
COPY "SAL*" TO "B:SAL*"

3. COPY "B:PROEF.TXT" TO "A:PROEF.TXT" of
COPY "B:PROEF.TXT" TO "A:"

4.
```
100 REM temperaturen bijschrijven in TEMP86
110 INPUT "Temperatuur(99=stop)"; T
120 OPEN "TEMP86" FOR APPEND AS #1
130 '***doe zolang niet 99 is getoetst
140 IF T=99 THEN 190
150     PRINT#1,T
160     INPUT "Temperatuur(99=stop)"; T
170     GOTO 140
180 '***
190 CLOSE1
200 END
```

5.
```
100 OPEN "cas:statop86" FOR INPUT AS#1
110 INPUT#1,N
120 DIM X(N)
130 FOR I=1 TO N
140    INPUT#1,X(I)
150 NEXT I
160 CLOSE1
170 GOSUB 3000 ' quicksort
180 M=N/2
190 IF N MOD 2 = 0
      THEN PRINT "Mediaan:";(X(M)+X(M+1))/2
      ELSE PRINT "Mediaan:";X(INT(M+1))
200 OPEN "A:statop86" FOR OUTPUT AS#1
210 FOR I=1 TO N
220    PRINT#1,X(I)
230 NEXT I
240 CLOSE1
250 END
```

6.
```
LOAD "PROG5-4"
MERGE"ROUTINE7"
SAVE "PR5-4R7"
KILL "PROG5-4"
KILL "ROUTINE7"
```

7.
```
100 REM voorbeeld met drie bestanden
110 MAXFILES=3
120 OPEN "cas:RUWEGEG" FOR INPUT  AS #1
130 OPEN "A:POSGEG"    FOR OUTPUT AS #2
140 OPEN "A:NEGGEG"    FOR OUTPUT AS #3
150 '===
160      INPUT#1,X
170      IF X>0 THEN PRINT#2,X
                ELSE IF X<0 THEN PRINT#3,X
180 IF NOT EOF(1) THEN 160
190 '===herhaal tot einde bestand 1
200 CLOSE
210 END
```

8. a.
```
100 REM Telefoonklapper
110 DIM NAAM$(70), NUMMER$(70)
120 ' Lees klapper in
130 OPEN "KLAPPER" FOR INPUT AS #1
140 FOR I = 1 TO 70
150    INPUT #1, NAAM$(I),NUMMER$(I)
160 NEXT I
170 CLOSE 1
180 '
190 INPUT "Wilt u een nummer gaan opzoeken "; ANTW$
195 '***doe zolang er nog gezocht moet worden
```

```
200 IF LEFT$(ANTW$,1) <> "J" AND LEFT$(ANTW$,1) <> "j" THEN 290
210       INPUT "Bij welke naam "; ZOEKNAAM$
220       I = 1
225       '***doe zolang naam niet gevonden
230       IF ZOEKNAAM$ = NAAM$(I) OR I >= 70 THEN 260
240            I = I + 1
250       GOTO 230
255       '***
260       IF ZOEKNAAM$ = NAAM$(I)
            THEN PRINT "Het nummer is: "; NUMMER$(I)
            ELSE PRINT "Naam niet gevonden"
270       INPUT "Wilt u nog meer nummers opzoeken "; ANTW$
280 GOTO 200
290 PRINT "Klapper geloten "
300 END
```

b.

```
195 '***doe zolang er nog egzocht moet worden
200 IF LEFT$(ANTW$,1) <> "J" AND LEFT$(ANTW$,1) <> "j" THEN 310
210       INPUT "Geef de eerste zoveel letters van de naam "; LS$
220       ALS = LEN(LS$)
230       GVN$ = "NEE"
240       FOR I = 1 TO 70
250          LN = LEN(NAAM$(I))
260          IF ALS <= LN
               THEN IF LS$ = LEFT$(NAAM$(I),ALS)
                 THEN PRINT "Naam, nummer: ";NAAM$(I);NUMMER$(I):GVN$="JA"
270       NEXT I
280       IF GVN$ = "NEE"
              THEN PRINT "Er is geen naam die zo begint.": PRINT
290       INPUT "Wilt u nog meer nummers opzoeken "; ANTW$
300 GOTO 200
305 '***
310 PRINT "Klapper gesloten "
320 END
```

Appendix C

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

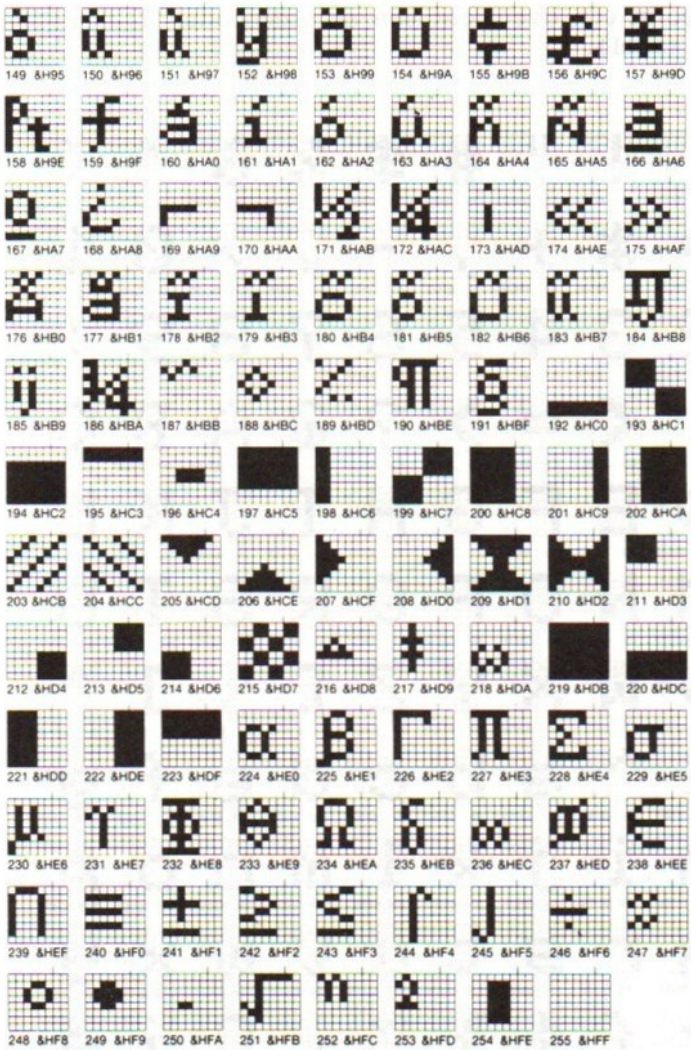
Appendix D

ASCII CODES (decimaal)

ASCII code	Teken	ASCII code	Teken	ASCII code	Teken	ASCII code	Teken
000	NUL	032	SP	064	@	096	'
001	SOH	033	!	065	A	097	a
002	STX	034	"	066	B	098	b
003	ETX	035	#	067	C	099	c
004	EOT	036	$	068	D	100	d
005	ENQ	037	%	069	E	101	e
006	ACK	038	&	070	F	102	f
007	BEL	039	'	071	G	103	g
008	BS	040	(	072	H	104	h
009	HT	041	)	073	I	105	i
010	LF	042	*	074	J	106	j
011	VT	043	+	075	K	107	k
012	FF	044	,	076	L	108	l
013	CR	045	-	077	M	109	m
014	SO	046	.	078	N	110	n
015	SI	047	/	079	O	111	o
016	DLE	048	0	080	P	112	p
017	DC1	049	1	081	Q	113	q
018	DC2	050	2	082	R	114	r
019	DC3	051	3	083	S	115	s
020	DC4	052	4	084	T	116	t
021	NAK	053	5	085	U	117	u
022	SYN	054	6	086	V	118	v
023	ETB	055	7	087	W	119	w
024	CAN	056	8	088	X	120	x
025	EM	057	9	089	Y	121	y
026	SUB	058	:	090	Z	122	z
027	ESC	059	;	091	[	123	{
028	FS	060	<	092	\	124	|
029	GS	061	=	093	]	125	}
030	RS	062	>	094	^	126	~
031	US	063	?	095	_	127	DEL

NUL - NULL
SOH - Start Of Heading
STX - Start of TeXt
ETX - End of TeXt
EOT - End Of Transmission
ENQ - ENQuiry
ACK - ACKnowledge
BEL - BELl
BS - BackSpace
HT - Horizontal Tabulation
LF - Line Feed
VT - Vertical Tabulation
FF - Form Feed
CR - Carriage Return
SO - Shift Out
SI - Shift In
DLE - Data Link Escape
DC - Device Controle
NAK - Negative AcKnowledge
SYN - SYNchronous idle
ETB - End of Transmission Block
CAN - CANcel
EM - End of Medium
SUB - SUBstitute
ESC - ESCape
FS - File Separator
GS - Group Separator
RS - Record Separator
US - Unit Separator
SP - SPace (blank)
DEL - DELete

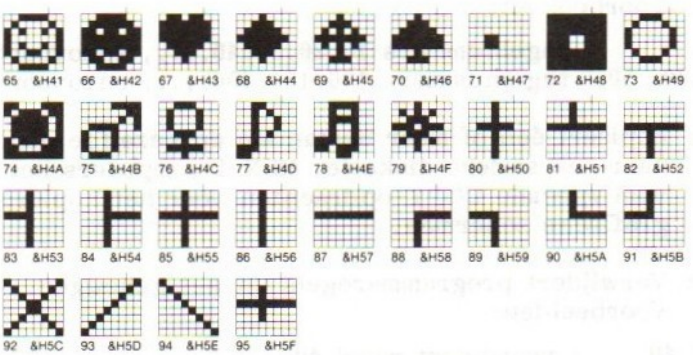
Hieronder volgt een overzicht van de videocodes 32 t/m 255 (zie ook voorgaande tabel).


149 &H95
150 &H96
151 &H97
152 &H98
153 &H99
154 &H9A
155 &H9B
156 &H9C
157 &H9D
158 &H9E
159 &H9F
160 &HA0
161 &HA1
162 &HA2
163 &HA3
164 &HA4
165 &HA5
166 &HA6
167 &HA7
168 &HA8
169 &HA9
170 &HAA
171 &HAB
172 &HAC
173 &HAD
174 &HAE
175 &HAF
176 &HB0
177 &HB1
178 &HB2
179 &HB3
180 &HB4
181 &HB5
182 &HB6
183 &HB7
184 &HB8
185 &HB9
186 &HBA
187 &HBB
188 &HBC
189 &HBD
190 &HBE
191 &HBF
192 &HC0
193 &HC1
194 &HC2
195 &HC3
196 &HC4
197 &HC5
198 &HC6
199 &HC7
200 &HC8
201 &HC9
202 &HCA
203 &HCB
204 &HCC
205 &HCD
206 &HCE
207 &HCF
208 &HD0
209 &HD1
210 &HD2
211 &HD3
212 &HD4
213 &HD5
214 &HD6
215 &HD7
216 &HD8
217 &HD9
218 &HDA
219 &HDB
220 &HDC
221 &HDD
222 &HDE
223 &HDF
224 &HE0
225 &HE1
226 &HE2
227 &HE3
228 &HE4
229 &HE5
230 &HE6
231 &HE7
232 &HE8
233 &HE9
234 &HEA
235 &HEB
236 &HEC
237 &HED
238 &HEE
239 &HEF
240 &HF0
241 &HF1
242 &HF2
243 &HF3
244 &HF4
245 &HF5
246 &HF6
247 &HF7
248 &HF8
249 &HF9
250 &HFA
251 &HFB
252 &HFC
253 &HFD
254 &HFE
255 &HFF

ALTERNATIEVE TEKENS

Behalve de tekens van de vorige twee pagina's kent MSX BASIC nog 31 alternatieve tekens met de codes 65...95. In een programma kunt u deze tekens oproepen door gebruik te maken van de instructie CHR$(1). Om het gezichtje af te drukken neemt u PRINT CHR$(1); CHR$(65).

CHR$(1) moet worden herhaald voor elk alternatief teken dat u wilt oproepen. Hebt u veel alternatieve tekens nodig, bijvoorbeeld om een kaderlijn te tekenen (codes 81 t/m 91), maak dan gebruik van de STRING$-functie.



(bron: gebruiksaanwijzing voor de PHILIPS MSX-thuiscomputer VG 8020)

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

Appendix F

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

Index

ACADEMIC SERVICE INFORMATICA UITGAVEN

AUTOMATISERING EN COMPUTERS

Computers en onze samenleving - M.A. Arbib
Computers in de negentiger jaren - G.L. Simons
De informatiemaatschappij - Jan Everink
Basiskennis informatieverwerking - Jan Everink
AIV, Automatisering van de informatieverzorging - Th.J.G. Derksen en H.W. Crins
Organisatie, informatie en computers - D.M. Kroenke
De Viewdata revolutie - S. Fedida en R. Malik

MICROCOMPUTERS

Microcomputers thuis en op school - K.P. Goldberg en R.D. Sherwood
Bouw zelf een Expertsysteem in BASIC - C. Naylor
Programmeercursus Microsoft BASIC - Nok van Veen
Werken met bestanden in BASIC - L. Finkel en J.R. Brown
40 Grafische programma's voor de Commodore 64 - M. Sutter
Doe-het-zelf programma's op de Commodore 64 - D. Kreutner
Programmeercursus BASIC op de Commodore 64 - Nok van Veen
TRS-80 BASIC - Bob Albrecht e.a.
TRS-80 BASIC voor gevorderden - Don Inman e.a.
Exidy sorcerer en BASIC - Nok van Veen e.a.
40 Grafische programma's voor de Electron en BBC - M. Sutter
Het Electron en BBC Micro boek - Jim McGregor en Alan Watt
Ontdek de ZX-Spectrum - Tim Hartnell
Werken met bestanden op de Apple - L. Finkel en J.R. Brown
Programmeercursus Applesoft BASIC - Nok van Veen
40 Grafische programma's voor de Apple II,IIe,IIc - M. Sutter
40 Grafische programma's in MSX BASIC - M. Sutter
Programmeercursus MSX BASIC - Nok van Veen

MICROPROCESSORS EN ASSEMBLEERTALEN

Procescomputers, basisbegrippen - dr.ir. J.E. Rooda en ir. W.C. Boot
Cursus Z-80 assembleertaal - Roger Hutty
6502 Assembleertaal en machinecode voor beginners - A.P. Stephenson

BESTURINGSSYSTEMEN

Inleiding besturingssystemen - A.M. Lister
Systeemprogrammatuur en software-ontwikkeling voor microcomputers - E. Verhulst
Bedrijfssystemen - EIT-serie, deel 4
CP/M het operating system voor microcomputers - J.N. Fernandez en R. Ashley
CP/M 86 - Nok van Veen
CP/M voor gevorderden - A. Clarke e.a.
PC DOS, het besturingssysteem van de IBM PC - R. Ashley en J.N. Fernandez
MS/DOS, het besturingssysteem voor 16 bit microcomputers - R. Ashley en J.N. Fernandez
UNIX, het standaard operating system - G.J.M. Austen en H.J. Thomassen
Werken met UNIX - Brian W. Kernighan en Rob Pike

PERSONAL COMPUTERS

Het werken met bestanden op de IBM PC - L. Finkel en J.R. Brown
De IBM PC en zijn toepassingen - Laurence Press
40 Grafische programma's voor de IBM PC - M. Sutter
Werken met VisiCalc - C. Klitzner en M.J. Plociak
Multiplan, een hulpmiddel bij de bedrijfsvoering - D.F. Cobb e.a.
Multiplan diskettes
Werken met Lotus 1-2-3 - D. Cobb en G. LeBlond

PROGRAMMEREN

Een methode van programmeren - prof.dr. Edsger W. Dijkstra en ir. W.H.J. Feijen
Programmeren, het ontwerpen van algoritmen (met Pascal) - ir. J.J. van Amstel
Inleiding tot het programmeren, deel 1 - ir. J.J. van Amstel
Inleiding tot het programmeren, deel 2 - ir. J.J. van Amstel
Programmeren, deel 2: van analyse tot algoritme - prof.drs. C. Bron
Inleiding programmeren en programmeertechnieken - EIT-serie, deel 1
Het Groot Pascal Spreuken Boek - H.F. Ledgard e.a.
JSP - Jackson Struktureel Programmeren - Henk Jansen
JSP Uitwerkingenboek - Henk Jansen

PROGRAMMEERTALEN

Aspecten van programmeertalen - ir. J.J. van Amstel en ir. J.A.A.M. Poirters
Programmeertalen, een inleiding - ir. J.J. van Amstel e.a.
BASIC - EIT-serie, deel 3
Cursus BASIC, een practicum-handleiding voor BASIC op de PRIME - ir. R. Bloothoofd e.a.
Cursus Pascal - prof.dr. A. van der Sluis en drs. C.A.C. Görts
Cursus eenvoudig Pascal - prof.dr. A. van der Sluis en drs. C.A.C. Görts
Inleiding programmeren in Pascal - C. van de Wijgaart
Systeemontwikkeling met Ada - Grady Booch
Cursus COBOL - A. Parkin
Cursus FORTRAN 77 - J.N.P. Hume en R.C. Holt
Aanvulling cursus FORTRAN 77 voor PRIME-computers - ing. J.M. den Haan
De programmeertaal C - ir. L. Ammeraal
Flitsend Forth - Alan Winfield
Programmeren in LISP - prof.dr. L.L. Steels

GEGEVENSSTRUCTUREN EN BESTANDSORGANISATIE

Informatiestructuren, bestandsorganisatie en bestandsontwerp - EIT-serie, deel 5
Programmeren, het ontwerpen van datastructuren en algoritmen - ir. J.J. van Amstel e.a.
Bestandsorganisatie - prof.dr. R.J. Lunbeck en drs. F. Remmen

DATABASE EN GEGEVENSANALYSE

Database, een inleiding - C.J. Date
Databases - drs. F. Remmen
Gegevensanalyse - R.P. Langerhorst

INFORMATIE-ANALYSE EN SYSTEEMONTWERP

Effectieve toepassingen van computers - M. Peltu
Voorbereiding van computertoepassingen - prof.dr. A.B. Frielink
Systeemontwikkeling volgens SDM - H.B. Eilers
Samenvatting SDM - Pandata
Informatie-analyse volgens NIAM - J.J.V.R. Wintraecken
Evaluation of methods and techniques for the analysis, design and implementation of information systems - ed. J. Blank en M.J. Krijger
Inleiding systeemanalyse, systeemontwerp - W.S. Davis
Systeemontwikkeling Zonder Zorgen - Paul T. Ward
Het ontwerpen van interactieve toepassingen en computernetwerken - J.A. Scheltens
EDP Audit - prof.dr. C. de Backer
Prototyping, een instrument voor systeemontwerpers - ed. T. Hoenderkamp en H.G. Sol
Simulatie, een moderne methode van onderzoek - drs. S.K. Boersma en ir. T. Hoenderkamp

EXPERT SYSTEMEN EN KUNSTMATIGE INTELLIGENTIE

Computerschaak - H.J. van den Herik
Expert systemen - Henk de Swaan Arons en Peter van Lith

THEORETISCHE INFORMATICA EN SYSTEEMPROGRAMMATUUR

Informatica, een theoretische inleiding - dr. L.P.J. Groenewegen en prof.dr. A. Ollongren
Systeemprogrammatuur - drs. H. Alblas
Vertalerbouw - H. Alblas e.a.

AANVERWANTE ONDERWERPEN EN OVERIGE TITELS

Lineaire programmering als hulpmiddel bij de besluitvorming - prof.dr. S.W. Douma
Inleiding programmeren - prof.dr. R.J. Lunbeck
Analyse van informatiebehoeften en de inhoudsbeschrijving van een databank - prof.dr. P.G. Bosch en ir. H.M. Heemskerk
Gegevensstructuren - R. Engmann e.a.
Cases en Uitwerkingenboek bij Cases - prof.dr. P.G. Bosch en H.A. te Rijdt
De tekstmachine - dr. M. Boot en drs. H. Koppelaar
Abstracte automaten en grammatica's - prof.dr. A. Ollongren en ir. Th.P. van der Weide
Onderneming en overheid in systeem-dynamisch perspectief - red. A.F.G. Hanken e.a.
Simulatie en sociale systemen - red. J.L.A. Geurts en J.H.L. Oud
Struktuur en stijl in COBOL - ir. E. Dürr en dr.ir. F. Mulder
Cursus ALGOL 60 - prof.dr. A. van der Sluis en drs. C.A.C. Görts

INFORMATIE OVER DEZE PUBLIKATIES BIJ:

Academic Service, Postbus 96996, 2509 JJ Den Haag, tel. 070-247238

Over de inhoud van dit boek

Dit boek is zowel een programmeercursus als een cursus MSX BASIC. De rode draad in het boek is het netjes leren programmeren. Tijdens dit leerproces worden bijna alle mogelijkheden en opdrachten van MSX BASIC behandeld. MSX BASIC staat in dit boek uitdrukkelijk in dienst van het leren programmeren en niet andersom.

Zo worden bijvoorbeeld bij het behandelen van herhalingsstrukturen een aantal MSX BASIC opdrachten geïntroduceerd. Daarnaast wordt een aantal programma's gegeven waarin een groot aantal verschillende programmastrukturen en MSX BASIC opdrachten verwerkt zijn. Een van deze programma's is een programma voor het op het scherm ontwerpen en veranderen van sprites.

Verder een aantal educatieve toepassingen, zoals een simulatieprogramma met graphics en geluid, een programma voor het tekenen van een 'Mondriaan', een programma voor het berekenen en tekenen van een regressielijn en nog veel meer.

Een hoofdstuk is gewijd aan het werken met bestanden op cassetteband en op diskette. Ook Disk BASIC krijgt hierin de nodige aandacht.

Ook vindt de lezer in dit hoofdstuk aanwijzingen en voorbeelden voor het programmeren van een foutafhandelingsroutine. Door de tekst heen staat een aantal opdrachten. Bijna elk hoofdstuk wordt besloten met een aantal oefenopgaven. De antwoorden van de opdrachten en opgaven zijn in een Appendix opgenomen.

De opzet van het boek is het ontwerpen en coderen van goed gestructureerde programma's, ongeacht of het programma's betreft voor administratieve, educatieve of recreatieve doeleinden.
Het boek is ook te gebruiken voor MSX-2-computers.

ISBN 90 6233 173 4