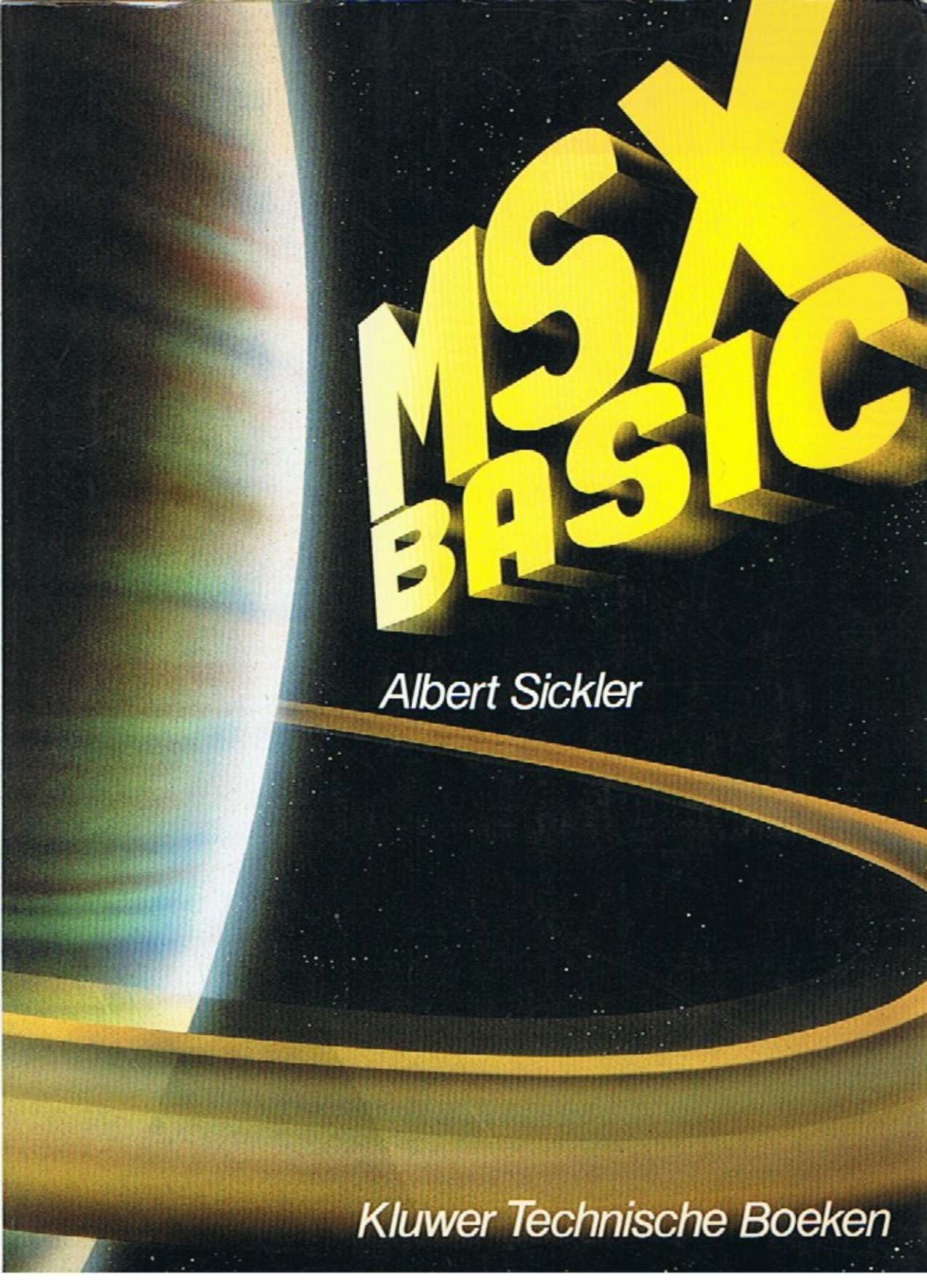

MSX
BASIC
Albert Sickler
Kluwer Technische Boeken

Erratum MSX BASIC van Albert Sickler

Op blz. 61: A=3 moet zijn: 10 A=12

Op blz. 82: 50 P(J)=J moet zijn: 50 P(AP)=J

Op blz. 96: aan het programma toevoegen:

45 INPUT K

65 INPUT K

Op blz. 98: aan het programma toevoegen:

80 PRINT "NEDERLANDS"

Op blz. 107: in programma: TEN in regel 430 moet THEN zijn.

Op blz. 113: 20 INPUT "X";Y moet zijn 20 INPUT "Y";Y

Op blz. 116: 50 PSET (X,Y) moet zijn 50 PSET (J,J)

Op blz. 121: 60 LINE -(X,4) moet zijn 60 LINE -(X,Y)

Op blz. 138: 40 C$=L8GA... moet zijn 40 C$="L8GA...

Appendix D betreft ook het gedeelte dat begint met PRINT op blz. 172 t/m WIDTH op blz. 178. De overige bladzijden (t/m 183) behoren tot de BASIC-functies.

Albert Sickler

MSX BASIC

Kluwer Technische Boeken b.v.
Deventer – Antwerpen

Omslag: Wim Niessink

ISBN 90 201 1819 6
D/1985/0108/165

1e druk 1985

Inhoud

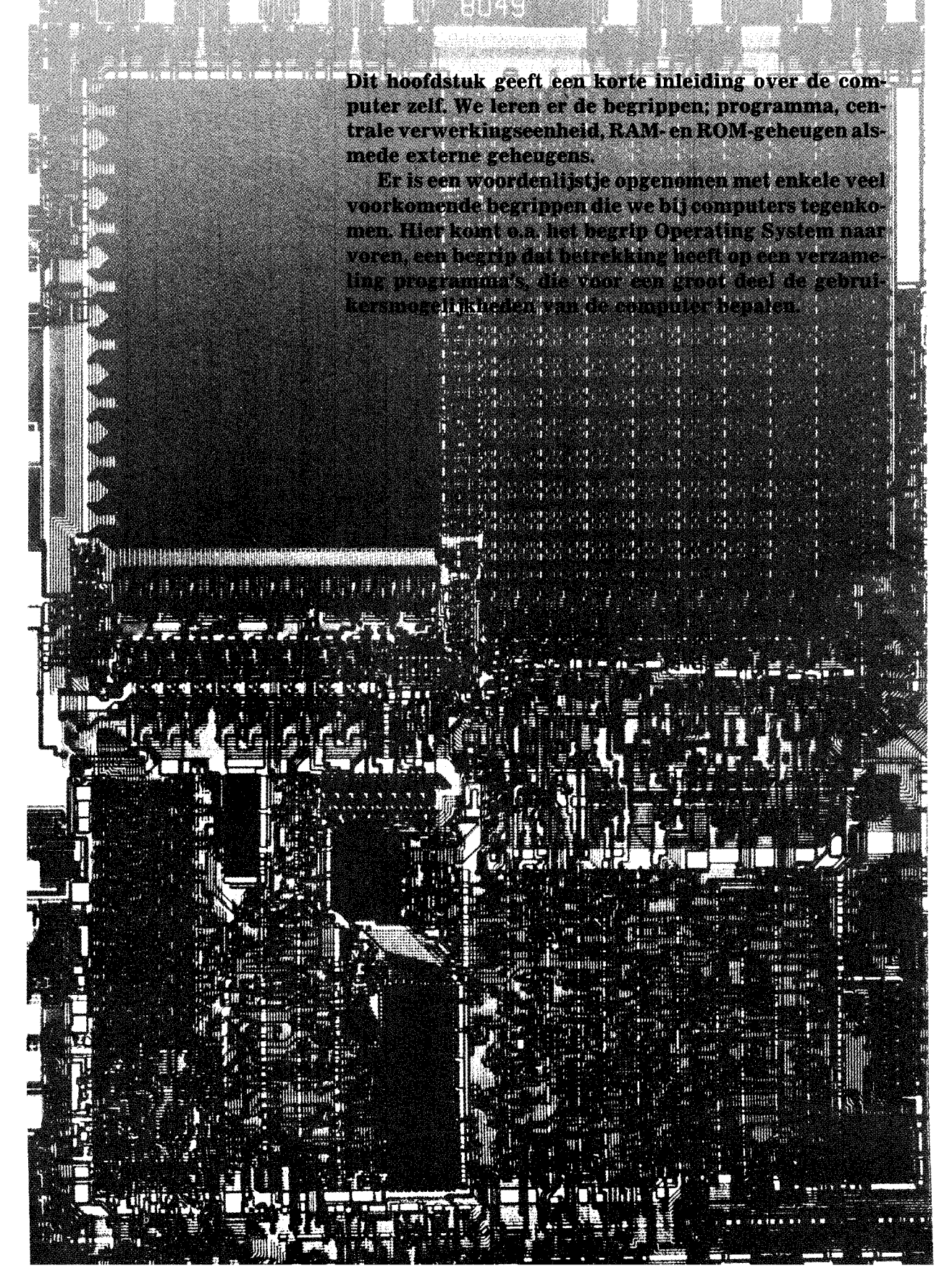
Woord vooraf

Achter de drie letters MSX gaat een hele wereld schuil. Een wereld die bepaald wordt door een grote hoeveelheid verschillende computermerken, die merkwaardigerwijze eerder opvallen door hun overeenkomst dan door hun verschil. Voor de eerste keer wordt in de wereld van de huiscomputers een *standaard* geïntroduceerd en de gevolgen zullen zonder twijfel enorm zijn.

Met de introductie van MSX-computers voor het grote publiek, zal ongetwijfeld een grote interesse ontstaan rond wat nu wel precies MSX-computers zijn en wat men met MSX-BASIC kan doen.

Voor velen zal het een eerste kennismaking zijn, een kennismaking waarbij de lezer stellig begrip hoopt te ontmoeten voor de drempels die hij moet nemen. Voor deze lezers is dit boek geschreven

Albert Sickler



Dit hoofdstuk geeft een korte inleiding over de computer zelf. We leren er de begrippen; programma, centrale verwerkingseenheid, RAM- en ROM-geheugen alsmede externe geheugens.

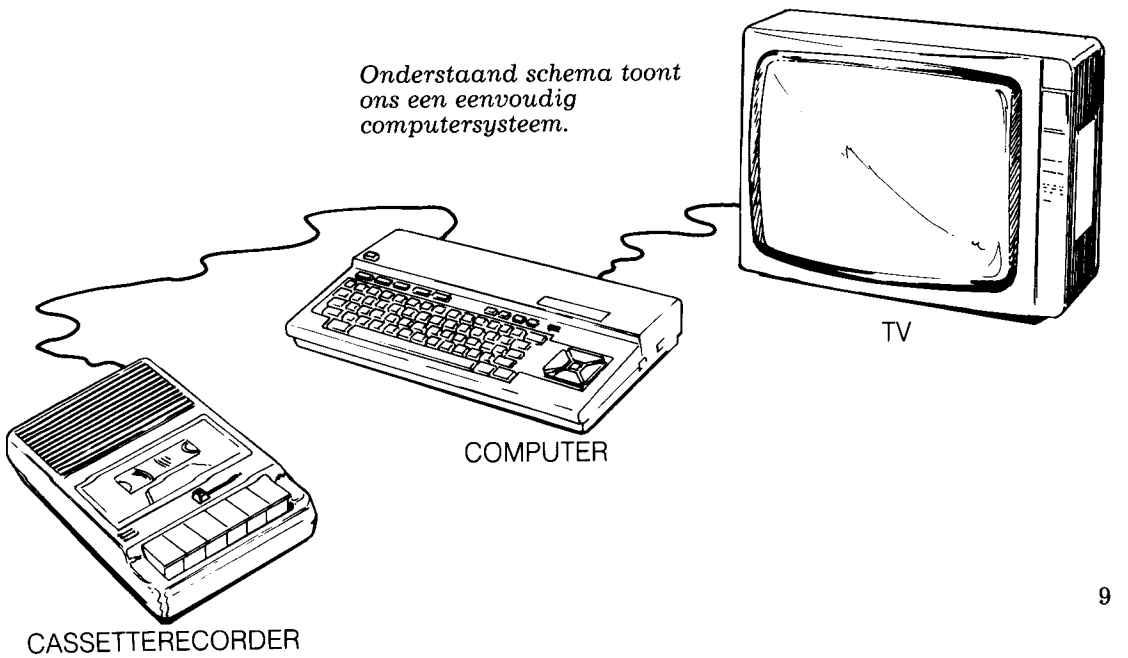
Er is een woordenlijstje opgenomen met enkele veel voorkomende begrippen die we bij computers tegenkomen. Hier komt o.a. het begrip Operating System naar voren, een begrip dat betrekking heeft op een verzameling programma's, die voor een groot deel de gebruikersmogelijkheden van de computer bepalen.

1. De computer: een inleiding

Inleiding Alhoewel dit boek zich allereerst ten doel stelt de beginner te laten zien hoe we met MSX-BASIC kunnen werken, is het verstandig eerst iets over de computer zelf te vertellen.

Tenslotte is MSX-BASIC niets anders dan een middel om de computer te vertellen wat er gedaan moet worden, daarom is het ook belangrijk om te weten wat nu wel een computer is!
De computer komt op het eerste gezicht over als een kast, waarop zich een toetsenbord bevindt. Als we een computer met enig ander bekend apparaat vergelijken, dan valt inderdaad de overeenkomst met een schrijfmachine op: ook bij de schrijfmachine overheerst het toetsenbord. Welnu, het verschil met die schrijfmachine is dat de computer is uitgerust met een groot aantal elektronische schakelingen, die nodig zijn voor het 'bewerken van informatie'. Wat dat bewerken van informatie nu wel precies inhoudt, zullen we straks zien.

Onderstaand schema toont ons een eenvoudig computersysteem.



In het voorgaande plaatje zien we dat onze computer tevens verbonden is met een TV en een cassetterecorder. Bij onze MSX-computer kunnen we inderdaad van een TV gebruikmaken. Deze dient dan om ons de gegevens te tonen die de computer produceert. Computerdeskundigen spreken in zo'n geval over een 'uitvoerapparaat'.

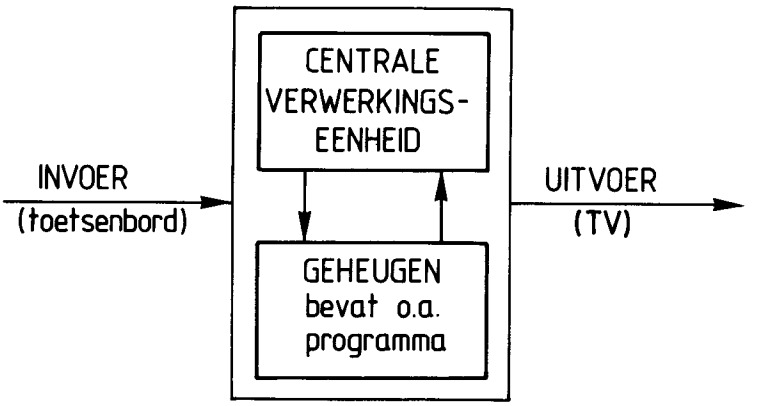
In de meest eenvoudige situatie typen we op het toetsenbord een *programma* in. Een programma bestaat uit een *reeks instructies* die de computer precies aangeeft wat deze moet doen.

Voor een beginner klinkt dat allemaal nogal moeilijk. Straks zullen we aan de hand van voorbeelden zien hoe eenvoudig alles in feite in zijn werk gaat. Een programma dat we intoetsen, wordt in een elektronisch geheugen opgeslagen. Het elektronische geheugen is één van de elektronische schakelingen die onze computer bevat.

Pas na het intoetsen van een speciaal woord zal de computer het ingetoetste programma ook werkelijk uitvoeren. Zo'n speciaal woord, waarmee we aangeven dat de computer een speciale actie moet uitvoeren, bijvoorbeeld het daadwerkelijk uitvoeren van een programma dat in het geheugen is opgeslagen, noemen we een *commando.*

De processor

Om de instructies die in het geheugen staan opgeslagen te kunnen uitvoeren, beschikt de computer over nog een bijzondere elektronische schakeling, deze noemen we de *centrale verwerkingseenheid.*
Het volgende schema verduidelijkt een en ander:



De centrale verwerkingseenheid staat in voortdurend contact met het geheugen. Het geheugen bevat o.a. de instructies die moeten worden uitgevoerd. Bovendien worden, bij het uitvoeren van berekeningen, ook tussenresultaten in het geheugen opgeslagen. Je kunt hierbij het beste denken aan een kladblok:

daar kun je ook tussenresultaten op noteren als je een ingewikkelde berekening moet uitvoeren.

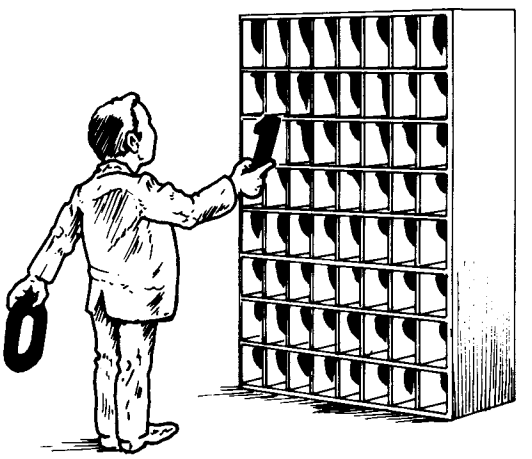
De taal die computerdeskundigen spreken is nogal sterk gekleurd door het gebruik van Amerikaanse termen. Dat hoeft ons niet te verbazen, want veel ontwikkelingen op het gebied van de computer hebben juist in Amerika plaatsgevonden.

Eén van die termen die men misschien wel eens gehoord heeft, is '*processor*'. Welnu, 'processor' is het Engelse woord voor 'verwerkingseenheid'. Als we het dus over de processor van een computer hebben, praten we over de elektronische schakelingen waarin de feitelijke bewerkingen plaatsvinden. Bij onze MSX-computer is deze processor een zeer complexe schakeling, die als één geheel op een plakje silicium is ondergebracht. Zo'n complexe schakeling op een plakje silicium noemen we een chip. Onze processor in chipvorm draagt zelfs een bijzondere naam; we spreken namelijk van een micro-processor... uiteraard omdat de schakeling zo bijzonder klein is.

Er zijn slechts een beperkt aantal fabrikanten die micro-processoren maken. Een van de bekendste typen microprocessoren is het type dat met de wat science fiction-achtige naam Z80 wordt aangeduid. Het is deze micro-processor waar alle MSX-computers zich van bedienen.

Het geheugen: RAM en ROM

Om de gegevens 'netjes' te kunnen opbergen, is het elektronische geheugen opgezet, alsof het om een grote ladenkast gaat.



Iedere lade is onderverdeeld in een serie vakjes. Bij de meeste huiscomputers, ook bij onze MSX-computer, maken we gebruik van een indeling in 8 vakjes; dus:

In één vakje kunnen we slechts een 1 of een 0 opslaan. We zeggen dan 'ieder vakje heeft een geheugencapaciteit van 1 bit'. Vaak zie je ook het woord byte; een byte komt overeen met een lade van 8 vakjes, kortom met een geheugencapaciteit van 8 bits.

Overigens, computerdeskundigen spreken doorgaans niet van laden maar van *woorden*. Als een woord dus 8 vakjes bevat, heeft het een opslagcapaciteit van 8 bits, dus van 1 byte. Computergeheugens bevatten duizenden van die woorden. Net zoals we 1000 meter afkorten met 1 km, spreken we bij een geheugen van bijvoorbeeld 16000 bytes van 16 Kbyte.

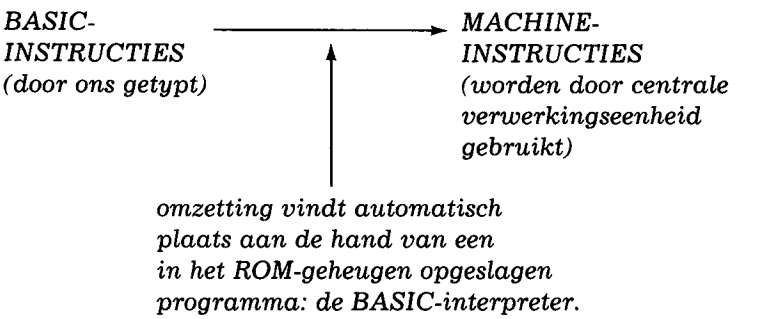
MSX-computers hebben in ieder geval een geheugencapaciteit van 8 Kbyte. Let wel, dat is een minimum, hetgeen inhoudt dat MSX-computers in de praktijk veel meer geheugencapaciteit hebben.

De geheugenschakeling van onze computer is zodanig gemaakt dat we in iedere willekeurige lade zowel gegevens in als uit kunnen lezen. We spreken in zo'n geval van een 'Willekeurig Toegankelijk Geheugen' (Engels: Random Access Memory of afgekort RAM).

Behalve het RAM-geheugen treffen we in de computer ook nog ROM-geheugen aan. Hier heeft de fabrikant al kant-en-klare programma's in opgeslagen. De term ROM slaat op Read Only Memory, hetgeen zoveel betekent als 'Lees Alleen Geheugen'.

Zo bevat het ROM-geheugen o.a. een programma dat de BASIC-instructies, die we straks gaan gebruiken, omzet in instructies waarvan de centrale verwerkingseenheid zich bedient, de 'machine-instructies'.

Dit in ROM opgeslagen programma draagt dan de naam BASIC-interpreter. In schema:



MSX-computers bevatten standaard een ROM-geheugen ter grootte van 32 Kbyte.

Alles met enen en nullen!

We komen nog even terug op het geheugen, waar kennelijk alleen maar enen en nullen in kunnen worden opgeslagen. Niet zo best... zul je zeggen als je naar het toetsenbord kijkt. Op het toetsenbord komen immers allerlei tekens voor en niet alleen de 1 en de 0. Het geheim dat achter dit alles schuilt, is dat alle letters en cijfers die we in het geheugen opslaan, worden *gecodeerd* met enen en nullen. Dat het mogelijk is letters in enen en nullen aan te geven, toont bijvoorbeeld al het Morse-alfabet.

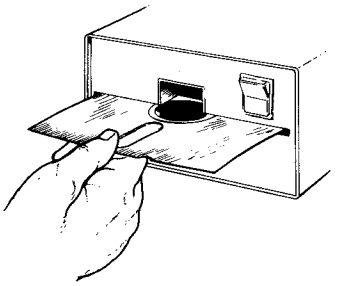
Bedenk dat het toetsenbord is gekoppeld aan een ingewikkelde elektronische schakeling die iedere toets die we indrukken onmiddellijk omzet in zo'n karakteristieke reeks enen en nullen.

Ook de uitvoer van de computer bestaat in principe uit reeksen enen en nullen. Gelukkig zijn er dan weer elektronische schakelingen, die deze reeksen in herkenbare letters en/of andere tekens op de TV omzetten.

Externe geheugens

Op de eerste tekening zien we bovendien nog een bandrecorder. Bij MSX-computers kunnen we inderdaad gebruik maken van een gewone, dan wel een aangepaste cassetterecorder. Voor zo'n cassetterecorder kan men dan cassettes kopen, waarop reeds programma's zijn vastgelegd, net zoals we muziek op cassettes kunnen verkrijgen. Met behulp van een speciaal commando kunnen we de computer opdracht geven het op cassette geregistreerde programma 'in te lezen', dat wil zeggen dat het in het geheugen van de computer wordt opgeslagen. Pas als het in het geheugen is opgeslagen, kan het ook werkelijk worden uitgevoerd.

Bovendien is het mogelijk programma's die men zelf ontwikkelt op cassette te registreren.



Fanatieke programmeurs beschikken zo over een uitgebreide collectie cassettes, waarop hun creatieve inzet is vastgelegd.

Bij MSX-computers kan men ook gebruik maken van zgn. floppy disks. De functie hiervan is geheel en al vergelijkbaar met de cassette. Op een floppy kunnen we echter veel sneller gegevens (bijv. een programma) door de computer laten opzoeken dan bij een cassetterecorder mogelijk is.

klein woordenboekje

In het nu volgende worden nog een aantal veel gebruikte termen uitgelegd.

hardware:

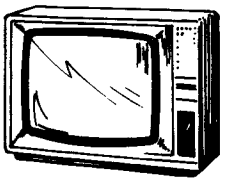
onder hardware verstaat men alles wat bij een computer tastbaar is of, populair gezegd, kapot kan gaan. Kast, elektronische schakelingen e.d. behoren tot de hardware.

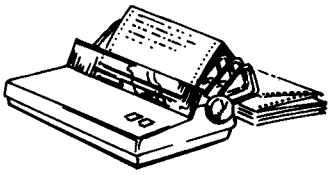
software: deze term wordt algemeen gebruikt als we het over 'programma's' hebben.

microprocessor: de centrale verwerkingseenheid bestaat uit een chip (complexe schakeling op een plakje silicium uitgevoerd). Er is slechts een beperkt aantal typen microprocessoren, zoals de Z80, de 6502 en de 6809. (Vergelijk deze namen met merknamen van auto's.) De machine-instructies van de genoemde microprocessoren verschillen onderling.
MSX-computers maken steeds gebruik van de Z80-microprocessor.

invoerapparaten: alle aan de computer aangesloten apparaten om gegevens in te voeren.

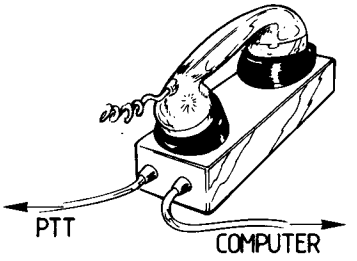
uitvoerapparaten: alle aan de computer aangesloten apparaten om gegevens uit te voeren. Bekende voorbeelden zijn: TV en monitor. Een monitor is een beeldscherm dat scherpere beelden levert dan een gewone TV.





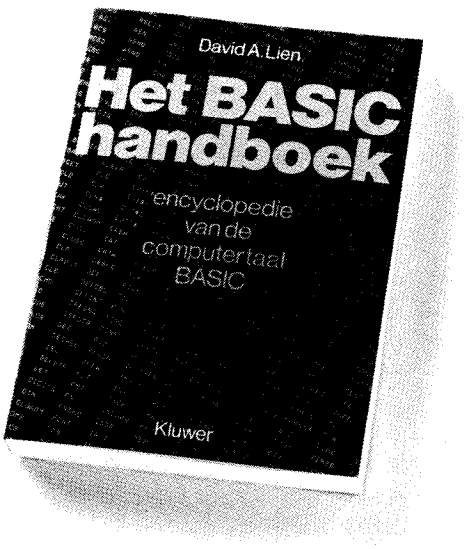
Printer: een printer is een afdrukapparaat om gegevens op papier vast te leggen. Het bekendste type is de matrixprinter, waarbij ieder symbool als een serie puntjes wordt weergegeven.

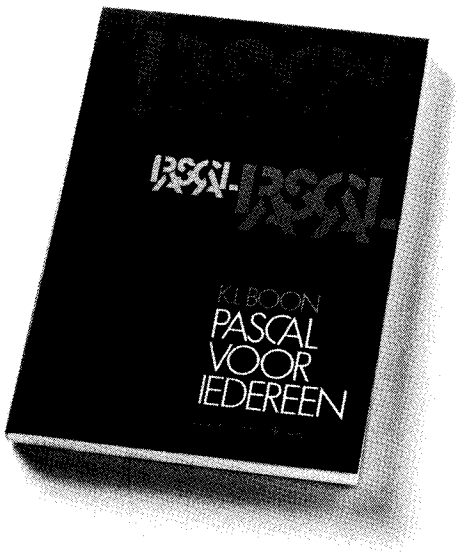
modem: een modem is een apparaat om de computer via het telefoonnet aan een andere computer te koppelen. Zo kunnen we eenvoudig een programma van de ene naar de andere computer overzenden.



hogere programmeertalen: een hogere programmeertaal is een taal die zich geheel en al richt op de feitelijke bewerkingen die de programmeur wil uitvoeren.
De term 'hogere' schrikt wellicht velen af, omdat hierbij snel het idee ontstaat dat het net zoals 'hogere wiskunde' om iets moeilijks zou gaan. Het tegendeel is waar: het programmeren met behulp van zo'n hogere programmeertaal is juist relatief eenvoudig.

Voorbeelden van hogere talen zijn BASIC en PASCAL.

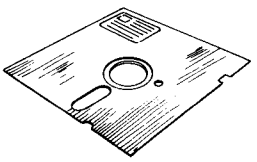




Bedenk dat iedere instructie van zo'n hogere taal door de computer eerst automatisch wordt omgezet in machine-instructies; de enige verteerbare instructies voor de centrale verwerkingseenheid.

Operating System: een Operating System is een verzameling programma's die de computer voor het uitvoeren van veel zaken nodig heeft.

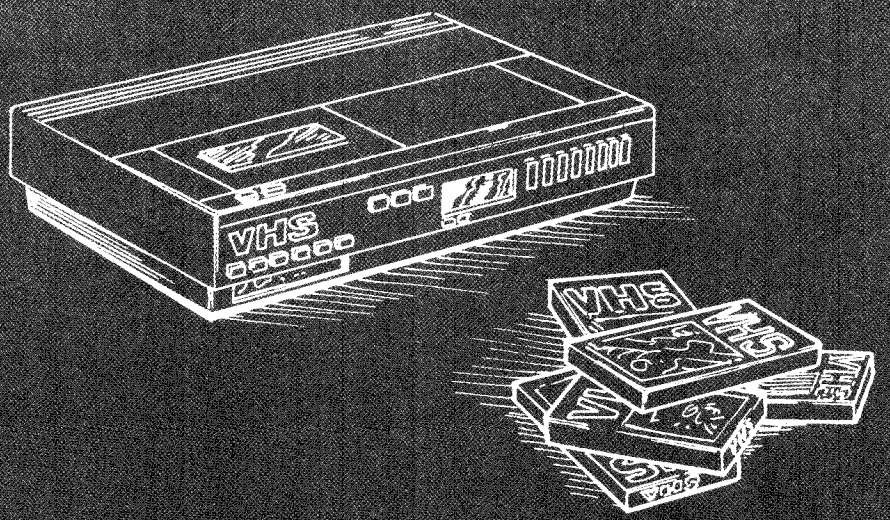
Denk maar eens aan:
- het inlezen van programma's van floppy disk of cassette;
- het afdrukken op een printer.



In feite worden deze acties steeds uitgevoerd onder besturing van deelprogramma's van het Operating System. Het Operating System voor MSX-computers die met floppy disks werken, draagt de naam MSX-DOS. De term DOS komt trouwens van *D*isc *O*perating *S*ystem.

De letters MSX geven een standaard aan. In dit hoofdstuk wordt allereerst uiteengezet wat we onder het woord standaard dienen te verstaan. Hierbij maken we een vergelijking met de standaards van video-recorders. Hierna wordt uiteengezet waar de standaard MSX betrekking op heeft.

Ten slotte wagen we ons aan een bespiegeling over de mogelijke gevolgen van een standaard.



2. MSX: een standaard

Inleiding

De titel van dit hoofdstuk luidt 'MSX: een standaard'. Wat willen we nu precies met 'een standaard' zeggen?

Welnu, de term standaard geeft aan dat verschillende onafhankelijke computerfabrikanten hun computers zodanig zullen bouwen dat deze, wat betreft een aantal zaken, volstrekt gelijk zijn.

Vergelijk de toestand maar eens met videorecorders. Ook daar zijn standaards ingevoerd. Het bekendste (meest gebruikte systeem) is VHS. Dit houdt in dat op alle video-recorders waarop VHS staat video-tapes afgespeeld kunnen worden waarop VHS staat (ongeacht het merk).

Welnu, hetzelfde geldt voor MSX-computers. Op iedere computer waar op het label staat

kunnen we programma's verwerken (bijv. aangeboden op cassette of floppy disk) waarop hetzelfde label staat.

Let wel, standaard wil hier beslist niet zeggen dat al die computers volstrekt gelijk zijn. Ook bij videorecorders treffen we verschillen aan. Zo biedt het ene merk de mogelijkheid om beelden stil te zetten en het andere merk niet. Zo zullen we ook zien dat de ene MSX-computer-fabrikant zijn computer met een aantal extra's uitrust die we bij een andere MSX-computer niet aantreffen. Voorbeelden van die extra's zijn:

- speciale programma's die worden aangeboden;
- speciale koppelingsmogelijkheden, bijv. met een laser-disk;
- speciale mogelijkheden met betrekking tot het spelen van muziek.

Na het vorige hoofdstuk kunnen we nu iets eenvoudiger onder woorden brengen wat nu wel precies de MSX standaard is.

De standaard De standaard geeft aan dat MSX-computers in ieder geval zijn uitgerust met de volgende zaken:

- Een Z80-microprocessor. In het voorafgaande hebben we reeds aangegeven dat dit het onderdeel is, waarmee de feitelijke bewerkingen worden uitgevoerd.

- Een Texas Instruments Video Chip (TMS 9918A). Wederom gaat het hier om een chip, dat wil zeggen een complexe elektronische schakeling die als één geheel op een plakje silicium is gefabriceerd. Deze chip bepaalt welke mogelijkheden we uiteindelijk hebben om plaatjes op het beeldscherm te krijgen. Deze mogelijkheden zijn dus voor iedere MSX-computer gelijk.

- Een General Instruments audiochip (AY-3-8910). Wederom een chip, in dit geval een chip die alle geluidsmogelijkheden van onze computer bepaalt. MSX-computers kunnen dankzij deze chip zeer fraaie geluiden over een bereik van 8 octaven produceren en wel met drie 'stemmen' tegelijkertijd. Dit laatste houdt in dat we 3 tonen tegelijkertijd kunnen produceren.

- 32 Kbyte ROM-geheugen. Hierin bevindt zich o.a. het programma dat onze BASIC-instructies vertaalt in de feitelijke instructies die de computer uitvoert; de machine-instructies. Aangezien dit programma voor een deel is ontwikkeld door het bekende Amerikaanse softwarebureau Microsoft spreken we vaak van Microsoft-BASIC.

- Een scherm, waarop standaard 40 tekens naast elkaar kunnen worden afgebeeld. Men duidt dit wel eens aan met de term '40-koloms-display'. Voor het weergeven van figuren is het scherm ingedeeld in 256 × 192 beeldpunten.

- De mogelijkheid om bij het opbouwen van figuurtjes enz. tot maximaal 16 verschillende kleuren te gebruiken.

- Standaard-koppeling voor aansluiting van een cassetterecorder.

- Standaard-mogelijkheid voor aansluiting van een floppy disk.

- Standaard-mogelijkheid om zgn. cartridges in de MSX-computer te schuiven. Cartridges zijn doosjes die in feite ROM-geheugens bevatten, waarin 'kant-en-klaar-programma's' zijn ondergebracht. Door het plaatsen van zo'n cartridge beschikt men meteen over zo'n programma. Er zijn bijzonder veel cartridges met spelletjes-programma's maar men treft ook car-

tridges voor meer serieuze toepassingen, zoals educatieve programma's, aan.

- Standaard-aansluiting voor 2 joy-sticks.

De gevolgen van een standaard

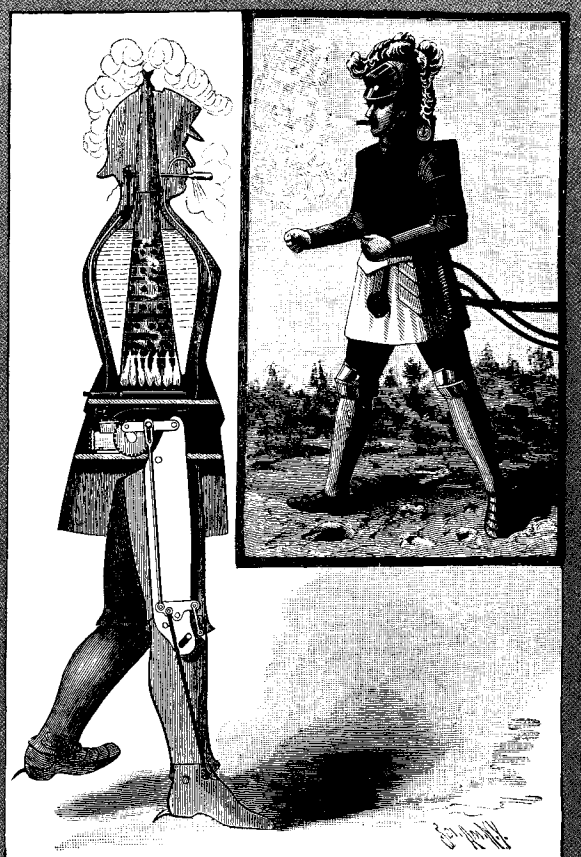
MSX-computers worden door veel onafhankelijke firma's aangeboden. Veel Japanse en Koreaanse firma's brengen MSX-computers op de markt, maar ook elektronicagiganten zoals Philips en Siemens doen mee. De gevolgen zullen naar onze stellige overtuiging enorm zijn. Dit is de eerste maal dat een standaard (zowel op hardware- als op softwaregebied) voor huiscomputers wordt ingevoerd. Bedenk dat het woord standaard hier niet alleen op BASIC slaat. Veel belangrijker nog is dat het totale concept is vastgelegd! Zo zullen softwarefirma's geweldige hoeveelheden software ontwikkelen, waaronder het reeds genoemde PASCAL.

Door het vastleggen van de standaard zal men ook zien dat nieuwe chips ontwikkeld worden die een *aantal* oude chips vervangen. Zo zullen goedkope MSX-computers op de markt verschijnen. Ogenschijnlijk zou zo iets voor veel fabrikanten dodelijk kunnen zijn. Bedenk echter weer dat men naast de standaard vrij is om zijn eigen machine met allerlei zaken op te tuigen.

De ontwikkelingen zullen inderdaad enorm zijn. Straks zal men zelfs zien dat MSX-chips standaard in de TV worden ingebouwd. MSX luidt, naar onze stellige overtuiging, een nieuw computertijdperk in!

In dit hoofdstuk tonen we hoe de computer als calculator, dat wil zeggen al apparaat voor het uitvoeren van directe berekeningen, gebruikt kan worden. We maken hierbij steeds gebruik van de zgn. PRINT-opdracht. Hierna worden de verschillende bewerkingstekens voor rekenkundige bewerkingen getoond. Hierbij komt ook het gebruik van haakjes aan de orde. Ten slotte laten we in dit hoofdstuk zien dat we ook teksten op het beeldscherm kunnen afdrukken.

In dit hoofdstuk gaan we werkelijk de eerste stappen nemen om onze computer te leren gebruiken.



THE STEAM MAN.

3. De eerste stappen: over PRINT en rekenkundige bewerkingen

PRINT voor het uitvoeren van directe berekeningen

Als we de computer aanzetten, wordt deze onmiddellijk in de toestand gebracht waarin we met MSX-BASIC kunnen werken. Dit is stellig een van de grote voordelen van MSX-BASIC t.o.v. andere hogere programmeertalen, zoals PASCAL. Om van deze laatstgenoemde programmeertaal gebruik te maken, moeten we steeds van te voren een aantal standaard handelingen verrichten, handelingen waarop we hier niet verder in zullen gaan. Goed dan... we kunnen dus direct beginnen.
Om dit te tonen, typen we in:

```
PRINT 2+3
```

Op het scherm verschijnt nu tevens de uitdrukking PRINT 2+3. Pas als we op de RETURN-toets drukken, zal de computer op deze opdracht reageren. Deze RETURN-toets wordt op de meeste computers ook met RETURN aangegeven.

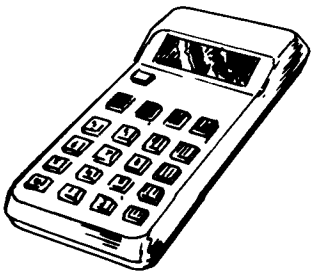
PRINT wil zeggen 'druk af' of liever 'beeld af' en direct na het indrukken van de RETURN-toets zien we ook dat iets wordt afgebeeld, namelijk:

```
5
```

en de term:

```
OK
```

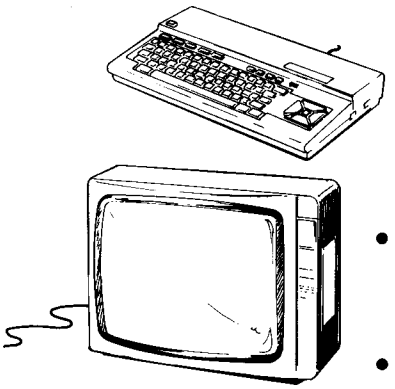
Het zal duidelijk zijn, dat 5 het antwoord is van de aangegeven optelling 2+3. De term OK geeft aan dat de computer klaar is met zijn taak en dat we dus weer een nieuwe taak kunnen opgeven. Het zal de lezer opvallen dat we de computer op deze wijze in feite als een calculator gebruiken.



Bij de calculator drukken we op:

```
2+3=
```

en hierna verschijnt in het display het antwoord: 5



Bij de computer toetsen we in:

```
PRINT 2+3
```

en na het indrukken van de RETURN-toets verschijnt het antwoord: 5

Opmerkingen:

- Bij de computer mogen we absoluut niet intypen 2+3= (en daarna bijvoorbeeld op de RETURN-toets drukken). Als we dit doen, verschijnt een mededeling – uiteraard in het Engels – dat we iets fout hebben gedaan.
- Bij de computer moet een opdracht, waarmee we onmiddellijk een resultaat op het scherm willen zien, altijd met de term PRINT beginnen.

Nog wat voorbeelden:

Vermenigvuldigen

```
PRINT 5*36
```

Na het indrukken van de RETURN-toets verschijnt:

```
180
```

Let op! Het vermenigvuldigteken wordt hier met een sterretje aangegeven, en wel om verwarring met de letter x te voorkomen.

Delen

De opdracht

```
PRINT 20/5
```

geeft als resultaat:

```
4
```

De deling wordt steeds met het teken / aangegeven en *niet* met het teken :

Haakjes

Haakjes mogen we toepassen zoals we dit op de schoolbanken hebben geleerd; zo wordt

$$\frac{5+15}{23\,(17+103)}$$

als volgt ingetoetst:

```
PRINT (5+15)/(23*(17+103))
```

Merk allereerst op dat de totale uitdrukking hier op één regel is geplaatst. Teller en noemer zijn nu tussen haakjes geplaatst. Als we dat niet hadden gedaan, zou de PRINT-instructie er als volgt uitzien:

```
PRINT 5+15/23*(17+103)
```

Deze instructie leidt echter tot een ander dan het gewenste resultaat (dit wordt pas duidelijk als we de zo dadelijk te bespreken prioriteitsregels voor berekeningen kennen).

Bovendien zien we dat in de noemer tussen 23 en 'haakjes openen' een vermenigvuldigteken is geplaatst, het gaat bij de oorspronkelijke uitdrukking van de noemer '23(17+103)' immers om een vermenigvuldiging van 23 met (17+103).

Volgorde waarin de bewerkingen worden uitgevoerd

Op de schoolbanken leerden we regels zoals 'Meneer Van Dalen Wacht Op Antwoord'. Daarmee werd aangegeven dat bijvoorbeeld Vermenigvuldigen vóór Delen gaat, of in deftiger taal; vermenigvuldigen heeft een hogere prioriteit dan delen. Bij MSX–BASIC worden de volgende prioriteitsregels in acht genomen:

1. eerst worden uitdrukkingen tussen haakjes uitgewerkt;
2. vermenigvuldigen en delen hebben gelijke prioriteit en gaan vóór optellen en aftrekken die ook gelijke prioriteit hebben;
3. bewerkingen met gelijke prioriteit worden van links naar rechts afgewerkt.

Bijvoorbeeld:

```
PRINT   15/3*5
```

wordt `5*5`

en dus `25`

Ergo, het resultaat is 25 (en niet 1!)

In Appendix F vindt men een overzicht van de volledige prioriteitsregels.

Tekst

Het werken met teksten is één van de belangrijke toepassingsgebieden van computers. Straks zullen we zien wat BASIC daartoe voor mogelijkheden biedt. Voorlopig tonen we dat we ook een gegeven tekst op het beeldscherm kunnen afbeelden.

De af te beelden tekst plaatsen we steeds tussen aanhalingstekens. Een voorbeeld:

```
PRINT "DIT IS EEN VOORBEELD"
```

geeft als resultaat:

```
DIT IS EEN VOORBEELD
```

Merk op dat de aanhalingstekens niet bij het resultaat verschijnen.

In dit hoofdstuk maken we voor het eerst kennis met een BASIC-programma. Een BASIC-programma is een serie instructies. Om aan te geven dat het om een programma gaat, hanteren we zgn. regelnummers. Deze nummers bepalen de volgorde waarin de instructies worden afgewerkt.

Daarna maken we voor de eerste keer kennis met het RUN-commando, een commando waarmee de computer wordt opgedragen het in het geheugen opgeslagen programma ook werkelijk uit te voeren. Ten slotte geven we in een nabeschouwing nog een korte uiteenzetting over verschillende programmeertalen. Kortom een hoofdstuk waarin we dieper op de software-mogelijkheden van onze computer ingaan.

De Telegraaf 4 JULI 1984

Software-industrie bundelt export

Van een onzer verslaggevers

DEN HAAG, donderdag

De Nederlandse software-industrie wil meer aan de export doen. Drie belangengroepen, die actief zijn op het gebied van computer services- en softwareprodukten en van alle daaraan gerelateerde diensten en produkten, gaan hun export-activiteiten bundelen. Zij hebben daarvoor een stichting onder de naam DECS (Dutch Exporters of Computer Services) in het leven geroepen. Aan het ministerie van Economische Zaken is een subsidie van ruim ƒ 250.000 gevraagd.

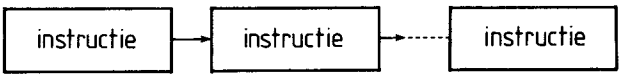
De drie belangengroepen — VIFKA (voornamelijk hard- COSSO (de grote soft- de NSV

e.d. Op grote pakketten voor b.v. financiële administratie uit ons land zit men in het buitenland niet bepaald te die zijn er volgens op

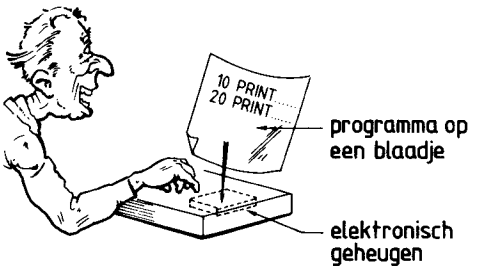
4. En nu een BASIC-programma!

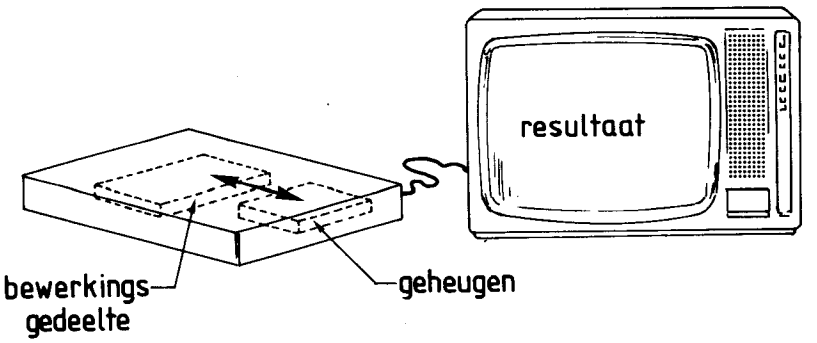
Een BASIC-programma: gebruik van regelnummers

Zoals in hoofdstuk 1 werd opgemerkt, is een programma niets anders dan een serie instructies die na een zeker start-commando automatisch door de computer wordt afgewerkt.
In schema:



In het meest eenvoudige geval kunnen we een programma, als ging het om een stuk tekst, gewoon op het toetsenbord intypen. Dit houdt in dat zo'n programma, kortom zo'n reeks instructies, eerst in het geheugen wordt opgeslagen.
In beeld:





Hoe krijgen we nu voor elkaar dat de instructies die het programma vormen eerst in het geheugen van de computer worden opgeslagen en niet onmiddellijk worden uitgevoerd? Het antwoord is eenvoudig:

door nummers voor de instructies te plaatsen.

In het nu volgende zullen we dit aan de hand van een voorbeeld illustreren.

Een voorbeeld: Stel dat we de computer als programma achtereenvolgens de volgende 3 instructies willen laten uitvoeren:

```
PRINT "DIT IS"
PRINT "HET EERSTE VOORBEELD"
PRINT "VAN EEN PROGRAMMA"
```

Dan moeten we bij het intypen van dit programma – het gaat tenslotte om een *reeks* instructies – vóór iedere instructie een nummer plaatsen.
We nummeren gewoonlijk met 10, 20, 30 enz.
Dus typen we in:

```
10 PRINT "DIT IS"
20 PRINT "HET EERSTE VOORBEELD"
30 PRINT "VAN EEN PROGRAMMA"
```

Aan het einde van iedere regel drukken we op de RETURN-toets, ten teken dat de regel volledig is ingetoetst (N.B. niet vergeten ook na de laatste regel op de RETURN-toets te drukken!).

Als zo alle regels zijn ingetoetst, is het programma in zijn geheel in het geheugen opgeslagen. We zien het trouwens ook op het scherm staan.

We gaan het programma uitvoeren! Als we nu intypen:

```
RUN    (gevolgd door RETURN-toets)
```

dan verschijnt op het scherm:

```
DIT IS
HET EERSTE VOORBEELD
VAN EEN PROGRAMMA
```

Hoera, we hebben een programma op de computer verwerkt!

In dit geval bestond het programma uit alleen maar PRINT-instructies met af te drukken tekst, maar niettemin ging het om een reeks instructies, met andere woorden om een programma.

We plaatsen hierbij de volgende opmerkingen:

- Nadat het programma is uitgevoerd – gerund, zeggen computerdeskundigen – blijft het nog steeds in het geheugen staan.

Alleen als we een zgn. geheugen-wis-commando geven, dan wel als we de spanning van de computer uitzetten, gaat het programma verloren.

- De nummers geven tevens de volgorde aan waarin de instructies worden afgewerkt en wel van het laagste tot het hoogste getal. We hadden zelfs mogen intypen:

```
10 PRINT "DIT IS"
30 PRINT "VAN EEN PROGRAMMA"
20 PRINT "HET EERSTE VOORBEELD"
```

Het resultaat zal hetzelfde zijn en wel omdat de regelnummers dat zo aangeven. De computer zal de instructies trouwens direct na het intypen in de juiste volgorde plaatsen.

- In plaats van nummers spreken we gewoonlijk van regelnummers. Als regelnummers mogen we alleen gehele getallen (van 0 tot 65529) gebruiken.
- Het feit dat we nu met 10, 20, 30 enz. nummeren, heeft het voordeel dat we naderhand nog instructies kunnen invoegen. Hoe dat precies in zijn werk gaat, wordt in hoofdstuk 5 uit de doeken gedaan.

Nabeschouwing

Dit was dan onze eerste kennismaking met een programma. We kunnen nog nauwelijks spreken van een kennismaking met het programmeren. Met dit laatste bedoelen we immers de wijze waarop we met de instructies een bepaald resultaat kunnen bereiken. Om meer inzicht te krijgen in het programmeren zelf, dienen we eerst wat meer instructies te behandelen.

Toch willen we hier een moment stilstaan bij die éne instructie die werd behandeld, omdat het een typisch voorbeeld is van een instructie van een hogere programmeertaal.
Als we de PRINT-instructie nogmaals bezien:

regelnummer PRINT 'af te drukken tekst'

dan valt op dat we er niets in terugvinden van wat in hoofdstuk 1 behandeld werd. We merken bijvoorbeeld niets van een centrale verwerkingseenheid, van ROM- en RAM-geheugen. Toch zijn al die zaken noodzakelijk voor het daadwerkelijk uitvoeren van die ene instructie. De PRINT-instructie richt zich geheel en al naar de taak die de programmeur wil uitvoeren, in dit geval het afdrukken van tekst. Dit kenmerk, het richten op de taak die de programmeur onder woorden wil brengen, is inderdaad het voornaamste kenmerk van een hogere programmeertaal. Hogere programmeertalen zijn in de loop der tijd ontstaan.

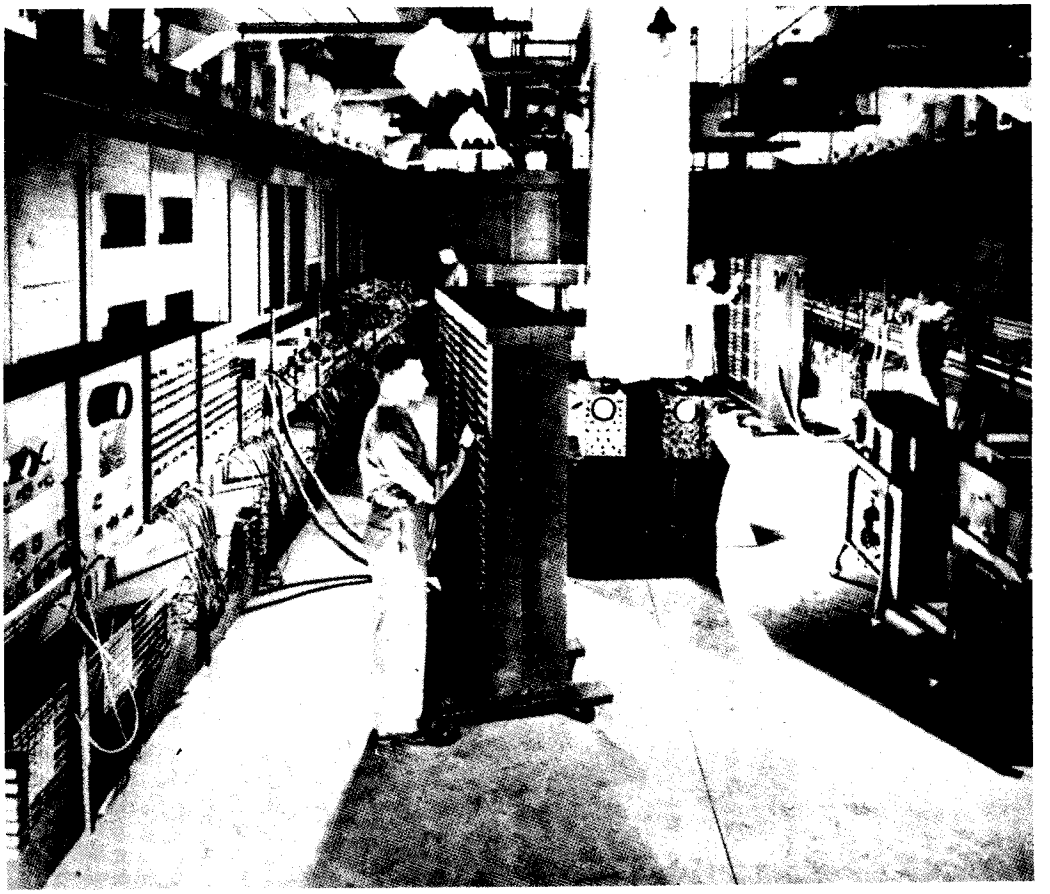
Bij de eerste computers bestond een programma nog geheel en al uit machinecodes. Deze werden dan met de hand, met behulp van schakelaars, één voor één in het geheugen geplaatst. Het zal geen verbazing wekken dat het programmeren hierbij

geheel en al bepaald wordt door de eigenschappen van de centrale verwerkingseenheid.

Vanaf het eerste moment heeft men naar eenvoudiger methoden voor het programmeren gezocht. Allereerst ontstonden toen de zgn. assembleer-talen. Dat zijn talen waarbij de machinecodes in korte, eenvoudig te onthouden, lettercodes worden aangegeven. Daarna ontstonden de eerste hogere programmeertalen, zoals FORTRAN en BASIC. Beide talen zijn trouwens nog springlevend. FORTRAN wordt vooral gebruikt op rekencentra, waar complexe wiskundige berekeningen worden uitgevoerd. BASIC heeft zich een zeer vaste plaats verworven, vooral bij de huiscomputers.

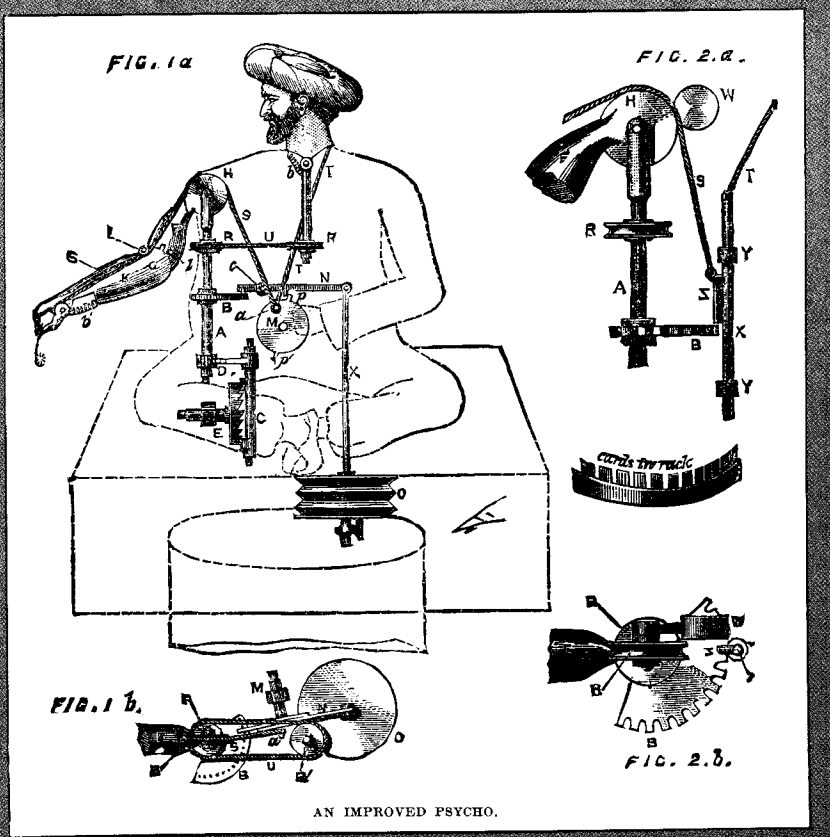
De ontwikkelingen staan uiteraard niet stil. Nieuwe programmeertalen worden voorgesteld. Verder ziet men dat er nieuwe 'talen' komen voor zeer specifieke toepassingen. Een zeer bekende toepassing is bijvoorbeeld het met behulp van de computer beheren van een grote hoeveelheid gegevens ('databestand').

De ENIAC uit 1946



Deze 'data-base-talen' komen stellig ook voor MSX-computers beschikbaar. Ten slotte willen we opmerken dat het belang van machinecodes en assembleertalen met de introductie van hogere programmeertalen geenszins is verdwenen. Uiteindelijk werkt iedere computer in laatste instantie met machinecodes. Voor een eerste kennismaking is dat echter een volstrekt onverteerbare materie, reden waarom we er niet verder op in zullen gaan.

In dit hoofdstuk worden met name een aantal commando's beschreven. RUN om een programma te laten verwerken, LIST om de instructies van het programma op het scherm te tonen, NEW om een programma te wissen. Ook bespreken we de commando's AUTO, voor het automatisch genereren van regelnummers, RENUM voor het automatisch hernummeren van regelnummers en DELETE voor het automatisch wissen van bepaalde gedeeltes van het programma. Ten slotte zien we hoe we regels kunnen toevoegen, invoegen en wijzigen. Een hoofdstuk dat nogmaals onderstreept dat de computer werkelijk een automaat is, dat wil zeggen veel handelingen zonder tussenkomst van de mens kan uitvoeren.



AN IMPROVED PSYCHO.

5. RUN, LIST, NEW, AUTO, RENUM en DELETE

Inleiding In het voorafgaande zagen we hoe we een programma konden intypen. Na het intoetsen van het RUN-commando werd het programma ook daadwerkelijk uitgevoerd. Het is op dit moment verstandig om eens stil te staan bij de vraag wat nu precies het verschil is tussen een commando en een instructie.

Welnu, een instructie vormt steeds een onderdeel van een programma, terwijl een commando een bevel is aan de computer om een bepaalde actie *direct* uit te voeren.

In schema:

instructie	commando
↓	↓
reeks instructies vormt een programma, tijdens het intypen worden de instructies in het geheugen opgeslagen	computer moet direct na het intoetsen van het commando de aangegeven actie uitvoeren

De bekendste commando's zijn RUN, LIST en NEW. We zullen deze in het nu volgende bespreken. Verder bespreken we nog een aantal commando's, die als extra beschouwd moeten worden; niet noodzakelijk maar wel plezierig dat ze geboden worden.

RUN Met het RUN-commando geven we aan dat een in het geheugen opgeslagen programma direct moet worden uitgevoerd.
Stel dat we van te voren hebben ingetoetst:

```
10 PRINT "DIT IS EEN"
20 PRINT "PROGRAMMA DAT BESTAAT"
30 PRINT "UIT 3 REGELS"
```

Dan verschijnt na het intypen vanRUN (uiteraard weer gevolgd door de RETURN-toets):

```
DIT IS EEN
PROGRAMMA DAT BESTAAT
UIT 3 REGELS
```

Men mag na RUN ook een regelnummer aangeven; vanaf de aangegeven regel zal het programma dan worden uitgevoerd. Voorbeeld:

```
RUN 20    (gevolgd door RETURN-toets)
```

geeft als resultaat:

```
PROGRAMMA DAT BESTAAT
UIT 3 REGELS
```

LIST Programmeren is een bij uitstek dynamische bezigheid. Een programma is zelden in één keer goed. We moeten instructies veranderen, invoegen, weglaten en het is vaak nodig dat we het oorspronkelijke programma weer op het scherm willen zien.

Voor dit doel biedt onze computer dan het LIST-commando. Stel dat het zojuist getoonde programma nog in het geheugen staat.
Als we nu intypen:

```
LIST    (gevolgd door RETURN-toets)
```

zal het volledige programma op het scherm verschijnen, dus we zien:

```
10 PRINT "DIT IS EEN"
20 PRINT "PROGRAMMA DAT BESTAAT"
30 PRINT "UIT 3 REGELS"
```

Door nà LIST eventueel regelnummers aan te geven, kunnen we delen van een programma oproepen:

```
LIST 20      : geeft alleen regel 20
LIST 20–30   : geeft regel 20 t/m 30
LIST –20     : geeft 'alles' tot en met regel 20
LIST 20–     : geeft het programma vanaf regel 20
```

NEW Het commande NEW is stellig het gevaarlijkste BASIC-commando dat we kennen. Met het NEW-commando geven we namelijk aan dat het in het geheugen opgeslagen programma gewist moet worden en wat eenmaal gewist is... is definitief weg!

Gebruik dit commando alléén als het werkelijk nodig is en wel;

- als het programma dat u wist werkelijk volstrekt oninteressant is;

- als u het programma al op cassettetape (of floppy disk) heeft staan;
- pas nadat u het programma, met een daartoe geschikt commando, veilig op cassette of floppy heeft opgeslagen (zie Appendix D).

AUTO

MSX-BASIC biedt vele mogelijkheden waarvoor enerzijds geldt dat ze beslist niet absoluut noodzakelijk zijn, maar waarvoor anderzijds geldt dat het wel plezierig is dat ze geboden worden.
Eén van die mogelijkheden is het AUTO-commando.
Na AUTO kan men één of twee getallen aangeven.

Bijvoorbeeld:

`AUTO 10` : in dit geval zal de computer automatisch de regelnummers 10, 20, 30, enz. produceren.

`AUTO 100, 10`: in dit geval worden 2 getallen aangeboden. Het eerste getal geeft het eerste regelnummer en het tweede met welke stapgrootte de volgende regelnummers gegenereerd worden. Bij het getoonde voorbeeld produceert de computer dus de reeks: 100, 110, 120, 130, enz.

Om ervoor te zorgen dat de computer stopt met het automatisch nummeren, drukken we met ingedrukte CTRL-toets op de toets met STOP. We geven deze laatste actie steeds met de volgende uitdrukking aan:

CTRL/STOP

Probeer als eerste oefening maar eens het zojuist gegeven programma met behulp van het AUTO-commando in te toetsen.

Het toevoegen, invoegen en hernummeren van regels en het RENUM-commando

Om de waarde van het RENUM-commando in te schatten, tonen we eerst weer een programma:

```
10 PRINT "DIT IS EEN"
20 PRINT "PROGRAMMA"
30 PRINT "TER ILLUSTRATIE"
```

Als we dit programma eenmaal hebben ingetoetst, kunnen we eenvoudig nieuwe instructies toevoegen en invoegen. Toevoegen verloopt min of meer vanzelfsprekend; kies een regelnummer dat volgt op het laatste regelnummer.
Als we bijvoorbeeld intoetsen:

```
40 PRINT "VAN HET WERKEN"
50 PRINT "MET REGELNUMMERS"
```

dan zijn deze beide regels onmiddellijk aan het voorafgaande toegevoegd. Om een regel in te voegen, kiezen we een regelnummer dat ligt tussen de regelnummers van de regels waartussen onze nieuwe regel moet komen te staan.
Als we bijvoorbeeld intoetsen:

```
15 PRINT "KORT EENVOUDIG"
```

en we geven hierna het LIST-commando, dan toont het scherm ons:

```
10 PRINT "DIT IS EEN"
15 PRINT "KORT EENVOUDIG"
20 PRINT "PROGRAMMA"
30 PRINT "TER ILLUSTRATIE"
40 PRINT "VAN HET WERKEN"
50 PRINT "MET REGELNUMMERS"
```

Regel 15 is inderdaad op de juiste plaats ingevoegd.

Overigens zien we nu ook hoe handig het was om in het begin steeds volgens de reeks 10, 20, 30 enz. te nummeren. Door deze nummering is het immers eenvoudig nieuwe regels in te voegen.

In de praktijk komt het invoegen van regels zó vaak voor, dat er dikwijls een 'ratjetoe' aan regelnummers ontstaat.

Om na veel invoegen weer een nette nummering te krijgen, kunnen we gebruik maken van het RENUM-commando.
Dit heeft o.a. de volgende mogelijkheden:

`RENUM`: in dit geval wordt na RENUM geen getal geplaatst en zal de computer alle regels weer volgens de reeks 10, 20, 30, enz. van regelnummers voorzien.

`RENUM 100`: als RENUM, maar nu wordt de reeks 100, 110, 120 enz. gebruikt. Voor RENUM 200 zal de reeks 200, 210, 220, enz. genomen worden.

`RENUM 100, 40, 10`: nu worden vanaf regel 40 alle regels opnieuw genummerd en wel volgens de reeks 100, 110, 120 enz.

We proberen het RENUM-commande en typen daartoe in;

```
RENUM
```

en via het LIST-commando zien we;

```
10 PRINT "DIT IS EEN"
20 PRINT "KORT EENVOUDIG"
30 PRINT "PROGRAMMA"
40 PRINT "TER ILLUSTRATIE"
50 PRINT "VAN HET WERKEN"
60 PRINT "MET REGELNUMMERS"
```

Inderdaad, de computer heeft de regels keurig opnieuw genummerd, volgens de reeks 10, 20, 30 enz.

DELETE

Als laatste commando in dit hoofdstuk behandelen we het DELETE-commando. Hiermee kunnen we één of meer regels uit een programma verwijderen. De opbouw is vrijwel overeenkomstig met het LIST-commando.
Voorbeelden:

```
DELETE 20      : verwijder regel 20
DELETE 20–40   : verwijder regel 20 t/m 40
DELETE –40     : verwijder alle regels t/m 40
DELETE 20–     : deze vorm is in tegenstelling tot bij
                 LIST, bij DELETE niet toegestaan.
```

Ook dit programma testen we met ons voorbeeld-programma uit.
We toetsen in:

```
DELETE 40–60
```

en na het LIST-commando toont de computer:

```
10 PRINT "DIT IS EEN"
20 PRINT "KORT EENVOUDIG"
30 PRINT "PROGRAMMA"
```

Herstellen van fouten

In het voorafgaande hebben we in feite al enkele mogelijkheden besproken om een programma te wijzigen. In deze paragraaf tonen we nog wat we kunnen doen als er een fout gemaakt is.

De ogenschijnlijk meest eenvoudige oplossing is dan de regel met de fout opnieuw in te typen, maar dan zonder fout.

Voorbeeld:
Typ nu in:

```
20 PRINT "KORD EENVOUDIG"
```

Na het LIST-commando zien we hoe deze regel met deze opvallende fout in het programma is opgenomen. We kunnen de fout dus herstellen door deze regel opnieuw, maar dan foutloos, in te typen.

Een meer elegante oplossing bestaat uit het werken met de cursor-toetsen. Dit zijn de toetsen met pijltjes rechts van het toetsenbord. Druk maar eens op deze toetsen en we zien hoe het witte blokje zich over het scherm verplaatst.

Dit blokje noemen we de cursor en op grond hiervan worden de toetsen met de pijltjes, de cursorbesturingstoetsen genoemd.

Verplaats de cursor nu zodanig dat deze precies op onze fout komt te staan, dus:

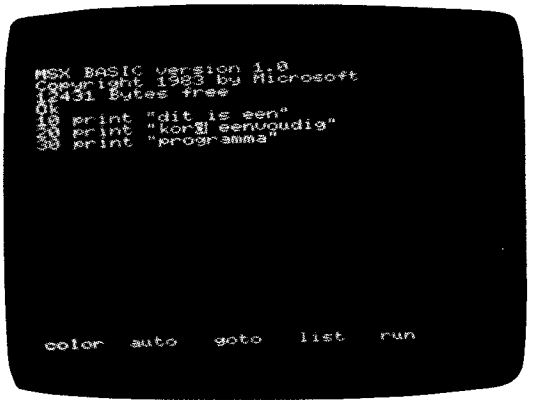
```
20 PRINT "KOR[D] EENVOUDIG"
          ↑
   wijst de cursor aan
```

Nu drukken we op T en op de RETURN-toets. Vervolgens verplaatsen we onze cursor weer tot onder het programma... en ziedaar de fout is hersteld!

Overigens kunnen we een cursor-toets ook ingedrukt houden, waardoor het effect ontstaat alsof herhaaldelijk op deze toets gedrukt wordt. Dit 'autorepeat-effect' geldt trouwens ook voor de overige toetsen.

1 toestand vóór het herstellen van de fout

2 cursor naar de regel met de fout verplaatst



3 cursor naar te verbeteren letter verplaatst: juiste letter intypen en op RETURN-toets drukken

4 cursor weer naar beneden verplaatst: de fout is nu hersteld

Speciale toetsen Ten slotte vermelden we nog dat we tijdens het typen onze tekst weer kunnen wissen en wel door op de toets met BS te drukken. Met de INS-toets kunnen we tekens invoegen en met de toets DEL kunnen we tekens wissen. Deze laatste twee toetsen zijn vooral van belang als we met de cursor-toetsen werken. Eerst verplaatsen we de cursor dan naar de plek waar de fout is geconstateerd om vervolgens naar wens tekens te veranderen, te wissen of toe te voegen.

In dit hoofdstuk maken we kennis met variabelen. Deze kunnen we ons voorstellen als dozen waarop een zekere naam is geplaatst. Het werken met variabelen wordt allereerst verduidelijkt aan de hand van een eenvoudig voorbeeld: het berekenen van de inhoud van een staaf. Het hoofdstuk eindigt met de behandeling van zgn. standaardfuncties. Deze zijn geheel en al te vergelijken met de functietoetsen die we bij een wetenschappelijke pocket-calculator aantreffen.

6. Variabelen en nog meer over rekenen

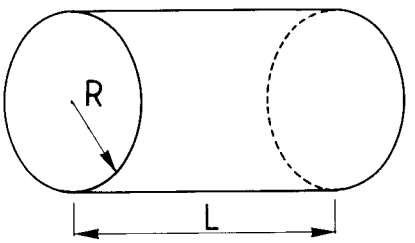
Introductie

De tot nu toe behandelde programma's zijn nog weinig inspirerend. Toch staan computers bekend als ware rekenwonders. Sterker nog, zonder de computer zouden vele wonderen van deze eeuw niet mogelijk zijn, denk bijvoorbeeld eens aan alle ruimtevluchten. Grote bedrijven, universiteiten, ingenieursbureaus, ze kunnen tegenwoordig niet meer zonder computer. Het is daarom stellig terecht dat we onze aandacht in het nu volgende eens richten op programma's, waarbij gerekend wordt.

Nu is rekenen bij velen echter niet zo'n geliefd onderwerp en met dit in het achterhoofd zullen we de zaken zo eenvoudig mogelijk houden. In het nu volgende zullen we een programma behandelen om de inhoud van een staaf te berekenen. Het eerste programma zal zeer eenvoudig van opzet zijn. Daarna zullen we zien hoe we dit programma voor algemeen gebruik geschikt zullen maken.

De inhoud van een staaf

Om de inhoud van een staaf te bepalen, dienen we het oppervlak van de doorsnede te kennen en die waarde te vermenigvuldigen met de lengte. In beeld:



oppervlakte van de doorsnede = 3,14159×R×R

inhoud van staaf = oppervlak doorsnede × L = 3,14159×R×R×L

We zien hoe de inhoud van een staaf in het algemeen van twee grootheden afhangt; de straal R en de lengte L.

Het volgende programma berekent de inhoud van een staaf en wel voor de waarden:

L=5
en R=7

Programma:

```
10 L=5
20 R=7
30 PI=3.14159
40 OPP=PI*R*R
50 I=OPP*L
60 PRINT I
```

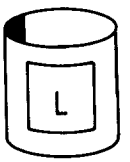
Na het RUN-commando verschijnt:

```
769.68955
```

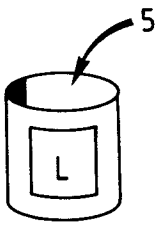
We zullen dit programma nu regel voor regel doornemen. In regel 10 ziet men de uitdrukking;

```
L=5
```

Het effect van deze uitdrukking is dat de computer een gedeelte van het geheugen reserveert en hier als het ware het label L opplakt. Zo'n gedeelte van het geheugen kunnen we het beste als een doos opvatten. In beeld:

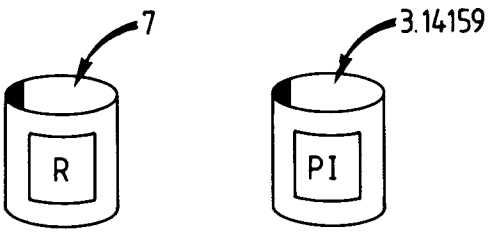


De uitdrukking L=5 wil dan zeggen dat in deze doos het getal 5 wordt opgeslagen. In beeld:



Als in het programma nu naar L verwezen wordt (regel 50), zal de computer in feite naar de *inhoud* van die doos kijken.

Regels 20 en 30 kunnen we evenzo in beeld brengen:



Merk op dat in PI het getal 3.14159 wordt opgeslagen en niet 3,14159. Kortom de komma, die wij gewoonlijk in getallen noteren, wordt in MSX-BASIC steeds met een punt aangegeven. Ook bij het resultaat van het programma zien we deze 'decimale punt'.

Als men al enige wiskundige achtergrond heeft, zal men in het getal 3.14159 het getal π (uitspraak: pi) hebben herkend.

Regel 40 toont een bewerking: in de doos met de naam OPP zal de waarde worden opgeslagen die men krijgt door PI met R×R te vermenigvuldigen.

Ergo, in OPP zal de waarde 3.14159 ×7×7 worden opgeslagen.

De zo gevonden waarde wordt weer in regel 50 gebruikt om uiteindelijk de waarde van de inhoud te bepalen. Deze wordt in de doos met het label I opgeslagen.

Ten slotte gebruiken we een PRINT-instructie om de inhoud van deze laatste doos af te beelden:

```
PRINT I
```

beeld af *de inhoud van de doos met het label I*

Let wel, hier wordt niet de letter I afgedrukt, dan zou de instructie PRINT 'I' moeten luiden, met andere woorden dan hadden we I tussen de bekende aanhalingstekens moeten plaatsen.

We plaatsen bij dit eerste programma de volgende opmerkingen:

- We hadden in plaats van

  ```
  10 L=5
  ```

  ook mogen schrijven

  ```
  10 LET L=5
  ```

  Letterlijk staat er dan 'Laat L gelijk aan 5 zijn' of nog beter 'L *wordt* 5'.

  Met de laatste uitdrukking benadrukken we dat de computer een *actie* uitvoert: er *wordt* iets in de doos met het label L

gestopt. Deze actie wordt helemaal duidelijk bij het volgende voorbeeld:

```
10 LET X=3
20 LET X=X+4
```

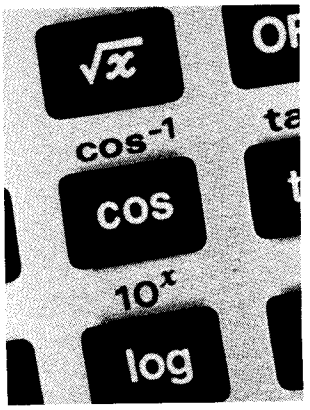
Bij de tweede regel wordt in X de oude waarde van X, met andere woorden 3+4 gestopt. Zo zal X uiteindelijk de waarde 7 bevatten, let wel, een uitdrukking als X=X+4 is in BASIC volledig correct, terwijl zij in de wiskunde niet is toegestaan.

- Het feit dat we een willekeurige waarde in zo'n doos kunnen stoppen, geeft aan dat de inhoud variabel is. Dit is dan ook de reden dat we doorgaans van een *variabele* in plaats van een doos spreken. Zo kent ons programma de variabelen L, R, PI, OPP en I.
  In het vervolg zullen we dan ook spreken van 'het toekennen van waarden aan een variabele' in plaats van 'het opslaan van een waarde in een doos'.

- Voor de naamgeving van variabelen gelden bepaalde regels. Deze zijn als volgt:
  - Alleen de eerste twee letters van een naam worden door de computer als *de* naam herkend. Zo zijn de namen EERSTE en EEN voor de computer gelijk: alleen EE wordt herkend.
  - We mogen geen gereserveerde woorden als naam gebruiken, zo mag men bijvoorbeeld IF niet als naam gebruiken, omdat deze term gereserveerd is. De gereserveerde namen ziet men in de appendices met functies en commando's.

Standaardfuncties

Het getoonde voorbeeld was nog zeer eenvoudig. We kunnen ons nauwelijks voorstellen dat we op deze manier ingewikkelde berekeningen kunnen uitvoeren. Met de behandeling van de zgn. standaardfuncties wordt het inzicht met betrekking tot wat we met MSX-BASIC zoal kunnen berekenen aanmerkelijk vergroot.



Om de standaardfuncties uit te leggen, maken we eerst weer een uitstapje naar de calculator en wel naar de calculator met wiskundige functies. Als we op zo'n calculator de wortel uit een getal moeten berekenen, voeren we een getal in en drukken daarna op de toets waar het wortel-teken op staat. Zo berekent men, door eenvoudig de daartoe aanwezige toets in te drukken, de gewenste functie.

Bij MSX-BASIC kunnen we even eenvoudig waarden voor functies berekenen. Zo'n functie roepen we dan aan met een bepaalde term, en tussen haakjes plaatsen we het getal waarvan de functiewaarde berekend moet worden, dus;

term (waarde)

geeft functie aan — *geeft waarde aan, waarvan functie bepaald moet worden*

Voorbeelden:

```
10 A=SQR(4)
20 PRINT A
```

Resultaat:

```
2
```

In regel 10 wordt aan A de wortel uit 4 toegekend. De term, die in dit voorbeeld de functie aanduidt, is gelijk aan SQR. Tussen haakjes staat de waarde waarvan de wortel bepaald moet worden, in dit geval (voor de eenvoud) 4.

We plaatsen hierbij nog de opmerking dat tussen de haakjes ook een variabele mag staan, of zelfs een rekenkundige uitdrukking.

Voorbeelden:

`B=SQR(A)` : hier wordt de waarde door middel van de variabele A aangegeven.

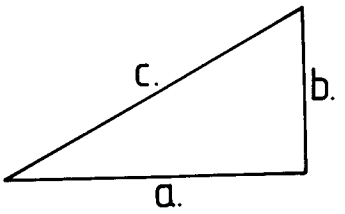
`C=SQR(SQR(5)+4+A)` : hier wordt de waarde door middel van de SQR(5)+4+A aangegeven. Merk op dat deze uitdrukking zelf ook weer een functie bevat.

In onderstaande tabel worden de bekendste functies getoond. Appendix E toont een overzicht van alle functies.

MSX-BASIC aanduiding	*Algebraïsche notatie*	*Betekenis*
ABS (X)	$\|x\|$	Geeft de absolute waarde van X. Dit is de waarde 'zonder teken'.
ATN (X)	arctan (x)	Geeft de arctangens van X. Uitkomst tussen –pi/2 en pi/2.
COS (X)	cos (x)	Geeft de cosinus van X:X in radialen.
EXP (X)	e^x	Geeft de e-macht van X.
INT (X)	geen	Rondt de waarde van X altijd naar beneden af. Zo geeft INT (3.8) de waarde 3 en INT (–3.1) de waarde –4. We kunnen een getal 'echt' afronden met INT (X+0.5).
LOG (X)	^{e}log (x)	Geeft de zgn. natuurlijke logaritme van X.
SGN (X)	geen	Geeft de waarde –1, 0, of 1 en wel afhankelijk van het feit of X negatief, 0 of positief is.
SIN (X)	sin (x)	Geeft de sinus van X:X in radialen.
SQR (X)	$\sqrt{x}$	Geeft de wortel uit X.
TAN (X)	tan (x)	Geeft de tangens van X:X in radialen.

We merken op dat machtsverheffen in deze tabel niet voorkomt. Hiervoor hanteren we het teken ∧. Zo geeft PRINT 2 ∧ 3 de waarde 8 als resultaat ($=2^3$).

Ten slotte geven we nog een eenvoudig voorbeeld van het werken met functies. Het volgende programma berekent de schuine zijde van een rechthoekige driehoek.



Programma:

```
10 INPUT A
20 INPUT B
30 Z2=A^2+B^2
40 C=SQR(Z2)
50 PRINT C
```

We laten het aan de lezer zelf over om het programma uit te proberen.

Meer instructies op één regel

MSX-BASIC biedt ons de mogelijkheid om meer instructies op één regel te plaatsen.

We dienen het teken : als scheidingsteken te gebruiken. Het volgende programma dat aangeeft hoe we twee variabelen A en B van waarde kunnen laten wisselen, illustreert dit

zonder scheidingsteken

```
10 A=10
20 B=5
30 H=B
40 B=A
50 A=H
60 PRINT A
70 PRINT B
```

resultaat

```
5
10
```

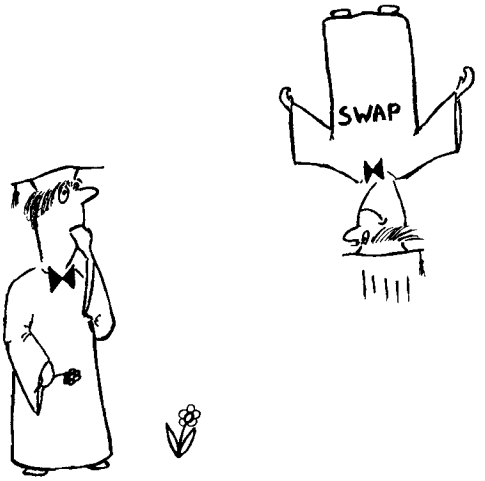
met scheidingstekens

```
10 A=10:B=5:H=B:B=A:A=H
20 PRINT A:PRINT B
```

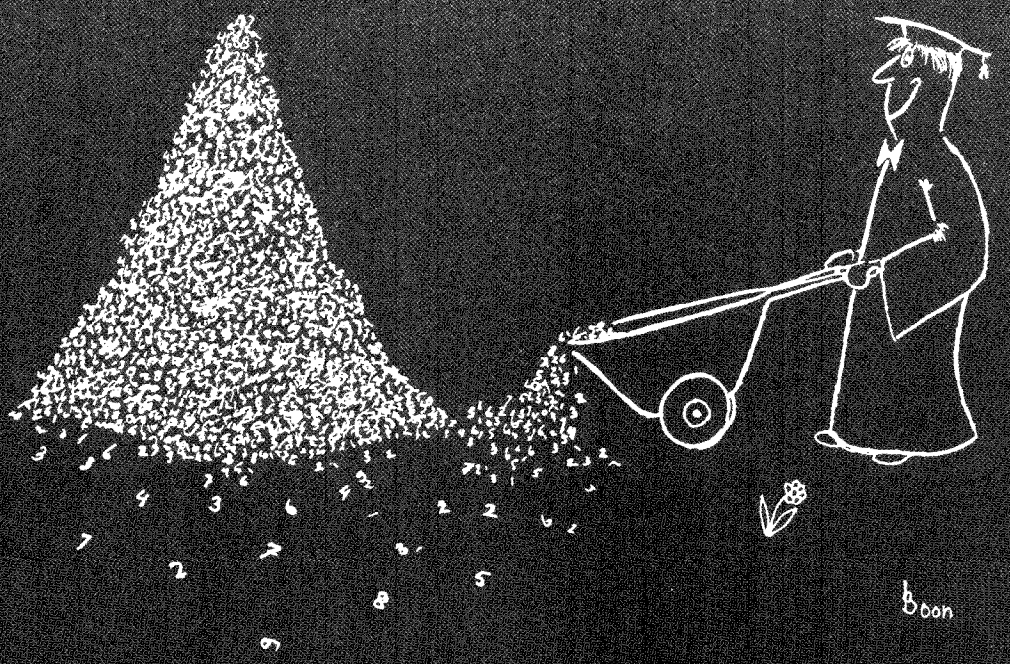
Overigens kent MSX-BASIC een afzondelijke instructie om deze 'wisseltruc' aan te geven. Als we de instructie

```
SWAP A,B
```

geven, zullen de waarden van A en B worden verwisseld.



In dit hoofdstuk breiden we onze programmeer-gereedschappen uit en wel met de instructies INPUT en READ...DATA. INPUT is een instructie die het mogelijk maakt een waarde aan een variabele toe te kennen zonder dat het programma daarbij gewijzigd hoeft te worden. Dit is zeer belangrijk, omdat het hiermee mogelijk wordt programma's voor algemeen gebruik op te stellen. De READ...DATA-vorm is van belang als we een relatief grote reeks waarden aan een relatief grote reeks variabelen willen toekennen. Achter READ komen dan de variabelen te staan waar het om gaat en achter DATA de desbetreffende waarden.



7. INPUT, READ en DATA

INPUT De eerste instructie die we nu behandelen, is de INPUT-instructie. Deze biedt ons de mogelijkheid om, tijdens het uitvoeren van een programma, waarden aan variabelen toe te kennen. Wellicht nog belangrijker is het feit dat deze instructie ons de mogelijkheid biedt programma's voor algemeen gebruik op te stellen. We bekijken allereerst nogmaals het programma van het vorige hoofdstuk:

```
10 L=5
20 R=7
30 PI=3.14159
40 OPP=PI*R*R
50 I=OPP*L
60 PRINT I
```

We hadden in plaats van dit programma natuurlijk ook kunnen intoetsen:

```
PRINT 3.14159*7*7*5
```

om eenzelfde resultaat te verkrijgen.

Dat we dat niet gedaan hebben, is omdat we wilden tonen hoe we een rekenschema in het geheugen van de computer kunnen opslaan, waarbij voor het bepalen van het resultaat, alleen maar waarden van R en L hoeven te worden ingevuld.

Om bijvoorbeeld de inhoud van een staaf met $L=34$ en $R=10$ te bepalen, kunnen we nu regel 10 en 20 wijzigen in:

```
10 L=34
20 R=10
```

en na het RUN-commando verschijnt het gewenste antwoord. Nu is het wijzigen van een programma, voor het uitvoeren van een berekening, ook niet erg aan te bevelen. We zouden immers fouten kunnen maken...

Het programma zou echter veel beter worden als na het

RUN-commando op het scherm mededelingen zouden verschijnen die aangeven dat respectievelijk waarden voor *L* en *R* moeten worden ingevoerd.

Deze mogelijkheid wordt inderdaad geboden en wel in de vorm van de INPUT-instructie. Deze heeft de volgende vorm:

```
INPUT 'af te beelden tekst';naam variabele
```

Bijvoorbeeld:

```
INPUT "VOER LENGTE IN"; L
```

Het resultaat van deze instructie is dat de computer de tekst afbeeldt, inclusief een vraagteken en onmiddellijk daarna het programma onderbreekt. De gebruiker dient nu eerst een getal in te toetsen, bijvoorbeeld:

```
5    (en RETURN-toets)
```

Onmiddellijk na het indrukken van de RETURN-toets wordt dit getal dan aan de variabele *L* toegekend en de computer zal weer doorgaan met het uitvoeren van het programma.

Ons programma komt er met INPUT-instructies nu als volgt uit te zien:

```
10 INPUT "VOER LENGTE L IN"; L
20 INPUT "VOER STRAAL R IN"; R
30 PI=3.14159
40 OPP=PI*R*R
50 I=OPP*L
60 PRINT "DE INHOUD="; I
```

Een voorbeeld van een resultaat is:

```
VOER LENGTE IN? 5
VOER STRAAL IN? 7
DE INHOUD=769.68955
```

We zien hoe bij de PRINT-instructie in dit programma tevens tekst is opgenomen en wel door deze tekst tussen aanhalingstekens te plaatsen en het teken; als scheidingsteken te gebruiken.

Het programma is ten opzichte van ons eerste programma aanmerkelijk verbeterd. De structuur ligt als het ware volledig vast. We hoeven geen veranderingen in het programma zelf aan te brengen, als we de inhoud van een *willekeurige* staaf willen berekenen.

Opmerkingen: Bij de INPUT-instructie plaatsen we nog de volgende opmerkingen:

- We mogen méér variabelen in een INPUT-instructie opnemen, bijvoorbeeld:

```
INPUT "VOER A, B EN C IN:"; A,B,C
```

Na de onderbreking van het programma toont de computer de tekst en een vraagteken en moeten we drie getallen, gescheiden door een komma, invoeren.

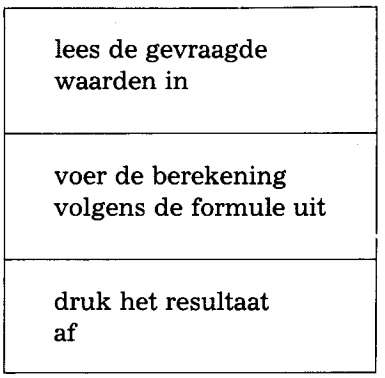
- De tekst in de INPUT-instructie mogen we ook weglaten, bijvoorbeeld:

```
INPUT A
```

Na de onderbreking toont de computer nu enkel nog het vraagteken ten teken dat we een getal moeten invoeren.

Opbouw van het programma

Het is verstandig om op dit punt een moment stil te staan bij de opbouw van het programma. Uitgangspunt was een formule, waarbij aan enkele grootheden van te voren een waarde moest worden toegekend. We zagen dat daartoe de volgende opzet werd gekozen.



Een dergelijke opzet treffen we altijd aan als het gaat om berekeningen die aan formules gekoppeld zijn.

READ en DATA

Op dit moment hebben we twee manieren besproken om waarden aan variabelen toe te kennen.

De eerste methode was gebaseerd op het toekennen van waarden door middel van een LET-instructie, zoals:

```
LET L=3
```

of korter

```
L=3
```

De tweede mogelijkheid ging uit van de INPUT-instructie, bijvoorbeeld:

```
INPUT L
```

De derde, nu te bespreken mogelijkheid haakt in op de wens om

aan een relatief grote reeks variabelen waarden toe te kennen. Hierbij geldt dan dat deze waarden doorgaans niet van programma tot programma wijzigingen zullen ondergaan.

We zouden in zo'n situatie natuurlijk ook een hele reeks LET-instructies kunnen opnemen, maar de mogelijkheid met READ en DATA is; zoals we zullen zien, aanzienlijk fraaier. We tonen onmiddellijk een voorbeeld:

```
10 READ A,B,C,D
20 F=A+B+C+D
30 PRINT F
40 DATA 3,7,4,8
```

Het gevolg van de READ-instructie op regel 10 is, dat de computer 'weet' dat er een aantal variabelen genoemd worden, waarvan de waarden worden opgesomd in de regel die met DATA begint.

Zo zal aan A de waarde 3, aan B de waarde 7, aan C de waarde 4 en aan D de waarde 8 worden toegekend.

Het resultaat van het programma bevestigt dit:

```
22
```

Merk op dat de DATA-regel op zich geen uitvoerbare instructie is. Het is gewoon een kladpapiertje, waarop de computer zijn gezochte waarden vindt. We hadden de READ-lijst overigens ook mogen splitsen:

```
10 READ A,B
15 READ C,D
```

en evenzo hadden we de DATA-lijst mogen splitsen, bijvoorbeeld als volgt:

```
40 DATA 3,7,4
50 DATA 8
```

Voor de computer maakt dat alles geen verschil; hij leest de gegevens, als vormen ze één continue lijst. Men kan het mechanisme het beste voorstellen, als plaatst de computer vooraf een pijl bij de eerste waarde van de DATA-lijst. Iedere keer als er nu door middel van een READ-instructie een waarde aan een variabele wordt toegekend, schuift deze pijl één plaats op:

```
40 DATA 3,7,4,8
          ↑
```

pijl schuift steeds één plaats op.

Als de laatste waarde van de DATA-lijst is bereikt, schuift de computer de pijl op naar de eerste waarde van een daaropvolgende DATA-lijst, tenminste als die er is.

We kunnen deze pijl ook terugzetten naar de oorspronkelijke positie en wel met behulp van de volgende instructie:

```
RESTORE
```

Voorbeeld:

```
10 READ, A,B
20 RESTORE
30 READ C,D
40 F=A+B+C+D
50 PRINT F
60 DATA 3,7,4,8
```

Resultaat:

```
20
```

Merk op dat de uitkomst nu 20 is. Dit is het gevolg van de RESTORE-instructie; zowel aan A en B als aan C en D wordt nu 3 en 7 toegekend.

Zijn met het bespreken van LET, INPUT, READ en DATA nu alle mogelijkheden voor het toekennen van gegevens behandeld? Het antwoord is nee!

Dat zou ook niet zo fraai zijn als we denken aan de vele administratieve toepassingen die in BASIC kunnen worden uitgevoerd.

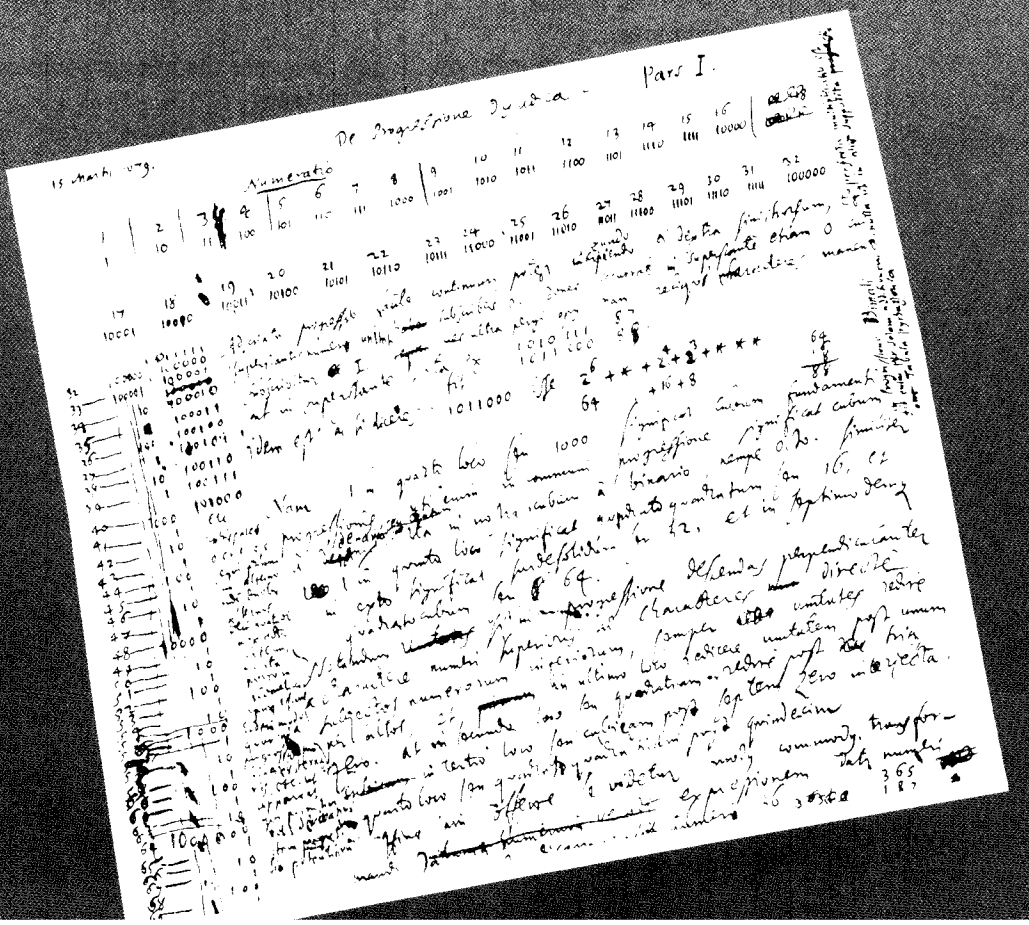
De laatste mogelijkheid, waarop we op dit moment niet verder willen ingaan, gaat uit van het inlezen van waarden vanuit bestanden.

Bestanden zijn verzamelingen gegevens die bijvoorbeeld op cassette of floppy disks zijn opgeslagen.

In dit hoofdstuk gaan we dieper in op verschillende getallen. Allereerst bekijken we hoe men zeer grote, dan wel zeer kleine getallen in de zgn. wetenschappelijke notatie kan weergeven.

Vervolgens wordt ingegaan op de begrippen dubbele en enkele precisie. Hierbij gaat het om de nauwkeurigheid, waarmee getallen in het geheugen van de computer worden opgeslagen.

Ten slotte geven we een inleiding over binaire, octale en hexadecimale getallen. Het gaat hierbij om notatie-vormen, waarbij de positie van een cijfer met een bepaalde macht overeenkomt.



8. Getallen, groot en klein en in allerlei gedaanten

Wetenschappelijke notatie

In het voorafgaande kwamen we reeds verschillende typen getallen tegen, namelijk de gehele getallen, die ook wel eens integers genoemd worden, en de getallen met een decimale punt.

Nu zullen we een nieuwe vorm bespreken om getallen met een decimale punt te representeren (voor te stellen). Deze vorm is met name geschikt als het om zeer grote dan wel zeer kleine getallen gaat. Dergelijke getallen komen bijvoorbeeld bij natuurkundige of sterrenkundige berekeningen voor.

Laten we, om een en ander tastbaar te maken, eens denken aan de elektrische lading van één elektron.
Deze lading, uitgedrukt in de eenheid coulomb, is:

0.000 000 000 000 000 000160219

Dit is inderdaad een zeer, zeer klein getal. De spaties hebben we hier met opzet in het getal aangegeven, om wat eenvoudiger het aantal nullen te kunnen tellen (bij de computer mogen we trouwens nooit spaties in getallen plaatsen). Hoe kunnen we zo'n getal nu op eenvoudige wijze noteren?

Het antwoord luidt: representeer het getal als een vermenigvuldiging van twee getallen. Het eerste getal geeft dan de 'werkelijke' cijfers in het oorspronkelijke getal aan en het tweede geeft direct aan hoeveel nullen er in het spel zijn.
Bijvoorbeeld voor het bovenste getal:

1.60219 × getal dat aangeeft hoeveel nullen er zijn

Voor het getal dat moet aangeven hoeveel nullen er in het oorspronkelijke getal voorkomen, wordt een zogenaamde macht van het getal 10 genomen. We geven in de tabel op de volgende pagina enkele voorbeelden van machten van 10.

We kunnen uit de tabel opmaken, dat bijvoorbeeld 10^4 overeenkomt met 10000, met andere woorden het aantal nullen komt hier direct met de macht (cijfer rechtsboven de 10) overeen.

in woorden	*notatie*	*betekenis*	*waarde*
10 tot de macht 1	10^1	10	10
10 tot de macht 2	10^2	10×10	100
10 tot de macht 3	10^3	10×10×10	1000
10 tot de macht 4	10^4	10×10×10×10	10000
10 tot de macht −1	10^{-1}	1/10	0.1
10 tot de macht −2	10^{-2}	1/(10×10)	0.01
10 tot de macht −3	10^{-3}	1/(10×10×10)	0.001
10 tot de macht 0	10^0	10/10	1

Voorbeeld:
Met welke macht komt 1000 000 000 overeen?

Antwoord: tel het aantal nullen, dit zijn er 9 en zodoende vinden we 10^9.

Uit de tabel blijkt tevens dat 0.001 overeenkomt met 10^{-3}. Ook bij getallen kleiner dan 1 kan men kennelijk het aantal nullen tellen om de macht te bepalen.

Voorbeeld:
Met welke macht komt onderstaand getal overeen?

0.000 000 000 000 000 000 1

Antwoord: tel het aantal nullen, dit zijn er 19 en zo vinden we 10^{-19}.

Dit was overigens precies het aantal nullen dat voorkwam in het getal dat de lading van het elektron aangaf.

We kunnen deze lading dus ook aangeven als:

1.60219×10^{-19}

Het eerste getal wordt *mantisse* genoemd en het tweede getal geeft de *exponent* aan.

Bij de computer geven we bovenstaand getal aan volgens de volgende notatie:

1.60219E−19

Met andere woorden, we gebruiken de letter E om de mantisse van de exponent te onderscheiden.

Om enige ervaring met deze notatie op te doen, kan men het volgende programmaatje gebruiken:

```
10 INPUT A
20 INPUT B
30 C=A*B
40 PRINT C
```

Voorbeelden:

A=987654	B=456789	geeft: 451149483006
A=98765434	B=987654321	geeft: 9.7546105778997E+17
A=.0000000008	B=.0000000007	geeft: 5.6E−19
A=8000000000	B=7000000000	geeft: 5.6E+19

Dubbele en enkele precisie

Voor de weergave van getallen gebruikt onze computer een beperkt aantal geheugenregisters (woorden). Dit betekent ook dat berekeningen een beperkte nauwkeurigheid hebben. MSX-BASIC biedt de gebruiker zowel de mogelijkheid om met zgn. dubbele precisie, als om met enkele precisie te werken. Bij dubbele precisie worden steeds 8 registers gebruikt voor de opslag van getallen en bij enkele precisie slechts 4. Standaard werkt onze computer steeds met dubbele-precisie-getallen. Om met enkele-precisie-getallen te werken, plaatsen we steeds een uitroepteken *(!) achter het getal.*

Variabelen waaraan we enkele-precisie-getallen toekennen, onderscheiden zich van dubbele-precisie-variabelen door het plaatsen van *!* direct *achter de naam.* De volgende voorbeelden illustreren het verschil tussen enkele- en dubbele-precisie-berekeningen.

```
10 A=10/3                    10 A!=10!/3!
20 PRINT A                   20 PRINT A!

resultaat                    resultaat

3.333 333 333 333 3          3.33333
```

We plaatsen hierbij nog de volgende opmerkingen:

- Bij dubbele-precisie-getallen mogen we aan het eind van het getal het teken # plaatsen. Evenzo mogen we dit teken als laatste teken van een naam opgeven. Zo geven we expliciet aan, met dubbele-precisie-getallen te werken.
- Als we bij de dubbele precisie-getallen de letter D in plaats van E gebruiken (bijv. 23.45156321D39) dan geven we daarmee expliciet aan dat we met dubbele precisie werken.
- Men dient voortdurend in de gaten te houden of het wel realistisch is, resultaten met zoveel cijfers weer te geven. Stel dat een timmerman een plank van 10 meter in 3 delen moet zagen, dan is volgens onze computer ieder deel 3.333 333 333 333 3 m lang... maar kan onze timmerman wel zo nauwkeurig zagen!
- Het gebruik van enkele-precisie-getallen kan vooral van belang zijn als we zuinig met het geheugen willen omspringen. Deze geheugenproblematiek komt in de praktijk vooral naar voren bij de nog te bespreken arrays (zie hoofdstuk 11).

Gehele getallen

In een aantal gevallen willen we in het programma aangeven dat het beslist om gehele getallen, met andere woorden om 'integers' gaat.

Bij de weergave van getallen in een programma levert dat geen problemen op: een getal zonder decimale punt is immers een geheel getal. Aan variabelen kan men echter niet zien of ze een geheel dan wel een getal met een decimale punt voorstellen. Welnu om dat onderscheid wel duidelijk te maken, plaatst men het teken % achter de naam.
Voorbeelden:

```
A%      B1%
```

Let wel, zo'n variabele kan alleen maar een geheel getal bevatten. Als de uitkomst van een berekening een niet geheel getal oplevert, en zo'n waarde wordt aan een integer-variabele toegekend, dan vindt automatisch afronding (naar beneden) plaats.
Voorbeeld:

```
10 A%=20
20 B%=3
30 C%=A%/B%
40 PRINT C%
```

Resultaat:

```
6
```

Merk nogmaals op dat de afronding altijd naar beneden toe plaatsvindt. Bij ons voorbeeld werd 6.6666.. naar 6 afgerond!

Binaire getallen

Om binaire getallen uit te leggen, kijken we eerst naar de opbouw van onze 'gewone', dat wil zeggen decimale getallen. Deze getallen zijn steeds uit machten van 10 opgebouwd (zie tabel aan het begin van dit hoofdstuk).

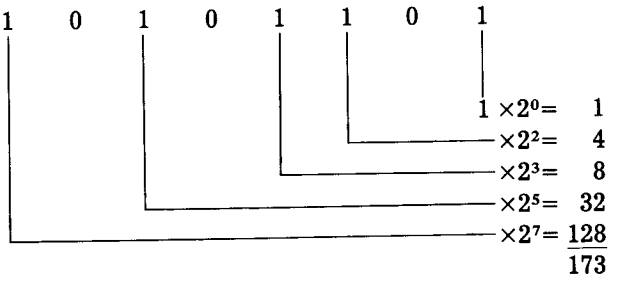
Voorbeeld:
Het getal 4736 is als volgt uit machten van 10 opgesteld.

```
4   7   3   6
|   |   |   |
|   |   |   6×10⁰=6×    1=    6
|   |   3———×10¹=3×   10=   30
|   7———————×10²=7×  100=  700
4———————————×10³=4×1000= 4000 +
                         ————
                         4736
```

We zien hoe ieder cijfer hier met een macht van 10 overeenkomt. Van rechts af tellen we de nulde, eerste, tweede en derde positie en deze posities komen dan precies met de machten van 10 overeen.

Welnu, binaire getallen bestaan alleen uit de cijfers 0 en 1 en de positie van zo'n cijfer komt weer met een macht overeen en wel met een macht van 2.
Voorbeeld:

Met welk decimaal getal komt het binaire getal 10101101 overeen? Uitwerking



Bij MSX-BASIC kunnen we binaire getallen ook direct aangeven en wel door &B voor het getal te plaatsen.
Voorbeeld:

```
10 A%=&B10101101
20 PRINT A%
```

Resultaat:

```
173
```

Octale en hexadecimale getallen

Na het voorafgaande kunnen we eenvoudig aangeven wat octale en hexadecimale getallen zijn.

Octale getallen zijn getallen waarvan het cijfer overeenkomstig onze uiteenzetting correspondeert met een macht van 8. Bij hexadecimale getallen komt de positie van een cijfer met een macht van 16 overeen.

Het is aardig om op te merken dat we bij decimale getallen precies 10 cijfers hebben (0 t/m 9) om deze getallen te presenteren. Bij binaire getallen zijn er maar twee cijfers (0 en 1) en bij octale zijn er natuurlijk 8 (0 t/m 7).

Tot zover geen problemen. Bij hexadecimale getallen zijn er natuurlijk 16 cijfers... Nu kennen we alleen maar de cijfers 0 t/m 9. Hoe geven we dan de overige cijfers bij hexadecimale getallen aan? Welnu, we gebruiken hiervoor de letters A t/m F.

De volgende tabel toont de decimale getallen 0 t/m 15 en deze worden ter illustratie tevens in binaire, octale en hexadecimale vorm getoond.

decimaal	*binair*	*octaal*	*hexadecimaal*
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Voorlopig lijkt het nut van dit alles nog ver te zoeken. Toch zullen we zien, (onder andere in het hoofdstuk over sprites), hoe deze zaken van pas komen.

Octale en hexadecimale getallen mogen als zodanig direct in programma's worden opgenomen. Voor een octaal getal plaatsen we dan &O en voor een hexadecimaal getal &H.
Voorbeelden:

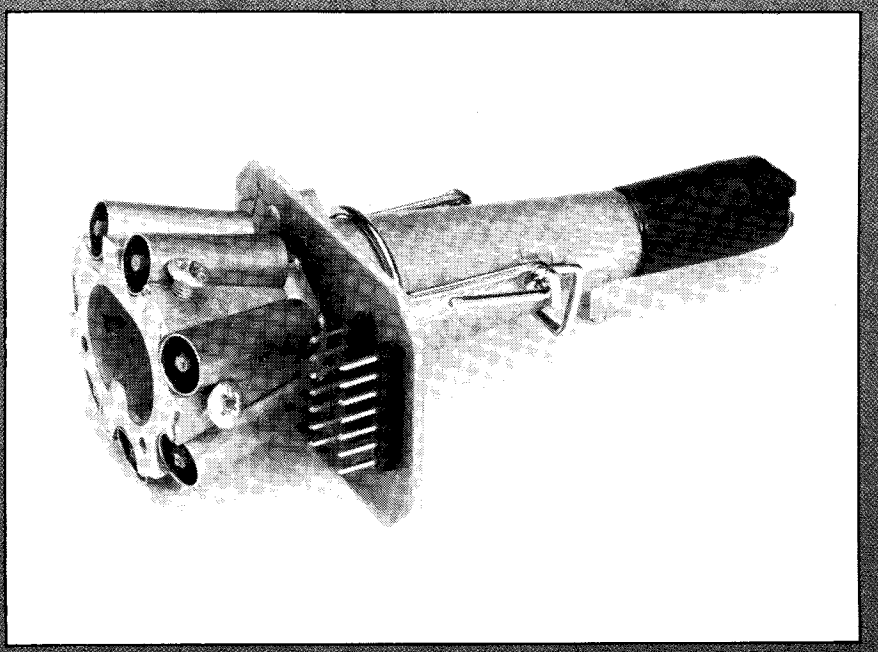
```
PRINT &017  geeft  15
PRINT &HF   geeft  15
PRINT &HFF  geeft  255
```

hex.code	mnemonic		hex.code	mnemonic	
00	NOP		23	INC	HL
018405	LD	BC,nn	24	INC	H
02	LD	(BC),A	25	DEC	H
03	INC	BC	2620	LD	H,n
04	INC	B	27	DAA	
05	DEC	B	282E	JR	Z,e
0620	LD	B,n	29	ADD	HL,HL
07	RLCA		2A8405	LD	HL,(nn)
08	EX	AF,AF'	2B	DEC	HL
09	ADD	HL,BC	2C	INC	L
0A	LD	A,(BC)	2D	DEC	L
0B	DEC	BC	2E20	LD	L,n
0C	INC	C	2F	CPL	
0D	DEC	C	302E	JR	NC,e
0E20	LD	C,n	318405	LD	SP,nn
0F	RRCA		328405	LD	(nn),A
102E	DJNZ	e	33	INC	SP
118405	LD	DE,nn	34	INC	(HL)
12	LD	(DE),A	35	DEC	(HL)
13	INC	DE	3620	LD	(HL),n
14	INC	D	37	SCF	
15	DEC	D	382E	JR	C,e
1620	LD	D,n	39	ADD	HL,SP
17	RLA		3A8405	LD	A,(nn)
182E	JR	e	3B	DEC	SP
19	ADD	HL,DE	3C	INC	A
1A	LD	A,(DE)	3D	DEC	A
1B	DEC	DE	3E20	LD	A,n
1C	INC	E	3F	CCF	
1D	DEC	E	40	LD	B,B
1E20	LD	E,n	41	LD	B.C
1F	RRA		42	LD	B,D
202E	JR	NZ,e	43	LD	B,E
218405	LD	HL,nn	44	LD	B,H
228405	LD	(nn),HL	45	LD	B,L

Een gedeelte van de machinecodes waarvan de Z80-processor gebruik maakt.

De machinecodes zijn symbolisch aangegeven (onder de kolom mnemonic). Merk op dat elke code ook d.m.v. een hexadecimale code is aangegeven. Zo worden hexadecimale getallen veelvuldig in programma's met machinecodes gebruikt.

In dit hoofdstuk gaan we uitgebreid in op de PRINT-instructie. Allereerst tonen we hoe we meer af te drukken zaken in een PRINT-instructie kunnen opnemen. Hierbij worden de tekens komma (,) en puntkomma (;) als scheidingstekens gebruikt. Vervolgens wordt getoond hoe we met de functie TAB en LOCATE de positie kunnen aangeven waar iets op het scherm moet komen te staan. De PRINT USING-vorm is van belang als het er om gaat in welke vorm getallen moeten worden gepresenteerd. De REM-instructie ten slotte biedt ons de mogelijkheid commentaar in programma's op te nemen.



9. PRINT, TAB, LOCATE, PRINT USING en REM

Scheidingstekens

In het nu volgende zullen we de aandacht vooral richten op het gebruik van scheidingstekens in de PRINT-instructie.

De PRINT-instructie is zeer belangrijk, want het afdrukken van resultaten, iets wat we aan de hand van de PRINT-instructies regelen, is immers één van de onderwerpen waar iedere programmeur mee te maken krijgt.

In vorige hoofdstukken zagen we al dat we meer af te drukken zaken achter een PRINT-instructie mochten plaatsen.

We illustreren de verschillende mogelijkheden weer met voorbeelden.

Voorbeeld 1:

```
10 A=3
20 B=4
30 PRINT A*B
```

Resultaat:

```
12
```

Het voorbeeld toont op eenvoudige wijze aan dat we niet alleen namen van variabelen in een PRINT-instructie mogen opnemen, maar dat we zelfs bewerkingen kunnen aangeven.

Voorbeeld 2:

```
A=3
20 PRINT "A=";A
30 PRINT "A=",A
```

Resultaat:

```
A=12
A=              12
```

Het verschil tussen het resultaat van regel 20 en dat van regel 30

schuilt in het scheidingsteken. Bij regel 20 gebruikten we de punt-komma als scheidingsteken en bij regel 30 de komma.

Bij een punt-komma komt het volgende af te drukken resultaat direct achter het vorige. Gebruikt men de komma, dan hanteert de computer automatisch een indeling in kolommen.

Bij deze indeling in kolommen wordt steeds gerekend met een tussenruimte van 14 posities.

Overigens mag men aan het einde van een PRINT-instructie deze scheidingstekens ook plaatsen, kijk maar eens naar het volgende voorbeeld:

```
10 PRINT "ABC";"DEFG";
20 PRINT "H";"IJ";"KLM"
```

Resultaat:

```
ABCDEFGHIJKLM
```

Als men bij het afdrukken van tekst een regel wil overslaan, dan gebruikt men een PRINT-instructie zonder nadere aanduiding. Voorbeeld:

```
10 PRINT "A"
20 PRINT
30 PRINT "B"
```

Resultaat:

```
A

B
```

Het is duidelijk dat tussen A en B nu een regel is overgeslagen.

TAB We kunnen aan de hand van de zgn. TAB-functie ook aangeven op welke kolom een af te drukken grootheid moet komen te staan.
Het volgende voorbeeld verduidelijkt het gebruik van TAB.

```
10 PRINT TAB(1);"MSX"
20 PRINT TAB(3);"MSX"
30 PRINT TAB(5);"MSX"
```

Resultaat:

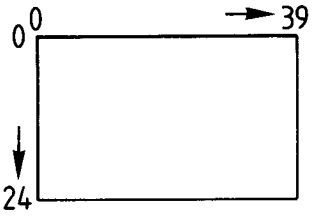
```
MSX
  MSX
    MSX
```

We zien hoe na TAB tussen haakjes steeds de kolom-positie wordt aangegeven vanaf waar de af te drukken zaak (in dit geval tekst) moet komen te staan. De getallen van de kolommen lopen overigens van 0 tot 39.

LOCATE De functie TAB regelt de afdrukpositie op de regel. LOCATE geeft zowel de horizontale als de verticale plaats aan. Het is een zeer krachtige instructie, waarmee we op eenvoudige wijze een mooie opmaak voor af te drukken tekst kunnen verkrijgen. De vorm van de LOCATE-instructie is als volgt:

```
LOCATE horizontale positie , verticale positie
```

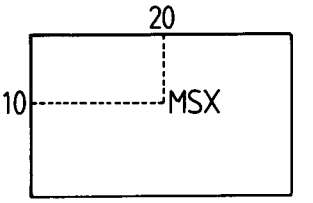
De scherm-indeling is daarbij als volgt:



Voorbeeld van een programma:

```
10 CLS
20 LOCATE 20,10
30 PRINT "MSX"
```

geeft als resultaat:



Het programma start met een nog niet eerder behandelde instructie, namelijk de CLS-instructie (*C*lear-*S*creen-instructie). Deze heeft als resultaat dat het scherm gewist wordt en de cursor in de linkerbovenhoek wordt geplaatst. De daaropvolgende LOCATE-instructie verplaatst de cursor naar de positie (20, 10). Dit houdt dan in dat de tekst MSX vanaf die positie wordt afgebeeld.

PRINT USING Bij zeer veel toepassingen willen we dat de resultaten steeds in een bepaalde vorm verschijnen. Als illustratief voorbeeld kunnen we aan financiële berekeningen denken. Uitkomsten dienen hier steeds in een vorm met twee cijfers achter de komma te verschijnen.

De wijze waarop we een getal afbeelden geven we aan met een zgn. PRINT USING-instructie.

Deze heeft steeds de volgende opbouw:

```
PRINT USING "informatie over representatie";getal
```

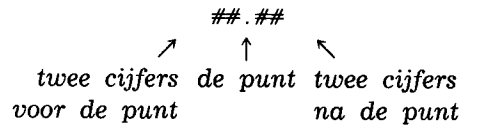
We geven een voorbeeld:

```
PRINT USING "##.##"; 32.7
```

Op het scherm verschijnt dan:

```
32.70
```

De uitdrukking tussen de aanhalingstekens, met andere woorden ##.## geeft aan hoe het getal gerepresenteerd moet worden. In dit geval met twee cijfers vóór de decimale punt en twee cijfers áchter de decimale punt.



De mogelijkheden met betrekking tot PRINT USING zijn bijzonder groot. Een volledig overzicht wordt in appendix D gegeven.

REM De laatste instructie die we in dit hoofdstuk bespreken, is de zgn. REM-instructie. De term REM komt van 'remark', hetgeen zoveel betekent als 'opmerking'. We kunnen hiermee commentaar aan een programma toevoegen. Dit commentaar dienen we dan steeds na de term REM te plaatsen.
Een voorbeeld:

```
10 REM ZO MAAR EEN VOORBEELD
20 PRINT "EINDE"
```

Resultaat na het RUN-commando:

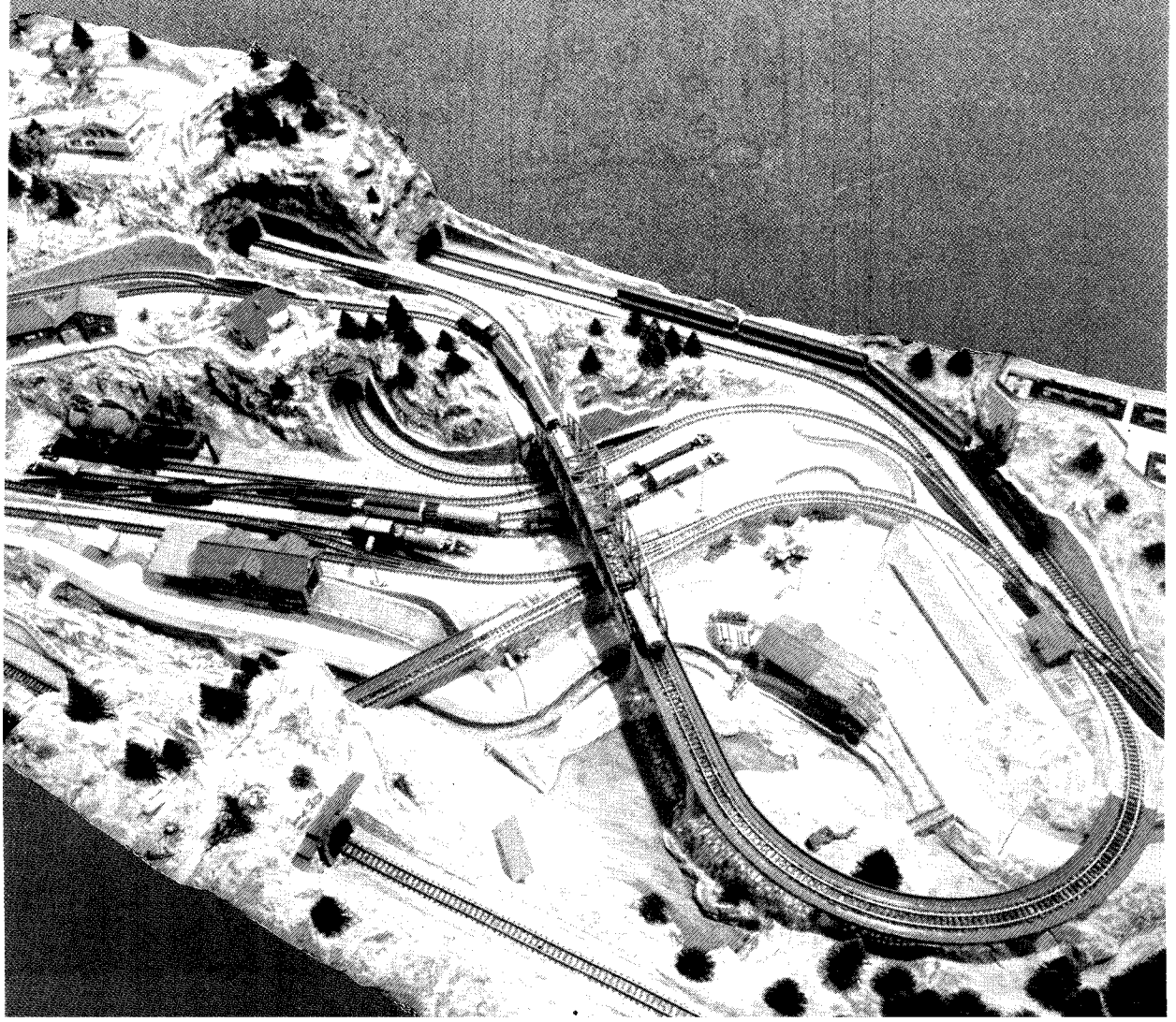
```
EINDE
```

We zien dat het na REM geplaatste commentaar na het RUN-commando niet wordt afgedrukt. We zien het alleen maar terug als we weer een LIST-commando geven.

Het opnemen van commentaar kan bijzonder belangrijk zijn bij de wat langere programma's. We gebruiken deze mogelijkheid dan om het hoe en waarom van het programma uit te leggen.

De REM-instructie . . . .

In dit hoofdstuk komen de zgn. besturings-instructies aan de orde. Dit zijn instructies waarmee de volgorde waarin de computer de instructies afwerkt, wordt bepaald. Eerst wordt de GOTO-instructie behandeld, een instructie waarmee een sprong naar een aangegeven regelnummer wordt uitgevoerd. Vervolgens komt de IF...THEN-instructie aan bod. Deze instructie biedt de mogelijkheid om op grond van een bepaalde voorwaarde al of niet bepaalde instructies uit te voeren. Ten slotte komen de FOR...NEXT-instructie en de ON...GOTO-instructie aan bod. De FOR...NEXT-instructie geeft de mogelijkheid een groep instructies een aantal malen uit te voeren. De ON...GOTO-instructie doet denken aan een keuzeschakelaar. Het programmeren gaat nu echt beginnen!



10. De besturings-instructies: GOTO, IF...THEN, FOR...NEXT en ON...GOTO

Inleiding Bij de tot nu toe behandelde programma's worden de instructies één voor één uitgevoerd. Deze opzet was, zo zagen we, met name geschikt om uitkomsten van formules te berekenen. In de eerste instructies krijgen bepaalde variabelen die in die formule voorkomen, een bepaalde waarde en in de laatste instructie(s) wordt de waarde van de formule berekend.

In zeer veel gevallen wil men echter over de mogelijkheid beschikken om in te grijpen op de volgorde waarin de instructies worden uitgevoerd.

De instructies die deze mogelijkheid bieden, noemen we besturings-instructies, omdat ze als het ware de loop waarin het programma wordt afgewerkt, besturen.

Deze instructies zijn:

`GOTO`	voor het uitvoeren van een sprong in het programma.
`IF...THEN`	voor het al of niet uitvoeren van een programmagedeelte, op grond van zelf aan te geven voorwaarde.
`FOR...NEXT`	voor het herhaald uitvoeren van een reeks instructies.
`ON...GOTO`	voor het uitvoeren van één van een reeks van mogelijke sprongen.

Deze instructies zullen in dit hoofdstuk aan bod komen. In hoofdstuk 14 bespreken we dan nog de volgende instructies:

`GOSUB...RETURN`	voor het aanroepen van een bepaalde reeks instructies vanuit een willekeurig punt in een programma.

ON...GOSUB	voor het aanroepen van één bepaalde reeks instructies, uit een aantal mogelijke reeksen instructies.
DEF FN	voor het definiëren van een functie.

We zullen de nu te bespreken instructies zo eenvoudig mogelijk behandelen. Zo kan men op de beste wijze het effect van zo'n instructie overzien. In volgende hoofdstukken zal het gebruik van deze instructies dan aan de hand van meer realistische voorbeelden worden toegelicht.

GOTO De GOTO-instructie is wellicht de meest bekritiseerde instructie van BASIC. De kritiek richt zich vooral op het feit dat voor veel constructies de GOTO-instructie noodzakelijk is en dat dit tot gevolg heeft dat programma's onoverzichtelijk worden. Maar laten we deze sombere geluiden even rusten en ons richten op de werking van de instructie zelf.
De GOTO-instructie heeft een zeer eenvoudige vorm en wel:

```
GOTO  regelnummer
```

Als de computer deze instructie tegenkomt, zal een sprong naar het aangegeven regelnummer worden uitgevoerd, alwaar het programma dan wordt voortgezet. Aangezien zo'n sprong *altijd* wordt uitgevoerd, spreken we hier van een *onvoorwaardelijke* sprong.

Een eenvoudig voorbeeld:

```
10 PRINT "DIT WORDT AFGEDRUKT"
20 GOTO 40
30 PRINT "DIT NIET"
40 PRINT "DIT WORDT OOK AFGEDRUKT"
```

Resultaat:

```
DIT WORDT AFGEDRUKT
DIT WORDT OOK AFGEDRUKT
```

Regel 20 geeft een sprong naar regel 40, zodat regel 30 altijd wordt overgeslagen. Op dit moment komt het gebruik van deze instructie waarschijnlijk als tamelijk onzinnig over. Waarom zouden we immers over instructies heen willen springen als die instructies kennelijk toch nooit behoeven te worden uitgevoerd?

Bij de behandeling van de volgende instructie zien we echter dat de GOTO-instructie in een aantal gevallen wél gebruikt moet worden.

IF...THEN De meest eenvoudige vorm van de IF...THEN-instructie is:

```
IF voorwaarde THEN instructie
```

We zien dat tussen de termen IF en THEN kennelijk een voorwaarde is aangegeven en wel een voorwaarde waarvoor geldt dat *ALS* (IF) aan deze voorwaarde voldaan is de na THEN (*DAN*) aangegeven instructie wordt uitgevoerd. We kunnen de constructie het beste aan de hand van een zeer eenvoudig voorbeeld verduidelijken:

```
10 INPUT "VOER GETAL IN";K
20 IF K=3 THEN PRINT "GETAL WAS DRIE"
30 PRINT "EINDE PROGRAMMA"
```

Het programma start met een INPUT-instructie, die aangeeft een getal in te voeren. Stel we voeren de waarde 3 in; aan K wordt dus de waarde 3 toegekend. In de daaropvolgende regel staat een IF...THEN-instructie.
De voorwaarde luidt:

```
K=3
```

of in woorden:

is K gelijk aan 3?

Merk op dat we de uitdrukking hier in vragende vorm hebben gesteld. Wel nu, zo'n uitdrukking kan alleen maar goed of fout zijn, of met betrekking tot het voorbeeld; K is inderdaad gelijk aan 3 of dit is niet het geval.

We spreken dan van het al of niet *waar* zijn van de aangegeven voorwaarde. Of om het nog deftiger te zeggen; de logische uitdrukking is al of niet waar. Welnu, aan K was de waarde 3 toegekend en de conclusie is dan ook dat aan de gestelde voorwaarde voldaan is.

In dit geval zal de computer de instructie na THEN uitvoeren, met als resultaat dat de mededeling 'GETAL WAS DRIE' volgt. Als K ongelijk aan 3 zou zijn, met andere woorden aan de gestelde voorwaarde was niet voldaan, dan zal de instructie na THEN gewoonweg worden overgeslagen. Als de computer de IF...THEN-instructie eenmaal heeft afgewerkt, gaat hij door met de daaropvolgende instructie.

In het nu volgende voorbeeld tonen we een programma waar na THEN een GOTO-instructie voorkomt:

```
10 PRINT "HOEVEEL IS 2+5?"
20 INPUT K
30 IF K=7 THEN GOTO 80
40 PRINT "NEE HOOR"
50 PRINT "HET ANTWOORD OP 2+5"
```

```
60 PRINT "IS NATUURLIJK 7"
70 GOTO 90
80 PRINT "DAT IS INDERDAAD HET ANTWOORD"
90 END
```

We zien hier dat na THEN de instructie GOTO 80 is opgenomen. Zo zal, als voor K de waarde 7 wordt ingevuld, een sprong naar regel 80 worden uitgevoerd.

Als voor K een waarde ongelijk aan 7 wordt ingevoerd, zal deze GOTO-instructie worden overgeslagen en komen zodoende de regels 40, 50, 60 en 70 aan bod. Merk op dat regel 70 weer een GOTO-instructie aangeeft. Deze is ook noodzakelijk, omdat anders na de tekst van regel 40, 50 en 60 de tekst van regel 80 zou verschijnen... op z'n minst zou dat wat merkwaardig overkomen.

De laatste instructie in dit programma is de zgn. END-instructie, die aangeeft dat het programma is afgelopen. Deze instructie mag eventueel worden weggelaten. In feite hebben we dit bij de voorafgaande programma's ook steeds gedaan.

We plaatsen bij dit programma nog de volgende aantekeningen:

- in plaats van IF K=7 THEN GOTO 80
  had men ook mogen schrijven
  IF K=7 THEN 80
  of met weglating van THEN
  IF K=7 GOTO 80

- na THEN hadden we eventueel meer instructies en wel gescheiden door de tekens : mogen plaatsen. Voorwaarde is dan wel dat de hele IF...THEN-uitdrukking op één BASIC-regel past (max. 255 tekens).

- in de IF...THEN-instructie wordt K met 7 vergeleken en wel met het teken =. Zo'n teken in een voorwaarde noemen we een relatie-teken. We mogen behalve het teken = ook de volgende tekens gebruiken:

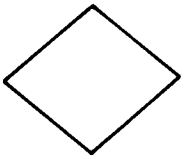
teken	*betekenis*
<	kleiner dan
>	groter dan
< =	kleiner dan of gelijk aan
= <	idem
> =	groter dan of gelijk aan
= >	idem
<>	ongelijk aan
><	idem

- We mogen logische uitdrukkingen ook combineren. De mogelijkheden hiertoe worden in Appendix F besproken.

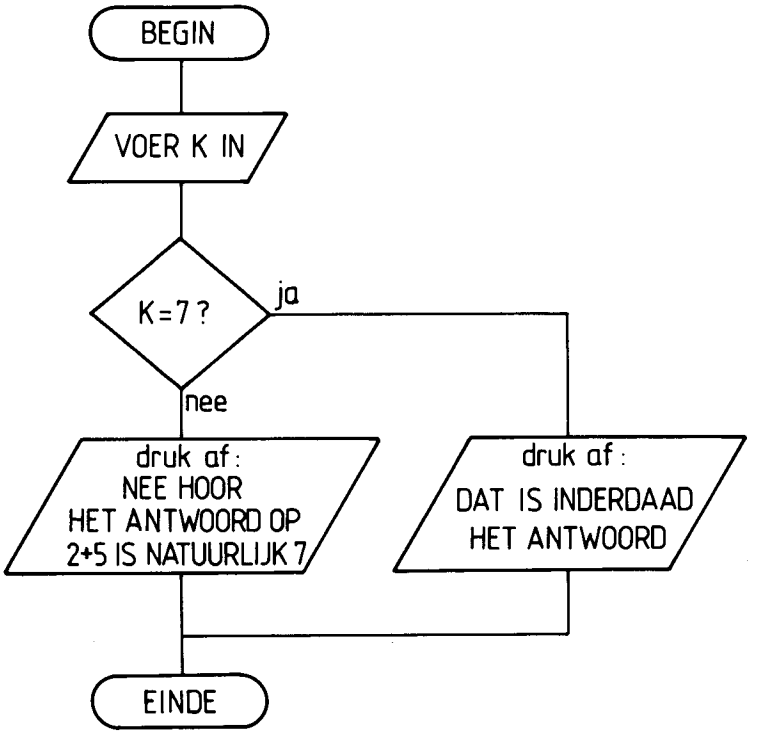
Stroomschema's

Om de loop van een programma beter in beeld te brengen, maken we soms gebruik van stroomschema's. In de praktijk maakt men zelfs vaak eerst zo'n stroomschema om dit daarna pas in een programma te vertalen of zoals men soms zegt 'in een programma te coderen'.

De volgende tekens worden in stroomschema's gebruikt:

teken	*betekenis*
	symbool om begin- en eindpunt van een programma aan te geven.
	symbool om een bewerking zoals A=3 aan te geven.
	symbool voor een in- of uitvoerinstructie (PRINT/INPUT).
	symbool om een IF...THEN-instructie aan te geven.

Met behulp van deze symbolen kunnen we nu ons laatste programma als volgt in kaart brengen.



IF...THEN ...ELSE

MSX-BASIC staat ook de IF...THEN-instructie toe, waarbij deze met de term ELSE is uitgebreid. We illustreren deze uitbreiding weer aan de hand van een eenvoudig voorbeeld:

```
10 INPUT K
20 IF K=7 THEN PRINT "K IS 7" ELSE
                PRINT "K IS ONGELIJK AAN 7"
30 END
```

De instructie na THEN wordt alleen uitgevoerd als de aangegeven voorwaarde (K=7?) opgaat, in alle andere gevallen wordt de instructie na ELSE uitgevoerd. Ook hier geldt weer, evenals bij THEN, dat we meer instructies na ELSE mogen aangeven en wel gescheiden door het teken dubbelepunt.

Nabeschouwing

Het karakter van programma's waarin IF...THEN-instructies zijn opgenomen is verschillend van de programma's waarin recht toe, recht aan formules worden uitgerekend. Het feit dat nu keuze-momenten worden ingevoerd, maakt dat het programma logische beslissingen voor ons neemt.

In het algemeen kan men zien dat keuze-momenten nodig zijn als:

a. Afhankelijk van bepaalde uitkomsten, variabelen specifieke waarden moeten krijgen. Denk bijvoorbeeld eens aan renteberekeningen, waarin de feitelijke rente door een aantal factoren wordt bepaald.

b. Er sprake is van een combinatie van voorwaarden, op grond waarvan een specifieke actie moet worden uitgevoerd. We kunnen hier als voorbeeld uitgaan van de vele denkspelletjes. De computer kan ons 'denkwerk' overnemen, op voorwaarde dat we de wijze om tot een oplossing te komen, ook in een reeks instructies kunnen vertalen.
Hoofdstuk 13 gaat hier dieper op in.

c. Er sprake is van een controle op het uitvoeren van het programma zelf. Een voorbeeld wordt in appendix A gegeven, waarbij door middel van een IF...THEN-instructie wordt nagegaan of alle waarden van een zgn. bestand zijn ingelezen.

Het is ten slotte verstandig om nog enige woorden te wijden aan de constructie van de IF...THEN-instructie zelf.
In veel situaties willen we een reeks instructies uitvoeren als aan een bepaalde voorwaarde voldaan is. Het programma wordt dan in feite eenvoudiger van opzet, als we de voorwaarde in de IF...THEN-instructie omkeren.

Zo is:

```
IF K=5 THEN regelnr. a
.....
.....
.....
GOTO b
```

regelnr. a
reeks uit te voeren
instructies als K=5

regelnr. b

Gelijk aan:

```
IF K<>5 THEN regelnr. b
   reeks uit te voren
   instructies
```

regelnr. b

Bij het laatste voorbeeld is de voorwaarde K=5 'omgedraaid' in K<>5. We zien nu dat de reeks uit te voeren instructies direct onder de IF...THEN-instructie zelf komt te staan en bovendien wordt nu een GOTO-instructie vermeden.

Het denken in IF...THEN- en GOTO-instructies kan soms verwarrend werken. Vaak doet zich de situatie voor die in woorden als volgt is aan te geven:

ZOLANG voorwaarde voldoet DOE reeks instructies.

Het is verstandig om deze constructie bij het opstellen van een programma aan te houden en pas in de laatste instantie de omzetting in IF...THEN en GOTO-instructies te gebruiken.

Een voorbeeld:

Stel dat we een reeks in te voeren getallen willen optellen en wel zolang de in te voeren getallen positief zijn.

Het 'programma' zou dan luiden:

```
INPUT K
ZOLANG K>0 DOE
           SOM=SOM+K
           INPUT K
```

Vertaald in BASIC wordt dit programma:

```
10 INPUT K
20 IF K <=0 THEN STOP
30 SOM=SOM+K
40 GOTO 10
```

(N.B. de instructie STOP heeft het onderbreken van het pro-

gramma tot gevolg). Hierbij plaatsen we nog de opmerking dat het bij dit voorbeeld verstandig is, de waarde van SOM aan het begin van het programma op 0 te stellen (het z.g. geven van een initiële waarde), bijvoorbeeld door:

```
5 SOM=0
```

Het werken met de ZOLANG...DOE-constructie kan aanmerkelijke voordelen opleveren bij het opstellen van programma's, omdat de structuur hiermee veel duidelijker wordt. De combinatie van IF...THEN en GOTO-instructies maakt een programma immers al snel onoverzichtelijk.

FOR...NEXT De FOR...NEXT-instructie dienen we op te vatten als een bijzonder handig programmeergereedschap, waarmee we kunnen aangeven dat een bepaalde serie instructies een aantal keren (herhaald) moet worden afgewerkt.
Het volgende voorbeeld verduidelijkt dit:

```
10 FOR A=1 TO 5
20 PRINT A; "DIT IS EEN DEMONSTRATIE"
30 NEXT A
```

Resultaat:

```
1 DIT IS EEN DEMONSTRATIE
2 DIT IS EEN DEMONSTRATIE
3 DIT IS EEN DEMONSTRATIE
4 DIT IS EEN DEMONSTRATIE
5 DIT IS EEN DEMONSTRATIE
```

De instructies die herhaald moeten worden uitgevoerd, worden steeds ingesloten tussen de regel die met FOR begint en de regel die met NEXT eindigt, dus in schema:

```
.. FOR naam variabele=...
.. ...  \
.. ...   } deze instructies worden herhaald uitgevoerd
.. ...  /
.. NEXT naam variabele
```

In ons geval gaat het maar om één instructie, namelijk de instructie van regel 20. Hoe vaak de instructies worden uitgevoerd, kan men geheel en al bepalen door wat in de regel, die met FOR begint, omschreven is.

Bij

```
FOR A=1 TO 5
```

zal het gedeelte precies 5 maal herhaald worden, waarbij A dan

achtereenvolgens de waarde 1, 2, 3, 4 en 5 aanneemt. De variabele die na FOR wordt aangegeven noemen we dan ook heel toepasselijk 'de tel-variabele'.

In dit geval wordt A steeds met 1 verhoogd. We mogen echter ook een stap-grootte opgeven en wel door de regel met FOR met de term STEP uit te breiden:

```
10 FOR A=1 TO 6 STEP 2
20 PRINT A; "DIT IS EEN DEMONSTRATIE"
30 NEXT A
40 PRINT A
```

Resultaat:

```
1 DIT IS EEN DEMONSTRATIE
3 DIT IS EEN DEMONSTRATIE
5 DIT IS EEN DEMONSTRATIE
7
```

We zien hoe de tel-variabele A hier achtereenvolgens de waarde 1, 3 en 5 krijgt en voor deze waarde 'de lus' steeds doorlopen wordt.

Als A gelijk aan 7 is, wordt de aangegeven grens in regel 10 (6) overschreden en vervolgt de computer het programma met regel 40. Nogmaals laten we de waarde van A afdrukken en zo wordt de waarde 7 afgedrukt.

De moraal die wij uit dit programmaatje kunnen trekken is duidelijk ... pas op als we na een FOR...NEXT-instructie de 'tel-variabele' verder in het programma gebruiken; deze heeft niet noodzakelijkerwijze de eindwaarde die in de regel met FOR wordt aangegeven!

Nog wat voorbeelden:

- `FOR A=10 TO 5 STEP-1`
  de reeks instructies wordt nu uitgevoerd voor
  A= 10, 9, 8, 7, 6 en 5
- `FOR A= -15 TO 15 STEP 3`
  de reeks instructies wordt nu uitgevoerd voor
  A=− 15, −12, −9, −6, −3, 0, 3, 6, 9, 12 en 15
- `FOR A= 1.4 TO 1.7 STEP .05`
  de reeks instructies wordt nu uitgevoerd voor
  A= 1.4, 1.45, 1.50, 1.55, 1.60, 1.65 en 1.70
- `FOR A=B TO C STEP K/L`

We zien hier hoe zowel de begin- en grenswaarde als de stapgrootte door variabelen worden aangegeven. De stapgrootte wordt zelfs door een rekenkundige uitdrukking (deling van 2 variabelen) aangegeven. Het is inderdaad toegestaan om begin-grens- en stapwaarde door dit soort uitdrukkingen aan te geven.

- FOR...NEXT-instructies mogen op zich ook weer FOR...NEXT-instructies omvatten. Voorwaarde is dan wel dat de ene instructie de andere volledig omvat. Zo is de constructie

```
FOR K=1 TO 10
...
FOR J=1 TO 5   } binnenlus wordt geheel
...            } omvat door buitenlus
...
NEXT J
NEXT K
```

wel toegestaan en is de constructie:

```
FOR K=1 TO 10 )
...           )
FOR J=1 TO 5  } 'K-lus'  )
...           )          } 'J-lus'
...           )          )
NEXT K        )          )
...                      )
NEXT J                   )
```

verboden. Hier is de binnen-lus immers niet volledig door de buitenlus omvat.

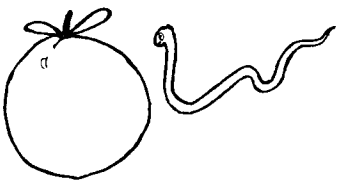
ON...GOTO De laatste instructie die we in dit hoofdstuk bespreken is de ON...GOTO-instructie. Deze instructie doet aan een keuze-schakelaar denken.

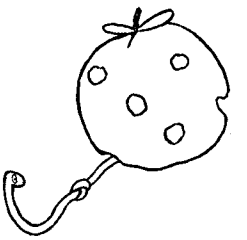
Op grond van de waarde die tussen ON en GOTO staat wordt een bepaalde sprong uitgevoerd.
Een voorbeeld verduidelijkt dit:

```
10 INPUT K
20 ON K GOTO 30,50
30 PRINT "K IS 1"
40 GOTO 70
50 PRINT "K IS 2"
60 GOTO 70
70 END
```

Als K gelijk is aan 1 zal een sprong naar het eerste aangegeven regelnummer volgen. Is K gelijk aan 2 dan volgt een sprong naar het tweede regelnummer.

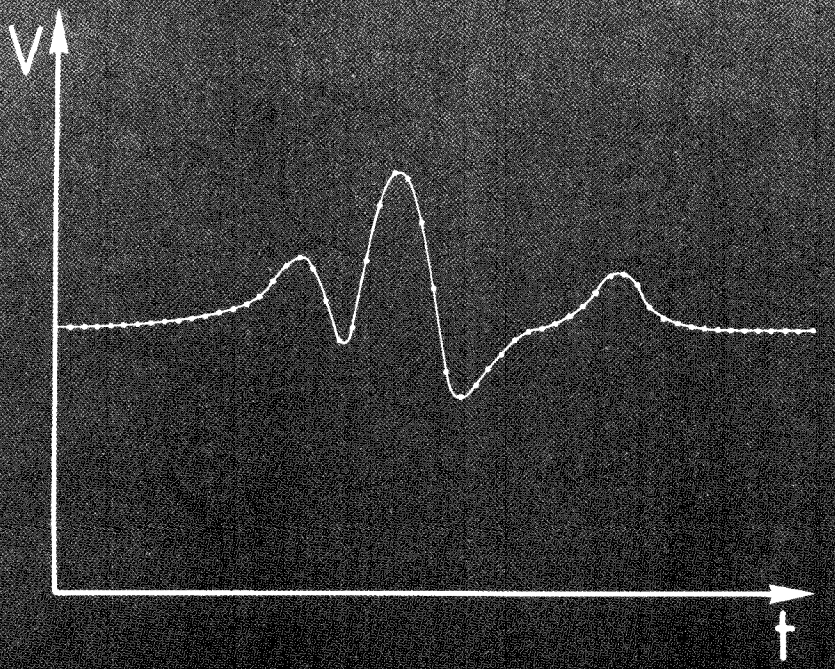
In dit voorbeeld werden na GOTO (regel 20) maar 2 regelnummers opgegeven, in feite kunnen dit er veel meer zijn. Merk tevens op dat de verschillende programma-gedeeltes waar naartoe gesprongen wordt weer worden afgesloten met een GOTO-instructie. Het gebruik van GOTO-instructies is hier weer onvermijdelijk.





indien lussen kunnen voorkomen pas dan extra op...

De array introduceert de mogelijkheid om op eenvoudige wijze een groot aantal variabelen in een programma te introduceren. De array wordt zowel in administratieve als rekenkundige toepassingen veel gebruikt. De combinatie van de FOR...NEXT-instructie en de array blijkt in de praktijk veel voor te komen. Aan het eind van dit hoofdstuk wordt dat geïllustreerd aan de hand van een klassiek probleem: het vinden van priemgetallen.



11. De array (de lijst)

Inleiding

In een aantal gevallen bestaat de behoefte om in een programma over een groot aantal variabelen te kunnen beschikken.

We zouden bijvoorbeeld kunnen denken aan een programma voor een voorraadadministratie. Als we voor ieder artikel dat we beheren in een programma een afzonderlijke variabele moeten introduceren, wordt het programma uiteraard zeer groot.

In het nu volgende bespreken we hoe BASIC deze problemen ondervangt en wel door de mogelijkheid te bieden via één instructie een groot aantal variabelen te introduceren.

DIM-instructie

De instructie waarmee men eenvoudig een groot aantal variabelen introduceert, is de DIM-instructie. Deze heeft steeds de volgende vorm:

```
DIM    naam (aantal)
```

bijvoorbeeld:

```
DIM A (100)
```

Met bovenstaande instructie introduceert BASIC een reeks van 101 variabelen. De *naam* van ieder van deze variabelen bestaat dan uit de naam, zoals die in de DIM-instructie voorkomt met een getalsaanduiding tussen haakjes. Zo vormen

```
A(0) A(1) A(2) A(3) ... A(99) A(100)
```

precies de 101 variabelen die behoren bij de instructie DIM A(100). We spreken in zo'n geval ook wel eens van de array A, die bij dit voorbeeld bestaat uit de elementen A(0) t/m A(100).

Al die variabelen kunnen we op gelijke wijze als onze 'gewone' variabelen gebruiken.

Een voorbeeld:

```
10 DIM A(100)
20 A(3)=6
30 A(27)=5
40 A(98)=A(3)+A(27)
50 PRINT A(98)
```

Resultaat:

```
11
```

Dit op zich wat merkwaardig aandoende programma toont dat we array-elementen A(3), A(27) en A(98) gebruikt hebben, als ging het om 'gewone' variabelen met bijvoorbeeld de namen A, B en C.

De mogelijkheden van array-variabelen worden vooral bepaald door het feit dat we het nummer tussen haakjes (vaak indexnummer genoemd) ook indirect mogen aangeven.
Voorbeelden:

`A(93)`	nummer wordt hier direct door een getal aangegeven.
`A(K)`	nummer wordt indirect door een variabele aangegeven.
`A(K+3)`	nummer wordt hier indirect door een uitdrukking aangegeven.

Het volgende programma geeft een eenvoudige illustratie:

```
10 DIM B(20)
20 FOR K=1 TO 20
30 B(K)=K
40 NEXT K
50 FOR K=20 TO 1 STEP -1
60 PRINT B(K);
70 NEXT K
```

Resultaat:

```
20  19  18  17  16  15  14  13  12
11  10  9  8  7  6  5  4  3  2  1
```

In regel 10 wordt array B geïntroduceerd. Regels 20 t/m 40 geven aan dat aan B(1) de waarde 1, B(2) de waarde 2, etc. moet worden toegekend. Dat die waarden inderdaad zijn toegekend, bewijzen de regels 50 t/m 70, als gevolg waarvan de waarden van de variabelen B(20), B(19), B(18), etc. worden afgedrukt. We merken tevens op hoe handig de FOR...NEXT-instructie in combinatie met arrays kan worden gebruikt.

We maken bij de array nog de volgende opmerkingen:

- als in een programma een array-variabele bijv. A(6) wordt gebruikt, zonder dat in dat programma een DIM-instructie is vermeld, neemt de computer een array met een bovengrens van 10 aan. Bij het voorbeeld gaat de computer er dus van uit dat de variabele A(6) behoort bij array A die als bovengrens 10 heeft (DIM A(10)).

- we mogen ook zogenaamde meer-dimensionale arrays in een programma gebruiken, d.w.z. tussen de haakjes worden nu meer getallen aangegeven. Zo introduceert de instructie:

```
DIM A(3,3)
```

de variabelen:

```
A(0,0)    A(0,1)    A(0,2)    A(0,3)
A(1,0)    A(1,1)    A(1,2)    A(1,3)
A(2,0)    A(2,1)    A(2,2)    A(2,3)
A(3,0)    A(3,1)    A(3,2)    A(3,3)
```

Meestal lopen we bij de introductie van meer-dimensionele arrays snel tegen de beperkingen van het geheugen aan.

- We mogen op één regel méér arrays introduceren.
Bijvoorbeeld:

```
10 DIM A(100), P(300), Z(50), P!(30), B%(5)
```

- we kunnen de gereserveerde ruimte naderhand indien gewenst, vrijmaken (bijvoorbeeld in verband met geheugenproblemen) door de instructie ERASE te gebruiken.
Bijvoorbeeld:

```
ERASE A,P
```

Het berekenen van priemgetallen. Een voorbeeld

Een priemgetal is een getal dat alleen deelbaar is door zichzelf en door 1. Uit deze definitie volgt dat 1 een priemgetal is. Het daarop volgende getal 2 is ook een priemgetal, het is immers alleen deelbaar door 2 en 1. Ook 3 is een priemgetal, 4 echter niet, want dit getal kan men door het priemgetal 2 delen. Het daaropvolgende getal 5 is weer een priemgetal, terwijl 6 zowel door 2 als door 3 deelbaar is en daarom niet in de verzameling der priemgetallen thuishoort.
Laten we nu eens kijken of we een recept kunnen verzinnen om bijvoorbeeld bij de getallen 1 t/m 250 na te gaan, welk getal een priemgetal is. Om dit recept op te stellen, kijken we eerst hoe we bij de getallen 1 t/m 10 kunnen nagaan welke getallen priemgetal zijn.

We schrijven daartoe eerst de getallen op:

1 2 3 4 5 6 7 8 9 10

Nu gaan we van links naar rechts priemgetallen zoeken en steeds als we een priemgetal vinden, strepen we alle veelvouden van dat priemgetal door. We weten dat 1 een priemgetal is en ook het volgende getal (2) is een priemgetal. Vervolgens strepen we alle veelvouden van 2 door, dus:

1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~

We kijken nu naar het volgende getal, dat is 3. Hier staat nog geen streepje door, ergo het is een priemgetal. We strepen nu ook alle veelvouden van 3 door:

1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~

Het volgende getal is 4. Hier staat al een streepje door en het is dus geen priemgetal. Het volgende getal is 5 waar geen streepje door staat. Ergo 5 is een priemgetal. Het volgende getal (6) staat weer een streep door, etc.

We zien hoe we aan dit proces van streepjes door getallen te zetten, als het ware alle priemgetallen uit onze oorspronkelijke reeks getallen zeven. Dit recept kenden de oude Grieken trouwens al en het staat bekend als 'de zeef van Eratosthenes'.

Het nu volgende programma is volledig gebaseerd op het besproken recept.

```
10 DIM N(250), P(250)
20 FOR J=2 TO 250
30 IF N(J)<0 THEN GOTO 70
40 AP=AP+1
50 P(J)=J
60 FOR K=J TO 250 STEP J:N(K)=-1:NEXT K
70 NEXT J
80 REM AFDRUKKEN
90 FOR K=1 TO AP: PRINT P(K);:NEXT K
```

Resultaat:

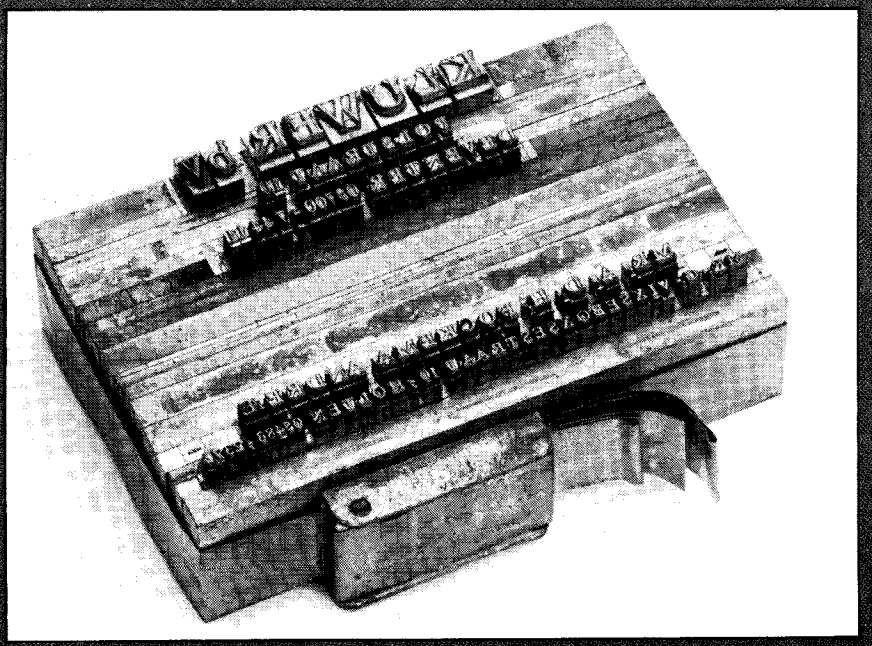
```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59
61 67 71 73 79 83 89 97 101 103 107 109 113 127
131 137 139 149 151 157 163 167 173 179 181 191
193 197 199 211 223 227 229 233 239 241
```

In regel 10 worden twee arrays geïntroduceerd. Array N is om te onthouden door welke getallen we streepjes hebben geplaatst. Dit doen we in dit programma door aan zo'n getal de waarde −1 toe te kennen (regel 60). Array P is om de gevonden priemgetallen op te slaan. Dit zijn er uiteraard minder dan 250 maar aangezien we het juiste aantal van tevoren niet kennen, hebben we deze array maar 'ruim' bemeten.

Regel 20 geeft het van links naar rechts opschuiven aan. Staat er door een getal een streepje (regel 30) dan wordt een volgend getal getest. Vindt men geen streepje dan wordt AP met 1 verhoogd.

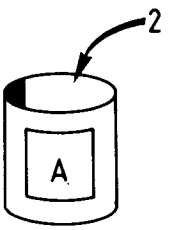
Met AP tellen we het aantal gevonden priemgetallen. Vervolgens wordt het gevonden priemgetal in de array P opgeslagen (regel 50) en worden de veelvouden doorgestreept (regel 60). De laatste regels zijn opgenomen om onze 'door de zeef' opgevangen priemgetallen af te drukken.

In dit hoofdstuk gaan we in op de mogelijkheid om reeksen letters aan variabelen toe te kennen. Zo'n reeks letters of, meer in 't algemeen, zo'n reeks tekens noemen we een string en de variabelen waaraan we zulke strings toekennen, noemen we string-variabelen. Er zal getoond worden dat er met betrekking tot strings maar één bewerkings-teken is, namelijk het teken +. Daarnaast zijn er wel veel functies die we bij strings kunnen toepassen.



12. Strings... spelen met letters

Inleiding Tot nu toe hebben we gezien dat we aan een variabele een getal kunnen toekennen. We hadden bij het uitleggen, hoe dit in zijn werk ging, gebruik gemaakt van een voorstelling met een doos. Zo werd de instructie LET A=2 als volgt aanschouwelijk gemaakt:



We kunnen aan variabelen ook reeksen letters toekennen. Een dergelijke reeks letters noemen we dan een string en dit is tevens de reden dat we dergelijke variabelen, string-variabelen noemen.

Uiteraard zullen we door een speciale schrijfwijze duidelijk moeten maken dat het om een string-variabele gaat. De afspraak is eenvoudig:

als we onmiddellijk na de naam van een variabele een dollarteken plaatsen, gaat het om een string-variabele.

Zo zijn A$, EERSTE$ en TEKST$ voorbeelden van namen die kennelijk betrekking hebben op string-variabelen.

De tekst die we aan zo'n string-variabele kunnen toekennen, moeten we altijd tussen aanhalingstekens plaatsen.
Het volgende programmaatje toont een voorbeeld:

```
10 LET A$= "DIT IS EEN STRING"
20 PRINT A$
```

Regel 10 kunnen we weer d.m.v. een plaatje verduidelijken.

Merk op dat in regel 10 deze tekst inderdaad tussen aanhalingstekens is geplaatst. Als we nu het resultaat van het programma bekijken, zal opvallen dat deze aanhalingstekens niet worden afgedrukt, als resultaat verschijnt:

```
DIT IS EEN STRING
```

We kunnen dus concluderen dat de aanhalingstekens geen deel uitmaken van de string zelf.

Hieronder volgen nog een aantal voorbeelden van strings die we aan een stringvariabele mogen toekennen.

`"HALLO JAN"`	dit is een string, die bestaat uit negen tekens, de spatie tussen HALLO en JAN telt namelijk ook mee!
`"164"`	dit is een string die bestaat uit drie tekens. Het feit dat het hier om cijfers gaat, is eigenlijk erg verraderlijk want de computer ziet slechts tekens en geen cijfers! Straks zullen we zien hoe we in dit soort bijzondere gevallen, zo'n 'schijngetal' in een echt getal kunnen omzetten.
`""`	dit is een string die bestaat uit geen enkel teken, de aanhalingstekens zijn namelijk direct na elkaar geplaatst. We zullen straks voorbeelden zien waarom zo'n 'lege-string' toch voordelen kan bieden. Deze string wordt ook wel eens aangeduid met de term nul-string.

Concateneren... of het aan elkaar rijgen van strings

Met behulp van het optelteken + kunnen we aangeven dat twee strings aan elkaar geregen moeten worden. Computerdeskundigen spreken dan van concateneren.

Een voorbeeld verduidelijkt dit:

```
10 A$="MSX-"
20 B$="BASIC"
30 C$=A$+B$
40 PRINT C$
```

Resultaat:

```
MSX-BASIC
```

We zien hoe in regel 10 en 20 de strings 'MSX-' en 'BASIC' aan A$ en B$ worden toegekend. In de daarop volgende regel worden deze strings 'opgeteld' en aan C$ toegekend.

Stringfuncties

Tot nu toe konden we nog niet zoveel met strings doen, hoogstens konden we twee of meer strings aan elkaar rijgen om op deze wijze een langere string te verkrijgen.

Gelukkig kent MSX-BASIC een groot aantal functies waarmee we allerlei handelingen met betrekking tot strings kunnen uitvoeren.
We kunnen deze stringfuncties in twee groepen verdelen:

groep 1: de functies die weer een string als resultaat hebben. De namen van deze functies eindigen steeds met $.

groep 2: de functies die als resultaat een getal opleveren. De meeste functies uit deze groep maken gebruik van een bepaalde afspraak. Deze afspraak heeft betrekking op het geven van getallen aan letters en andere tekens. De namen van de functies uit deze groep eindigen niet op een dollar-teken.

LEN

Deze functie eindigt niet op een dollar-teken, kennelijk gaat het hier om een functie die een getal oplevert. Inderdaad, met behulp van de LEN-functie bepalen we namelijk het aantal tekens van de string.
We geven direct een voorbeeld:

```
10 A$="MSX"
20 B=LEN (A$)
30 PRINT B
```

Resultaat:

```
3
```

In regel 10 wordt aan A$ de string 'MSX' toegekend. Deze string bestaat dus uit drie tekens. In de volgende regel wordt aan B het resultaat van LEN(A$) toegekend met andere woorden het aantal tekens waaruit A$ bestaat.
Het resultaat bevestigt dit.

**LEFT$** De LEFT$-functie is de eerste string-functie waarmee we uit een string als het ware een gedeelte kunnen snijden. Zo'n gedeelte van een string noemen we dan een substring.
Het volgende voorbeeld illustreert de LEFT$-functie:

```
10 A$="MSX-BASIC"
20 B$=LEFT$(A$,3)
30 PRINT B$
```

Resultaat:

```
MSX
```

De LEFT$-functie van regel 20 geeft aan dat van string A$ de eerste drie tekens moeten worden genomen. Het resultaat bevestigt dit.

**RIGHT$** Weer een string-functie om een substring te verkrijgen. Ging het bij de LEFT$-functie om de eerste tekens van een aangegeven string, bij de RIGHT$-functie gaat het om de laatste tekens. Weer een voorbeeld:

```
10 A$="MSX-BASIC"
20 B$=RIGHT$(A$,5)
30 PRINT B$
```

Resultaat:

```
BASIC
```

In regel 20 wordt door middel van RIGHT$ (A$,5) aangegeven dat aan B$ de laatste vijf tekens van A$ moeten worden toegekend. Het volgende voorbeeld geeft nog een illustratie van zowel de LEFT$- als de RIGHT$-functie.

```
10 A$="MSX-BASIC"
20 N=LEN(A$)
30 FOR K=1 TO N
40 B$=LEFT$(A$,K)
50 C$=RIGHT$(A$,K)
60 PRINT B$,C$
70 NEXT K
```

Resultaat:

```
M             C
MS            IC
MSX           SIC
MSX-          ASIC
MSX-B         BASIC
MSX-BA        -BASIC
MSX-BAS       X-BASIC
MSX-BASI      SX-BASIC
MSX-BASIC     MSX-BASIC
```

**MID$** Kunnen we met LEFT$ en RIGHT$ respectievelijk het eerste en het laatste deel van een string 'uitsnijden', met behulp van MID$ kunnen we een willekeurig gedeelte uit een string snijden. Daartoe geven we natuurlijk eerst op van welke string we uitgaan, vervolgens wordt het teken opgegeven van waar de sub-string dient te worden uitgesneden. Eventueel geeft men dan nog een derde getal op. Dit getal geeft dan het aantal tekens aan, waaruit de substring zal bestaan. Laten we dit laatste getal weg, dan worden alle tekens vanaf het aangegeven punt genomen. Het volgende voorbeeld verduidelijkt dit:

```
10 A$="MSX-BASIC"
20 B$=MID$(A$,2,3)
30 C$=MID$(A$,2)
40 PRINT B$
50 PRINT C$
```

Resultaat:

```
SX-
SX-BASIC
```

We zien hoe in regel 20, een substring bestaande uit drie tekens vanaf teken 2 wordt opgebouwd. In regel 30 wordt in de MID$-functie het derde getal weggelaten. We zien hoe nu 'de rest' vanaf teken 2 als substring wordt genomen.

ASC Deze functie eindigt niet op een dollar-teken, waaruit we kunnen concluderen dat het resultaat kennelijk een getal is.

Dit is juist: het getal waar het hier om gaat, is de zogenaamde ASCII-code. Volgens een bepaalde afspraak wordt aan ieder teken (letter, cijfer, leesteken) een getal toegekend. (zie appendix B).

Voor de hoofdletters A t/m Z ziet men in onderstaande tabel deze ASCII-getallen.

ASCII-waarden voor de hoofdletters A t/m Z.

letter	*getal*	*letter*	*getal*	*letter*	*getal*	*letter*	*getal*
A	65	H	72	O	79	V	86
B	66	I	73	P	80	W	87
C	67	J	74	Q	81	X	88
D	68	K	75	R	82	Y	89
E	69	L	76	S	83	Z	90
F	70	M	77	T	84		
G	71	N	78	U	85		

Uit deze tabel kunnen we opmaken dat bijvoorbeeld A met het getal 65 overeenkomt en K met 75. Het gegeven, dat met iedere letter ook een getal correspondeert, blijkt in de praktijk in bijzonder veel situaties van pas te komen. Denk maar eens aan het sorteren van namen. Door nu getallen te vergelijken kunnen we de klus klaren.

Het volgende voorbeeld toont een voorbeeld van de ASC-functie:

```
10 N=ASC("B")
20 PRINT N
```

Resultaat:

```
66
```

We zien hoe hier de ASCII-waarde van B door middel van de ASC-functie wordt bepaald.

**CHR$** Met behulp van de CHR$-functie kunnen we het teken bepalen dat bij een zekere ASCII-waarde hoort.
Een voorbeeld:

```
PRINT CHR$(66)
```

Resultaat:

```
B
```

Bezien we dit voorbeeld naast het vorige voorbeeld dan is duidelijk dat we de CHR$-functie als een soort 'omgekeerde' ASC-functie kunnen beschouwen.

VAL Met behulp van deze functie kunnen we een string die een getal aangeeft ook werkelijk in dit getal omzetten.
Voorbeeld:

```
10 A$="123"
20 B=VAL(A$)
30 PRINT B
```

Resultaat:

```
123
```

Merk op dat A$ een string-variabele is. In regel 20 vindt dan de feitelijke omzetting naar het getal 123 plaats.

**STR$** De STR$-functie kunnen we als een soort omgekeerde VAL-functie opvatten: we zetten er een getal mee om in een string die uit dezelfde tekens bestaat.

Voorbeeld:

```
10 A=123
20 B$=STR$(A)
30 PRINT B$
```

Resultaat:

```
123
```

We zien hoe aan de getal-variabele A het getal 123 wordt toegekend. Vervolgens wordt dit getal door middel van de STR$-functie als string aan de variabele B$ toegekend.

Hierbij merken we nog op dat als een getal positief is, er bij het omzetten naar een string altijd een spatie voor de string geplaatst wordt.

**STRING$** De functie STRING$ gebruiken we om een bepaald teken, aangegeven als ASCII-code, een aantal malen achter elkaar te plaatsen.
De algemene vorm is:

```
STRING$ (K,ASCII-waarde)
```

Met K geven we dan aan hoeveel maal het gegeven teken herhaald moet worden. We kunnen eventueel ook een string aangeven. Bij de STRING$-functie wordt dan naar het *eerste* teken van de string gekeken.
Voorbeelden:

```
PRINT STRING$ (5,65)        resultaat: AAAAA
PRINT STRING$ (5,"MSX")     resultaat: MMMMM
```

**SPACE$** Met de SPACE$-functie kunnen we een gegeven aantal spaties afdrukken.
Voorbeeld:

```
10 FOR K=1 TO 5
20 A$=SPACE$(K)
30 PRINT "A"; A$; "B"
40 NEXT K
```

Resultaat:

```
A B
A  B
A   B
A    B
A     B
```

INSTR$ (X$,Y$) Dit is een krachtige zoek-instructie. We bepalen er namelijk mee op welke plaats de string Y$ in de string X$ voorkomt. Voorbeeld:

```
10 X$="WERELD"
20 Y$="REL"
30 K=INSTR(X$,Y$)
40 PRINT K
```

Resultaat:

```
3
```

Inderdaad, de string REL komt vanaf de derde positie in onze uitgangsstring WERELD voor.

INKEY$ en een spelletje

INKEY$ is in feite geen functie maar een variabele. Als de computer bij het 'runnen' van een programma de term INKEY$ aantreft, 'kijkt' de computer momentaan naar het toetsenbord. Als op dat moment een toets wordt ingedrukt, wordt het betreffende teken aan INKEY$ toegekend. Als we op dat moment geen toets indrukken, zal de lege string "" aan INKEY$ worden toegekend.

Het volgende programma illustreert hoe INKEY$ voor een eenvoudig reactiespelletje gebruikt wordt:

```
10 PRINT "DRUK OP EEN TOETS"
20 PRINT "ALS HET SCHERM"
30 PRINT "DE WAARDE 25 TOONT"
40 FOR K=1 TO 50
50 PRINT K
60 A$=INKEY$
70 IF A$ <> "" THEN GOTO 90
80 NEXT K
90 PRINT "EINDE"
```

Nog wat opmerkingen In het voorafgaande hebben we de belangrijkste functies met betrekking tot strings besproken. Een overzicht van alle toegestane instructies treft men in appendix E aan.
Tenslotte plaatsen we nog de volgende opmerkingen:

- We mogen, evenals dat bij getallen is toegestaan, ook stringarrays introduceren, bijvoorbeeld:

```
DIM A$(100)
```

- De computer heeft bij het aanzetten een bepaalde geheugenruimte gereserveerd voor de opslag van strings. Standaard kunnen zo 200 tekens worden opgeslagen. We kunnen deze ruimte

vergroten met een zogenaamde CLEAR-instructie, bijvoorbeeld:

```
CLEAR 500
```

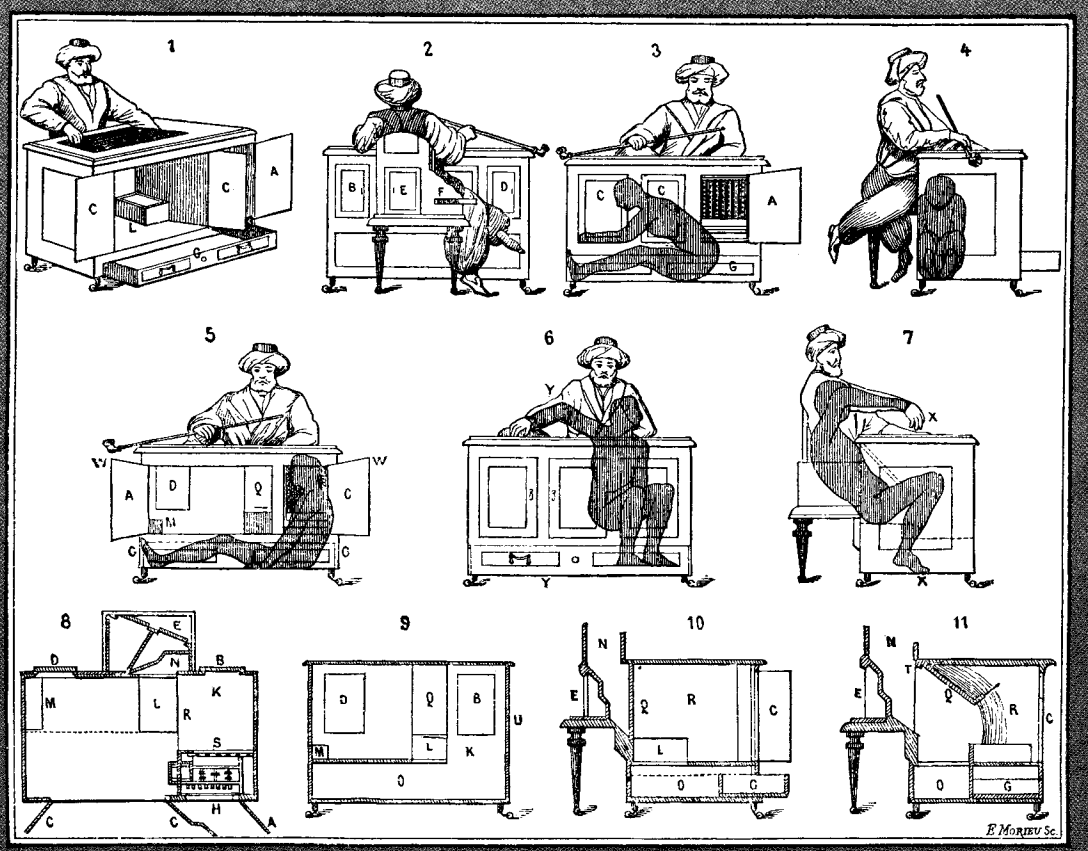
Met deze CLEAR-instructie reserveren we een ruimte voor de opslag van 500 tekens.

- De string die we aan een string-variabele mogen toekennen, heeft een maximale lengte van 255 tekens.

In dit hoofdstuk komen geen nieuwe instructies aan bod. We zullen onze tot nu toe behandelde programmeergereedschappen gebruiken om te zien in hoeverre we de vraag 'kunnen computers denken?' kunnen beantwoorden. In dit hoofdstuk worden daartoe twee programma's besproken.

Bij het eerste programma daagt de computer ons uit een denkspelletje te spelen.

Bij het tweede programma krijgen we te maken met patroonherkenning. In het voorbeeld gaat het om het herkennen van de taal waarin een zin is gesteld.



13. Intermezzo: Kunnen computers denken?

Inleiding

'Kunnen computers denken?' is een vraag die men in veel artikelen aantreft. Dat die vraag gesteld wordt, is niet vreemd want ieder van ons vergelijkt een computer toch stilletjes met de hersenen van de mens.

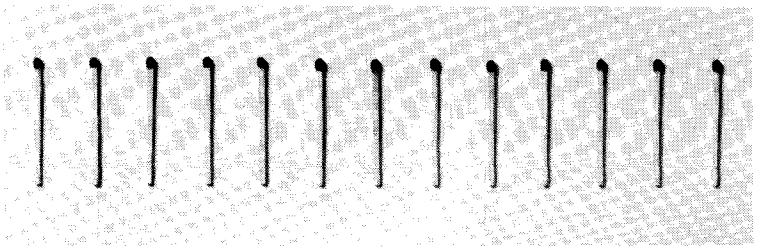
En wat bijvoorbeeld te denken van de schaakprogramma's, die we op onze computer kunnen verwerken. Vormen die programma's geen prachtig voorbeeld van het kunnen denken, zoals wij mensen doen?

Het plaatje dat we zien, toont ons de beroemde schakende Turk. Achteraf weten we dat het hier om bedrog ging. Wat als schaakautomaat aan het publiek werd gepresenteerd, was in feite niet meer dan een slimme verpakking van een goede schaker. Maar het feit dat dergelijke automaten werden gepresenteerd, geeft aan hoe uitdagend het toch kennelijk steeds weer voor de mens is, om zijn denken in automaten over te brengen.

Een eenvoudig denkspel?

Laten we, om de vraag 'kunnen computers denken?' te kunnen beantwoorden, eens dieper op een bepaald denkspelletje ingaan.

Het spelletje waar het hier om gaat, is het spel van 13 lucifers. Twee spelers mogen om beurt 1, 2 of 3 lucifers pakken ... of liever, om beurten mogen de computer en u 1, 2 of 3 lucifers pakken. Het doel is eenvoudig: wie de laatste lucifer pakt, heeft verloren.



U moet beginnen. Stel, u pakt 2 lucifers, de computer pakt ook 2 lucifers en er resteren dus nog 9 lucifers. Voorzichtig pakt u nu 1 lucifer ... de computer pakt nu 3 lucifers; resultaat, nog 5 lucifers. Op dit moment zet u dat u al verloren heeft want wat u ook pakt, de computer zal een zodanig aantal lucifers pakken dat er nog 1 lucifer overblijft! Als u 2 lucifers pakt, neemt de computer ook 2 lucifers en er blijft nog maar 1 lucifer liggen. Als u er één pakt, neemt de computer er uiteraard 3, etc. Kortom u bent verloren en als u de truc *niet* doorziet, komt de computer natuurlijk best als een slim (in termen van denken) apparaat over. Enig denkwerk leert dat de computer bij de eerste twee keren altijd zo veel lucifers neemt dat er respectievelijk 9 en 5 lucifers overblijven. Of anders: de som van uw lucifers en die van de computer moet steeds 4 zijn.

Met deze kennis is het natuurlijk niet moeilijk meer om een programma op te stellen. Kijk maar eens naar het volgende programmaatje; het doet zijn taak voortreffelijk.

```
10 PRINT "U MOET BEGINNEN"
20 INPUT K
30 PRINT "IK NEEM ER... "; 4-K
40 PRINT "NU BENT U WEER"
50 PRINT "IK NEEM ER... "; 4-K
60 PRINT "UW LAATSTE BEURT!"
70 PRINT "IK NEEM NATUURLIJK"; 4-K
80 PRINT "HA HA ... U HEEFT VERLOREN"
```

Niet zo leuk, zult u misschien zeggen, een natuurlijke reactie als we de truc doorzien. Maar geldt dat niet steeds voor 'denkwerk' waarvan we kennelijk precies kunnen aangeven hoe het moet! Laten we daarom naar een volgend voorbeeld kijken.

Herkennen van Nederlandse en Engelse zinnen

Bij dit voorbeeld zal de computer uitmaken of u een Engelse dan wel een Nederlandse zin intypt!

Het vraag- en antwoordspel van de computer verloopt als volgt:

```
COMPUTER:  TOETS EEN LANGE NEDERLANDSE OF EN-
           GELSE ZIN IN (GEBRUIK HOOFDLETTERS)
       U:  PLEASE CAN YOU TELL ME IF YOU CAN
           SEE IF I SPEAK ENGLISH
COMPUTER:  ENGELS
```

Zonder het programma te tonen, moet u wel toegeven dat dit werkelijk een knappe prestatie is. Weer geldt, als u niet weet hoe de computer hier uitkomt, die computer als een werkelijke denkmachine naar voren komt. Welnu, we laten eens zien hoe de computer deze klus klaart.

Uitgangspunt is dat in lange Engelse zinnen doorgaans de letter y voorkomt, terwijl dat in Nederlandse zinnen niet het geval is... voelt u het al?

Wat we dus moeten doen, is kijken of er in zo'n zin de letter y voorkomt.

Stel dat het programma de volgende instructie bevat:

```
LINE INPUT A$
```

Dat wil zeggen, de gehele zin wordt als één string aan A$ toegekend.

We moeten dan vervolgens nagaan of de letter y in die zin voorkomt. We kunnen met behulp van de volgende constructie de letters een voor een uit de string lichten.

```
N=LEN(A$)
FOR K=1 TO N
B$=MID$(A$,K,1)
NEXT
```

Met N wordt het aantal tekens geteld en de daarop volgende constructie kent aan B$ achtereenvolgens de afzonderlijke tekens van de string A$ toe.

Om deze constructie uit te testen, kan men het volgende programmaatje eens intypen:

```
10 LINE INPUT A$
20 N=LEN(A$)
30 FOR K=1 TO N
40 B$=MID$(A$,K,1)
50 PRINT B$
60 NEXT K
```

Als we nu bijvoorbeeld COMPUTER intypen, zien we als resultaat:

```
C
O
M
P
U
T
E
R
```

Nu moeten we er nog voor zorgen, dat de computer ieder teken bekijkt en als dit inderdaad een Y is, moet de uitspraak 'ENGELS' volgen.

Kortom de instructie die we in ons programma nodig hebben, is:

```
IF B$='Y' THEN PRINT 'ENGELS'
```

In dit geval is het handig om ook de STOP-instructie na de PRINT-instructie te gebruiken. Immers als we eenmaal weten dat het om een Engelse zin gaat, hoeven we niet verder te zoeken. Hier volgt ons volledig programma:

```
10 PRINT "TOETS EEN LANGE NEDERLANDSE OF ENGELSE ZIN IN
(GEBRUIK HOOFDLETTERS)"
20 LINE INPUT A$
30 N=LEN(A$)
40 FOR K=1 TO N
50 B$=MID$(A$,K,1)
60 IF B$="Y" THEN PRINT "ENGELS": STOP
70 NEXT K
```

Wederom zult u achteraf het idee hebben gekregen enigszins beet genomen te zijn... zo eenvoudig was de oplossing.

Toch is dit programmaatje erg illustratief, omdat het ons toont hoe wij 'denkwerk' door middel van de IF...THEN-instructie kunnen inbouwen.

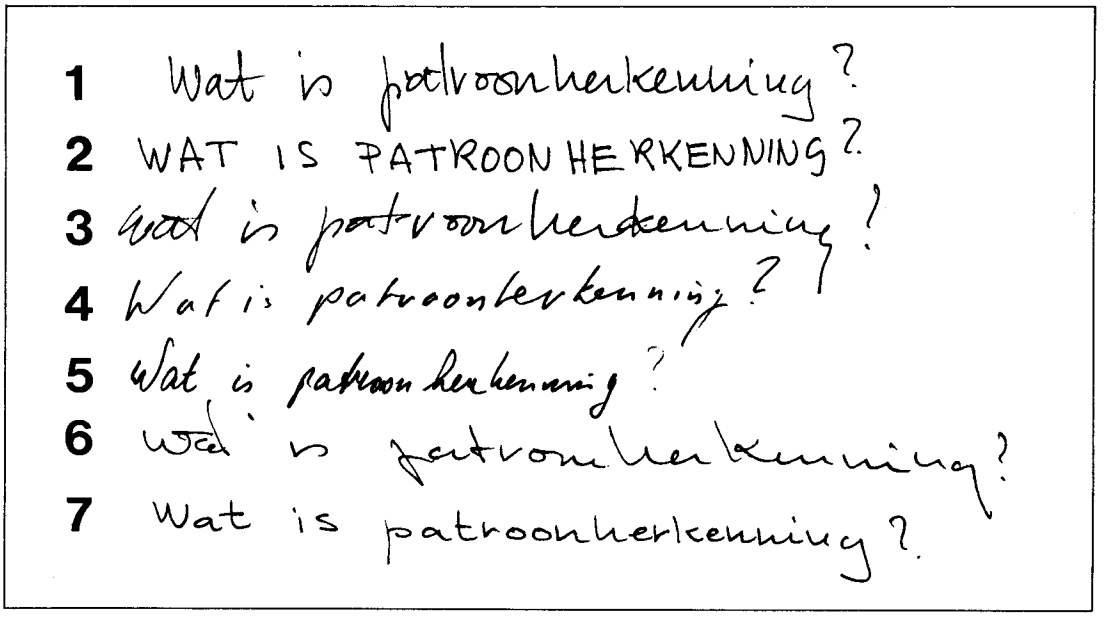
Patroonherkenning

Na de behandeling van dit programma is het nogmaals zinvol om de vraag 'kunnen computers denken?' te bestuderen. Het zal duidelijk zijn dat computers kunnen denken, als wij dat denken kunnen vertalen in het afwerken van een reeks BASIC-instructies. In veel gevallen komt dat denken neer op wat men wel eens met het woord 'patroonherkenning' aangeeft. Bij patroonherkenning gaat het om het constateren of een bepaald patroon al of niet aanwezig is. Om de daartoe noodzakelijke test(s) uit te voeren, dienen we uiteraard wel kenmerken van het te zoeken patroon te kennen. Zo was een kenmerk van een Engelse zin, dat de letter Y er relatief veel in voorkomt.

Bedenk dat wij zelf ook van dergelijke zoekmechanismen uitgaan. In onze schooltijd zijn ons allerlei Engelse woorden, zoals YOU, PLEASE, TELL ME en ENGLISH in het hoofd geprent. Als wij een willekeurige zin zien of horen, *herkennen* we onmiddellijk deze woorden. Met andere woorden, ons patroonherkennings-mechanisme constateert onmiddellijk overeenkomsten tussen woorden in een zin en woorden die wij van te voren hebben geleerd.

Ongetwijfeld zijn er sterke overeenkomsten in menselijk denken en het denken van de computer. We moeten daarbij echter wel inzien, dat de programma's die in onze hersenen worden 'gerund' ons volstrekt onbekend zijn. Hieruit volgt dan weer dat een werkelijk goede beantwoording van onze uitgangsvraag waarschijnlijk nooit gevonden kan worden.

Niettemin kunnen we stellen dat computers ons eens te meer dwingen te denken over ons eigen denken en dit is op z'n minst een belangrijke constatering!



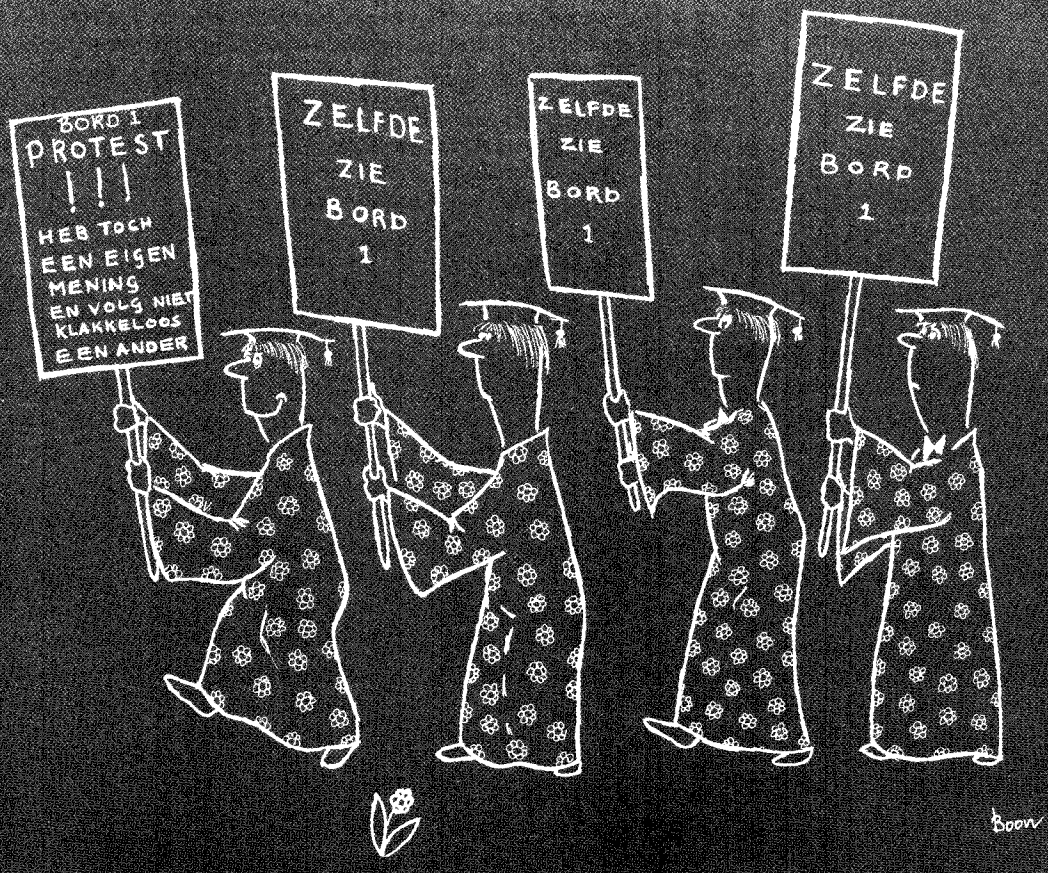
Bovenstaande figuur toont ons 7 zinnen waarbij iedere keer is geschreven 'Wat is patroonherkenning?'

Het toont ons op bijzonder illustratieve wijze waar de werkelijke problemen bij patroonherkenning liggen. Wij mensen zijn uitgerust met een zeer goed patroonherkennings-mechanisme. Als we dergelijke zinnen met een computer zouden moeten herkennen zou iedere letter eerst afgezonderd moeten worden. Bij regel 2 en 7 is dit misschien mogelijk... maar wat te zeggen van bijvoorbeeld regel 6?

Het herkennen van handgeschreven zinnen m.b.v. de computer is nu al een klassiek probleem op het gebied van de patroonherkenning!

In dit hoofdstuk wordt de subroutine besproken. Een subroutine is in feite een reeks instructies, die als ging het om een afzonderlijk programma, vanuit een willekeurig punt van ons 'hoofd-programma' kan worden opgeroepen. Het belangrijkste winstpunt van subroutines is, dat het ons de mogelijkheid biedt programma's gestructureerd op te bouwen.

Dit punt vormt dan ook het belangrijkste onderdeel van dit hoofdstuk. We zullen daarbij onder andere zien hoe we een probleem splitsen in deelproblemen, die op zich ook weer uitgesplitst kunnen worden. Deze methode staat bekend onder de naam van 'methode van stapsgewijze verfijning'. Tenslotte zullen we zien hoe we met behulp van de DEF FN-instructie zelf functies kunnen definiëren.



14. Subroutines

Inleiding

Bij de wat grotere programma's kan men in de opbouw vaak gedeelten herkennen die men zou kunnen opvatten als min of meer op zichzelf staande programma's. Sterker nog, als we een programma ontwikkelen, is het verstandig om te zien hoe we het probleem kunnen opdelen in deelproblemen die op zich overzichtelijker zijn. De programmeerwijze die op deze methode van het verder uitsplitsen van deelproblemen is gebaseerd, staat bekend onder de naam 'stapsgewijze verfijning'.

Bij deze methode geldt dan met name dat men deze opzet alleen goed kan uitvoeren, als de programmeertaal gereedschappen biedt om die zelfstandige deelproblemen ook duidelijk als zelfstandige programma-gedeelten, dat wil zeggen als zogenaamde 'subroutines', te ontwikkelen.

In het nu volgende voorbeeld behandelen we de opzet van een wat groter programma. Hierbij zal het oorspronkelijke probleem in fijnere, op zich logische deelproblemen, worden opgedeeld. Bij het omzetten van deze deelproblemen in op zichzelf staande deelprogramma's maken we dan gebruik van de subroutine-structuur die MSX-BASIC biedt.

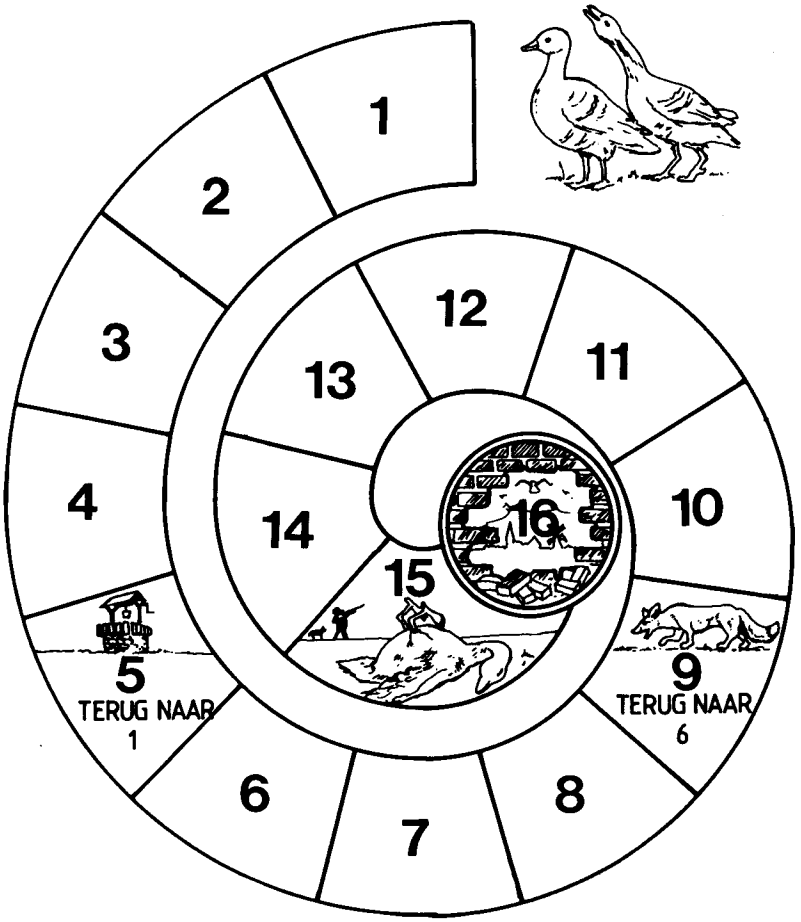
Mini-ganzenbord!

Het programma dat we nu opzetten, biedt ons de mogelijkheid het bekende oerhollandse spel 'Ganzenbord' te spelen.

Om nu een en ander overzichtelijk te houden, hebben we ons ganzenbord wat eenvoudiger opgezet. Als oefening kan men dan de nu te bespreken techniek toepassen om een programma voor het originele ganzenbord te schrijven.

De figuur op de volgende pagina toont ons mini-ganzenbord.

Het af te leggen pad bestaat uit 16 velden. De spelers starten op veld 1 en gooien om beurten met één dobbelsteen. Het aantal ogen dat gegooid wordt, is bepalend voor het aantal stappen dat men mag nemen.



Zoals dat bij een ganzenbord hoort, zijn er een aantal velden met een bijzondere betekenis:

veld 5: als men hierop terecht komt, moet het spel weer van voren af aan begonnen worden, kortom men keert terug naar veld 1.

veld 9: ga terug naar veld 6.

veld 15: dit is het bekende struikelblok op het einde van het spel. Als men hierop terecht komt, moet men weer van voren af aan beginnen. Ook hier geldt dus weer 'keer terug naar veld 1'.

Het doel van het spel is om in veld 16 te komen. Dit kan alleen door in de laatste beurt precies het juiste aantal ogen te gooien. Geeft de dobbelsteen een te groot aantal aan, dan moet worden teruggeteld. Stel bijvoorbeeld dat men op veld 14 staat en dat men een 4 gooit, dan zou men op veld 14+4=18 komen, maar nu wordt teruggeteld en men komt zodoende op veld 14 terecht.

Opzet van het programma

We gaan er van uit dat er twee spelers zijn, namelijk u zelf en de computer. De stand van beide spelers, dat wil zeggen het veld waarop ze staan, kunnen we natuurlijk aan een variabele toe-

kennen. Laten we bijvoorbeeld de variabele A gebruiken om uw stand aan te geven en B om de stand van de computer te onthouden. We zouden het spel dan als volgt in woorden kunnen omschrijven:

1 geef zowel A als B de startwaarde 1. Dit zijn de startposities.
2 beurt van de speler.
3 beurt van de computer.
4 ga terug naar stap 2.

We zien hoe hier het oorspronkelijke probleem in een viertal deelproblemen is gesplitst. We kijken nu hoe deelprobleem 2 verder is uit te splitsen:

2a werp een dobbelsteen. Stel dat het aantal ogen aan de variabele D wordt toegekend.
b bereken het nieuwe veld, m.a.w. A=A+D.
c ga na of het nieuwe veld een bijzonder veld is en corrigeer zonodig het nieuwe veld. Bijvoorbeeld als A+9 zou worden, wordt A volgens de regels van het spel gecorrigeerd (A wordt dan 6).

We merken hierbij op dat uitwerken van stap 3, de beurt van de computer, tot eenzelfde uitsplitsing leidt:

3a werp een dobbelsteen. Stel dat het aantal ogen aan de variabele D wordt toegekend.
b bereken het nieuwe veld, met andere woorden B=B+D.
c ga na of het nieuwe veld een bijzonder veld is en corrigeer zonodig het nieuwe veld.

De stappen a en c van deelproblemen 2 en 3 zijn volstrekt identiek en het zou dus plezierig zijn als we hier hetzelfde programmadeel voor zouden kunnen gebruiken.

We zullen nu een 'BASIC-achtig' programma opschrijven, waarbij de afzonderlijke subroutines nog in woorden worden genoteerd.

```
A=1 }
B=1 }  start-positie
PRINT "UW BEURT"
```

subroutine TOETS

roep subroutine aan waarin de speler op een willekeurige toets moet drukken. Dit heeft het effect dat de speler iets moet doen.

subroutine DOBBEL

roep subroutine aan om het aantal ogen van de dobbelsteen D te bepalen.

```
A=A+D
```

subroutine CORRECTIE

kijk of dit een bijzonder veld is en corrigeer zonodig de stand. Nieuwe stand is N.

```
A=N
PRINT "NIEUWE STAND VAN U IS"; N
PRINT "COMPUTER'S BEURT"
```

subroutine DOBBEL

roep subroutine aan om het aantal ogen van de dobbelsteen D te bepalen.

```
B=B+D
```

subroutine CORRECTIE?

kijk of dit een bijzonder veld is en corrigeer zonodig de stand. Nieuwe stand is N.

```
B=N
PRINT "NIEUWE STAND COMPUTER IS"; N
```

Ga terug naar instructie `PRINT "UW BEURT"`

GOSUB... RETURN

We zijn nu op het punt beland om dit 'pseudo-programma' in een werkelijk programma om te zetten.

Alvorens dit te doen, geven we allereerst de subroutine-constructie van MSX-BASIC weer.

Deze constructie houdt in dat we naar een willekeurig programmagedeelte kunnen springen en wel met de instructie:

```
GOSUB  regelnummer
```

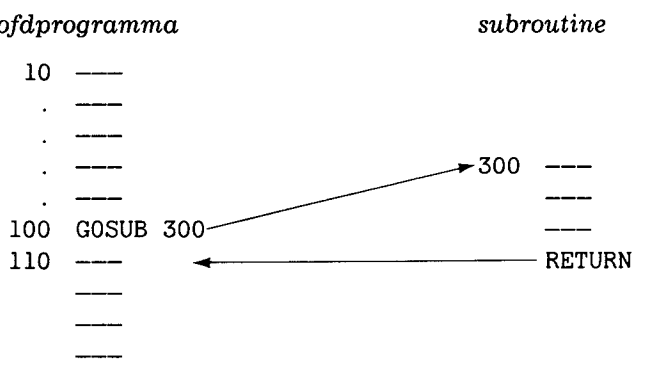
en dat de computer na het herkennen van de term

```
RETURN
```

in dit programma-gedeelte weer terugspringt naar het oorspronkelijke programma. De computer vervolgt het programma dan met de instructie die volgt op de regel met de GOSUB-instructie.

Zie schema op volgende pagina.

In dit schema is voor de overzichtelijkheid de subroutine rechts van het 'hoofdprogramma' aangegeven. In werkelijkheid zullen de programma's één lange lijst instructies vormen.



We bespreken nu één voor één de subroutines van ons ganzenbord-programma.

subroutine TOETS

We willen in deze subroutine het effect van een onderbreking oproepen. De speler moet eerst op een toets drukken, bij wijze van spreken, om de dobbelsteen te werpen.

Uiteraard maken we hier weer van de INKEY$-constructie gebruik. We gebruiken de volgende subroutine:

```
200 PRINT "DRUK EEN TOETS IN"
210 A$=INKEY$
220 IF A$="" THEN GOTO 210
230 RETURN
```

subroutine DOBBEL

Deze subroutine moet zorgdragen voor een simulatie van een dobbelsteen.

We maken hierbij gebruik van de zogenaamde RND-functie. Deze heeft de volgende vorm

```
RND(1)
```

Deze functie levert dan een willekeurig getal tussen 0 en 1 op.

Om nu een willekeurig getal tussen 1 en 6 te krijgen, gebruiken we de volgende constructie:

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

De dobbelsteen-subroutine wordt dan:

```
300 D=INT(RND(1)*6+1)
310 PRINT "DE DOBBELSTEEN TOONT EEN"; D
320 RETURN
```

subroutine CORRECTIE

Bij de subroutine CORRECTIE gaan we er van uit dat de stand die getest moet worden eerst aan de hulp-variabele W is toegekend. De gecorrigeerde stand zullen we dan aan de hulp-variabele N toekennen.

Het testen of we op speciale velden staan, kunnen we eenvoudig aan de hand van IF...THEN-instructies aangeven.
We gebruiken de volgende subroutine:

```
400 IF W=5 THEN W=1: PRINT "VIA VELD 5"
410 IF W=9 THEN W=6: PRINT "VIA VELD 9"
420 IF W=15 THEN W=1: PRINT "VIA VELD 15"
430 IF W>16 THEN W=16-(W-16): PRINT
"TE VER....": GOTO 400
440 IF W=16 THEN PRINT "GEWONNEN!": STOP
450 N=W
460 REM WACHTTIJD
470 FOR K=1 TO 400
480 NEXT K
490 RETURN
```

De regels 400 t/m 450 spreken voor zich. De PRINT-instructies zijn toegevoegd om de speler duidelijk te maken *hoe* het nieuwe veld bereikt is. Het laatste gedeelte is toegevoegd om een zekere wachttijd te introduceren. Zo is het net alsof er ook tijd voor nodig is om de pionnen te verplaatsen...

Volledig programma

Na de voorafgaande bespreking wordt nu het totale programma gegeven. We zien hoe hier onze oorspronkelijke opzet volledig is aaangehouden.

Mini-ganzenbord:

```
 10  A=1
 20  B=1
 30  PRINT "UW BEURT"
 40  GOSUB 200
 50  GOSUB 300
 60  A=A+D
 70  W=A
 80  GOSUB 400
 90  A=N: PRINT "NIEUWE STAND VAN U IS";
     N:PRINT
100  PRINT "COMPUTER'S BEURT"
110  GOSUB 300
120  B=B+D
130  W=B
140  GOSUB 400
```

```
150  B=N: PRINT "NIEUWE STAND COMPUTER IS";
     N:PRINT
160  GOTO 30
200  PRINT "DRUK EEN TOETS IN"
210  A$=INKEY$
220  IF A$="" THEN GOTO 210
230  RETURN
300  D=INT (RND(1)*6+1)
310  PRINT "DE DOBBELSTEEN TOONT EEN"; D
320  RETURN
400  IF W=5 THEN W=1: PRINT "VIA VELD 5"
410  IF W=9 THEN W=6: PRINT "VIA VELD 9"
420  IF W=15 THEN W=1: PRINT "VIA VELD 15"
430  IF W>16 TEN W=16-(W-16): PRINT "TE VER
     ...": GOTO 400
440  IF W=16 THEN PRINT "GEWONNEN!": STOP
450  N=W
460  REM WACHTTIJD
470  FOR K=1 TO 400
480  NEXT K
490  RETURN
```

Nabeschouwing Het is verstandig om na deze eerste bespreking van de subroutine nog eens stil te staan bij de wezenlijke winstpunten die de subroutine-constructie oplevert. In veel tekstboeken wordt vooral gewezen op het feit dat de subroutine winst oplevert als we een bepaalde serie instructies meer keren moeten afwerken.

Uiteraard is dat waar, maar in onze ogen is de waarde van de subroutine vooral gelegen in het feit dat we bij de opzet van het programma de min of meer op zich staande onderdelen als zelfstandige programma-gedeelten kunnen ontwikkelen.

Het hoofdprogramma bestaat dan uit een reeks instructies waarin onder andere de subroutines worden opgeroepen. Zo verkrijgt men de volgende structuur:

```
10 REM HOOFDPROGRAMMA  ┐
   ---                 │
   ---                 │
   GOSUB---            } hoofdprogramma
   ---                 │
   GOSUB---            │
   etc.                ┘
-----------
   ---  ┐
   ---  } subroutine 1
   ---  ┘
   ---  ┐
   ---  } subroutine 2
   ---  ┘
```

Een op deze wijze opgezet programma biedt een duidelijke structuur en dat is in feite het hoofddoel van een goed programma!

We plaatsen tenslotte nog de volgende opmerkingen:

- subroutines kunnen zelf ook weer subroutines aanroepen.
- evenals de ON...GOTO-instructie kan men de ON...GOSUB-instructie gebruiken. De opzet is volledig gelijk aan de ON...GOTO-instructie en daarom zullen we er niet verder op ingaan.

DEF FN

MSX-BASIC biedt tevens de mogelijkheid zelf functies te definiëren. Zo'n functie kunnen we als een soort subroutine opvatten en dit is de reden dat we deze mogelijkheid in dit hoofdstuk bespreken.
De DEF FN-instructie heeft steeds de volgende vorm:

DEF FN naam (reeks variabelen gescheiden door komma's) = uitdrukking waarin deze variabelen voorkomen

Bijvoorbeeld:

```
DEF FNA (X,Y)=X^2+Y^2
```

De functienaam bestaat uit één letter (in ons voorbeeld de letter A). Bij dit voorbeeld wordt de functie A gedefinieerd door:

X^2+Y^2

Deze functie wordt dan opgeroepen met de uitdrukking FNA, met andere woorden met de term FN gevolgd door de naam van de functie.
Het volgende programma illustreert dit:

```
10 DEF FNA(X,Y)=X^2+Y^2
20 A=3
30 B=4
40 C=FNA(A,B)
50 PRINT C
```

Resultaat:

```
25
```

Regel 10 definieert, op de reeds besproken wijze, de functie. In eerste instantie negeert de computer deze instructie. Regel 20 en 30 spreken voor zich.

Regel 40 toont de term FNA. De computer leidt hieruit af dat een functie wordt geroepen en wel de functie met de naam A. Vervolgens wordt naar regel 10 gekeken en overal waar daar X staat wordt nu A ingevuld en evenzo wordt voor Y de variabele B ingevuld.

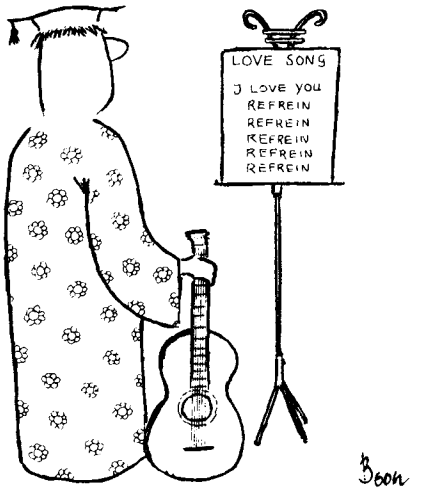
Zodoende wordt C gelijk aan:

A^2+B^2

en aangezien A=3 en B=4 wordt C gelijk aan:

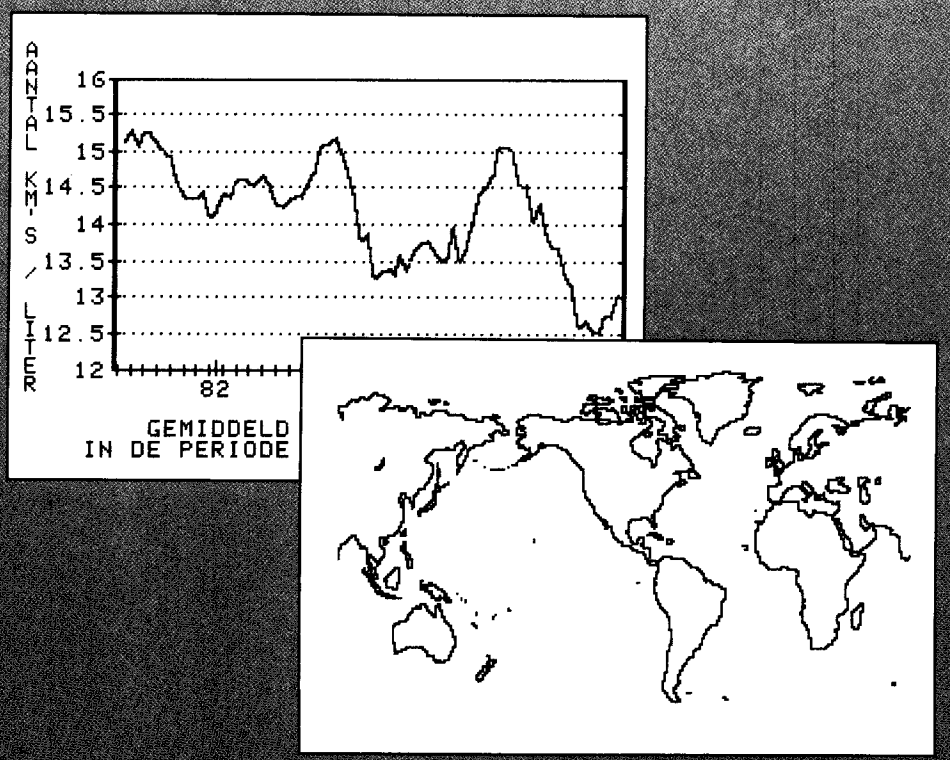
3^2+4^2 $(=3^2+4^2)$

Het resultaat bevestigt dit.



Bij de tot nu toe behandelde programma's bestond de uit te voeren informatie steeds uit tekst en cijfers. In dit hoofdstuk maken we kennis met de mogelijkheid ook allerlei grafische voorstellingen op het scherm weer te geven.

Achtereenvolgens bespreken we de SCREEN-instructie om het zgn. grafische scherm te definiëren, de PSET-instructie om een stip op het scherm weer te geven, COLOR om de kleur aan te geven en PRESET om een reeds aangegeven stip weer te wissen.



15. Grafische voorstellingen: SCREEN, PSET, COLOR en PRESET

Inleiding

Een grafische voorstelling is niets anders dan een deftig woord voor een plaatje. Plaatjes bouwen we op met punten, lijnen en krommen.

Onze MSX-computer heeft bijzonder veel mogelijkheden om allerlei fraaie figuren op het scherm te tonen. Dat is belangrijk want veel toepassingen zijn juist gebonden aan het werken met boeiende plaatjes. Als illustratief voorbeeld kunnen we hier denken aan allerlei educatieve programma's. Onderwerpen zoals bijvoorbeeld rekenen worden bij deze programma's dan gekoppeld aan boeiende spektakels op het scherm, zodat het leren zonder meer veraangenaamd wordt.

Het leidt dan ook geen twijfel of computers zullen zich op dit gebied een niet meer af te stane plaats veroveren.

Behalve de educatieve programma's moeten we, als we tenminste aan plaatjes denken, natuurlijk ook wijzen op de vele spelletjes die voor computers aangeboden worden.

Ondanks de duidelijke gevaren (bij sommige spelletjes) voor verslaving kunnen we niet ontkennen dat de produkten die we zien soms buitengewoon fraai zijn.

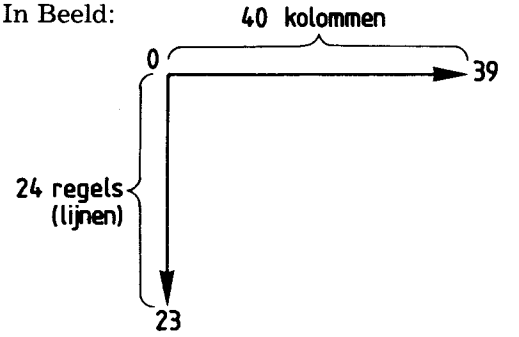
Het gaat ons veel te ver om alles wat met spel te maken heeft te veroordelen. Persoonlijke ervaring leert dat het zelf schrijven van een spel een uitermate boeiende en zeker niet minder leerzame bezigheid is.

SCREEN

Om inzicht te krijgen hoe we met een MSX-computer plaatjes opbouwen, moeten we eerst iets meer over de indeling van ons scherm (Engels: screen) vertellen.

Bij al de tot nu toe geboden voorstellingen hanteerde de computer een indeling in 24 regels, waarbij iedere regel 40 tekens kan bevatten.

In Beeld:



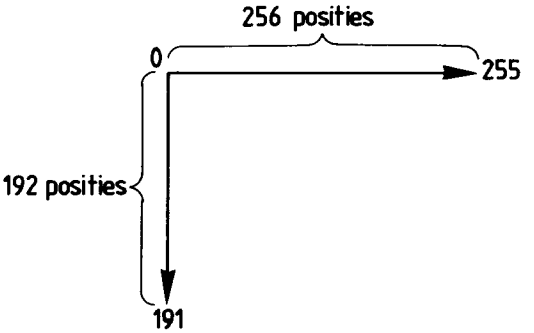
Merk op dat we ons assenstelsel min of meer op z'n kop tekenen: dat is bij computers gebruikelijk.

Welnu, als we fijne tekeningen willen weergeven, is deze indeling niet toereikend. Als we bijvoorbeeld op een scherm van 24×40 posities, door middel van kruisjes een lijn tekenen, dan komt wel een erg grove figuur naar voren.

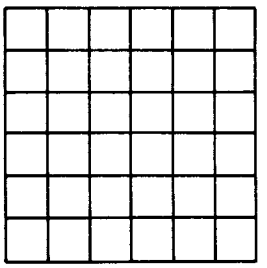
Om toch lijnen met 'hoge fijnheid' weer te geven (folders spreken van een hoge resolutie), moeten we de computer eerst omschakelen, zodat een andere schermindeling wordt gehanteerd. Na de instructie:

```
SCREEN 2
```

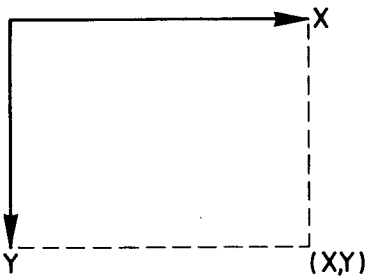
hanteert de computer een indeling die er als volgt uitziet:



We zien dat ons TV-scherm nu in een veel fijner raster is onderverdeeld. Als we nu een deel van het raster bekijken, zien we een aaneenschakeling van vierkantjes.



Ieder vierkantje noemen we voortaan een beeldpunt (of kortweg punt) en de coördinaten van ieder punt geven we dan door een X- en een Y-waarde aan en wel volgens de volgende afspraak:



Hierbij mag x van 0 tot 255 variëren en y van 0 tot 191.

Punt (0,0) komt dus neer op het punt linksboven en (191,255) met het punt rechtsonder.

Als we met behulp van de instructie SCREEN 2 de computer eenmaal in de toestand hebben gebracht dat het fijne raster gebruikt wordt, kunnen we stippen en lijnen met daartoe beschikbare functies aangeven.

PSET We starten met de instructie PSET (*pointset*, dat wil zeggen, geef een punt weer) waarmee we stippen op het scherm kunnen plaatsen.
In de eenvoudigste vorm ziet deze instructie er als volgt uit:

PSET (x-coördinaat, y-coördinaat)

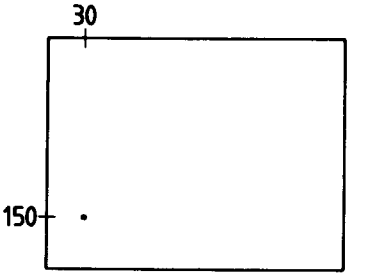
We starten maar meteen met een voorbeeld:

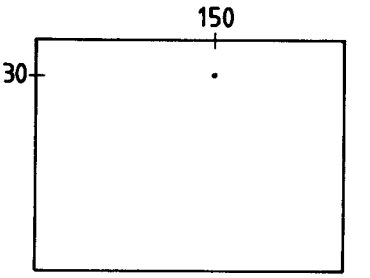
```
10 INPUT "X=";X
20 INPUT "X=";Y
30 SCREEN 2
40 PSET (X,Y)
50 GOTO 50
```

Regel 10 en 20 kennen een waarde aan X en Y toe. In regel 30 staat de noodzakelijke instructie SCREEN 2. Daarna wordt met de PSET-instructie de stip getekend. De laatste instructie zorgt er voor dat het programma niet wordt beëindigd, zodat we in feite niets te zien zouden krijgen (laat deze regel maar eens weg!)

We kunnen het programma dan ook alleen maar onderbreken door niet ingedrukte CTRL-toets op STOP te drukken (CTRL/STOP).

Voer nu maar eens een paar waarden in. Merk op dat kleine waarden punten kunnen opleveren die buiten het beeld vallen, zodat we dergelijke punten uiteindelijk niet te zien krijgen. Onderstaande tekeningen tonen twee voorbeelden:





We kunnen onze PSET-instructie nog uitbreiden en wel door er een getal achter te plaatsen die de kleur aangeeft.
Wijzig het programma maar eens als volgt:

```
10 INPUT "X=";X
20 INPUT "Y=";Y
30 INPUT "KLEUR=";K
40 SCREEN 2
50 PSET (X,Y),K
60 GOTO 60
```

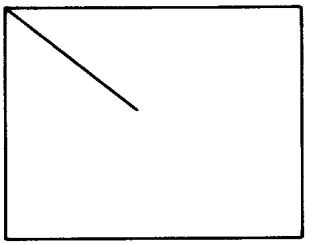
Voor de kleuren gelden de volgende cijfers:

cijfer/kleur	*cijfer/kleur*	*cijfer/kleur*	*cijfer/kleur*
0 transparant	4 donkerblauw	8 rood	12 donkergroen
1 zwart	5 lichtblauw	9 lichtrood	13 magenta
2 groen	6 donkerrood	10 donkergeel	14 grijs
3 lichtgroen	7 hemelsblauw	11 lichtgeel	15 wit

Waarschijnlijk ziet u in een aantal gevallen de gekleurde stip niet duidelijk of in het geheel niet. Als we eens een serie punten plaatsen, is het effect in ieder geval veel duidelijker.
Het volgende programma biedt daartoe de mogelijkheid:

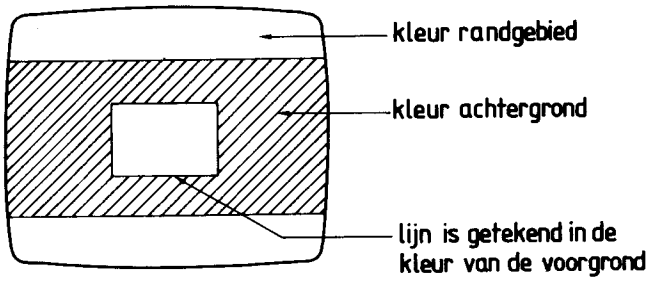
```
10 INPUT "N=";N
20 INPUT "KLEUR=";K
30 SCREEN 2
40 FOR J=1 TO N
50 PSET (J,J),K
60 NEXT J
70 GOTO 70
```

Als we nu bijvoorbeeld voor N de waarde 100 invullen en voor K de waarde 1 dan zien we de volgende zwarte lijn:



Het voorbeeld toont tevens dat we lijnen kunnen tekenen door in feite veel punten naast elkaar te plotten. Straks zullen we zien dat MSX-BASIC bovendien nog een aantal zeer krachtige instructies kent voor het tekenen van allerlei soorten lijnen.

COLOR We kunnen bovendien nog de kleur van het scherm en de zgn. kleur van de randgebieden instellen. Bedenk dat het beeld daartoe als volgt is ingedeeld:



De instructie die dit alles bepaalt, heeft de volgende vorm:

`COLOR` kleur voorgrond, kleur achtergrond, kleur randgebied

Voorbeelden:

`COLOR 1` : maak alleen de kleur van de voorgrond zwart (1), met andere woorden teken zwarte lijnen.

`COLOR ,2` : verander de kleur van de achtergrond in groen. Merk op dat we een komma voor het getal plaatsen.

`COLOR 1,2,11` : maakt de kleur van de voorgrond zwart (1), van de achtergrond groen (2) en van de randgebieden lichtgeel (11).

`COLOR ,,11` : verander alleen de kleur van de randgebieden. Merk op dat voor het cijfer twee komma's zijn geplaatst.

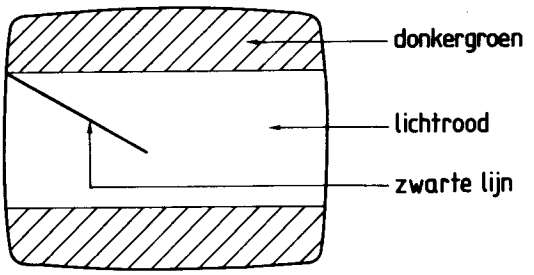
We dienen bovendien nog te bedenken dat als we de kleur van de achtergrond willen wijzigen, we voor de COLOR-instructie altijd een CLS-instructie ('*cl*ear *s*creen': dat wil zeggen wis het scherm) moeten plaatsen.
Als we bijvoorbeeld ons vorige programma uitbreiden met:

```
32 CLS
35 COLOR 1,9,12
```

en we laten bovendien de kleur-aanduiding bij de PSET-instructie weg

```
50 PSET (X,Y)
```

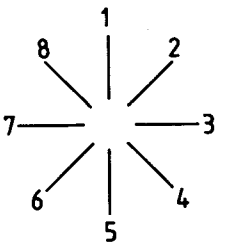
dan verschijnt de volgende figuur:



Tekenen met de joystick

Aan het einde van dit hoofdstuk zullen we een aardige demonstratie geven van wat we zoal met de PSET-instructie kunnen doen. In dit voorbeeld zullen we een programma bespreken, waarmee we een tekening in hoge resolutie op ons scherm kunnen maken. Als tekenpen gebruiken we dan de joystick.

Om de stand van de joystick uit te lezen, gebruiken we de functie STICK (joystick-nummer). Deze functie levert de waarden 1 t/m 8 op en wel volgens onderstaand schema:



Zo zal bij de instructie D=STICK(1) de waarde 3 worden toegekend als joystick nummer 1 naar rechts wordt getrokken.

In het programma wordt tevens gebruik gemaakt van de mogelijkheid een punt te wissen. Dit gebeurt dan met de PRESET-instructie:

```
PRESET (X,Y)
```

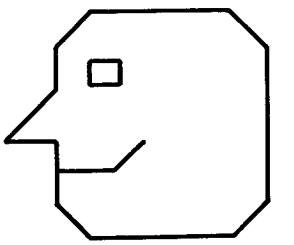
Ten slotte wordt er in het programma gekeken of de vuurknop van joystick 1 wordt ingedrukt.
Dit gebeurt aan de hand van de volgende instructie:

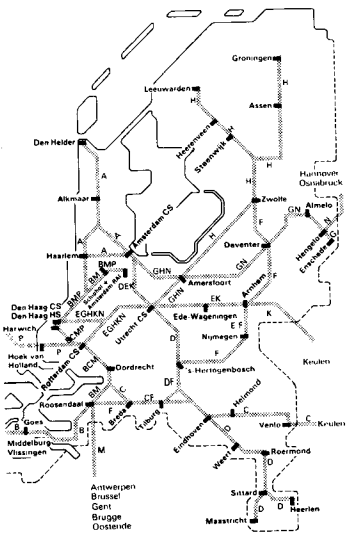
```
IF STRIG(1) THEN ....
```

De instructie na THEN wordt hier uitgevoerd als de vuurknop wordt ingedrukt. Hieronder volgt het volledige programma:

```
 10 X=100
 20 Y=100
 30 CLS
 40 SCREEN 2
 50 D=STICK(1)
 60 IF D=1 THEN Y=Y-1
 70 IF D=2 THEN X=X+1:Y=Y-1
 80 IF D=3 THEN X=X+1
 90 IF D=4 THEN X=X+1:Y=Y+1
100 IF D=5 THEN Y=Y+1
110 IF D=6 THEN X=X-1:Y=Y+1
120 IF D=7 THEN X=X-1
130 IF D=8 THEN X=X-1:Y=Y-1
140 IF STRIG(1) THEN GOTO 170
150 PSET (X,Y)
160 GOTO 50
170 PSET (X,Y)
180 FOR K=1 TO 10
190 NEXT K
200 PRESET (X,Y)
210 GOTO 50
```

Voorbeeld van een resultaat:



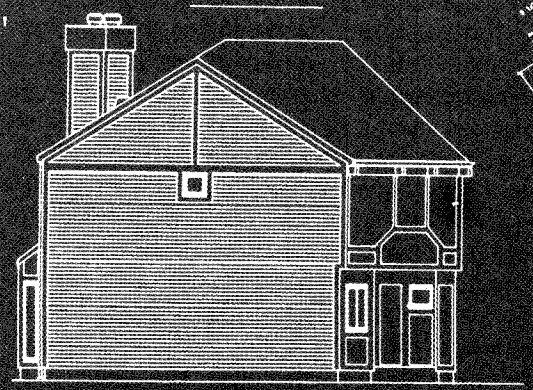


In regel 10 en 20 wordt een uitgangspunt voor de tekening aangegeven. In regel 50 wordt aan D de stand van de joystick toegekend. Vervolgens (regel 60 t/m 130) worden aan de hand van D de nieuwe coördinaten berekend. Als de vuurknop wordt ingedrukt (regel 140) volgt een sprong naar regel 170. Hier wordt het punt geplot en wel voor de duur die door de FOR...NEXT-instructie wordt bepaald. Vervolgens wordt het punt weer gewist. Zo kan men met ingedrukte vuurknop als het ware de tekenpen verplaatsen, zonder daarbij een lijn achter te laten!

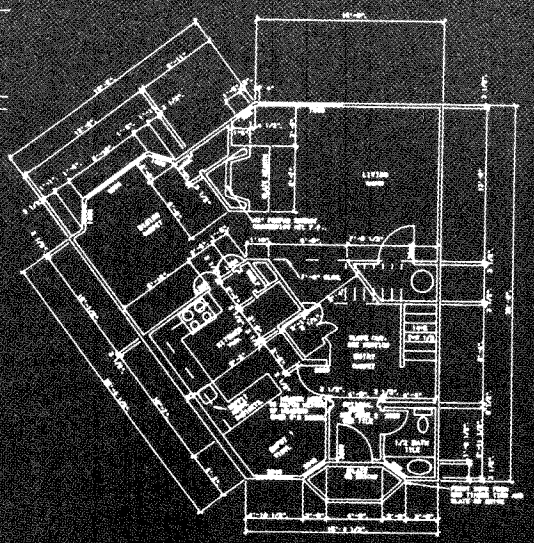
De tekeningen dragen vooral het karakter dat ze uit verticale, horizontale en lijnen onder een hoek van 45° bestaan. Het bekende spoorkaartje van de NS is ook zo opgebouwd ... misschien kun je het wel natekenen.

Het opbouwen van lijn-figuurtjes verloopt eenvoudig met standaard-instructies. Met de LINE-instructie kunnen we een rechte lijn van een gegeven beginpunt naar een gegeven eindpunt trekken. De instructie DRAW is ook gericht op het trekken van rechte lijnen. Achter DRAW komt een string te staan, waarmee op eenvoudige wijze staat aangegeven hoe een aaneenschakeling van rechte lijnen moet worden afgebeeld. De CIRCLE-instructie dient voor het afbeelden van cirkels en ellipsen. Aangezien we ook delen hiervan kunnen afbeelden, is het in het algemeen een instructie om bogen weer te geven. De laatste instructie PAINT biedt ons de mogelijkheid gesloten vlakken in te kleuren.

FRONT VIEW



SIDE VIEW



FIRST FLOOR

16. Grafische voorstellingen: LINE, DRAW, CIRCLE en PAINT

Inleiding In dit hoofdstuk behandelen we de overige mogelijkheden om lijnen op het beeldscherm weer te geven. Alhoewel we met de PSET-instructie in feite alle lijnen kunnen plaatsen die we maar wensen, blijken de nu te behandelen instructies in de praktijk zeer handig te zijn. De instructies zijn er op gebaseerd dat we in veel gevallen figuurtjes uit 'standaard-lijnen', namelijk rechte lijnen en bogen, kunnen opbouwen. De instructies LINE en DRAW bieden de mogelijkheid om op eenvoudige wijze rechte lijnen aan te geven, terwijl we met CIRCLE bogen (onderdelen van een cirkel of een ellips) op het scherm kunnen tekenen.

Al deze instructies gaan uit van het grafische scherm, zoals dit in het vorige hoofdstuk besproken werd.

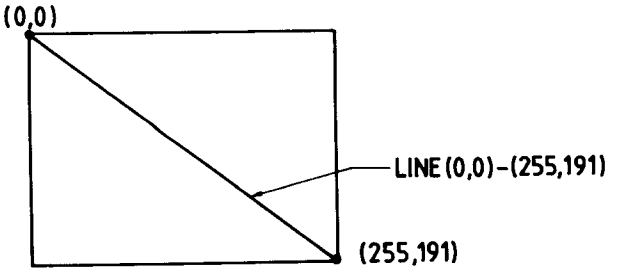
LINE De meest eenvoudige vorm van de LINE-instructie is:

```
LINE (X1,Y1)-(X2,Y2)
```

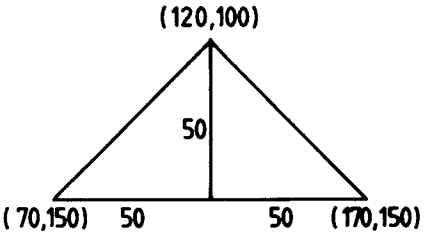
Met (X1,Y1) wordt dan het beginpunt van de lijn opgegeven en met (X2,Y2) het eindpunt.
Voorbeeld:

```
10 SCREEN 2
20 LINE(0,0)-(255,191)
30 GOTO 30
```

Als resultaat zien we nu een schuine lijn over het beeldscherm lopen.



In het nu volgende programma zullen we een driehoek tekenen en wel volgens onderstaande definitie:



Programma:

```
10 SCREEN 2
20 LINE ( 70,150)-(170,150)
30 LINE (170,150)-(120,100)
40 LINE (120,100)-( 70,150)
50 GOTO 50
```

Eventueel kunnen we de beginpositie weglaten. In dit geval wordt als beginpunt het eindpunt genomen dat aan de hand van een vorige instructie werd bereikt. Zo verkrijgen we dezelfde driehoek met het volgende programma:

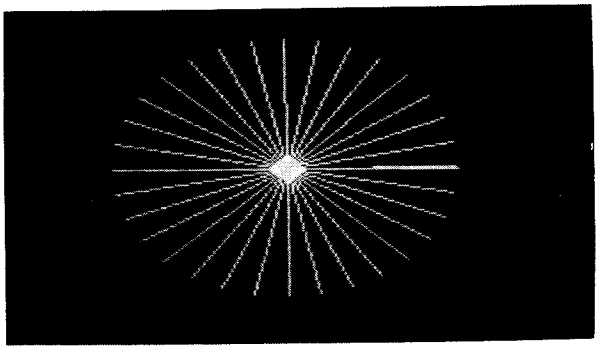
```
10 SCREEN 2
20 LINE ( 70,150)-(170,150)
30 LINE -(120,100)
40 LINE -( 70,150)
50 GOTO 50
```

In het volgende programma zullen we de LINE-instructie gebruiken om een fraaie ster te tekenen.

Programma:

```
10 SCREEN 2
20 AL=2*3.14159
30 FOR K=0 TO AL STEP AL/32
40 X=INT (70*COS(K))+122
50 Y=INT (70*SIN(K))+95
60 LINE (122,95)-(X,Y)
70 NEXT K
80 GOTO 80
```

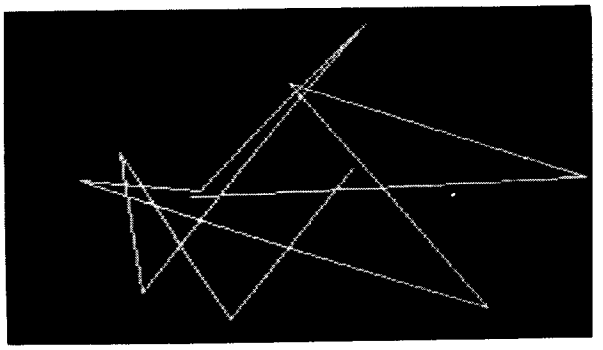
Resultaat:



Als laatste voorbeeld behandelen we een programma waarmee we een wirwar van lijnen op het beeld krijgen.

```
10 SCREEN 2
20 PSET (100,100)
30 FOR K=1 TO 10
40 X=INT(RND(1)*255)
50 Y=INT(RND(1)*191)
60 LINE -(X,4)
70 NEXT K
80 GOTO 80
```

Resultaat:



Als men in plaats van FOR K=1 TO 10 bijvoorbeeld FOR K=1 TO 100 neemt dan wordt het patroon werkelijk zeer grillig.

We kunnen bovendien een kleur aan een lijn opgeven en wel volgens dezelfde methode als bij PSET, kortom door het plaatsen van een komma en een kleurcode, dus;

```
LINE (X1,Y1)-(X2,Y2), kleurcode
```

We dienen hierbij wel te bedenken dat als twee lijnen met verschillende kleur elkaar kruisen, er op het kruispunt onregelmatigheden kunnen voorkomen. De lijn die gekruist wordt, kan voor een klein gedeelte rond het snijpunt (maximaal 8 punten) de kleur van de andere lijn aannemen.

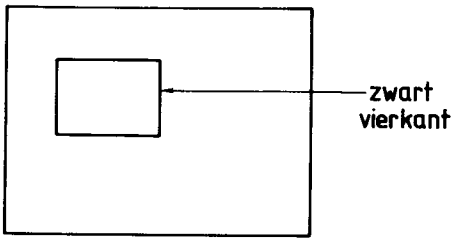
Ten slotte kunnen we met de LINE-instructie eenvoudig blokken tekenen. We breiden de instructie daartoe uit met ,B dus:

```
LINE (X1,Y1)-(X2,Y2), kleur ,B
```

Bijvoorbeeld:

```
10 SCREEN 2
20 LINE (50,50)-(100,100),1,B
30 GOTO 30
```

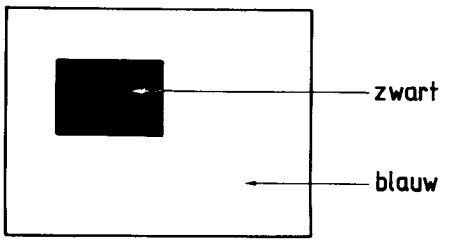
Geeft als resultaat:



Merk op dat we het blok kennelijk volledig gedefinieerd hebben door twee schuin tegenover elkaar liggende hoekpunten in de LINE-instructie aan te geven. Als we in plaats van B nu BF invullen, wordt het gehele vierkant ingekleurd, bijvoorbeeld:

```
10 SCREEN 2
20 LINE (50,50)-(100,100),1,BF
30 GOTO 30
```

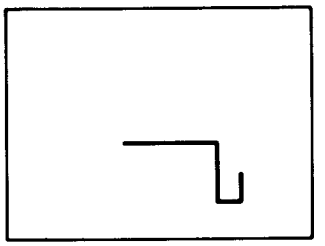
geeft:



DRAW De DRAW-instructie is een bijzonder krachtige instructie om horizontale, verticale en schuine lijnen te tekenen.
We geven een eenvoudig voorbeeld:

```
10 SCREEN 2
20 PSET (127,95)
30 DRAW "R50D50R25U25"
40 GOTO 40
```

Resultaat:



Regel 20 geeft het startpunt van onze tekenoefening. De daarop volgende DRAW-instructie geeft door middel van een string aan wat getekend moet worden en wel volgens:

R50 teken een lijn door 'de pen' 50 punten naar rechts te trekken.
D50 ga nu 50 punten naar beneden (D komt van 'down' hetgeen wil zeggen 'naar beneden').
R25 ga vervolgens weer 25 punten naar rechts.
U25 en vervolgens 25 punten naar boven (U komt van 'up' hetgeen wil zeggen 'naar boven')
Een volledig overzicht van de mogelijkheden van DRAW wordt in appendix D gegeven.

CIRCLE Dit is de laatste instructie die we bespreken om lijnen, in dit geval curven, te tekenen. De algemene vorm is:

```
CIRCLE (X,Y) , straal, kleur, beginhoek, eindhoek,
               aangezichtsverhouding
```

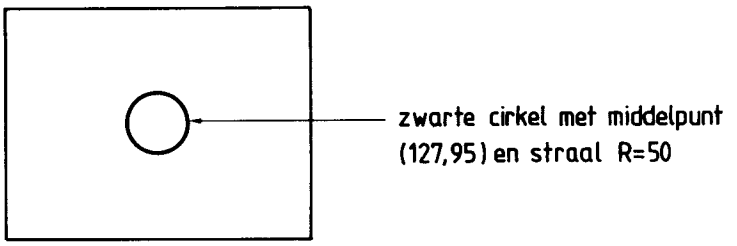
met:

(X,Y)	coördinaten van middelpunt
straal	waarde voor de straal
kleur	code voor de kleur
beginhoek	hoek vanaf waar de boog getekend wordt (opgeven in radialen: 0..2π)
eindhoek	hoek met betrekking tot waar de boog wordt getekend (opgeven in radialen: 0..2π)
aangezichts-verhouding	getal dat opgeeft in hoeverre de cirkel afgeplat, dat wil zeggen als een ellips wordt getekend. Vult men hier de waarde 1.4 voor in, dan verkrijgt men een cirkel.

We geven enkele voorbeelden:

```
10 SCREEN 2
20 CIRCLE (127,95),50,1,,,1.4
30 GOTO 30
```

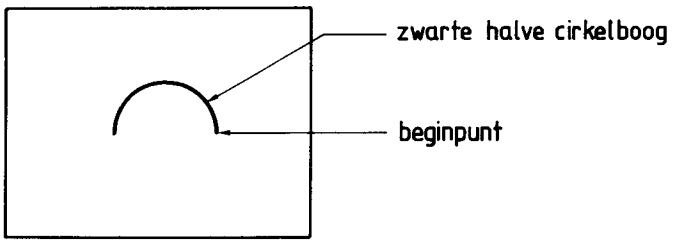
Resultaat:



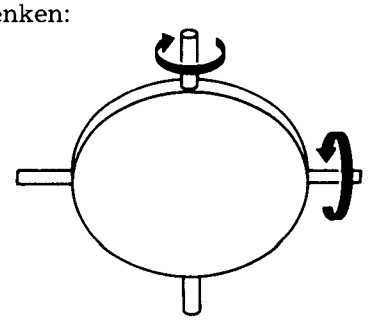
Het effect van de begin- en eindhoek kunnen we met behulp van het volgende programmaatje uitzoeken:

```
10 INPUT B
20 INPUT E
30 SCREEN 2
40 CIRCLE (127,95),50,1,B,E,1.4
50 GOTO 50
```

Als men nu voor B de waarde 0 en voor E de waarde 3.1415 invult, verkrijgt men de volgende halve cirkel:

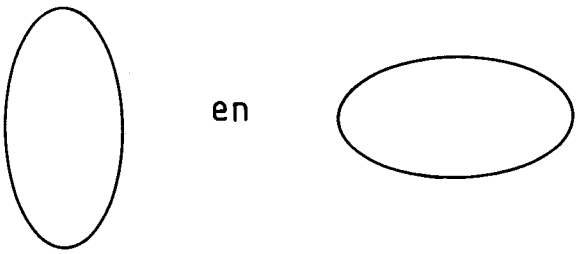


Het laatste getal van de CIRCLE-instructie bepaalt de zgn. aangezichtsverhouding. We kunnen hierbij het beste aan het volgende beeld denken:



Als de schijf om één van de assen draait, zullen we de schrijf steeds onder een andere hoek, dat wil zeggen aangezichtsverhouding zien.

Het effect is dat we een ellips in de volgende gedaanten kunnen zien:



draaiing om verticale as

draaiing om horizontale as

We kunnen het effect het beste aan de hand van het volgende programmaatje bestuderen:

```
10 INPUT K
20 SCREEN 2
30 CIRCLE (127,95),50,1,,,K
40 GOTO 40
```

PAINT De laatste instructie die we in dit hoofdstuk bespreken, is de PAINT-instructie. Hiermee kunnen we bepaalde vlakken inkleuren en wel door een willekeurig punt in zo'n in te kleuren vlak aan te wijzen.

De algemene vorm luidt:

```
PAINT (X,Y), kleurvlak, kleurlijn
```

met

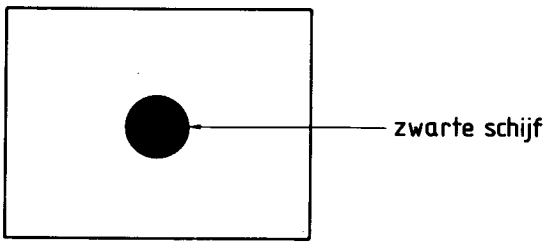
(X,Y)	coördinaten van het aan te wijzen punt
kleurvlak	kleur waarmee het vlak moet worden ingekleurd
kleurlijn	kleur waarmee de grens van het vlak moet worden aangegeven

Gewoonlijk laten we de laatste code weg, omdat voor het grafische scherm geldt dat de kleur van de grenslijn gelijk moet zijn aan de kleur van het vlak. Aangezien we altijd de grenzen van het vlak moeten aangeven, alvorens we kunnen inkleuren, ligt de code voor de kleur in feite ook vast.

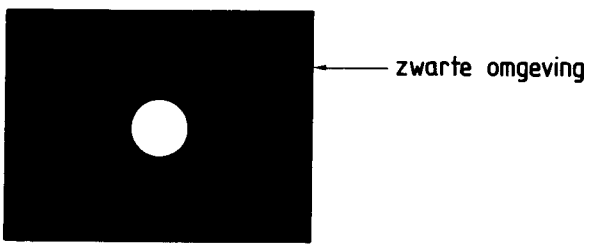
Een voorbeeld:

```
10 INPUT X,Y
20 SCREEN 2
30 CIRCLE (127,95),50,1,,,1.4
40 PAINT (X,Y),1
50 GOTO 50
```

Als we met PAINT nu een punt binnen de cirkel aanwijzen, krijgen we het volgende resultaat:



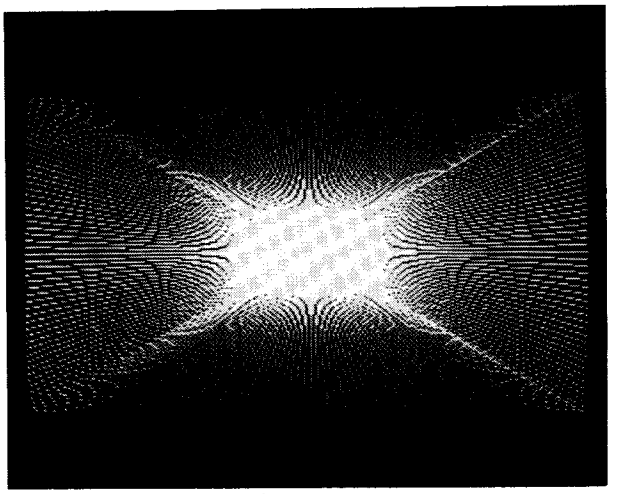
We zien hoe nu het vlak binnen de cirkel wordt ingekleurd. Als we echter een punt buiten de cirkel aanwijzen, krijgen we:



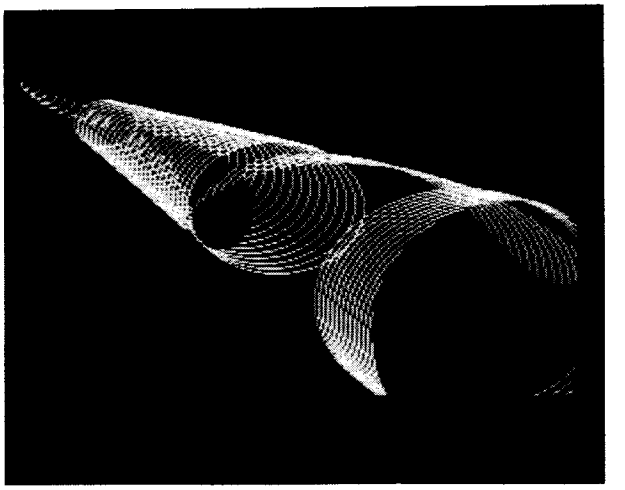
In dit geval wordt dus het andere gebied gekleurd.

Voorbeelden

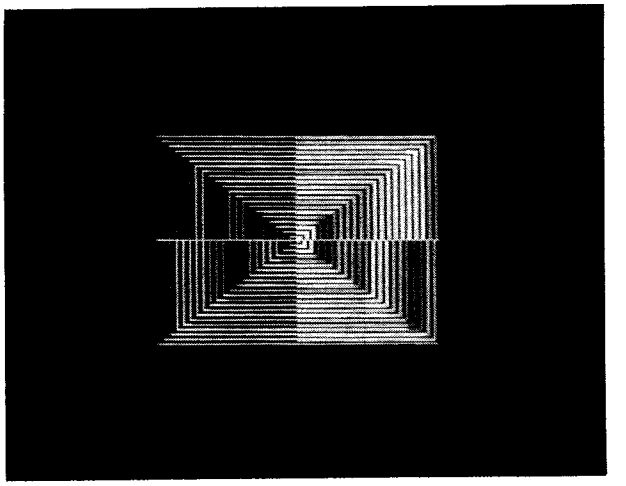
```
10 REM STER VAN LIJNEN
30 SCREEN 2
40 FOR T=0 TO 256 STEP 4
50 LINE (T,0)-(256-T,192)
60 LINE (0,T)-(255,192-T)
70 NEXT T
80 GOTO 80
```



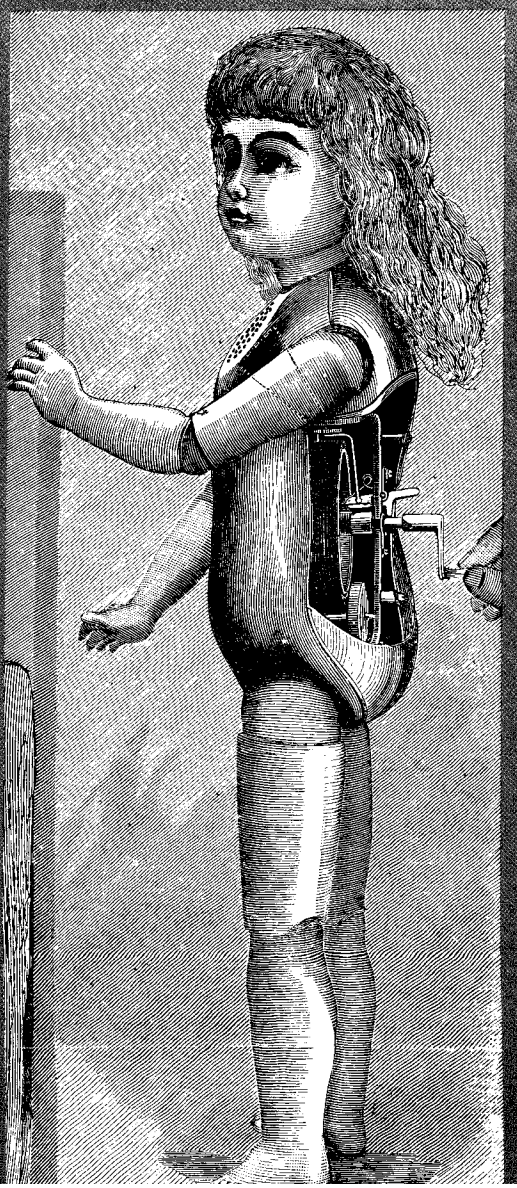
```
10 REM TUNNEL
20 PI=3.14159
30 SCREEN 2
40 FOR R=1 TO 80
50 CIRCLE(3*R,2*R),R,15,PI+SIN(R/
   40*PI)*PI,PI+COS(R/80*PI)*PI
60 NEXT R
70 GOTO 70
```



```
10  REM VIERKANTEN
20  SCREEN 2
30  DRAW "BM128,96"
40  FOR I=4 TO 255 STEP 3
50  XC=I MOD 13+2
60  XA=(I/4) MOD 3
70  DRAW"S=I;C=XC;A=XA;URDL2UR2DL"
80  NEXT I
90  GOTO 90
```



Dit hoofdstuk biedt wederom een intermezzo, dat wil zeggen er worden geen nieuwe instructies behandeld. Aan het einde van de behandeling van BASIC is het aardig om eens een blik terug te werpen. We gaan daarbij dieper in op de ontwikkeling van automaten. Daarbij worden achtereenvolgens de perioden besproken waarin de mens methoden vond om getallen weer te geven, automaten uitvond en ging denken over zijn eigen denken.



17. Intermezzo: over toen en nu

Inleiding

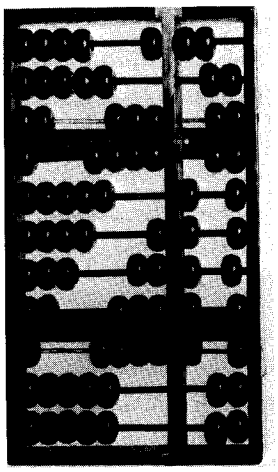
Computer zijn ingewikkelde apparaten. Ondanks alle verhalen over de eenvoud, moeten we bedenken dat er toch een lange ontwikkelingsperiode vooraf is gegaan aan de computer die wij nu kennen.

Het is interessant om te weten dat de mens eigenlijk voortdurend aan het zoeken is geweest om zaken automatisch uit te voeren. Onder automatisch dienen we dan te verstaan: dat een gehele reeks acties verloopt zonder tussenkomst van de mens. Er zijn prachtige voorbeelden bekend van zeer vernuftige apparaten.

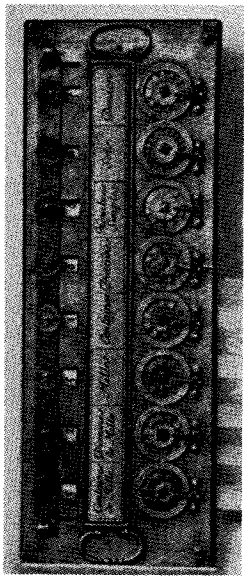
Ontwikkelingen

De ontwikkelingen die de geschiedenis ons toont, kunnen eigenlijk worden teruggebracht op een drietal punten:

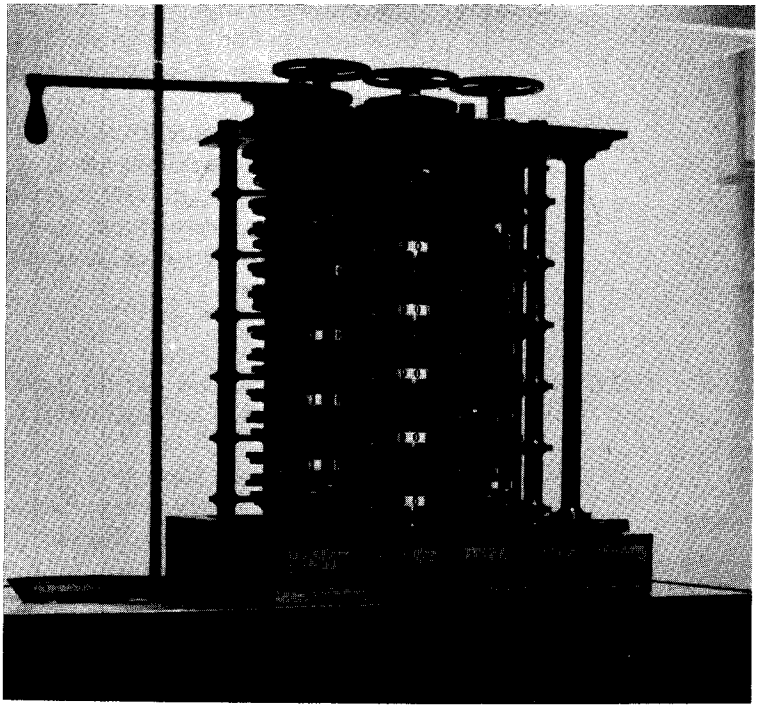
1 De mens verzon methoden om grootheden, zoals getallen, op een bepaalde manier vast te leggen, zodat we er handelingen mee konden uitvoeren. Waarschijnlijk is bovenstaande zin wat moeilijk te begrijpen.
Een voorbeeld verduidelijkt de situatie.
In de figuur hiernaast zien we een telraaam. Een telraam is in feite niets anders dan een handig middel om getallen *voor te stellen* (te presenteren). Handig, omdat we met zo'n telraam bijzonder gemakkelijk allerlei wiskundige berekeningen kunnen uitvoeren. Het werkelijke rekenen met zo'n telraam wordt natuurlijk door de mens uitgevoerd; hij schuift de bolletjes over de tralies. Op dat punt is er dus in totaal nog geen automaat te herkennen.



2 De mens verzon mechanismen om wiskundige berekeningen automatisch uit te voeren. Als illustratief voorbeeld kunnen we hier denken aan de rekenmachine die de beroemde Franse wiskundige en filosoof Blaise Pascal in 1642 introduceerde. Weer

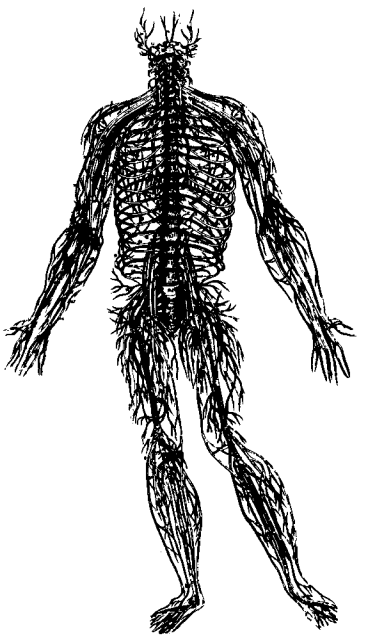


gaat het allereerst om een methode om getallen op een handige manier voor te stellen en wel aan de hand van wieltjes. De stand van zo'n wiel gaf dan een getal aan. Als zodanig is er geen verschil met het telraam van straks; ook daar werd een handige methode gebruikt om getallen te representeren. Het verschil is er echter wel degelijk, want bij Pascal zijn de wieltjes met een zodanig mechanisme met elkaar verbonden, dat we er allerlei berekeningen mee kunnen uitvoeren. We zien dus hier dat er behalve een handige vorm om getallen voor te stellen, een mechanisme wordt geïntroduceerd waarmee we de feitelijke bewerkingen kunnen uitvoeren.
Latere ontwikkelingen, zoals de 'Analytic Engine' van Charles Babbage (1791-1871), bouwden steeds op dit uitgangspunt voort.



De Analytic Engine kan men in het beroemde Londense Science Museum bewonderen. Het is een uitermate ingewikkeld mechanisch apparaat, bedoeld om zeer ingewikkelde berekeningen uit te voeren. Het is een toonbeeld van menselijk vernuft en vakmanschap.

3 De mens ging zich steeds meer bezinnen op de vraag 'hoe werken wij mensen zelf?'. Opzettelijk heb ik de wat laag bij de grondse term 'werken' gebruikt.



Bron: K. L. Boon: Elektrofysiologische lokalisatie, rapport Bio-informatica THT, 1968

De afbeelding hiernaast toont ons een van de prachtige platen die Andreas Vesalius (1514-1564), lijfarts van Karel V, ons achterliet. Zijn anatomische werk ligt in het logische vervolg van Leonardo da Vinci, die in 1519 stierf. De plaat toont ons het zenuwstelsel van de mens, althans buiten het gedeelte dat we nu als het allerbelangrijkste herkennen; de menselijke hersenen. Het inzicht dat men toentertijd in het zenuwcentrum had, ging niet verder dan dat het banen waren, waarlangs berichten werden gestuurd, bijvoorbeeld om spieren de opdracht te geven zich aan te spannen. Het belang van de hersenen (en in zekere mate het ruggemerg) als centrum van waaruit deze berichten – deze informatie – werden gecoördineerd, werd nog niet ingezien. Dat inzicht kwam pas omstreeks het midden van de negentiende eeuw (P. Broca 1824-1887); toen ontdekte men dat bepaalde gebreken van de hersenen tot onvermogen om te spreken leiden.

Vanaf dat moment is het denken over ons zenuwstelsel en het denken over 'ons denken' in een stroomversnelling terecht gekomen. Steeds meer werd het denken over ons zenuwstelsel voedingsbodem voor het ontwerpen van nieuwe apparaten.

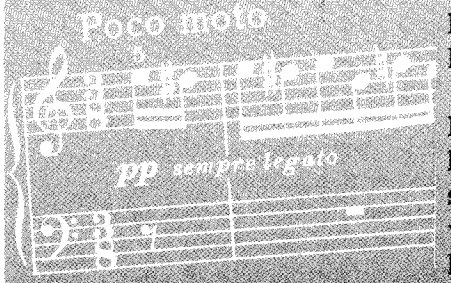
Heel belangrijk in dit proces is de ontdekking dat de signalen, die we in zenuwen kunnen 'oppikken', altijd een zeer eenvoudige vorm hebben.

De foto hiernaast toont zo'n registratie van signalen die uit zenuwbanen van de hersenen zijn opgemeten. Het meest opvallende is dat de signalen uit kortdurende pulsen, de 'spikes', bestaan. Als men hiermee geconfronteerd wordt, dan rijpt snel het idee dat informatie in onze hersenen gekoppeld is aan deze pulstreinen. Bovendien rijpt dan het idee dat onze hersenen kennelijk zijn uitgerust met schakelingen waarmee handelingen op dergelijke puls-treinen worden uitgevoerd.

10.1 sec.

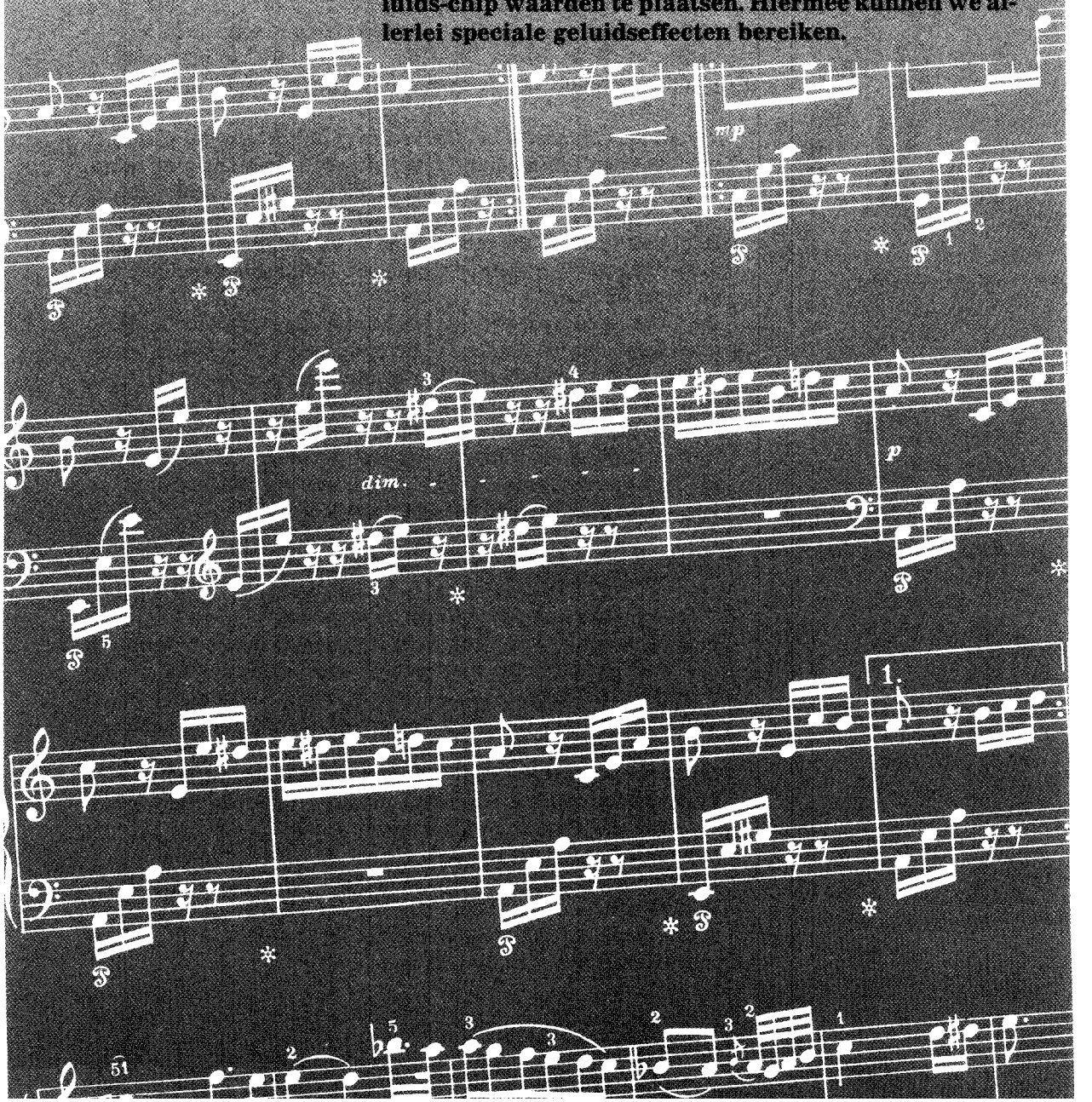
Als we nu niets van computers zouden weten en we zouden zo maar eens een van de elektrische signalen die langs de zenuwen (de elektrische geleiders – draden) van de computer lopen, dan zullen we ook dergelijke signalen oppikken. Let wel, we willen met deze uiteenzetting niet zeggen dat computers precies hetzelfde werken als onze hersenen. Wel willen we zeggen dat we nu zijn beland op een punt dat grootheden (getallen, letters etc.) kennelijk op een heel algemene manier worden voorgesteld en dat er kennelijk ook heel algemene schakelingen zijn voor de bewerking van deze signalen.

In dit verband mogen we niet vergeten dat één van de grondleggers van de moderne computers, John v. Neumann, zo gebiologeerd was door dit onderwerp, dat hij er zelfs een boek over schreef (Het zenuwstelsel als computer).



In dit hoofdstuk bespreken we de mogelijkheden om geluiden op onze computer voort te brengen.

MSX-BASIC biedt hier drie instructies voor. BEEP, PLAY en SOUND. Met BEEP brengen we slechts een korte pieptoon voort. PLAY is een zeer krachtige instructie. Aan de hand van een string die achter PLAY wordt geplaatst, bepalen we de melodie die we willen horen. Zo brengt PLAY "CDE" achtereenvolgens de tonen C, D en E ten gehore. De SOUND-instructie wordt gebruikt om in de registers van de afzonderlijke geluids-chip waarden te plaatsen. Hiermee kunnen we allerlei speciale geluidseffecten bereiken.



18. Geluid: BEEP, PLAY en SOUND

Introductie Geluid speelt in ons leven een belangrijke rol en het mag ons dan ook niet verbazen dat onze computer hier ruime mogelijkheden biedt. Sterker nog, onze computer biedt hier veel ruimere mogelijkheden dan de grote computers die voor het bedrijfsleven en in grote universitaire rekencentra worden gebruikt. De reden van dit opvallende verschil is dat we onze computer o.a. voor educatieve programma's en spelletjes willen gebruiken en daarbij is het produceren van allerlei melodietjes nu eenmaal een niet te verwaarloze smaakmaker. Onze MSX-computer maakt gebruik van een speciaal daartoe vervaardigde 'geluidschip', met andere woorden onze computer beschikt over een afzonderlijke chip waarmee geluiden en wel in drie stemmen (drie onafhankelijke geluidskanalen) kunnen worden opgewekt. Dit biedt o.a. de mogelijkheid om drie-stemmige akkoorden te spelen. In het nu volgende zullen we achtereenvolgens de instructies BEEP, PLAY en SOUND bespreken.

BEEP BEEP wil zo veel zeggen als 'even piepen'. We gebruiken BEEP alleen als waarschuwingssignaal, bijvoorbeeld om de gebruiker aan te geven dat een berekening is afgelopen. Stel bijvoorbeeld eens dat de computer een lange berekening moet uitvoeren. Het is vervelend dat u dan gedurende al die tijd naar het beeldscherm moet turen om te zien wanneer het resultaat verschijnt. Veel plezieriger is het om in die tijd bijvoorbeeld iets na te lezen, en dat u pas na het horen van het piep-signaal, uw blik op het beeldscherm hoeft te werpen.
Onderstaand programma illustreert de opzet:

```
10 PRINT "BEGIN"
20 FOR K=1 TO 1000
30 PRINT K
40 NEXT K
50 BEEP
60 PRINT "KLAAR"
```

We zien hoe de BEEP-instructie hier aan het eind van het programma is geplaatst. Als de computer klaar is met zijn rekenwerk hoort men een korte pieptoon.

PLAY De PLAY-instructie is geheel en al gericht op het eenvoudig aangeven van melodieën.

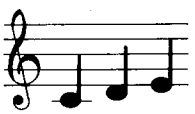
Het is een buitengewoon krachtige instructie waarin we o.a. de volgende zaken kunnen aangeven:

- het tempo waarin gespeeld moet worden.
- de octaaf waar een noot betrekking op heeft.
- de duur van een noot.
- de noot (toon) zelf.
- rustmaten.
- het volume waarmee een noot wordt gespeeld.
- bepaalde geluidseffecten.

PLAY kan als instructie, maar ook als commando worden gebruikt.
Toets maar eens in:

```
PLAY "CDE"
```

Na het indrukken van de RETURN-toets horen we:



Kortom de noten C, D en E en wel die behoren bij de octaaf met de G-sleutel, we zullen deze octaaf voortaan aangeven met 'octaaf 4'.

Het volgende programmaatje maakt van de PLAY-instructie gebruik om een eenvoudig orgeltje van onze MSX-computer te maken.

```
10 A$=INKEY$
20 PLAY A$
30 GOTO 10
```

In regel 10 wordt aan A$ de momentaan ingedrukte toets toegekend, die dan vervolgens in de PLAY-instructie wordt gebruikt etc.

Ons orgeltje is zeker nog niet perfect, want we kunnen o.a. het tempo in het geheel niet regelen. Gelukkig zijn er veel mogelijkheden om een en ander wel te regelen.

Meer geluidskanalen

Als we drie strings, gescheiden door komma's, na PLAY opnemen, zal de MSX-computer iedere string voor een afzonderlijk geluidskanaal gebruiken.
Bijvoorbeeld: bij

```
PLAY "CDE"
```

zal er steeds van één kanaal gebruik worden gemaakt en bij

```
PLAY "CDE", "EFG"
```

zullen tegelijkertijd twee kanalen gebruikt worden en bij

```
PLAY "CDE", "EFG", "GAC"
```

zullen de drie kanalen tegelijkertijd worden gebruikt. Dit houdt in dat de noten C, E en G en hierna D, F en A en ten slotte E, G en C tegelijkertijd ten gehore worden gebracht.

Tempo

Het tempo geven we met een zgn. T-code aan.
Geef maar eens het volgende commando:

```
PLAY "T200CDE"
```

De noten C, D en E worden nu in een hoger tempo dan voorheen gespeeld. Dit klopt, want zonder nadere mededeling neemt de computer 'de stand' T120 aan en T200 geeft zodoende een hoger tempo aan. De T-waarde kan liggen tussen 32 en 256.

Volume

Het volume geven we met een V-code aan.
Ter illustratie:

```
PLAY "T200V13CDE"
```

We horen nu dezelfde wijs als bij het vorige voorbeeld alleen de volume-knop (V) staat kennelijk in een hogere stand. Ook dit klopt want zonder nadere aanduiding neemt de computer 'de stand' V8 aan en V13 komt zodoende op een groter volume neer. De V-code kan variëren tussen 0 (minimaal) en 15 (maximaal).

De noten en de octaaf

De noten geven we aan met de letters C, D, E, F, G, A en B. Bij halve tonen, bijvoorbeeld de CIS, gebruiken we het teken – (of +). Bijvoorbeeld:

```
PLAY "CC#" en evenzo PLAY "CC+"
```

brengt achtereenvolgens de C en de CIS ten gehore. De vraag is nu 'hoe geven we de hoge C aan?' Welnu, de hoge C behoort bij het volgende octaaf. Nu hebben we in het voorafgaande al vermeld dat zonder nadere mededeling, de computer octaaf 4 (O4) veronderstelt.

Om nu de hoge C te produceren, moeten we dus O5 aangeven.
Bijvoorbeeld:

```
PLAY "CDEO5CDE"
```

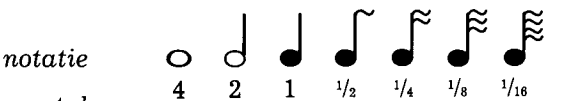
brengt tweemaal de reeks CDE ten gehore, maar bij de laatste reeks wordt op een octaaf hoger gespeeld.

De octaaf-code (O-code) kan minimaal 1 en maximaal 8 zijn en hieruit volgt dan dat onze MSX-computer muziek over acht octaven kan spelen.

Duur van de noten

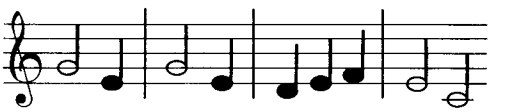
Tot nu toe klonken alle noten even lang. Wie echter een muziekboek openslaat ziet dat tonen een verschillende duur kunnen hebben.

In muziekschrift geven we dat aan door het bolletje al dan niet op te vullen en eveneens door vlaggetjes aan de stok te tekenen.



De duur van een noot geven we met de L-code aan. Zonder nadere opgave neemt de computer L4 aan en dit komt dan neer op een hele noot.

In het volgende illustreren we de L-code aan de hand van 'boer, wat zeg je van m'n kippen'.



Omgezet in een PLAY-commando wordt dit:

```
PLAY "L2GL4EL2GL4EDFGL2EL4C"
```

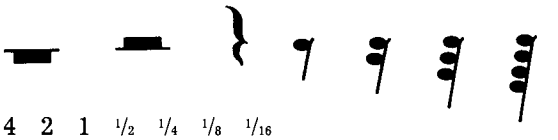
We mogen dit ook korter opschrijven en wel door onmiddellijk achter een noot de duur aan te geven, dus voor ons voorbeeld:

```
PLAY "G2EG2EDEFE2C"
```

Rusttekens

In muziekschrift komen we ook rusttekens tegen. Deze geven aan dat een bepaalde periode, of liever een bepaald aantal tellen, geen toon gespeeld mag worden. We gebruiken hiervoor de R-code.

Het volgende overzicht toont de tekens van het muziekschrift die hiervoor gebruikt worden en wel samen met de R-codes.



aantal tellen	4	2	1	1/2	1/4	1/8	1/16
R-code	1	2	4	8	16	32	64

Het volgende voorbeeld geeft een illustratie:

```
10 PLAY "CDER4DECDECR2": GOTO 10
```

Na de eerste E is er een rust van één tel en na de laatste C een rust van twee tellen. Door PLAY op deze wijze in een programma op te nemen, kan men de maat van de muziek nog beter bestuderen.

We onderbreken de 'schoone wijs' met CTRL/STOP. In deze paragraaf over rusttekens merken we ten slotte nog op dat we achter een noot een punt mogen plaatsen. Deze noot zal dan anderhalf maal zolang klinken, bijvoorbeeld:

```
PLAY "CD.E"
```

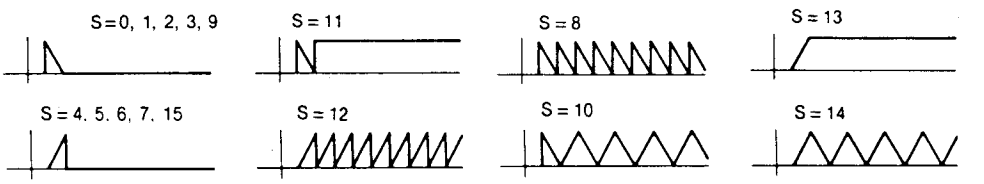
Ook hier kunnen we dus het ritme van onze muziek mee bepalen.

De S- en de M-code

De laatste codes die we bij de PLAY-instructie mogen gebruiken, zijn de S- en de M-code. De S-code heeft te maken met een bepaalde klankkleur die we mee kunnen geven en de M-code bepaald in hoeverre het patroon van die klankkleur herhaald moet worden.

De S-code geeft als het ware aan, hoe ons geluid volgens een bepaald patroon versterkt en verzwakt moet worden. Hierbij gebruiken we de volgende plaatjes:

5=0,1,2,3



De M-code geeft dan de lengte van de cyclus weer, waarin onze met S gekozen vorm herhaald moet worden. Zonder nadere aanduiding neemt de computer voor S de waarde 1 en voor M de waarde 255 aan (M kan liggen tussen 1 en 65535).
Voorbeelden:

```
PLAY "S10M510CDE"
```

en

```
PLAY "S8M6000CDE"
```

We horen bij het eerste voorbeeld bij iedere toon een opvallend stotende klank. Bij het tweede voorbeeld lijkt het wel of de tonen op een piano worden gespeeld. Probeer nu zelf verschillende combinaties uit om de verschillende mogelijkheden van de S- en M-codes te ontdekken.

Een uitgewerkt voorbeeld

Het volgende voorbeeld laat de computer de canon 'vader Jacob' spelen. Men kan zien hoe de strings, die in de PLAY-instructie worden gebruikt, van te voren met string-variabelen worden opgebouwd.

```
 10 CLEAR 500
 20 A$="L4CDECR64CDEC"
 30 B$="EFG2EFG2"
 40 C$=L8GAGFL4ECL8GAGFL4EC"
 50 D$="C03G04C2C403G04C2"
 60 W$=A$+B$+C$+D$
 70 W1$=W$
 80 W2$="R1R1"+W$
 90 W3$="R1R1"+W2$
100 PLAY W1$,W2$,W3$
```

(N.B. in de string in regel 50 worden hoofdletters O gebruikt.)

De CLEAR-instructie wordt gebruikt om voldoende ruimte voor de opslag van strings te reserveren. In regel 20, 30, 40 en 50 wordt de melodie in de kenmerkende stukken aangegeven. Regel 60 geeft de combinatie van de strings en zo de hele wijs weer. Regel 70 en 80 bepalen de tweede en derde stem.

SOUND

De laatste instructie die we in dit hoofdstuk bespreken, is de SOUND-instructie. Deze instructie kan men alleen goed begrijpen door te beseffen dat onze afzonderlijke geluids-chip is uitgerust met een aantal registers (zie hoofdstuk 1). Door nu bepaalde waarden in die registers te plaatsen, produceert die chip allerlei geluiden.

Onze geluids-chip is uitgerust met 13 registers die de volgende betekenis hebben:

Register-nummer	*wordt gebruik voor*
0,1	de combinatie van de inhoud van register 0 en 1 bepaalt de frequentie van de toon voor geluidskanaal A.
2,3	evenzo, maar nu voor kanaal B.
4,5	evenzo, maar nu voor kanaal C.
6	bepaalt de klankkleur van een voort te brengen ruisachtig geluid. De waarde kan van 0 tot 31 variëren.
7	bepaalt of een bepaald geluidskanaal ruis, dan wel een toon weergeeft.
8	bepaalt volume van kanaal A (waarde tussen 0 en 15).
9	als 8, maar nu voor kanaal B.
10	als 8, maar nu voor kanaal C.
11, 12	bepaalt de klankkleur en wel de zgn. modulatie van een toon.
13	bepaalt de klankkleur (aanzwel- en uitsterfpatronen).

Om de frequentie van een bepaalde toon te berekenen, maken we gebruik van het volgende hulpprogramma:

```
10 INPUT "GEWENSTE FREQUENTIE"; F
20 A=1789772.5
30 B=INT(A/(16*F))
40 C=INT(B/256)
50 D=B-C
60 PRINT D,C
```

De waarden van D en C zijn dan de waarden die in het eerste en tweede register geplaatst moeten worden om de desbetreffende toon te verkrijgen.

De volgende tabel geeft enkele veel voorkomende frequenties.

getal	*toon*	*getal*	*toon*
110	A	440	a'
117	ais, bes	466	ais', bes'
123	B	494	b'
131	c	523	c"
139	cis, des	554	cis", des"
147	d	587	d"
156	dis, es	622	dis", es"
165	e	659	e"
175	f	698	f"
185	fis, ges	740	fis", ges"

getal	*toon*	*getal*	*toon*
196	g	784	g"
208	gis, as	831	gis", as"
220	a	880	a"
233	ais, bes	932	ais", bes"
247	b	988	b"
262	c'	1047	c"'
277	cis', des'	1109	cis"', des"'
294	d'	1175	d"'
311	dis', es'	1245	dis"', es"'
330	e'	1319	e"'
349	f'	1397	f"'
370	fis', ges'	1480	fis"', ges"'
392	g'	1568	g"'
415	gis', as'	1661	gis"', as"'
		1760	a"'

Voorbeelden Stel, we willen de A produceren (frequentie 440). Ons programmaatje produceert de waarden:

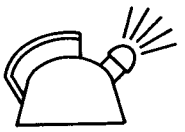
```
254        0
```

Dit leidt dan tot de volgende SOUND-instructies:

```
10 SOUND 0,254
20 SOUND 1,0
30 SOUND 8,11
```

We horen nu de toon A (probeer ook PLAY 'A'). Het geluid kunnen we onderbreken met CTRL/STOP.

Ruis In de praktijk zullen we de SOUND-instructie gebruiken om allerlei opvallende geluidseffecten, zoals gillende sirenes en plofgeluiden te maken. De volgende voorbeelden geven enkele mogelijkheden aan:



```
10 REM FLUITKETEL
20 FOR K=255 TO 0 STEP-1
30 PRINT K
40 SOUND 0,K
50 SOUND 1,0
60 SOUND 8,10
70 NEXT K
```

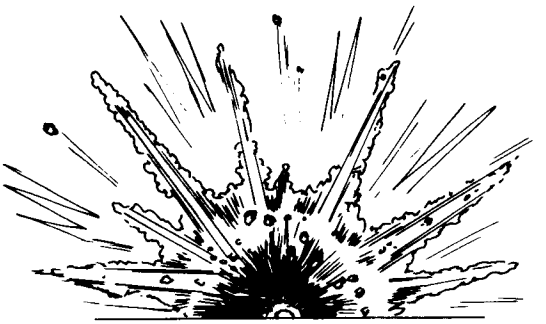
Laat men regel 30 weg dan verkrijgt men een versnelde variatie van het geluid. Het volgende programmaatje heeft het geluidseffect van een stervende packman.

```
10 REM STERVENDE PACKMAN
20 FOR K=0 TO 255
30 SOUND 0,K
40 SOUND 1,0
50 SOUND 8,10
60 NEXT K
```

Met het volgende voorbeeld kunnen we nog verschillende effecten uitproberen:

```
10 INPUT K,L,M
20 SOUND 0,250:SOUND 1,0
30 SOUND 6,K:SOUND 7,L:SOUND 13,M
40 FOR J=15 TO 0 STEP -0.05
50 SOUND 8,J
60 NEXT J
```

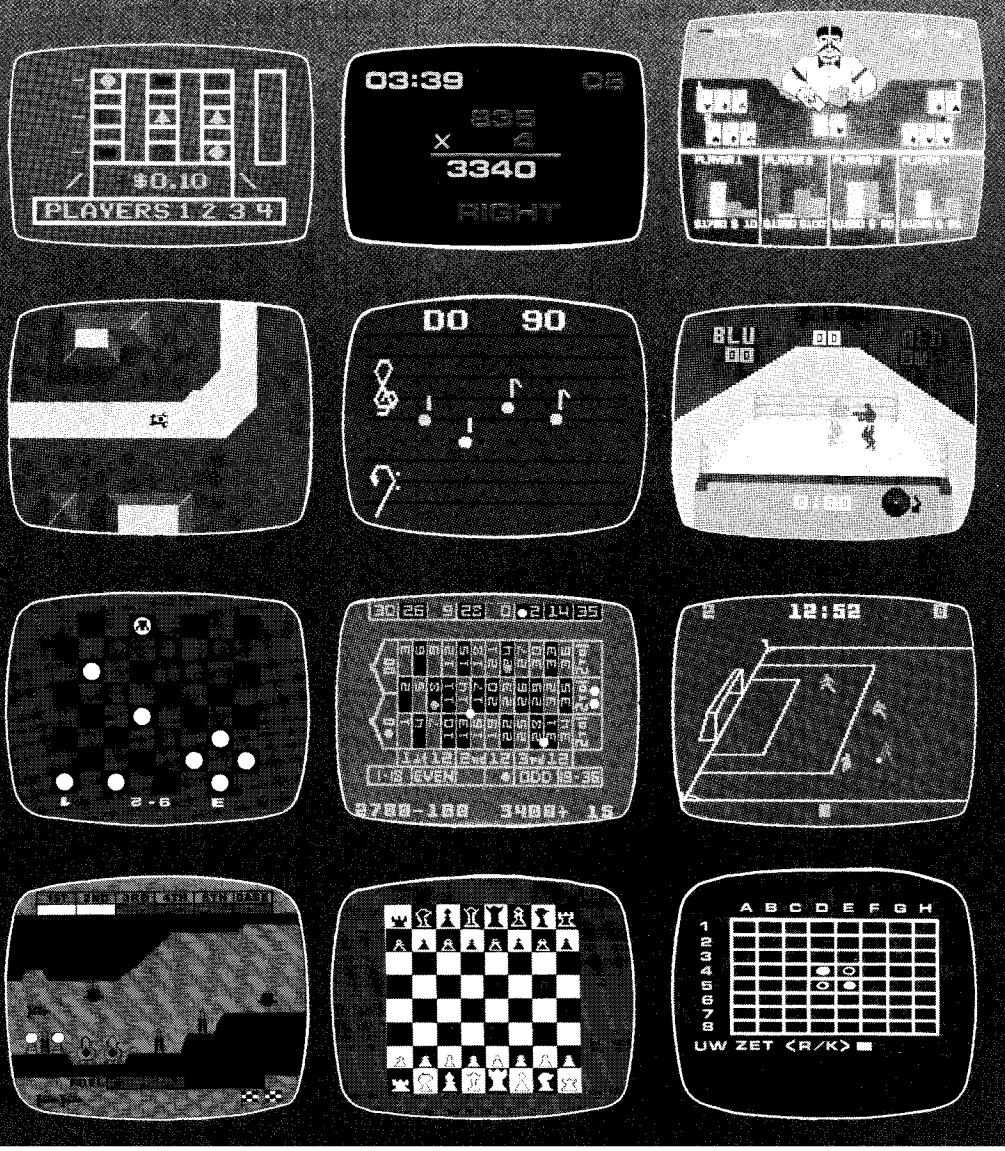
Voor K=10, L=10 en M=10 levert dit een uitstervende toon, maar voor K=20, L=20 en M=20 hoort men een explosie.



In dit hoofdstuk wordt de mogelijkheid om met sprites te werken besproken. Een sprite is een figuurtje dat we zelf kunnen definiëren. Dit gebeurt dan aan de hand van de SPRITE-instructie.

Vervolgens kunnen we met behulp van de PUT SPRITE-instructie dit figuurtje op een willekeurige positie op het scherm plaatsen.

Als we de coördinaten van een sprite op bepaalde wijze veranderen, heeft dat tot effect dat het figuurtje over het scherm beweegt. Met behulp van de ON SPRITE GOSUB-instructie kunnen we naar een willekeurige subroutine springen, als twee sprites elkaar overlappen ('botsen').



19. Sprites

Sprite

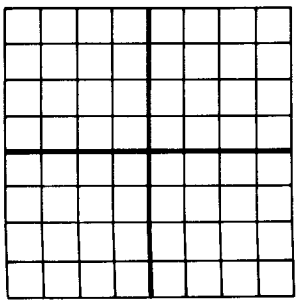
De allereerste vraag die we ons in dit hoofdstuk stellen, is natuurlijk 'wat is precies een sprite?'. Welnu een sprite is een figuurtje dat we kunnen vastleggen met behulp van een ruitjespatroon. Een dergelijke figuur geven we vervolgens een nummer. Daarna kunnen we aan de hand van een speciale instructie allerlei manipulaties met zo'n figuurtje uitvoeren. De belangrijkste manipulatie is dat we zo'n figuurtje eenvoudig over het scherm kunnen verplaatsen.

Uiteraard kunnen we ook met PRINT- en PSET-instructies allerlei figuurtjes definiëren. Voor het verplaatsen van dergelijke met PRINT- en PSET-instructies opgebouwde figuurtjes zijn echter relatief veel instructies nodig, met als resultaat dat een snelle verplaatsing van een zo opgebouwde figuur in feite niet mogelijk is.

Het werken met sprites geeft dus wezenlijk meer mogelijkheden. Het vormt met name een krachtig gereedschap om bijvoorbeeld packman-achtige wezens over het scherm te laten dwalen.

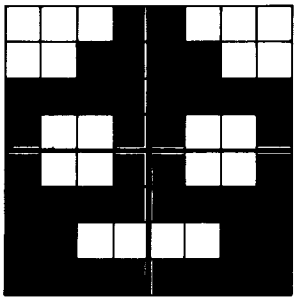
Het vastleggen van een sprite

In het nu volgende zullen we tonen hoe we zo'n figuurtje, met andere woorden zo'n sprite, definiëren. Hierbij gaan we uit van een 8×8 matrix.



Vervolgens kleuren we dan bepaalde vakjes zwart om de figuur te verkrijgen die we wensen.

Om bijvoorbeeld een spookje te verkrijgen, kunnen we vakjes als volgt inkleuren:



```
0001 : 1000   18
0011 : 1100   3C
1111 : 1111   FF
1001 : 1001   99
1001 : 1001   99
1111 : 1111   FF
1100 : 0011   C3
1111 : 1111   FF
```

Links zien we het spookje en in de linkerkolom ziet men hoe we met behulp van enen en nullen (zwart en wit) dezelfde figuur kunnen omschrijven.

De stippellijn is een 'hulplijn'; deze splitst iedere regel in twee rijtjes van vier enen en/of nullen. De bovenste regel bestaat bijvoorbeeld uit de reeksen 0001 en 1000.

In de rechterkolom ziet men een notatie met andere codes. Deze codes kan men direct uit de volgende tabel afleiden. (N.B. het gaat hier om de zgn. hexadecimale notatie, die in hoofdstuk 8 werd besproken).

reeks code	*reeks code*	*reeks code*	*reeks code*
0000 0	0100 4	1000 8	1100 C
0001 1	0101 5	1001 9	1101 D
0010 2	0110 6	1010 A	1110 E
0011 3	0111 7	1011 B	1111 F

Met behulp van de zo gevonden codes definiëren we nu in een programma de sprite en wel met de volgende instructie:

SPRITE$ (nummer)= reeks CHR$-instructies

Voor ons voorbeeld:

```
SPRITE$(1)=CHR$(&H18)+CHR$(&H3C)+CHR$(&HFF)+
           CHR$(&H99)+CHR$(&H99)+CHR$(&HFF)+
           CHR$(&HC3)+CHR$(&HFF)
```

Merk op dat hier de reeks codes 18, 3C, FF, etc. in CHR$-instructies zijn opgenomen, waarbij iedere code wordt voorafgegaan door de tekens &H.

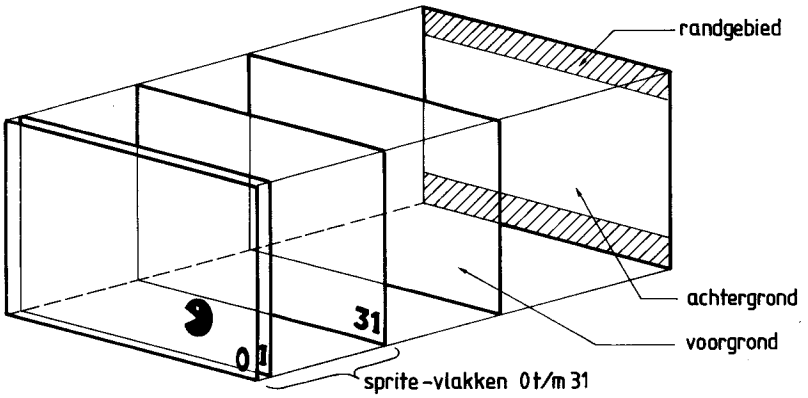
PUT SPRITE

Op de zojuist besproken wijze is onze sprite (nummer 1) volledig vastgelegd en we kunnen vervolgens het figuurtje op een willekeurige plaats op het scherm plaatsen. Hiervoor gebruiken we de PUT SPRITE-instructie, die de volgende vorm heeft:

```
PUT SPRITE vlak, (x,y), kleur, sprite-nummer
```

Hierin is:

vlak een nummer tussen 0 en 31. Men moet hierbij denken aan een volgende opzet:



De sprites bewegen zich als het ware over doorzichtige glazen schermen, die op bovenstaande wijze achter elkaar zijn geplaatst. De ligging van een vlak bepaalt welke sprite wordt afgedekt als twee sprites elkaar overlappen.

(x, y) de coördinaten van de positie waar de sprite moet worden afgebeeld.

kleur het getal waarmee de kleur van de sprite wordt aangegeven.

nummer nummer dat aan de sprite gegeven is en wel aan de hand van de SPRITE$ (nummer)-instructie.

ON SPRITE GOSUB en SPRITE ON

Bij veel spelletjes is het belangrijk om te weten of op een gegeven moment twee sprites elkaar overlappen, of populair gezegd, of er een botsing optreedt.

Om dit te kunnen constateren, biedt MSX-BASIC de ON SPRITE GOSUB-instructie. Deze heeft steeds de volgende vorm:

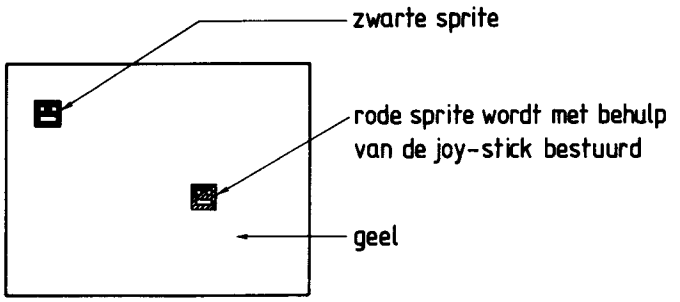
```
ON SPRITE GOSUB regelnummer
```

Als de computer nu constateert dat twee sprites elkaar overlappen, wordt naar een subroutine via het aangegeven regelnummer gesprongen.

Het is dan wel noodzakelijk dat we de waakzaamheid van de

computer daartoe activeren. Dat gebeurt met behulp van de SPRITE ON-instructie.

Een voorbeeld Het volgende programma toont een voorbeeld. Op het scherm komen twee spookjes te staan. Het ene (zwarte) spookje staat stil, terwijl het andere (rode) spookje door middel van de joy-stick bestuurd wordt. Het besturingsmechanisme hebben we reeds in hoofdstuk 15 besproken.
In beeld:



Het volledige programma luidt:

```
 10 CLS
 20 COLOR,11,11
 30 SCREEN 2
 40 X=100:Y=100
 50 FOR K=1 TO 2
 60 SPRITE$(K)=CHR$(&H18)+CHR$(&H3C)+CHR$(&HFF)+
               CHR$(&H99)+CHR$(&H99)+CHR$(&HFF)+
               CHR$(&HC3)+CHR$(&HFF)
 70 NEXT K
 80 SPRITE ON
 90 PUT SPRITE 0,(40,40),1,1
100 D=STICK(1)
110 IF D=1 THEN Y=Y-1
120 IF D=2 THEN X=X+1:Y=Y-1
130 IF D=3 THEN X=X+1
140 IF D=4 THEN X=X+1:Y=Y+1
150 IF D=5 THEN Y=Y+1
160 IF D=6 THEN X=X-1:Y=Y+1
170 IF D=7 THEN X=X-1
180 IF D=8 THEN X=X-1:Y=Y-1
190 PUT SPRITE 1,(X,Y),6,2
200 ON SPRITE GOSUB 220
210 GOTO 100
220 PLAY "CCC"
230 GOTO 230
240 RETURN
```

Regel 10 t/m 30 spreken voor zich. Regel 40 bepaalt de beginpositie van de rode sprite. Hierna worden in regel 50 t/m 70 twee sprites gedefinieerd, die overigens precies dezelfde vorm hebben (namelijk de reeds besproken vorm).

Hierna wordt de computer in de toestand 'attent op botsingen' geplaatst en wel met de SPRITE ON-instructie (regel 80). Regel 90 plaatst de zwarte sprite op het scherm en wel op positie (40,40). Vervolgens ziet men de bekende constructie om de stand van de joystick uit te lezen. Deze stand wordt dan meegegeven aan de PUT SPRITE-instructie van regel 190. Hiermee wordt dan de rode sprite geplaatst.

Regel 200 toont de 'botsings-proef'; als er een botsing optreedt, springt de computer naar de subroutine van regel 220.

Opmerkingen We plaatsen bij het werken met sprites ten slotte nog de volgende opmerkingen:

- we kunnen de sprite een factor 2 vergroten en wel door regel 30 te vervangen door: SCREEN 2,1
- we mogen sprites ook definiëren in een matrix van 16×16. Een dergelijke matrix bestaat dus uit 4 blokken van een 8×8-matrix. Deze blokken worden als volgt genummerd

1	3
2	4

De eenvoudigste manier is dan om in vier strings dit patroon vast te leggen:

```
A$=CHR$()+ etc.      (definitie blok 1)
B$=CHR$()+ etc.      (definitie blok 2)
```

en evenzo C$ en D$ en vervolgens

```
SPRITE$(1)=A$+B$+C$+D$
```

In het geval dat men met 16×16-sprites werkt, gebruikt men de instructie SCREEN 2,2 of SCREEN 2,3 in het geval men een vergroting met een factor 2 wenst.

- we mogen maximaal 256 sprite-patronen met behulp van een 8×8-matrix definiëren en 64 patronen met behulp van een 16×16-matrix. Bedenk echter dat op elk sprite-vlak hoogstens één sprite mag worden weergegeven.
- men kan maximaal 4 sprite-patronen afbeelden die horizontaal op een rij geplaatst staan.

Appendix A Bestanden

Inlezen

Een bestand (Engels: file) is een verzameling gegevens die onder een bepaalde naam is opgeslagen. De gegevens worden steeds in een bepaalde volgorde opgeslagen en op grond van dit feit kan men de gegevens ook terugvinden. De volgorde is in feite zeer eenvoudig: de gegevens worden na elkaar (sequentieel) opgeslagen.
Een voorbeeld verduidelijkt dit.

```
10 REM DEMO INLEZEN BESTAND
20 A$="AAP"
30 B$="NOOT"
40 C$="MIES"
50 OPEN "CAS:DATA" FOR OUTPUT AS#1
60 PRINT#1,A$;B$;C$
70 PRINT#1,B$
80 PRINT#1,C$
90 CLOSE#1
```

In dit voorbeeld worden de strings A$, B$ en C$ opgeslagen.

Daartoe wordt eerst een bestand geopend, dat wil zeggen we definiëren hier een bestand door een naam aan te geven (DATA), dit bestand een label te geven (#1) en aan te geven dat hier de cassette als opslag-medium (CAS) wordt gebruikt.

De algemene vorm van de OPEN-instructie is:

OPEN "naam opslag medium: [naam bestand]" FOR OUTPUT AS[# nummer van bestand].

Voor 'naam opslag medium' kan men gebruiken

CAS	cassetterecorder
CRT	tekst-beeldscherm
GRP	grafisch beeldscherm
LPT	printer

De 'naam bestand' mag uit maximaal 6 tekens bestaan. Als het bestand op deze wijze eenmaal is geïntroduceerd kunnen we er gegevens in opslaan. Dit gaat via de PRINT-instructie, waarachter nu de tekens #, zijn geplaatst (regel 60).

Na afloop moet, zoals men dit noemt 'het bestand gesloten worden'. Dit gebeurt met de CLOSE-instructie die de volgende vorm heeft:

CLOSE [# nummer van bestand]

Plaatsen van tekst op het grafische scherm

Op deze wijze kan men ook schrifttekens op het grafisch scherm (GRP) plaatsen.
De constructie is dan bijvoorbeeld:

```
OPEN "GRP:BEELD" FOR OUTPUT AS #1
PRESET (50,50)
PRINT #, "RESULTAAT"
```

Met de PRESET-instructie wordt hier bepaald op welke plaats de tekst moet verschijnen.

Uitlezen

Bij het uitlezen van een bestand ziet men een vrijwel gelijke constructie als bij het inlezen. Wederom wordt het bestand geopend maar nu met de uitdrukking FOR INPUT en in plaats van de PRINT#,-instructie gebruiken we nu de INPUT#,-instructie.
Het volgende voorbeeld sluit op ons eerste voorbeeld aan:

```
10 REM DEMO UITLEZEN BESTAND
20 OPEN "CAS:DATA" FOR INPUT AS#1
30 INPUT #1, A$,B$,C$
40 PRINT A$,B$,C$
50 CLOSE #1
```

De PRINT-instructie op regel 40 is alleen opgenomen om het resultaat te kunnen controleren. Indien men van te voren niet weet hoeveel gegevens het bestand bevat, kan men de volgende constructie gebruiken:

```
10 OPEN "CAS:DATA" FOR INPUT AS#1
20 IF EOF(1)=-1 THEN GOTO 60
30 INPUT #1, X$
40 PRINT X$
50 GOTO 20
60 CLOSE #1
```

Nu zal iedere waarde van het bestand worden afgedrukt en als het einde van het bestand wordt aangetroffen, volgt een sprong naar regel 60.

Ten slotte plaatsen we nog de volgende opmerkingen:

- het maximale aantal bestanden dat tegelijkertijd geopend mag zijn, bedraagt 1 tenzij met een MAXFILE-commando een groter aantal is opgegeven (bijv. MAXFILE=3)
- in plaats van PRINT #, kan men ook de vorm PRINT USING #, gebruiken
- in plaats van INPUT #, mag men ook de vorm LINE INPUT #, gebruiken
- in plaats van strings kan men ook getallen in bestanden opslaan

Appendix B

Tekens en hun codes

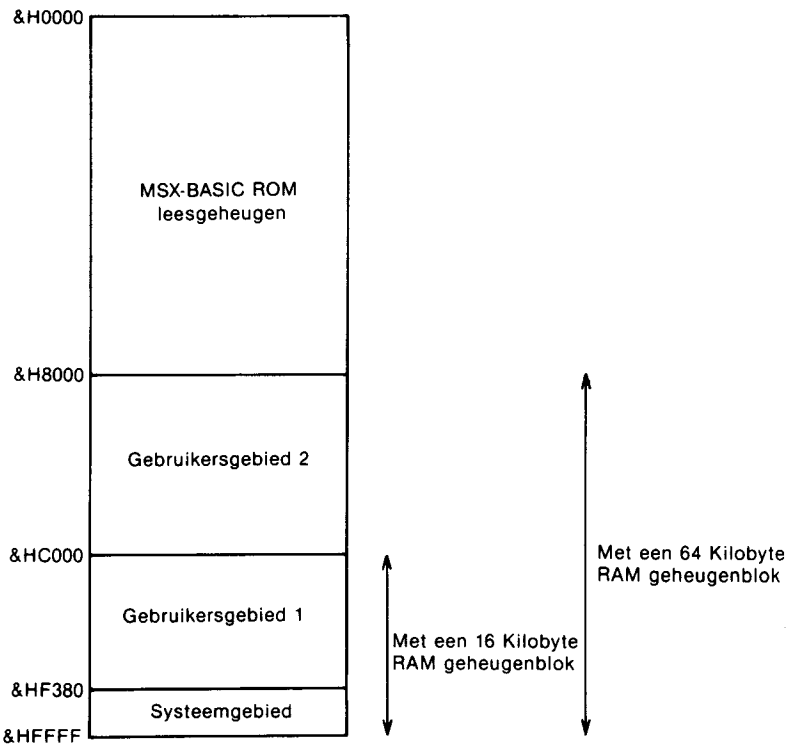
De volgende tabel geeft een overzicht van alle tekens en hun bijbehorende codes.

code	teken	code	teken	code	teken	code	teken
0	(nul)	32	(spatie)	64	@	96	`
1	☺	33	!	65	A	97	a
2	☻	34	"	66	B	98	b
3	♥	35	#	67	C	99	c
4	♦	36	$	68	D	100	d
5	♣	37	%	69	E	101	e
6	♠	38	&	70	F	102	f
7	•	39	'	71	G	103	g
8	◘	40	(	72	H	104	h
9	○	41	)	73	I	105	i
10	◙	42	*	74	J	106	j
11	♂	43	+	75	K	107	k
12	♀	44	,	76	L	108	l
13	♪	45	—	77	M	109	m
14	♫	46	.	78	N	110	n
15	☼	47	/	79	O	111	o
16	┼	48	Ø	80	P	112	p
17	┴	49	1	81	Q	113	q
18	┬	50	2	82	R	114	r
19	┤	51	3	83	S	115	s
20	├	52	4	84	T	116	t
21	┼	53	5	85	U	117	u
22	│	54	6	86	V	118	v
23	─	55	7	87	W	119	w
24	┌	56	8	88	X	120	x
25	┐	57	9	89	Y	121	y
26	└	58	:	90	Z	122	z
27	┘	59	;	91	[	123	{
28	╳	60	<	92	\	124	¦
29	╱	61	=	93	]	125	}
30	╲	62	>	94	^	126	~
31	┼	63	?	95	_	127	

code	teken	code	teken	code	teken	code	teken
128	Ç	160	á	192		224	α
129	ü	161	í	193		225	β
130	é	162	ó	194		226	Γ
131	â	163	ú	195		227	Π
132	ä	164	ñ	196		228	Σ
133	à	165	Ñ	197		229	σ
134	å	166	ª	198		230	μ
135	ç	167	º	199		231	γ
136	ê	168	¿	200		232	Φ
137	ë	169	⌐	201		233	Θ
138	è	170	¬	202		234	Ω
139	ï	171	½	203		235	δ
140	î	172	¼	204		236	∞
141	ì	173	¡	205		237	Ø
142	Ä	174	«	206		238	∈
143	Å	175	»	207		239	∩
144	É	176	Ã	208		240	≡
145	æ	177	ã	209		241	±
146	Æ	178	Ĩ	210		242	≥
147	ô	179	ĩ	211		243	≤
148	ö	180	Õ	212		244	⌠
149	ò	181	õ	213		245	⌡
150	û	182	Ũ	214		246	÷
151	ù	183	ũ	215		247	≈
152	ÿ	184	Ĳ	216	△	248	°
153	Ö	185	ĳ	217	ǂ	249	•
154	Ü	186	¾	218	ω	250	·
155	¢	187	∽	219		251	√
156	£	188	◇	220		252	η
157	¥	189	‰	221		253	²
158	₧	190	¶	222		254	■
159	ƒ	191	§	223		255	

Appendix C Geheugenindeling

Geheugenindeling De volgende schema's tonen de indeling van het geheugen. De adressen zijn hexadecimaal opgegeven.



Appendix D

Overzicht van de BASIC-instructies en -commando's

In deze appendix worden alle BASIC-instructies en -commando's kort behandeld. Van iedere instructie en van ieder commando wordt de volledige syntax (dat wil zeggen de volledige schrijfwijze) gegeven. Daarbij worden twee symbolen gebruikt met de volgende betekenis:

[...] alles tussen vierkante haken is optioneel, dat wil zeggen: kan eventueel worden weggelaten;

< ... > gegevens tussen deze haken moeten door de gebruiker zelf worden ingevuld.

AUTO [<regelnummer>],[<stapgrootte>]
Genereert automatisch regelnummers tijdens het invoeren van een programma. Als geen <regelnummer> wordt opgegeven, dan wordt begonnen met regelnummer 10 en anders met het opgegeven <regelnummer>. Is geen <stapgrootte> opgegeven dan wordt het regelnummer steeds met 10 verhoogd, anders met de opgegeven <stapgrootte>.

Het commando wordt beëindigd met CRTL/C, dat wil zeggen met ingedrukte CTRL-toets wordt op C gedrukt.

BEEP
Genereert een pieptoon met een beperkte duur.

BLOAD "<dev>[<bestandsnaam>]"[,R][, <verschuiving>]
Hiermee wordt een in machinecode geschreven programma geladen dat met een BSAVE-commando is opgeslagen.

Vult men voor dev de term CAS in dan wordt daarmee de cassetterecorder aangegeven. De term bestandsnaam slaat op de naam die is meegegeven (max 6 lettertekens, bijvoorbeeld PROG1). Geeft men de R op dan wordt na het laden het programma onmiddellijk 'gerund'.

Geeft men een verschuiving op (geheel getal) dan wordt het nieuwe startadres gelijk aan het oude + de verschuiving

BSAVE "<dev>[<bestandsnaam>]",<beginadres, eindadres> [<beginadres voor start van het programma>]
Hiermee wordt een in machinecode geschreven programma opgeslagen op een extern medium. De term dev en bestandsnaam zijn bij BLOAD verklaard. Begin- en eindadres markeren het geheugengebied dat op het externe medium wordt gekopieerd.

CALL <naam>[<argument>,<argument>,...)]
Algemeen commando om bepaalde subroutines die in ROM zijn opgeslagen uit te voeren.

CIRCLE [STEP] (<x,y>),<r>,[<kleur>],[<beginhoek>], [<eindhoek>][,<afplatting>]
Genereert een ellips dan wel (met afplatting 1.4) een cirkel. Met (x,y) wordt het middelpunt opgegeven, r geeft de straal aan. Eventueel kan men slechts een gedeelte tekenen door een begin- en eindhoek op te geven (opgeven in radialen). Laat men STEP weg dan wordt het gewone coördinatenstelsel genomen. Mét STEP schuift het coördinatenstelsel op naar het na STEP genoemde punt.

CLEAR[<geheugenruimte voor strings>][,<hoogste adres>]
Hiermee worden alle variabelen gelijk aan 0 gemaakt en bovendien wordt een geheugengebied voor het opslaan van characters (strings, letters) gereserveerd.

Het 'hoogste adres' bepaalt het hoogste adres voor gebruik in BASIC. Ten slotte heeft CLEAR tot gevolg dat alle eventueel nog geopende bestanden gesloten worden. Bedenk dat zonder CLEAR de computer slechts 200 woorden reserveert voor het opslaan van letters etc.

CLOAD ["<bestandsnaam>"]
Commando om een programma, eventueel onder naam, (max. 6 letters en/of cijfers) op cassette op te slaan. Zo heeft CLOAD "PROG4" tot gevolg dat uw programma vanuit het geheugen onder de naam PROG4 op cassette wordt opgeslagen. Als er iets mis gaat, dient men het laden met CTRL/STOP te onderbreken, de band terug te spoelen en het laden opnieuw te starten.

CLOAD? ["<bestandsnaam>"]
Commando om een eenmaal op cassette opgeslagen programma te vergelijken met het in het geheugen opgeslagen programma. Is alles goed, dan verschijnt "OK". Als er een fout optreedt, verschijnt "Device I/O error".

CLOSE [#][<bestandsnummer>][,<bestandsnummer>,...]
Sluit de bestanden met de opgegeven nummers. Geeft men geen nummer op dan worden alle bestanden gesloten.

COLOR [<kleur voorgrond>],[<kleur achtergrond>],[<kleur randgebied>]
Hiermee kan men de kleur van de aangegeven gebieden bepalen. De kleur geeft men steeds met een cijfer aan:

cijfer	*kleur*	*cijfer*	*kleur*
0	transparant	8	rood
1	zwart	9	lichtrood
2	groen	10	donkergeel
3	lichtgroen	11	lichtgeel
4	donkerblauw	12	donkergroen
5	lichtblauw	13	magenta
6	donkerrood	14	grijs
7	hemelsblauw	15	wit

De uitgangssituatie (situatie bij aanzetten van computer) komt overeen met COLOR 15,4,4

Voorbeelden:

COLOR 2
alleen de voorgrond wordt groen

COLOR „5
alleen de achtergrond wordt lichtblauw

Bij grafische afbeeldingen kan men alleen de kleur van de achtergrond wijzigen als eerst CLS is uitgevoerd.

CLS
Veegt het beeld schoon

CONT
Met dit commando wordt de uitvoering van het prograamma hervat en wel vanaf de plaats waar het werd onderbroken met CTRL/STOP dan wel met de STOP- of END-instructie

CSAVE "bestandsnaam" [,<snelheid in band>]
Commando om een programma op cassette op te slaan (zie ook CLOAD). Voor de snelheid in baud geldt:

1 1200 baud
2 2400 baud

Laat men deze aanduiding weg dan wordt een snelheid van 1200 baud aangenomen.

DATA n1,n2,n3,n4, ...
Met deze instructie kan men een lijst met vaste waarden opnemen in een programma die dan één voor één kunnen worden

opgehaald met de READ-instructie. De lijst mag zowel uit numerieke waarden bestaan als uit strings (tussen aanhalingstekens (")) en ze moeten door komma's worden gescheiden. READ haalt de waarden sequentieel op. Men kan weer vooraan beginnen met het uitlezen van een DATA-lijst door eerst de RESTORE-instructie uit te voeren.

DEF FN<functienaam>[(<argumentenlijst>)]=<functiedefinitie>
Hiermee definieert men eigen functies. De naam waaronder de functie moet worden aangeroepen is de <functienaam> voorafgegaan door FN. Eventueel kan men aan de functie argumenten meegeven, die dan worden gebruikt in de <functiedefinitie>. De namen van de argumenten dienen alleen om de functie te definiëren en hebben geen invloed op eventuele variabelen in het hoofdprogramma met dezelfde naam. De argumenten moeten worden gescheiden door een komma.

In de <functiedefinitie> mogen ook variabelennamen voorkomen die niet als argument worden meegegeven in de <argumentenlijst>, maar voor de aanroep van de functie reeds een waarde hebben gekregen in het hoofdprogramma. De <functiedefinitie> mag maximaal één regel lang zijn.
Voorbeeld:

```
1Ø DEF FNAB(X,Y)=X^2+Y*Z
2Ø Z=5
3Ø I=2:J=3
4Ø T=FNAB(I,J)
5Ø PRINT T
6Ø END

RUN
 19
```

DEF <type> <letter(s)>
Deze instructie zorgt er voor dat alle variabelen die beginnen met de gegeven <letter(s)> van het opgegeven <type> zijn. De mogelijke <typen> zijn:

INT integer
SGN real
DBL real met dubbele precisie
STR string

Een type-declaratieteken (dus %, #, $) heeft voorrang boven de DEF-instructie. Voorbeelden:

`10 DEFDBL A-E` Alle variabelen die beginnen met A, B, C, D of E zijn dubbele-precisievariabelen.

`10 DEFSTR A` Alle variabelen die beginnen met de letter A zijn stringvariabelen.

DEF USR[<getal>]=<integer-uitdrukking>
Hiermee specificeert men het startadres van een subroutine in machinetaal. <getal> moet tussen 0 en 9 liggen en wordt als naam toegewezen aan de machinetaalroutine. Laat men <getal> weg dan wordt 0 aangenomen. Op grond van dit <getal> en met behulp van de USR-functie kan men nu de betreffende subroutine in machinetaal aanroepen. Voorbeeld:

```
10 DEF USR0=1000
20 X=USR0(9^2)
```

DELETE [<regelnummer>][–[<regelnummer>]]
Verwijdert alle opgegeven regels. Na uitvoering van dit commando wordt altijd teruggekeerd naar de BASIC-commandomode. Voorbeelden:

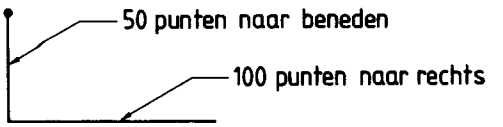
DELETE	verwijder alle regels
DELETE 10	verwijder regel 10
DELETE 10–40	verwijder de regels 10 t/m 40
DELETE –40	verwijder alle regels vanaf 1 t/m 40
DELETE 40–	verwijder alle regels vanaf 40

DIM <array-naam>
(<maximum index>)[,<array-naam>
Creëert geheugenruimte voor de gespecificeerde array(s) en initialiseert de array-elementen op nul. Wanneer aan een array wordt gerefereerd die niet van tevoren door een DIM-instructie is gecreëerd, dan wordt 10 als maximum index aangenomen. De laagste index is altijd 0, tenzij we met OPTION BASE deze op 1 hebben gezet. Met ERASE kunnen we een array wissen.

DRAW "<string>"
Met behulp van DRAW kan men allerlei rechte lijnen aangeven met behulp van eenvoudige codes. De codes vormen steeds de string.
Voorbeeld:

```
10 SCREEN 2
20 DRAW "D50R100"
30 GOTO 30
```

Geeft twee aan eensluitende rechte lijnen. De eerste gaat over 50 beeldpunten naar beneden ("down 50" of afgekort D50) en de tweede 100 punten naar rechts (R100). Het resultaat is dus als volgt

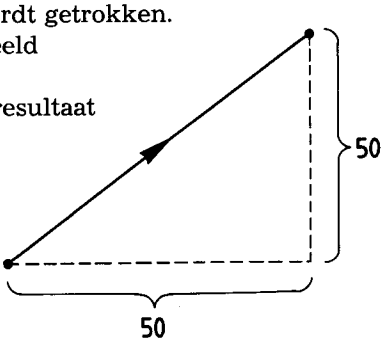


Voor ieder lijnstuk mogen we ook een kleur opgeven en wel door middel van de letter C en een kleurcode, bijvoorbeeld:

```
DRAW "C8D50C10R100"
```

De volgende tabel geeft een overzicht van de mogelijkheden:

Code	*Betekenis*
S	Geeft de schaal aan, bijvoorbeeld S4 geeft aan dat de fijnste eenheid uit 4 stippen bestaat
A	Na A komt 0, 1, 2 of 3 te staan. Hiermee verdraaien we het coördinatensysteem in stappen van 90°. Zonder A opgave neemt de computer A0 aan.
C	Geeft de kleur aan. Zie bovenstaand voorbeeld.
M	Voor het tekenen van een schuine lijn. Na M komen twee getallen gescheiden door een komma, bijvoorbeeld: M30,50 In dit voorbeeld zal een schuine lijn getrokken worden vanaf de laatste positie, tot het punt dat we krijgen door bij de x-coördinaat 30 en bij de y-coördinaat 50 op te tellen. De waarden x en y mogen ook negatief zijn.
U	U slaat op 'up' dat wil zeggen 'naar boven'. Bijvoorbeeld: U30 geeft aan dat vanaf de laatste positie een lijn naar boven getrokken moet worden en wel over 30 beeldpunten.
D	D slaat op 'down'. Voor het trekken van een lijn naar beneden.
R	R slaat op 'right'. Voor het trekken van een lijn naar rechts.
L	L slaat op 'left'. Voor het trekken van een lijn naar links.
E	Voor een lijn die onder een hoek van 45° naar rechtsboven wordt getrokken. Bijvoorbeeld E50 geeft als resultaat



F Voor een lijn die onder een hoek van 45° naar rechtsonder wordt getrokken (zie ook E)

G Voor een lijn die onder een hoek van 45° naar linksboven wordt getrokken (zie ook E)

H Voor een lijn die onder een hoek van 45° naar linksonder wordt getrokken (zie ook E)

Merk op dat lijnen steeds vanaf de laatst bereikte positie worden getrokken. Een vaste beginpositie kunnen we verkrijgen met BMx,y waarin x en y de begincoördinaten voorstellen. Voorbeeld:

```
DRAW "BM50,50D30"
```

geeft vanaf punt (50,50) een lijn naar beneden over 30 punten.

Men kan een laatst bereikte punt ook als startpunt voor een nieuwe lijn handhaven en wel door vóór de code een N te plaatsen. Men kan de string ook door middel van een stringvariabele aangeven, bijvoorbeeld:

```
....
50 B$="BM50,50D30"
60 DRAW B$
```

Men kan een gedeelte van een string in een DRAW-instructie ook door middel van een variabele weergeven. In het gedeelte van de DRAW-instructie plaatsen we dan een X.
Voorbeeld:

```
....
50 B$="BM50,50D30"
60 DRAW "R100XB$"
```

Ten slotte kan men de getallen die in de DRAW-string voorkomen ook door middel van een variabele aangeven. De variabele komt dan tussen de tekens = en ; te staan.
Bijvoorbeeld:

```
...
50 K=30
60 DRAW "B10,10R=K;"
```

END
Beëindigt een programma. Deze instructie mag overal in het programma voorkomen. Een END aan het einde van een programma is optioneel, dat wil zeggen niet verplicht.

EOF (bestandsnaam)
Bepaalt of het einde van een bestand (file) is bereikt en wel tijdens het inlezen van gegevens. Bijvoorbeeld:

```
IF EOF(2) THEN CLOSE #2
```

ERASE naam array-variabele [,naam array-variabele]...
Met ERASE kunnen we een array wissen waardoor weer geheugenruimte vrijkomt (zie ook DIM).
Bijvoorbeeld:

```
 10 DIM K(100), X$(50)
    ...
500 ERASE K,X$
```

ERR- en ERL-variabelen
Als in een programma een fout optreedt, bevat de variabele ERR het foutnummer van de opgetreden fout (zie appendix G, Overzicht foutmeldingen) en bevat ERL het regelnummer waarin de fout is opgetreden.

ERR en ERL kunnen gebruikt worden in door de gebruiker geschreven fout-afhandelingsroutines (zie ook ON ERROR GO-TO), meestal in IF...THEN-constructies. Bijvoorbeeld: IF ERR=11 AND ERL=160 THEN ... etc.

ERROR<foutnummer>
Drukt de bij het opgegeven <foutnummer> behorende foutcode af op het scherm. Het <foutnummer> moet groter zijn dan 0 en kleiner dan 255. Appendix G geeft een overzicht van de foutnummers. Voorbeeld: typ in ERROR 11. Op het beeldscherm verschijnt /0 Error.

Niet alle foutnummers tussen 0 en 255 worden door BASIC gebruikt. De vrije nummers kan men gebruiken om eigen foutboodschappen te genereren.
Voorbeeld:

```
10 ON ERROR GOTO 400
20 INPUT "X:-",A
30 IF A>100 THEN ERROR 210
   ...
   ...
400 IF ERR=210 THEN PRINT "MAXIMAAL 100!"
410 RESUME 20
```

FOR...NEXT
De volledige syntax luidt:

FOR <variabele>=n TO m [STEP k]
instructie 1
instructie 2
...
...
NEXT [<variabele>]

waarbij n, m en k numerieke uitdrukkingen zijn.

Alle instructies tussen FOR en NEXT worden herhaald uitgevoerd. Net zo vaak totdat de teller (=<variabele>) de waarde

van m overschrijdt. De beginwaarde van de teller is n en de teller wordt, iedere keer als alle instructies zijn uitgevoerd, met k verhoogd. Als men geen k specificeert dan wordt de teller steeds met 1 verhoogd.

FOR...NEXT-instructies mogen op hun beurt andere FOR...NEXT-instructies omvatten.

GOSUB <regelnummer> ... RETURN

Deze instructie maakt het mogelijk om naar een subroutine te springen. <regelnummer> is het nummer van de eerste regel van de subroutine. Een RETURN moet ergens in de subroutine voorkomen en zorgt ervoor dat wordt teruggekeerd naar de eerstvolgende instructie na de GOSUB-instructie. Subroutines mogen op hun beurt weer naar andere subroutines verwijzen.

GOTO <regelnummer>

De programma-uitvoering wordt vervolgd bij het opgegeven <regelnummer>.

IF...THEN

De volledige syntax luidt:

IF <boolean expressie>[<boolean operator><boolean expressie>...]THEN <regelnummer of instructie(s)>[ELSE <regelnummer of instructie(s)>]

N.B. Met 'boolean expressie' wordt bedoeld 'logische voorwaarde' die al of niet juist is. In de <boolean expressies> mogen als relatieteken (operator) voorkomen:

<	:kleiner dan	<=	:kleiner dan gelijk aan
>	:groter dan	>	:groter dan gelijk aan
=	:gelijk aan	<>	:ongelijk aan

Plaats men na THEN of ELSE meer instructies dan moeten deze gescheiden zijn door een dubbelepunt:

De totale uitdrukking moet wel een regel beslaan.
In de uitdrukkingen mogen OR, AND of NOT worden gebruikt.

Als de hele expressie (uitdrukking) tussen IF en THEN waar is, worden de instructies na THEN uitgevoerd. Volgt na THEN geen instructie(s) maar een regelnummer dan wordt verder gegaan met de opgegeven regel.

Als de hele expressie tussen IF en THEN niet waar is, worden de instructies na ELSE uitgevoerd of wordt naar de opgegeven regel gesprongen. Is geen ELSE aanwezig dan wordt verder gegaan met de regel die direct volgt op de IF...THEN-instructie.

De instructies na THEN of ELSE mogen weer een IF...THEN-instructie bevatten, mits het geheel op één regel staat. Dus de instructie:

```
IF X>Y THEN PRINT "GROTER" ELSE IF Y>X THEN PRINT
"KLEINER" ELSE PRINT "GELIJK"
```

is toegestaan. Iedere ELSE in zo'n instructie wordt gekoppeld aan de dichtstbijzijnde IF waarvoor nog geen bijbehorende ELSE was gevonden. Dus de instructie:

```
IF X=Y THEN IF Y=Z THEN PRINT "X=Z" ELSE PRINT "X<>Z"
```

met X<>Y drukt *niet* X<>Z af.

INPUT ["<tekst>" <; of ,>]<variabele 1>[,<variabele 2>,...]
Leest waarden in die door de gebruiker moeten worden ingetoetst op het toetsenbord. Op het scherm verschijnt steeds een vraagteken (?).

Wordt <tekst> meegegeven, dan verschijnt deze tekst vóór het vraagteken op het scherm. Wordt de <tekst> gevolgd door een , in plaats van ; dan wordt het vraagteken achterwege gelaten.

De ingetoetste waarden worden toegekend aan de <variabelen>. Er moeten net zoveel waarden, gescheiden door een komma, worden ingetoetst als er variabelen voorkomen in de INPUT-instructie. Ook string-variabelen zijn toegestaan maar er mag geen komma voorkomen in de ingetoetste string. Komt er een fout voor in de ingetoetste waarden dan verschijnt de foutmelding ?Redo en wordt de INPUT-instructie opnieuw uitgevoerd.

INPUT$ (<n>)
INPUT$ (<n>,[#]<filenummer>)
Bij INPUT$ (<n>) wordt invoer van het toetsenbord verwacht en het aangegeven aantal tekens n wordt aan een string-variabele toegekend. Bijvoorbeeld:

```
50 A$=INPUT$ (3)
```

Bij de uitgebreide versie kan men een bestand opgeven van waaruit de tekens worden verwacht.

INPUT #<filenummer>,<variabele 1>[,<variabele 2>,...]
Deze instructie is vergelijkbaar met INPUT maar nu gaat het om de waarden in een file of die via de RS-232C-interface worden aangeboden. <filenummer> is het nummer dat aan de desbetreffende file of aan de RS-232C-interface is toegekend met OPEN. De waarden in de file moeten in dezelfde volgorde staan als de variabelen in de INPUT-instructie. Tijdens het inlezen van een string wordt het einde van de string bepaald door een komma, RETURN of line feed. Is het eerste teken van de string echter een aanhalingsteken ("), dan wordt een tweede aanhalingsteken als einde van de string aangenomen.

INTERVAL ON
INTERVAL OFF
INTERVAL STOP
Hiermee kan men aangeven of een onderbreking die door de ingebouwde klok wordt aangegeven; wordt ingeschakeld, uitgeschakeld of aangehouden.

KEY <n>,"<commandostring>"
Voor het toekennen van een <commandostring> aan een functietoets. <n> is het nummer van de functietoets (1 t/m 10). De <commandostring> moet tussen aanhalingstekens (") staan en mag uit maximaal 15 schrifttekens bestaan. Een RETURN kan men in de <commandostring> opnemen met de toevoeging +CHR$(13). Typ bijvoorbeeld in:

```
KEY 1,"RUN"+CHR$(13)
```

Wordt nu functietoets 1 gebruikt dan wordt het programma in de P-partitie waarin gewerkt wordt direct uitgevoerd, omdat het commando RUN is afgesloten met een RETURN. De RETURN wordt op het scherm weergegeven met ∧M.

Een aanhalingsteken (") kan aan de commandostring worden toegevoegd met +CHR$(34). Typ bijvoorbeeld het commando PRINT "HALLO" in:

```
KEY1,"PRINT"+CHR$(34)+"HALLO"+CHR$(34)+CHR$(13)
```

Na indrukken van functietoets 1 wordt het commando PRINT "HALLO" uitgevoerd en verschijnt het woordje HALLO op het scherm.

KEY LIST
Geeft een overzicht van de commandostrings van de 10 functietoetsen.

KEY ON
KEY OFF
Bepalen of de betekenis van de functietoetsen al of niet op het scherm worden weergegeven. Eventueel kan men één van de toetsen aangeven, bijvoorbeeld:

```
KEY (1) ON
```

[LET] <variabele>=<waarde>
Deze instructie kent een waarde aan een variabele toe. De term LET mag eventueel worden weggelaten.

LINE [[STEP](x1,y1)]-[STEP](x2,y2),[kleur],[BF of B]
Tekent een lijn tussen (X1,Y1) en (X2,Y2). Met STEP kunnen we aangeven dat het coördinatenstelsel naar het aangegeven punt

wordt verplaatst. Geeft men een diagonaal aan en eindigt men deze instructie met B dan wordt een vierkant met overeenkomstige diagonaal getekend. Met BF wordt het vierkant ingekleurd.

LINE INPUT ["<tekst>";]<string-variabele>
Leest een string in die wordt ingetoetst op het toetsenbord. De string mag uit alle mogelijke schrifttekens bestaan en een willekeurige lengte hebben tot maximaal 255 tekens.

LINE INPUT #<filenummer>,<string-variabele>
Deze instructie is vergelijkbaar met LINE INPUT, maar leest de string in van een file of via de RS-232C-interface. <filenummer> is het nummer dat aan de desbetreffende file of aan de RS-232C-interface is toegekend met OPEN. Een string wordt in z'n geheel ingelezen tot aan een RETURN.

LIST [<regelnummer>[-[<regelnummer>]]]
Dit commando geeft de gespecificeerde regels weer op het scherm. Geeft men geen regelnummer op, dan wordt het hele programma weergegeven.

Voorbeelden:

`LIST`	geeft het hele programma weer
`LIST 100`	geeft alleen regel 100 weer
`LIST 100-150`	geeft alle regels tussen 100 en 150 weer
`LIST -100`	geeft alle regels tussen 1 en 100 weer
`LIST 100-`	geeft alle regels vanaf 100 weer

LLIST [<regelnummer>[-[<regelnummer>]]]
Hetzelfde als LIST, alleen worden de regels nu afgedrukt op de printer.

LOCATE [X],[Y],[cursor aan/uit]
Verplaatst de cursor naar de aangegeven plaats. De cursor kunnen we respectievelijk aan- of uitzetten door een 1 of een 0 aan te geven.

NEW
Heeft tot gevolg dat het in het geheugen opgeslagen BASIC-programma wordt gewist

ON ERROR GOTO <regelnummer>
Tengevolge van deze instructie zal, als er een fout optreedt in het programma, de fout niet worden gemeld op het scherm, zoals gebruikelijk, maar zal het programma worden voortgezet op het gespecificeerde <regelnummer>. Daardoor heeft men

de mogelijkheid om eigen fout-afhandelingsroutines te schrijven. <regelnummer> is de eerste regel van de fout-afhandelingsroutine. In de routine kan men gebruik maken van de variabelen ERR en ERL.

Met RESUME keert men vanuit de fout-afhandelingsroutine terug naar het hoofdprogramma. De instructie kan weer ongedaan worden gemaakt met ON ERROR GOTO 0. Het verdient aanbeveling om in een fout-afhandelingsroutine de instructie ON ERROR GOTO 0 uit te voeren voor de afhandeling van onvoorziene fouten. Voorbeeld:

```
10 ON ERROR GOTO 400
20 OPEN "CAS:PROG1" FOR OUTPUT AS#1
   . . .
   . . .
400 IF ERR=53 THEN PRINT "OPENEN MISLUKT. PROBEER OP-
   NIEUW!":RESUME 20

410 ON ERROR GOTO Ø
```

ON <integer-expressie>GOTO<regelnummer 1>[,<regelnummer 2>,...]
Maakt het mogelijk om de programma-uitvoering voort te zetten op een regelnummer dat wordt bepaald door de waarde van de <integer-expressie>. Levert de <integer-expressie> bijvoorbeeld de waarde 3 op, dan zal het programma worden voortgezet op het derde regelnummer dat na GOTO is gespecificeerd. De waarde van de <integer-expressie> mag niet negatief, nul of groter zijn dan het aantal opgegeven regelnummers.

ON <integer-expressie>GOSUB<regelnummer 1>[,<regelnummer 2,...>]
Werkt op dezelfde manier als de instructie ON...GOTO, maar nu moeten de opgegeven regelnummers de eerste regel zijn van een subroutine.

ON INTERVAL=<tijd>GOSUB<regelnummer>
Hiermee wordt aangegeven dat na een door INTERVAL aangegeven wachttijd, het programma naar de aangegeven subroutine moet springen.
Voorbeeld:

```
. . .
ON INTERVAL=100 GOSUB 300
. . .
INTERVAL ON
. . .
```

Het getal na het =–teken geeft de duur van de onderbreking in eenheden van 1/50 seconden aan.

ON KEY GOSUB <regelnummer 1>[,<regelnummer 2>,...]
Geeft aan naar welke subroutine gesprongen moet worden als op één van de toetsen F1 t/m F5 wordt gedrukt.

ON SPRITE GOSUB <regelnummer 1>,[,<regelnummer 2>,...]
Geeft aan naar welke subroutine gesprongen moet worden als twee sprites elkaar overlappen. Van te voren dient dan wel de SPRITE ON-instructie gegeven te worden.
Voorbeeld:

```
...
SPRITE ON
ON SPRITE GOSUB 300
...
```

ON STOP GOSUB <regelnummer>
Geeft aan naar welke subroutine gesprongen moet worden als het programma met CTRL/STOP onderbroken wordt.

```
    ...
STOP ON
ON STOP GOSUB 300
...
300 PRINT "CTRL/STOP INGEDRUKT!"
310 RETURN
```

ON STRIG GOSUB <regelnummer 1>[,<regelnummer 2>,...]
Geeft aan naar welke subroutine gesprongen moet worden. Voor regelnummer 1, 2, 3, 4 en 5 wordt gekeken naar:

nr. vuurknop of spatiebalk
1 spatiebalk
2 joystick 1, vuurknop 1
3 joystick 2, vuurknop 1
4 joystick 1, vuurknop 2
5 joystick 2, vuurknop 2

De desbetreffende vuurknop/spatiebalk moet wel geactiveerd worden. Voorbeeld:

```
... ...
100 ON STRIG GOSUB 300,400,500
110 STRIG(0) ON: REM SPATIEBALK
120 STRIG(1) ON: REM JOYSTICK 1
130 STRIG(2) ON: REM JOYSTICK 2
... ...
```

OPEN "<devicenaam>:[<filenaam>]" FOR <verwerking> AS#[<nummer>]
Hiermee wordt een file (bestand) geopend.
Voor <devicenaam> kan het volgende worden ingevuld:

CAS: cassetterecorder
CRT: tekst-beeldscherm
GRP: grafisch-beeldscherm
LPT: printer

Voor <filenaam> mag een naam bestaande uit maximaal 6 characters worden genomen. Als men voor <verwerking> OUTPUT neemt, gaat het om een bestand waar gegevens op worden geplaatst. Neemt men INPUT dan gaat het om een bestand van waaruit gegevens worden ingelezen.
Voorbeeld:

```
OPEN "CAS:PROG1" FOR OUTPUT AS#1
```

OUT <poortnummer, uitdrukking>
Het door <uitdrukking> aangegeven gegeven wordt naar de door <poortnummer> aangegeven poort gestuurd.

```
OUT &H80,3
```

LOAD <devicenaam>[filenaam]
Voor het laden van een programma dat met behulp van een SAVE-commando op cassette was opgeslagen.
Voorbeeld:

```
LOAD "CAS:PROG1"
```

wil zeggen; Lees van cassette het programma dat onder de naam PROG1 is opgeslagen

LOCATE [x-coördinaat],[y-coördinaat],[cursor aan/uit]
Bepaalt positie van de cursor op het beeldscherm. De grootheid x mag variëren tussen 0 en 39 en y tussen 0 en 24. Voorbeeld: LOCATE 15,13

Door voor "code aan/uit" 0 in te vullen, geeft men aan dat de cursor niet mag worden aangegeven. Met code 1 wordt de cursor weer aangegeven (bijv. LOCATE 15,13,1).

LPRINT
Als PRINT, maar nu wordt de printer aangestuurd.

LPRINT USING
Als PRINT USING, maar nu wordt de printer aangestuurd.

MERGE "<devicenaam:filenaam>"
Voegt een op een extern geheugen opgeslagen programma toe

aan de in het geheugen aanwezige programma's. Het programma moet in ASCII-formaat zijn opgeslagen.

De programmaregels worden in oplopende volgorde van het regelnummer geplaatst. Hebben twee regels hetzelfde regelnummer dan wordt de regel in het geheugen vervangen door de regel van het ingelezen programma. Het nieuw ontstane programma blijft in het geheugen staan.
Voorbeeld:

```
MERGE "CAS:PROG1"
```

MOTOR [ON/OFF]
Hiermee kan men een taperecorder op afstand besturen. MOTOR ON heeft het effect alsof men bij de recorder op de toets "play" drukt en MOTOR OFF schakelt de motor weer uit.

PAD (<getal>)
Leest de stand van het aanraakpaneel uit.
Voor <getal> gelden de volgende afspraken.

getal	*betekenis*
0 of 4	0: paneel wordt niet aangeraakt –1: paneel wordt aangeraakt
1 of 5	x-coördinaat
2 of 6	y-coördinaat
3 of 7	0: schakelaar wordt niet ingedrukt –1: schakelaar wordt ingedrukt

PAINT [STEP](<x,y>),[<kleurvlak>],[<kleurlijn>]
Geeft aan dat een bepaald gebied moet worden ingekleurd. Met STEP wordt coördinatenstelsel verplaatst. Met x en y geven we een punt aan. Het vlak waarbinnen dit punt valt, wordt ingekleurd. Als de randlijn niet volledig is ononderbroken, wordt het gehele scherm ingekleurd.

Bij het grafisch scherm met hoge resolutie (SCREEN 2) geldt dat als de kleur van het vlak en de kleur van de reeds gegeven randlijn niet gelijk zijn, het hele scherm wordt ingekleurd.
Voorbeeld:

```
10 CLS
20 SCREEN
30 CIRCLE (100,100),50,10,,,1.4
40 PAINT (100,100),10
50 GOTO 50
```

Geeft een donkergeel ingekleurde cirkel te zien.

PLAY "string"
Instructie om geluid volgens de door "string" bepaalde aanwijzingen voort te brengen. PLAY "CDE" laat bijvoorbeeld de tonen C, D en E horen.

Vóór de aanduiding van een noot (C, D, E, F, G, A, B en evenzo C+ voor CIS, etc.) kan men letters en getallen plaatsen die een speciale betekenis hebben:

Code *betekenis*

On Geeft betreffende octaaf aan, bijvoorbeeld PLAY "O5CDE": wil zeggen speel C, D en E van octaaf 5. Voor n geldt; 1 ≤n≤8. Zonder nadere aanduiding neemt de computer O4 aan.

Ln Geeft de duur van een toon aan en wel volgens

duur	code
4 tellen	L1
2 tellen	L2
1 tel	L4
1/2 tel	L8
1/4 tel	L16
1/8 tel	L32
1/16 tel	L64

Zonder nadere aanduiding neemt de computer L4 aan.

Nn Met n tussen 0 en 96: geeft een toon met nummer n aan.

A t/m 9 Geeft toon aan. Geeft men bijvoorbeeld A5 aan dan komt dit overeen met 05A, kortom de A bij de vijfde octaaf.

Rn Geeft rustmaat weer en wel in stappen van 2. R4 komt op een hele tel rust neer en R1 op vier tellen rust. De opeenvolgende mogelijkheden zijn dan: R1, R2, R4, R8, R16, R32 en R64.

Vn Geeft het volume aan: n=15 komt met 'maximaal volume' overeen. Zonder nadere aanduiding neemt de computer V8 aan.

• verlengt de duur van een noot.

Sn Geeft de klankkleur aan (0≤n≤15). Zonder nadere aanduiding neemt de computer S1 aan.

Mn Geeft de mate aan waarin de klankkleur varieert. Zonder nadere aanduiding neemt de computer M255 aan.

POKE <adres, gegeven>
Plaatst gegeven in aangegeven geheugenregister. Voorbeeld:

```
POKE 5000,100
```

PRESET [STEP] (<x,y>) [,<kleur>]
Zet of wist een stip op (x,y) van het grafische scherm. Als een kleur is opgegeven, is het effect van PRESET gelijk aan dat van PSET. Zonder deze aanduiding wordt een reeds aangegeven punt gewist. Met STEP kan men eventueel het coördinatenstelsel verplaatsen.

Appendix E

Overzicht van de BASIC-functies

Deze appendix geeft de volledige syntax van alle BASIC-functies. Daarbij worden twee notaties gebruikt met de volgende betekenis:

<X> een numerieke variabele of uitdrukking die de gebruiker zelf moet invullen;

<X$> een string-variabele of uitdrukking die de gebruiker zelf moet invullen.

ABS(<X>)
Geeft de absolute waarde van <X>.

ASC(<X$>)
Geeft de ASCII-code van het eerste schriftteken van <X$>.

ATN(<X>)
Geeft de arctangens van <X> in radialen en in enkele precisie. Het resultaat ligt tussen $-\pi/2$ en $\pi/2$.

CDBL(<X>)
Zet <X> over naar een getal met dubbele precisie.

CHR$(<ASCII-code>)
Geeft het schriftteken dat behoort bij de <ASCII-code>.

CINT(<X>)
Rondt de waarde van <X> af op een geheel getal (een integer). De waarde moet liggen tussen –32768 en 32767.

COS(<X(in radialen)>)
Geeft de cosinus van <X>.

CSGN(<X>)
Zet de waarde van <X> om naar een getal met enkele precisie.

CSRLIN
Geeft de y-coördinaat van de positie waarop de cursor zich bevindt.

EOF (<filenummer>)
Geeft aan of het einde van de aangegeven file bereikt is.

ERL
Zie Appendix D.

ERR
Zie Appendix D.

PRINT [<uitdrukking>[, of ;][<uitdrukking><, of ;>...]]
Geeft op het scherm de waarde weer van variabelen, numerieke uitdrukkingen of strings. Strings moeten tussen aanhalingstekens (") staan.

De positie op het scherm waar de gegevens worden geplaatst, wordt bepaald door het scheidingsteken. Is dit scheidingsteken een puntkomma (;) of een spatie, dan worden de gegevens direct achter elkaar geplaatst. BASIC verdeelt een regel in zones van 14 posities breed. Is het scheidingsteken tussen twee gegevens een komma (,) dan wordt het tweede gegeven in de volgende zone geplaatst.

Sluit men de PRINT-instructie af met een komma of puntkomma, dan worden de gegevens bij de volgende PRINT-instructie op dezelfde regel geplaatst, anders op de volgende regel.

PRINT USING "<notatiecode>";<uitdrukking>[;<uitdrukking>...]
Deze instructie is een uitbreiding op de PRINT-instructie, waarbij men met behulp van <notatiecode> zelf kan opgeven in welke vorm men de gegevens wil weergeven. De <uitdrukkingen> moeten worden gescheiden door een puntkomma (;).

Bij het weergeven van strings kan men kiezen uit één van de volgende drie <notatiecodes>:

"!" Alleen het eerste schriftteken van de string wordt weergegeven.

"\n spaties\" Nu worden de eerste 2+n schrifttekens van de string weergegeven.

"&" De string wordt volledig weergegeven.

Voor het weergeven van numerieke waarden heeft men de keuze uit de volgende speciale schrifttekens:

Geeft het aantal posities voor en na de komma aan. Te grote

getallen worden afgerond. Te kleine getallen worden door middel van nullen en spaties passend gemaakt. Is het getal te groot voor de gespecificeerde <notatiecode>, dan wordt het procentteken (%) voorafgaand aan het getal weergegeven.
Voorbeelden:

```
PRINT USING "##.##";1.2345
1.23
PRINT USING "##.##";99.996
%100.00
```

+ Een plusteken vóór of aan het eind van de <notatiecode> zorgt er voor dat een plus- of minteken vóór of aan het eind van het getal wordt weergegeven.

− Een minteken aan het eind van de <notatiecode> zorgt er voor dat bij negatieve getallen een minteken aan het eind wordt weergegeven.

** Een dubbele asterisk aan het begin van de <notatiecode> zorgt er voor dat eventuele aan het getal voorafgaande spaties met asterisk-tekens worden aangegeven. Bovendien tellen deze twee asterisk-tekens voor twee extra posities.

££ Een dubbel £-teken aan het begin van de <notatiecode> zorgt er voor dat een £-teken voorafgaand aan het getal wordt weergegeven. De wetenschappelijke notatie (zie verderop) kan niet worden gebruikt in combinatie met ££.

****£** Geeft een combinatie van de twee voorgaande effecten.

, Een komma aan de linkerkant van de decimale punt zorgt er voor dat een komma wordt weergegeven om de drie posities.
Voorbeeld:

```
PRINT USING "####,.##";1234.5
1,234.50
```

Een komma aan het einde van de <notatiecode> wordt gewoon weergegeven.
Voorbeeld:

```
PRINT USING "####.##,";1234.5
1234.50,
```

^^^^ Door vier pijlen aan het eind van de <notatiecode> te plaatsen, wordt het getal in wetenschappelijke notatie weergegeven.
Voorbeelden:

```
PRINT USING "##.##^^^^";123.45
1.23E+0.1
```

tekst Tekst voor of achter de <notatiecode>, mits van de <notatiecode> gescheiden door een spatie, wordt gewoon weergegeven.
Voorbeeld:

```
PRINT USING "FL ##.## GULDEN";12.34
FL 12.34 GULDEN
```

PRINT # en PRINT # USING
De volledige syntax luidt:

```
PRINT        #<filenummer>,[USING"<notatiecode>";
]<uitdrukking>[;<uitdrukking>...]
```

Hiermee worden gegevens naar file geschreven. <filenummer> is het nummer dat aan de file is toegekend met OPEN. Voor <notatiecode> gelden dezelfde mogelijkheden als beschreven bij de PRINT USING-instructie. De <uitdrukkingen> worden achter elkaar naar file geschreven en men dient als scheidingsteken alleen de puntkomma (;) te gebruiken. Gebruikt men komma's dan worden ook de spaties, die worden toegevoegd voor het weergeven op het scherm, naar file geschreven.

Ook strings worden achter elkaar geplaatst in de file en zijn daardoor niet meer als afzonderlijke strings herkenbaar. Stel bijvoorbeeld dat A$=JAN en B$=KLAAS, dan schrijft de instructie:

```
PRINT #1,A$;B$
```

Als resultaat JANKLAAS naar file. Dit kan alleen maar als één string worden teruggelezen. Men kan dit probleem voorkomen door zelf scheidingstekens tussen te voegen. Bijvoorbeeld:

```
PRINT #1,A$;",";B$
```

heeft als resultaat JAN,KLAAS.

Een andere mogelijkheid is om de strings in twee instructies naar file te schrijven:

```
PRINT #1,A$
PRINT #1,B$
```

Door de PRINT-instructie niet met puntkomma af te sluiten, wordt automatisch een RETURN toegevoegd aan het eind van de string. De RETURN fungeert nu als scheidingsteken.

Wanneer de string zelf een komma bevat, wordt deze bij het teruglezen als twee strings beschouwd. Bijvoorbeeld A$=JAN,KLAAS en de instructie

```
PRINT #1,A$
```

levert in de file JAN,KLAAS op en zal na inlezen met:

```
INPUT #1,A$,B$
```

als resultaat opleveren dat A$=JAN en B$=KLAAS. Dit probleem kan men vermijden door de string tussen aanhalingstekens (") in de file te plaatsen (zie ook de INPUT-instructie). De instructie wordt nu:

```
PRINT #1,CHR$(34);A$;CHR$(34).
```

34 is de ASCII-code voor aanhalingstekens (").

PSET [STEP](<x,y>)],<kleur>]
Zie PRESET; nu gaat het om het weergeven van een stip.
Voorbeeld:

```
10 CLS
20 SCREEN 2
30 PSET (100,50)
40 GOTO 40
```

PUT SPRITE <vlaknummer>,[[STEP](<x,y>)],[<kleur>], [<sprite-nummer>]
Plaatst sprite met aangegeven kleur en nummer op het door <vlaknummer> gegeven vlak en wel op positie (x,y).
Voorbeeld:

```
PUT SPRITE 0, (100,50),1,3
```

Met STEP kan men eventueel het coördinatenstelsel verplaatsen. Voor een 8×8-sprite kan men als <sprite-nummer> een getal tussen 0 en 255 nemen. Voor een 16×16-sprite een getal tussen en 0 en 63.

READ <variabele 1>[,<variabele 2>...]
READ wordt altijd gebruikt in combinatie met DATA en kent de door DATA gespecificeerde waarden toe aan de opgegeven <variabelen>. Zie verder DATA.

REM <commentaar>
Hiermee is het mogelijk commentaar aan een programma toe te voegen. Alle informatie na REM wordt door BASIC genegeerd.

RENUM [[<nieuw regelnummer>][,[<oud regelnummer>] [,<stapgrootte>]]]
Dient voor het opnieuw nummeren van de regels. <nieuw regelnummer> is het beginnummer van de eerste regel. Specificeert men geen <nieuw regelnummer> dan wordt met 10 begonnen.

<oud regelnummer> geeft aan vanaf welke regel het hernummeren moet beginnen. Geeft men geen <oud regelnummer> op dan wordt met de eerste regel begonnen.

<stapgrootte> is het getal waarmee het regelnummer steeds moet worden verhoogd. Standaard is dat 10.

RESTORE [<regelnummer>]
Hiermee kunnen lijsten met waarden die door DATA zijn gespecificeerd weer van het begin af aan met READ worden gelezen. Zie verder DATA. Eventueel kan men met <regelnummer> naar een betreffende regel met DATA springen.

RESUME <regelnummer>
RESUME wordt alleen gebruikt in fout-afhandelingsroutines en heeft dezelfde functie als RETURN in subroutines. <regelnummer> geeft de regel aan waar de uitvoering van het programma moet worden hervat.

RUN [<regelnummer>]
Start de uitvoering van een programma vanaf <regelnummer>. Wordt geen <regelnummer> opgegeven dan start de uitvoering op de eerste regel.

SAVE "<devicenaam>[<filenaam>]"
Voor opslag van een bestand (file).

<devicenaam>	betekenis
CAS	cassette (in ASCII-codes)
CRT	tekst-beeldscherm
GRP	grafisch-beeldscherm
LPT	printer

Voorbeeld:

```
SAVE "CAS:PROG1"
```

wil zeggen: laadt het BASIC-programma onder de naam PROG1 op cassette.

SCREEN [<modus>],[<formaat sprites>],[<i>],[<s>],[<a>]
Met de SCREEN-instructie wordt de computer aangegeven hoe het scherm gebruikt moet worden. Gewoonlijk gebruikt men alleen SCREEN 2 om aan te geven dat het grafische scherm met hoge resolutie gebruikt moet worden. Zonder nadere aanduiding neemt de computer de instructie SCREEN 0,0,1,1,0 aan.

De volgende afspraken gelden:

<modus>	betekenis
0	tekst scherm: 40×24 (regels)
1	tekst scherm: 32×24 (regels)
2	grafisch scherm (hoge resolutie)
3	grafisch scherm (lagere resolutie)

<formaat sprite>	
0	8×8, zonder vergroting
1	8×8, met vergroting
2	16×16, zonder vergroting
3	16×16, met vergroting

<i>	(wel of geen geluid bij indrukken toets)
0	geeft geen piep-signaal bij indrukken toets
ander getal	geeft piep-signaal bij indrukken toets

<s> (snelheid bij in/uitvoer van cassette)
1 1200 band
2 2400 band

<a> (soort afdrukeenheid)
0 MSX-type
1 anders

SOUND <registernummer, getal>
Instructie om bepaald geluid te genereren. De volgende afspraken gelden:

register	*bereik getal*	*betekenis*
0	0–255	frequentie kanaal A
1	0–15	frequentie kanaal A
2	0–255	frequentie kanaal B
3	0–15	frequentie kanaal B
4	0–255	frequentie kanaal C
5	0–15	frequentie kanaal C
6	0–31	frequentie ruis
7	0–63	keuze kanaal: toon en ruis
8	0–15	volume kanaal A
9	0–15	volume kanaal B
10	0–15	volume kanaal C
11	0–255	frequentie van patroon-variatie
12	0–255	frequentie van patroon-variatie
13	0–14	keuze patroon

SPRITE ON/OFF/STOP
Met ON, OFF en STOP wordt aangegeven of computer attent is op het optreden van 'botsende' sprites. Zie ON SPRITE GOSUB. Met SPRITE STOP wordt aangegeven dat de 'onderbrekings-toestand' moet worden aangehouden.

STOP
Onderbreekt het uitvoeren van een programma.

STOP ON/OFF/STOP
Regelt de modus bij een door CTRL/STOP veroorzaakte onderbreking (zie ON STOP GOSUB). Bij STOP STOP wordt de onderbrekings-toestand aangehouden.

STRIG ON/OFF/STOP
Regelt de modus bij een door de joystick of spatiebalk veroorzaakte onderbreking (zie ON STRIG GOSUB). Bij STRIG STOP wordt onderbrekingstoestand aangehouden.

SWAP <variabele>,<variabele>
Wisselt de waarden van twee variabelen. Voorbeeld:

```
SWAP A,B
```

TROFF en TRON
Met behulp van het commando TRON schakelt men de computer in een toestand waarbij tijdens het uitvoeren van het programma eveneens het regelnummer wordt afgebeeld van de regel waarmee de computer bezig is.

Dit vereenvoudigt het opsporen van fouten. Met TROFF schakelt men deze toestand weer uit.

USR [nummer](<waarde>)
Voor het verwerken van een in machinecode opgesteld programmagedeelte.

VPOKE <adres, gegeven>
Als POKE maar nu heeft <adres> betrekking op een adres van het video-gedeelte van het RAM-geheugen.

WAIT <poortnummer, uitdrukking 1>,[<uitdrukking 2>]
Heeft betrekking op het inlezen van gegevens via I/O-poort met aangegeven poortnummer.

De ingelezen waarde wordt met <uitdrukking 2> via de zgn. exclusive OR-voorwaarde gecombineerd. Deze uitkomst wordt via een AND-operatie met <uitdrukking 1> gecombineerd. Als het resultaat hiervan 0 oplevert wordt het inlezen voortgezet en zo niet dan vervolgt de computer het programma met de volgende instructie. Als <uitdrukking 2> wordt weggelaten wordt de waarde 0 aangenomen.

WIDTH (<aantal kolommen>)
Stelt het aantal kolommen in van het tekst-scherm. Zonder nadere aanduiding wordt het scherm in 40 kolommen ingedeeld. Voorbeeld:

```
WIDTH 20
```

Wil zeggen: per regel kunnen maximaal 20 karacters worden afgebeeld.

EXP(<X>)
Berekent de e-macht van <X>.

FIX(<X>)
Geeft het gehele deel van <X>.

FRE(0) of FRE(<X$>)
Is het argument een 0 dan geeft FRE de grootte van de nog niet

gebruikte geheugenruimte in aantal bytes. Is het argument een string-variabele, dan geeft **FRE** het aantal vrije bytes in de geheugenruimte die gereserveerd is voor string-variabelen.

HEX$(<X>)
Geeft een string die de hexadecimale waarde voorstelt van <X>. <X> wordt eerst afgerond naar een geheel getal.

INKEY$
Deze functie wordt in de vorm <string-variabele>=INKEY$ gebruikt. De functie controleert of er een toets is ingedrukt op het toetsenbord. Zo ja, dan wordt het betreffende schriftteken aan <string-variabele> toegekend. Zo niet, dan bevat <string-variabele> een lege string (zogenaamde null-string). Voorbeeld:

```
10 PRINT "TYP EEN H: ";
20 A$=INKEY$
30 IF A$<>"H" THEN 20
40 PRINT A$
50 END
```

INP (<poortnummers>)
Geeft het poortnummer aan voor invoer en uitvoer van gegevens
Onderstaande tabel geeft enkele voorbeelden:

soort	*nummer*	*toepassing*
RS232–C:	&H80	in- en uitvoer van gegevens
	&H81	modus (bij inlezen) en status (bij uitlezen)
printer	&H90	signaalpoorten (inlezen) status (uitlezen)
	&H91	inlezen van gegevens

INPUT$(<X>)
Deze functie wordt in de vorm <string-variabele>=INPUT$ (<X>) gebruikt. De functie leest een string in van het toetsenbord van <X> schrifttekens. Alle schrifttekens worden geaccepteerd behalve CTRL/C.

INSTR([I,]<X$>,<Y$>)
Controleert of <Y$> voorkomt in <X$> en geeft dan de positie in <X$>. Komt <Y$> niet voor in <X$> dan geeft INSTR de waarde **0**. INSTR zoekt vanaf het begin van <X$> tenzij men met [I,] een andere startpositie opgeeft. Is <Y$> een lege string dan geeft INSTR de waarde 1 of I.

INT(<X>)
Deze functie geeft het grootste gehele getal dat kleiner of gelijk is aan <X>.

LEFT$(<X$>,<I>)
Geeft een string die bestaat uit de eerste <I> schrifttekens van <X$>. <I> moet liggen tussen 0 en 255.

LEN(<X$>)
Geeft het aantal schrifttekens waaruit <X$> bestaat. Ook spaties worden geteld.

LOG(<X>)
Geeft de natuurlijke logarithme van <X>. <X> moet groter zijn dan 0.

LPOS(<X>)
X kan hier een willekeurig getal zijn. LPOS geeft de positie van de printer in de printbuffer aan

MID$(<X$>,<I>[<J>])
Geeft een string die een deel is van <X$>. De string begint op positie <I> en is <J> schrifttekens lang. Wordt <J> niet opgegeven dan worden alle schrifttekens vanaf <I> genomen.

OCT$(<X>)
Geeft een string die de octale waarde voorstelt van <X>. <X> wordt eerst afgerond tot een geheel getal.

PAD(<X>)
Geeft informatie over het aanraakpaneel en wel volgens onderstaande tabel:

waarde X	*betekenis van functiewaarde*
0 of 4	bepaalt of paneel is aangeraakt; 0 (niet aanraken) of 1 (aanraken)
1 of 5	X-coördinaat van aangeraakte plaats
2 of 6	Y-coördinaat van aangeraakte plaats
3 of 7	bepaalt of schakelaar is ingedrukt (0=niet en –1=wel)

PDL(<X>)
Geeft de stand van de paddle (1 t/m 12). Waarde varieert van 0 tot 255.

PEEK(<X>)
Geeft de inhoud van geheugenadres X. In de vorm van een decimaal getal moet X tussen 0 en 65536 liggen.

POINT(<X>,<Y>)
<X>: positie op de regel (0 – 255).
<Y>: regel (0 – 191).

Geeft een getal dat de kleur aangeeft van het punt op het scherm met coördinaten (<X>,<Y>). De mogelijke getallen zijn:

0 wit (dus geen zichtbaar punt)
1 zwart
2 en 3 kleuren bij gebruik van een kleuren-TV

POS(X)
Geeft de positie van de cursor op de regel. De meest linkse positie is positie 0. X is slechts een schijnargument en heeft verder geen betekenis.

RIGHT$(<X$>,<I>)
Geeft een string die bestaat uit de laatste <I> schrifttekens van <X$>.

RND(<X>)
Geeft een random-getal tussen 0 en 1. Welke serie random-getallen wordt gegenereerd hangt af van het begingetal.

<X> = negatief er wordt vooraan begonnen van de serie random-getallen;
<X> = positief geeft het volgende random-getal in de serie;
<X> = 0 geeft het laatste random-getal nogmaals.

SGN(<X>)
Bij <X> = positief wordt SGN(<X>) = 1
Bij <X> = 0 wordt SGN(<X>) = 0
Bij <X> = negatief wordt SGN(<X>) = –1

SIN(<X(in radialen)>)
Geeft de sinus van <X>.

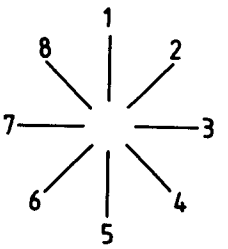
SPACE$(<X>)
Geeft een string bestaande uit <X> spaties. <X> wordt op een geheel getal afgerond en moet tussen 0 en 255 liggen.

SPC(<X>)
Geeft <X> spaties weer op het scherm. SPC mag alleen worden gebruikt binnen een PRINT- of LPRINT-instructie. <X> moet liggen tussen 0 en 255. Voorbeeld:

```
PRINT "VOOR";SPC(10);"BEELD"
VOOR          BEELD
```

SQR(<X>)
Geeft de wortel uit <X>. <X> moet groter dan of gelijk zijn aan nul.

STICK(<X>)
Geeft de stand van de joystick (1 of 2). Vult men voor X de waarde 0 in, dan wordt gekeken naar de cursor-toetsen.
De waarden corresponderen met de volgende richtingen



STR$(<X>)
Geeft een string die de decimale waarde van <X> voorstelt

STRIG(<X>)
Geeft aan of op vuurknop van joystick wordt gedrukt (−1=wel en 0=niet)

<X>	*betekenis*
0	toetsenbord: spatiebalk
1	joystick 1
2	joystick 2

STRING$(<I>,<J>) of STRING(<I>,(<X$>)
Geeft een string bestaande uit <I> dezelfde schrifttekens die als ASCII-code <J> hebben òf overeenkomen met het eerste schriftteken van <X$>.

TAB(<X>)
Plaatst de cursor op positie <X> van de regel. Als de cursor reeds voorbij de positie <X> staat, dan gebeurt er niets.
<X> moet liggen tussen 0 en 255. 0 is de meest linkse positie.
TAB kan alleen worden gebruikt in een PRINT- of LPRINT-instructie.
Voorbeeld:

```
PRINT "VOOR";TAB(10);"BEELD"
VOOR      BEELD
```

TAN(<X(in radialen)>)
Geeft de tangens van <X>.

USR[<n>](<X>)
Hiermee wijst men een machinetaal-programma aan. <n> is het nummer dat aan het machinetaal-programma is toegekend door DEF USR. Wordt <n> weggelaten dan wordt USR0 verondersteld.
<X> is een argument dat wordt doorgegeven aan het machinetaalprogramma.

VAL(<X$>)
Geeft de numerieke waarde aan van <X$>. Als het eerste schriftteken van <X$> geen +, −, & of cijfer is, dan wordt VAL=0.

VARPTR(<variabele>)
Geeft het adres van het eerste byte van de data die hoort bij de <variabele>. Aan de <variabele> moet een waarde zijn toegekend voordat de instructie VARPTR wordt uitgevoerd.

VPEEK(<X>)
Hier geeft <X> een adres van het video-RAM-geheugen aan. Met deze functie kan dan de inhoud van deze plaats bepaald worden.

Appendix F

Speciale tekens en uitdrukkingen

Bewerkingstekens

Onderstaande tabel geeft een overzicht van de tekens die voor wiskundige bewerkingen gebruikt mogen worden. Het laatste cijfer geeft de prioriteitswaarde aan: uitdrukkingen worden volgens deze prioriteitswaarden uitgewerkt (van het laagste tot het hoogste cijfer). Bij gelijke prioriteitswaarde wordt de uitdrukking van links naar rechts uitgewerkt

teken	*betekenis*	*voorbeeld*	*prioriteit*
+	optellen	3+5	6
–	aftrekken	5–3	6
*	vermenigvuldigen	5×3	3
/	delen	5/3	3
^	machtsverheffen	2^3	1
–	verandert teken	–3	4
MOD	rest bij integerdeling	12 MOD 3	5

Tekens in logische uitdrukkingen

De volgende symbolen mogen in uitdrukkingen voorkomen waarbij wordt nagegaan of ze al of niet waar zijn (bijv. na de term IF)

teken	*betekenis*	*voorbeeld*
=	is gelijk aan	IF A= B THEN...
<	kleiner dan	IF A<B THEN...
>	groter dan	IF A>B THEN...
<> of ><	ongelijk aan	IF A<>B THEN...
<= of =>	kleiner dan of gelijk aan	IF A<=B THEN...
>= of =>	groter dan of gelijk aan	IF A>=B THEN...

Termen om logische uitdrukkingen te combineren

Uitdrukkingen zoals A=B en C<>D kan men combineren met specifieke termen.

De volgende tabel geeft de verschillende termen aan met de zgn. waarheidstabellen. Hierin geeft 1 aan dat de uitspraak 'waar' is (klopt) en 0 dat de uitspraak 'onwaar' is (klopt niet, fout). Zo ziet men bijvoorbeeld bij de term AND

U1 AND U2

is 1 als U1 en U2 gelijk aan 1 zijn. Met U1 en U2 worden dan uitspraken als A=B en C<>D bedoeld. De tabel geeft in dit geval aan dat de volledige uitspraak waar is (1) als U1 en U2 ook waar zijn (1).

term	*waarheidstabel*		
NOT	U1	NOT U1	.
(ontkenning)	1	0	
	0	1	
AND	U1	U2	U1 and U2
(logisch produkt)	1	1	1
	1	0	0
	0	1	0
	0	0	0
OR	U1	U2	U1 OR U2
(logische som)	1	1	1
	1	0	1
	0	1	1
	0	0	0
XOR	U1	U2	U1 XOR U2
(exclusieve OR)	1	1	0
	1	0	1
	0	1	1
	0	0	0
EQV	U1	U2	U1 EQV U2
	1	1	1
	1	0	0
	0	1	0
	0	0	1
IMP	U1	U2	U1 IMP U2
(logische implicatie)	1	1	1
	1	0	0
	0	1	1
	0	0	1

Bewerkingstekens bij strings

Bij strings mag alleen het teken + gebruikt worden en wel om twee strings aan elkaar te koppelen. Voorbeeld:
"AB"+"CD" wordt "ABCD"

Appendix G Foutmeldingen

Bad file name (56)
De naam van het bestand (file) is onjuist. De naam voor het 'device' in een OPEN-, SAVE- of LOAD-commando is onjuist.

Bad file number (52)
Nummer van een file is groter dan toegestaan dan wel men geeft een file aan die nog niet 'geopend' is.

Can't CONTINUE (17)
Men probeert een programma 'te runnen' dat of niet bestaat of niet uitgevoerd kan worden omdat er een fout is opgetreden.

Device I/O error (19)
Het laden van een programma of bestand gaat fout. Is de cassetterecorder wel goed ingesteld? Is de juiste I/O-poort aangegeven?

Direct statement in file (57)
In een programma dat geladen wordt komt een instructie zonder regelnummer voor, dan wel; het bestand dat geladen wordt, is geen programma.

Division bij zero (11)
Delen door 0 is niet geoorloofd. Deze fout kan ook optreden als men deelt door een variabele waaraan van te voren nog geen waarde is toegekend.

File already open (54)
Men kan geen bestand openen dat al geopend is.

File not OPEN (59)
Er wordt naar een bestand verwezen dat nog niet door een OPEN-instructie is geopend.

Illegal direct (12)
Er wordt getracht een instructie als commando uit te voeren terwijl dat bij deze instructie niet is toegestaan.

Illegal function call (5)
Er wordt een BASIC-functie aangeroepen met een verkeerd argument. Deze melding treedt op bij:

1 een negatieve array-index of indices die buiten het array-bereik vallen;
2 een negatief argument bij LOG of SQR;
3 een negatieve mantisse met een real exponent;
4 het gebruik van de USR-functie zonder dat een startadres is gespecificeerd;
5 een verkeerd argument bij MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, ON...GOTO.

Input past end (55)
Er wordt getracht een gegeven in te lezen terwijl alle gegevens reeds zijn ingelezen. Kan ook optreden als het bestand leeg is, dat wil zeggen in het geheel geen gegevens bevat

Internal error (51)
De BASIC-interpreter zelf vertoont een fout, bijvoorbeeld door beschadiging van het ROM-geheugen.

Line buffer overflow (25)
De buffer voor de invoer van één regel is vol.

Missing operand (24)
Er is iets mis met één van de gegevens die in een commando moeten worden gebruikt.

NEXT without FOR (1)
Er is een NEXT-instructie gevonden zonder bijbehorende FOR-instructie.

NO RESUME (21)
De gebruikte fout-herstel-routine bevat geen RESUME-instructie.

OUT of DATA (4)
Geprobeerd wordt met een READ-instructie gegevens te lezen die niet in een DATA-lijst zijn vastgelegd.

Out of memory (7)
Het programma is te groot voor de beschikbare geheugenruim-

te of het programma bevat te veel variabelen, GOSUB's of 'geneste' FOR-loops.

Out of string space (14)
De beschikbare geheugenruimte voor strings is vol. Met de CLEAR-instructie kan men deze geheugenruimte uitbreiden.

Overflow (6)
Een berekening levert een te groot getal op (zgn. overflow). Het kan ook zijn dat een opgegeven waarde buiten het toegestane bereik valt.

RESUME without error (22)
Er is een RESUME-instructie aangetroffen zonder bijbehorende fout-afhandelingsroutine.

RETURN without GOSUB (3)
Er is een RETURN-instructie aangetroffen zonder dat via GOSUB naar een subroutine gesprongen was.

Redimensioned array (10)
Men probeert met DIM tweemaal dezelfde variabele te specificeren. Kan ook voorkomen als een DIM-instructie volgt op een instructie waarin reeds een array-variabele voorkomt.

String formula too complex (16)
De string-uitdrukking is te ingewikkeld en moet dus in kleinere stukken worden verdeeld.

String too long (15)
De string bestaat uit meer dan 255 tekens.

Subscript out of range (9)
Men gebruikt array-indexnummers die buiten het gespecificeerde bereik vallen.

Syntax error (2)
Algemene foutaanduiding: de uitdrukking voldoet niet aan de regels van BASIC.

Type mismatch (13)
Men tracht aan een string-variabele een numerieke waarde toe te kennen of omgekeerd.

Undefined line number (8)
Er vindt verwijzing naar een niet bestaand regelnummer plaats.

Undefined user function (18)
Een FN-functie wordt aangeroepen die niet met behulp van een DEF FN-instructie van te voren is gedefinieerd.

Unprintable error (23,26-49,60-255)
Voor de aangegeven code bestaat geen standaard-foutmelding.

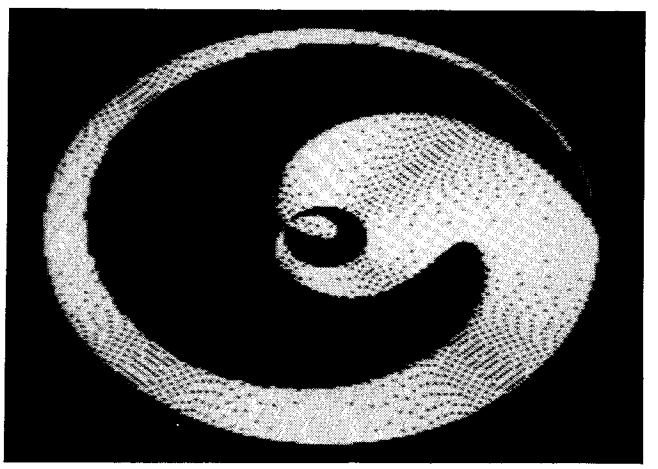
Verify error (20)
Er wordt een verschil geconstateerd tussen het opgenomen programma en het in het geheugen aanwezige programma.

Programma's

1. Grafische potpourri

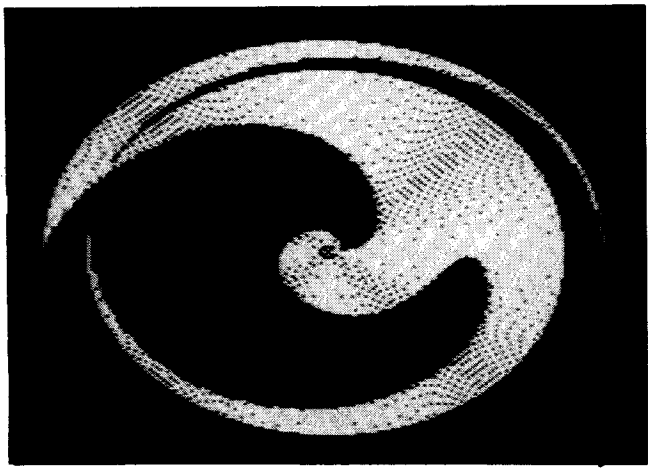
Aan het einde van hoofdstuk 16 toonden we reeds enkele voorbeelden van fraaie plaatjes die met zeer korte programma's verkregen kunnen worden. In het nu volgende worden nog enkele boeiende voorbeelden getoond.

```
10 REM SLANG
20 PI=3.14159
30 SCREEN 2
40 FOR R=1 TO 80
50 CIRCLE(128,96),80-R,15,SIN(R/80*PI)*2*PI,R/80*PI
60 NEXT R
70 GOTO 70
```



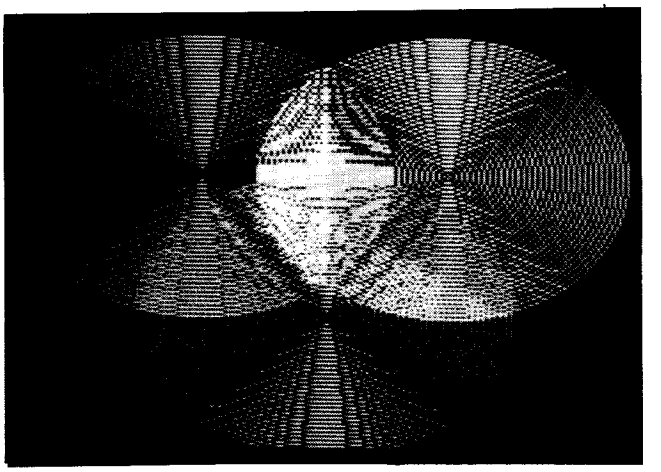
Slang

```
10 REM VOGEL
20 PI=3.14159
30 SCREEN 2
40 FOR R=1 TO 80
50 CIRCLE(128,96),R,15,
SIN(R/80*PI)*2*PI,R/80*PI
60 NEXT R
70 GOTO 70
```



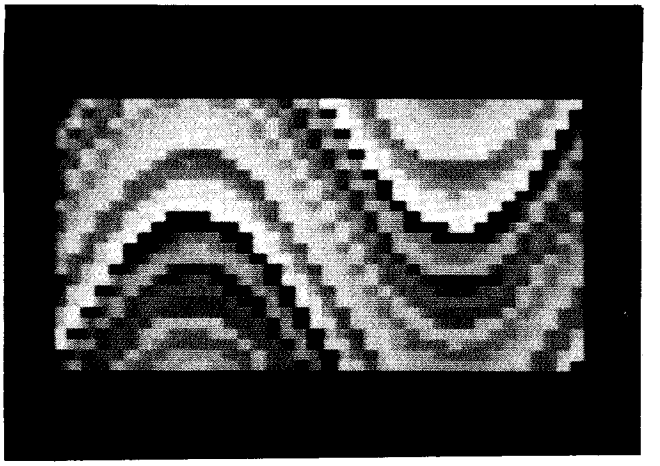
Vogel

```
10 REM 3 CIRKELS
20 SCREEN 2
30 FOR R=1 TO 65 STEP 2
40 CIRCLE(85,65),R,2
50 CIRCLE(170,65),R,7
60 CIRCLE(128,130),R,8
70 NEXT R
80 GOTO 80
```



Drie cirkels

```
10 REM KLEUREN SINUS
20 SCREEN 3
30 PI=3.14159
40 FOR X=50 TO 200 STEP 4
50 FOR Y=45 TO 145 STEP 4
60 PSET(X,Y),((Y/6.25+SIN((X-50)
/75*PI)*8+8)MOD 15+1)
70 NEXT Y
80 NEXT X
90 GOTO 90
```



Kleurensinus

2. Zeilen

Zeilen is een zeer boeiend programma. Na het intypen van het RUN-commando zien we rechts in het beeld een pijl. Hiermee woordt de windrichting aangegeven. Links zien we onze boot en bovendien zien we nog de boei waar we naar toe moeten varen.

Door op de ESC-toets te drukken kunnen we achtereenvolgens de stand van het roer en van het zeil selecteren. Het werkelijk instellen van de stand gebeurt met de cursor-toetsen (←: met de klok mee draaien en →: tegen de klok in draaien).

Goede vaart!

```
10 TIME=0
20 OPEN "GRP:" FOR OUTPUT AS #1
30 PI=3.14159265#
40 X=20:Y=160
50 B=1:H=PI
60 SCREEN 2:COLOR 15,4,4
70 PRESET(200,120):PRINT #1,"O"
80 DRAW"BM255,91L10E5G5F5"
90 Q$=INKEY$
100 IF Q$<>"" THEN AS=ASC(Q$)
110 D$=INKEY$:IF D$<>"" THEN 110
120 IF AS=27 THEN R=NOT R
130 PRESET(230,0)
140 IF R THEN LINE(220,0)-(255,8),4,BF:PRESET(220,0):PRIN
T #1,"Roer"
150 IF NOT R THEN LINE(220,0)-(255,8),4,BF:PRESET(220,0):
PRINT #1,"Zeil"
160 IF AS=29 THEN IF NOT R THEN NZ=ZH+PI/16 ELSE NR=RH+PI
/50
170 IF AS=28 THEN IF NOT R THEN NZ=ZH-PI/16 ELSE NR=RH-PI
/50
180 AS=0
190 V=V+COS(NZ-PI/2+PI-H)*SIN(NZ-PI/2+PI-H)-ABS(V/10)^1.5
*SGN(V)
200 NH=H-SIN(NR)*ABS(V)
210 NX=X-ABS(V)*SIN(PI-NH)
220 NY=Y-ABS(V)*COS(PI-NH)
230 C=4:GOSUB 300
240 GOSUB 340
250 X=NX:Y=NY
260 H=NH:RH=NR:ZH=NZ
270 C=15:GOSUB 300
280 IF X>196 AND X<204 AND Y>116 AND Y<124 THEN SCREEN 0:
PRINT TIME/50;"SEC":END
290 GOTO 90
300 LINE(X,Y)-(X+6*COS(ZH+H),Y+6*SIN(ZH+H)),C
310 LINE(A,B)-(D,E),C
320 CIRCLE(X,Y),4,C
330 RETURN
340 A=NX+4*SIN(NH)
350 B=NY-4*COS(NH)
360 D=A+6*SIN(NH+NR)
370 E=B-6*COS(NH+NR)
380 RETURN
```

3. Integreren

Met het onderstaande programma kan men de integraal van een functie tussen twee grenzen berekenen.

Allereerst wordt het aantal stappen opgevraagd waarmee de integraal zal worden berekend. In het programma wordt gebruik gemaakt van de regel van Simpson.
De functie is in een subroutine ondergebracht: regel 220-230. In dit geval gaat het als voorbeeld om de derde macht van X.

Programma

```
10 CLS
20 INPUT"AANTAL STAPPEN:";N
30 INPUT"ONDERGRENS     :";A
40 INPUT"BOVENGRENS     :";B
50 H=(B-A)/N
60 FOR K=1 TO N-1
70 X=A+K*H
80 C=Q*2+4
90 Q=NOT(Q)
100 GOSUB 220
110 SOM=SOM+C*Y
120 NEXT K
130 X=A
140 GOSUB 220
150 FA=Y
160 X=B
170 GOSUB 220
180 FB=Y
190 I=H/3*(FA+SOM+FB)
200 PRINT "INTEGRATIE LEVERT:";I
210 END
220 Y=X*X*X
230 RETURN
```

Voorbeeld

Aantal stappen: 20
Ondergrens: 0
Bovengrens: 2
Integratie levert: 4

4. Regressie en correlatie

De regressie- en correlatieberekening houdt zich bezig met de vraag hoe men door een gegeven aantal punten zo goed mogelijk een rechte lijn kan trekken. Deze rechte lijn wordt door twee grootheden getypeerd, nl. de hellingshoek M en het punt waar de lijn de Y-as doorsnijdt (B). Ieder punt wordt getypeerd door een X- en een Y-coördinaat. Het programma vraagt allereerst hoeveel punten er worden ingevoerd.

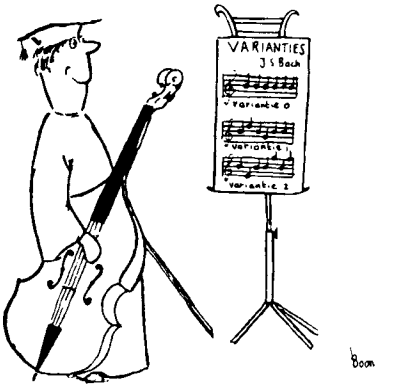
Na het invoeren van de coördinaten worden dan de waarden M en B berekend. Ten slotte wordt de correlatie-coëfficiënt R getoond. Als alle punten precies op een rechte lijn liggen zal de absolute waarde van R gelijk aan 1 zijn. Indien de punten niet precies op een rechte liggen zal de absolute waarde van R minder dan 1 zijn. Kennelijk geeft de waarde van R aan, hoe goed de punten wel op de berekende rechte lijn liggen.

Programma

```
10 CLS
20 INPUT "Aantal punten:";N
30 FOR K=1 TO N
40 PRINT "X";K;
50 INPUT"=";X
60 PRINT "Y";K;
70 INPUT"=";Y
80 XS=XS+X
90 YS=YS+Y
100 X2=X2+X*X
110 Y2=Y2+Y*Y
120 XY=XY+X*Y
130 NEXT K
140 M=(XS*YS/N-XY)/(XS*XS/N-X2)
150 PRINT "M=";M
160 PRINT "B=";YS/N-M*XS/N
170 NM=SQR((N*X2-XS*XS)*(N*Y2-YS*YS))
180 T=N*XY-XS*YS
190 PRINT "R=";T/NM
```

Voorbeeld

Aantal punten: 2
X1 = 1
Y1 = 2
X2 = 2
Y2 = 4
M = 2
B = 0
R = 1



5. Magneet

Wederom een orgineel programma. Een ieder weet dat hij met een magneet een andere magneet kan wegduwen en wel omdat gelijke polen elkaar afstoten. Welnu op dit reeds lang bekende natuurkundige verschijnsel is het volgende programma gebaseerd.

U kunt uw magneet met de cursor-toetsen besturen. Tracht in zo kort mogelijke tijd het aangegeven doel te bereiken!

Programma

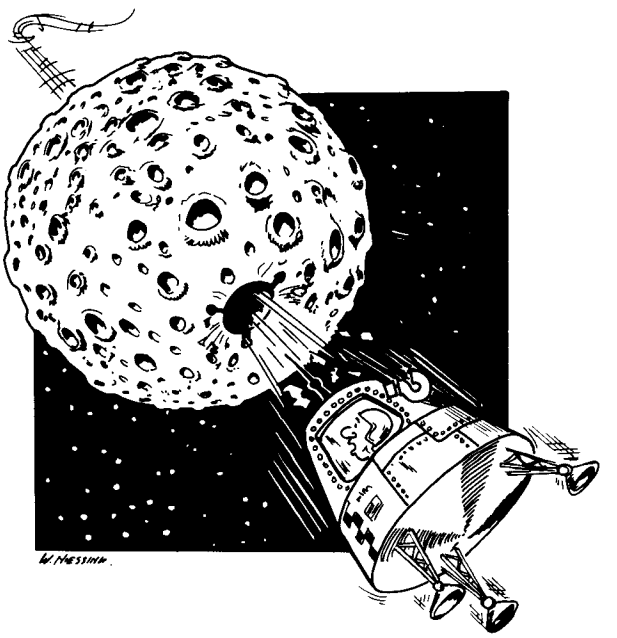
```
10 SCREEN 1,0
20 FOR T=0 TO 32:VPOKE6880+T,0:NEXT T
30 A=20:B=10
40 X=5:Y=5
50 TIME=0
60 SPRITE$(0)=CHR$(&H7E)+CHR$(&H42)+CHR$(&H81)+CHR$(&H81)+CHR$(&H81)+CHR$(&H81)+
CHR$(&H42)+CHR$(&H7E)
70 SPRITE$(1)=CHR$(&H18)+CHR$(&H7E)+CHR$(&H7E)+CHR$(&HFF)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H7E)+CHR$(&H18)
80 VPOKE6480,15
90 Q$=INKEY$
100 IF Q$<>"" THEN AS=ASC(Q$)
110 IF AS=30 THEN Y=Y-3
120 IF AS=31 THEN Y=Y+3
130 IF AS=28 THEN X=X+3
140 IF AS=29 THEN X=X-3
150 AS=0
160 PUT SPRITE1,(X,Y),15,1
170 PUT SPRITE0,(A,B),15,0
180 D=SQR((A-X)^2+(B-Y)^2)
190 F=1/D
200 C=A-COS((A-X)/D)*F*20*SGN(X-A)
210 E=B+SIN((B-Y)/D)*F*20
220 IF C>0 AND C<255 THEN A=C
230 IF E>0 AND E<191 THEN B=E
240 T=INT(TIME/50)
250 LOCATE0,0:PRINT INT(T/60);":";T MOD 60
260 IF INT(A)=127 AND INT(B)=78 THEN SCREEN 0:PRINT INT (T/60);"MIN.";T MOD 60;"
SEC.":END
270 GOTO 90
280 END
```

6. Jupiterlander

Het volgende programma toont ons de Jupiterlander die wij met behulp van onze cursortoetsen veilig op de bodem van Jupiter moeten laten neerkomen.

Zou het lukken... of zou Jupiter het werkelijke definitieve einddoel zijn van onze reis... als u begrijpt wat ik bedoel!

```
10 SCREEN 1
20 FOR T=0 TO 32:VPOKE6880+T,194:NEXT T
30 V=-20
40 S=180
50 TA=200
60 BR=1
70 SPRITE$(1)=CHR$(&H8)+CHR$(&H1C)+CHR$(&H3E)+CHR$(&H3E)+CHR$(&H3E)+CHR$(&H3E)+
CHR$(&H7F)+CHR$(& H7F)
80 Q$=INKEY$:IF Q$<>"" THEN AS=ASC(Q$)
90 IF AS=30 THEN BR=BR+.1
100 IF AS=31 AND BR>0 THEN BR=BR-.1
110 AS=0
120 FOR T=0 TO 100:NEXT T
130 TA=TA-BR
140 A=BR-1
150 V=V+A
160 S=S+(V-A/2)/10
170 IF TA>0 THEN LOCATE6,0:PRINT"      ":LOCATE 0,0:PRINT "Tank:";TA
180 LOCATE 24,0:PRINT"      "
190 LOCATE 12,0:PRINT "Raketkracht:";BR
200 LOCATE 9,1:PRINT"         "
210 LOCATE 0,1:PRINT"Snelheid:";V
220 PUT SPRITE 0,(120,175-S),15,1
230 IF ABS(V)<.5 AND ABS(S)<.5 THEN LOCATE 0,3:PRINT"GELUKT":END
240 IF TA<=0 THEN LOCATE 0,0:PRINT "Tank leeg":BR=0
250 IF S>0 THEN 80
260 PUT SPRITE 0,(120,175-S),0,1
270 P=219:GOSUB 1000
280 P=215:GOSUB 1000
290 P= 15:GOSUB 1000
300 P=  7:GOSUB 1000
310 P= 46:GOSUB 1000
320 P= 32:GOSUB 1000
330 END
1000 VPOKE6860,P
1010 VPOKE6861,P
1020 VPOKE6862,P
1030 VPOKE6863,P
1040 VPOKE6864,P
1050 VPOKE6865,P
1060 VPOKE6866,P
1070 VPOKE6867,P
1080 VPOKE6829,P
1090 VPOKE6830,P
1100 VPOKE6831,P
1110 VPOKE6832,P
1120 VPOKE6833,P
1130 VPOKE6834,P
1140 FOR T=0 TO 10:NEXT T
1150 RETURN
```



Eeuwkalender

Het nu volgende programma bepaalt voor iedere datum van deze en de vorige eeuw de bijbehorende dag.

Zo kunt u eindelijk eens nagaan op wat voor dag u geboren bent. Het getoonde voorbeeld heeft betrekking op 30 augustus 1981.

Programma

```
5 CLS
10 INPUT"Jaar:";J
20 INPUT"Maand:";M
30 INPUT"Dag:";D
40 IF M<=2 THEN J=J-1:M=M+12
50 E=INT((13/5)*(M+1))
60 F=INT(5*J/4)
70 G=INT(J/100)
80 H=INT(J/400)
90 T=D+E+F-G+H
100 DA=1+INT((T/7-INT(T/7))*7+.1)
110 ON DA GOTO 120,121,122,123,124,125,126
120 PRINT "Zaterdag":GOTO 130
121 PRINT "Zondag":GOTO 130
122 PRINT "Maandag":GOTO 130
123 PRINT "Dinsdag":GOTO 130
124 PRINT "Woensdag":GOTO 130
125 PRINT "Donderdag":GOTO 130
126 PRINT "Vrijdag"
130 END
```

Voorbeeld

Jaar: 1981
Maand: 8
Dag: 30
Zondag

8. Marsmannetjes!

Brrr... op sommige plaatsen in de ruimte kunnen veel marsmannetjes voorkomen. Pas op voor botsingen met die glimlachende wezens!
Als kapitein op uw ruimteschip heeft u de cursor-toetsen echter stevig in handen.
Hoelang kunt u deze vloed van marsmannetjes ontwijken?

Programma

```
10 TIME=0
20 SCREEN 1
30 X=128
40 FOR T=0 TO 32:VPOKE(6880+T),0:NEXT T
50 SPRITE$(0)=CHR$(&H81)+CHR$(&H42)+CHR$(&H18)+CHR$(&H3C)+CHR$(&H66)+CHR$(&HC3)+
CHR$(&H81)+CHR$(&H18)
60 Q$=INKEY$
70 IF Q$<>""THEN AS=ASC(Q$)
80 IF AS=28 AND X<246 THEN X=X+2
90 IF AS=29 AND X>16  THEN X=X-2
100 AS=0
110 PUT SPRITE 0,(X,0),15,0
120 IF VPEEK(6144+INT(X/8))=1 OR VPEEK(6144+INT(X/8+.85))=1 THEN BEEP:P=P+1
130 IF P=10 THEN 230
140 R1=30*RND(1)+1
150 R2=30*RND(1)+1
160 C1=INT(1.3*RND(1))
170 C2=INT(1.3*RND(1))
180 VPOKE 6784+R1,1+C1
190 VPOKE 6784+R2,1+C2
200 LOCATE 0,23:PRINT
210 VPOKE 6888,48+P
220 GOTO 60
230 SCREEN 0
240 PRINT TIME/50;"SEC"
```

9. 3D

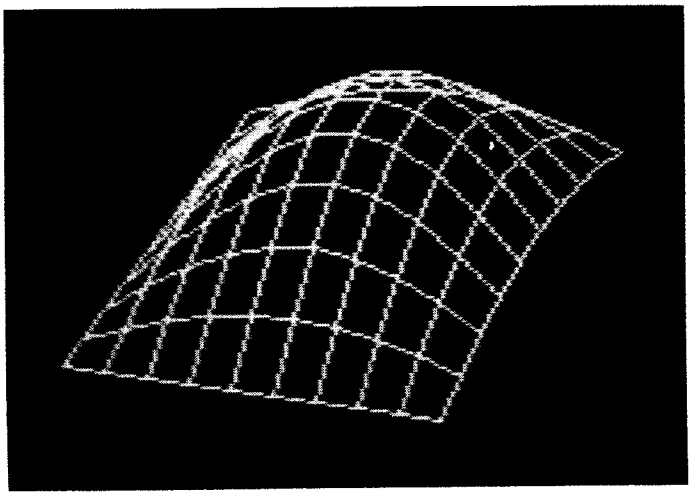
De uitdrukking '3D' heeft betrekking op 'driedimensionaal' of met andere woorden met 'ruimtelijk'. Met behulp van onze MSX-computer kunnen we allerlei fraaie ruimtelijke afbeeldingen maken.

Het volgende programma toont een eenvoudig maar daarom niet minder boeiend voorbeeld.

Programma

```
10 PI=3.14159265#
20 DIM MX(10,10),MY(10,10)
30 CLS
40 FOR X=0 TO 9
50 FOR Y=0 TO 9
60 GOSUB 240
70 ZX=50+X*10+Y*5
80 ZY=120-Y*9-Z+X*2
90 MX(X,Y)=ZX
100 MY(X,Y)=ZY
110 LOCATE 0,0:PRINT X;Y
120 NEXT Y
130 NEXT X
140 SCREEN 2
150 FOR X=0 TO 8
160 FOR Y=0 TO 8
170 LINE(MX(X,Y),MY(X,Y))-(MX(X+1,Y),MY(X+1,Y))
180 LINE(MX(X,Y),MY(X,Y))-(MX(X,Y+1),MY(X,Y+1))
190 NEXT Y
200 LINE(MX(X,9),MY(X,9))-(MX(X+1,9),MY(X+1,9))
210 LINE(MX(9,X),MY(9,X))-(MX(9,X+1),MY(9,X+1))
220 NEXT X
230 GOTO 230
240 Z=50*(SIN(X/10*PI)*SIN(Y/10*PI))
250 RETURN
```

Resultaat



10. Ra, ra... wat is dat?

Dit programma geeft een typisch letterspel. Telkens geeft de computer een aantal letters en u moet raden om welk woord het gaat.

Het is vooral een spel dat de jongeren onder ons kan boeien. De ouderen kunnen dan als oefening proberen na te gaan hoe het programma is opgebouwd.
Tenslotte nog een waarschuwing. Druk bij dit spel eerst op de CAPS-toets, met andere woorden gebruik alleen hoofdletters.

Programma

```
10 PRINT "Druk toets in om te beginnen"
20 W=RND(1):IF INKEY$=""THEN 20
30 DIM A$(10,5)
40 DIM B$(5)
50 G$=""
60 P$="LOHCSARBOCREDDAGEILVREGYTWUEELLEREMKEONSSRAABFARIG"
70 FOR K=1 TO 10
80 FOR L=1 TO 5
90 B1=(K-1)*5
100 A$(K,L)=MID$(P$,B1+L,1)
110 NEXT L
120 NEXT K
130 R=INT(RND(1)*10+1)
140 FOR K=5 TO 1 STEP -1
150 B$(K)=A$(R,K)
160 G$=G$+B$(K)
170 NEXT K
180 X$=""
190 R=INT(RND(5)*5+1)
200 FOR K=R TO R+4
210 L=K
220 IF K>5 THEN L=L-5
230 X$=X$+B$(L)
240 NEXT K
250 PRINT X$;"is een:";
260 INPUT M$
270 IF M$<>G$ THEN PRINT "FOUT":GOTO 250
280 PRINT "GELUKT"
```

Voorbeeld

CARBO is een: ? RACBO
FOUT
CARBO is een: ? COBRA
GELUKT

11. Mercuriusmonsters

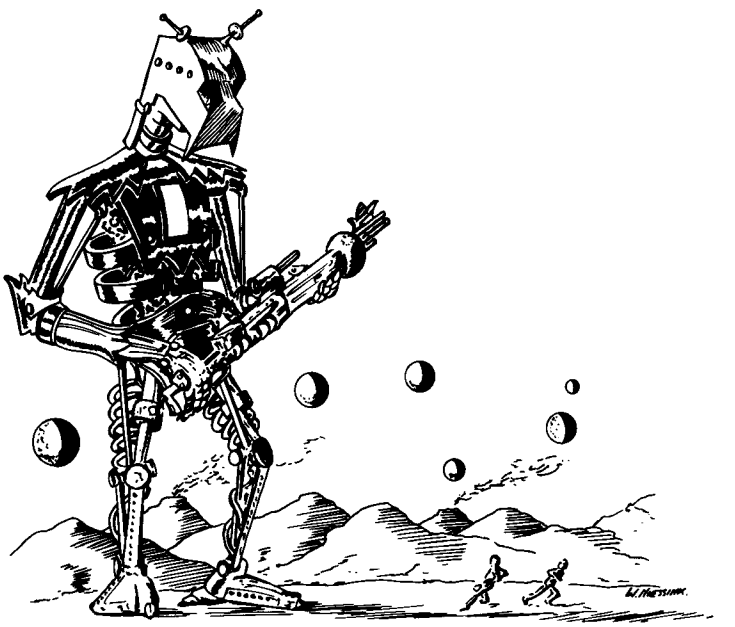
Alweer dreigt er gevaar... de mercuriusmonsters komen! U bent de moedige piloot in een jager en met de ESC-toets onder uw handen moet u het gevaar kunnen weren.

Ten slotte kunt u deze verschrikkelijke monsters ontwijken door op het juiste moment de cursor-toetsen te gebruiken. Pas op, de monsters worden steeds agressiever!

Programma

```
10 OPEN "GRP:" FOR OUTPUT AS #1
20 SCREEN 2:COLOR 15,4,4
30 SPRITE$(0)=CHR$(&H10)+CHR$(&H30)+CHR$(&HF0)+CHR$(&HFF)
+CHR$(&HFF)+CHR$(&HF0)+CHR$(&H30)+CHR$(&H10)
40 SPRITE$(1)=CHR$(&H6)+CHR$(&HC)+CHR$(&H1C)+CHR$(&HFF)+C
HR$(&HFF)+CHR$(&H1C)+CHR$(&HC)+CHR$(&H6)
50 YK=96
60 SPEED=0
70 XE=246:YE=160
80   'hoofdlus
90 GOSUB 180
100 N=N+1
110 IF N>999 THEN GOSUB 640
120 GOSUB 370
130 IF SC> 60 THEN FIRE=0:SPEED=2
140 IF SC>100 THEN FIRE=-1:SPEED=1
150 IF SC>150 THEN FIRE=-1:SPEED=2
160 IF SC>200 THEN GOSUB 630
170 GOTO 90
180 IF HIT<>0 THEN HIT=HIT-1:RETURN
190 Q$=INKEY$
200 R=RND(1)
210 IF Q$<>"" THEN AS=ASC(Q$)
220 IF AS=30 AND YK>8 THEN YK=YK-8
230 IF AS=31 AND YK<186 THEN YK=YK+8
240 PUT SPRITE 1,(12,YK),15,0
250 IF AS<>27 THEN 350
260 LINE(247,YK+4)-(14,YK+4)
270 BEEP
280 LINE(247,YK+4)-(14,YK+4),0
290 SC=SC-2
300 IF YK-4>YE OR YK+4<YE THEN 350
310 X=XE:Y=YE
320 GOSUB 520
330 SC=SC+10
340 FLAGE=0
350 AS=0
360 RETURN
370   'aanvallers
380 IF FLAGE=0 THEN FLAGE=1:XE=247:YE=INT(RND(1)*22+2)*8
390 DY=SPEED*INT(RND(1)*3-1)*3
400 XE=XE-SPEED
410 YE=YE+DY
420 IF YE<8   THEN YE=8
```

```
430 IF YE>186 THEN YE=186
440 IF XE<10  THEN FLAGE=0:GOTO 500
450 PUT SPRITE 0,(XE,YE),15,1
460 IF NOT FIRE OR RND(1)>.15 THEN 500
470 LINE(10,YE+4)-(XE,YE+4)
480 LINE(10,YE+4)-(XE,YE+4),0
490 IF YE-3<YK AND YE+3>YK THEN HIT=5:X=14:Y=YK:GOSUB 520
500 RETURN
510   'explosie
520 PRESET(X,Y):PRINT #1,CHR$(1);CHR$(71)
530 PUT SPRITE 0,(0,0),0,1
540 PRESET(X,Y):PRINT #1,CHR$(249)
550 PRESET(X,Y):PRINT #1,CHR$(248)
560 PRESET(X,Y):PRINT #1,CHR$(1);CHR$(79)
570 PRESET(X,Y):PRINT #1,CHR$(1);CHR$(73)
580 PRESET(X,Y):PRINT #1,CHR$(215)
590 PRESET(X,Y):PRINT #1,CHR$(219)
600 FOR T=0 TO 20:NEXT T
610 LINE(X,Y)-(X+8,Y+8),0,BF
620 RETURN
630 SCREEN 0:PRINT"U bent erin geslaagd de vijand te    v
erdrijven,hartelijk gefeliciteerd.","U had";N/100;"lichts
econden nodig.":END
640 SCREEN 0:PRINT"Het is niet mogelijk gebleken u als  j
ager in dienst te houden en u wordt derhalve gedegradeerd
.":END
```



12. B52-ganzen

Het type ganzen dat met de codenaam B52 wordt aangeduid is een zeer gevaarlijk type ganzen. In feite gaat het hier niet om een zuiver natuurprodukt... Deze ganzen zijn immers in staat tot het uitvoeren van bombardementen en welke ganzen kunnen dat?

Het zal dan ook duidelijk zijn dat deze ganzen moeten worden neergeschoten.
Uw kanon kunt u met de cursor-toetsen richten en met de spatie-balk afschieten. Pas op, als de B52-ganzen uw kanon raken.
In eerste instantie kunt u dan nog wel schieten maar u bent al uw bewegingsvrijheid kwijt.
Gelukkig zien we deze verschrikkelijke B52-ganzen niet al te vaak!

Programma

```
10 SCREEN 2
20 COLOR 15,4,14
30 S$=CHR$(&H0)+CHR$(&H0)+CHR$(&H0)+CHR$(&HE)+CHR$(&HFF)
40 SPRITE$(1)=CHR$(&H0)+CHR$(&H4)+CHR$(&HC)+CHR$(&H1F)+CHR$(&HFE)
50 SPRITE$(2)=S$
60 SPRITE$(3)=CHR$(&H0)+CHR$(&H0)+CHR$(&H0)+CHR$(&H0)+CHR$(&HFF)+CHR$(&H1C)+CHR$
(&HC)+CHR$(&H8)
70 SPRITE$(4)=S$
80 SPRITE$(0)=CHR$(&H8)
90 SPRITE$(5)=CHR$(&H8)+CHR$(&H8)+CHR$(&H3E)+CHR$(&H3E)+CHR$(&H7F)+CHR$(&H7F)+CHR
$(&H41)+CHR$(&H22)
100 SPRITE$(6)=CHR$(&H0)+CHR$(&H0)+CHR$(&H0)+CHR$(&H24)+CHR$(&H24)+CHR$(&H66)+CHR
$(&H7E)+CHR$(&HFF)
110 SPRITE$(7)=""
200 '
210 'NIEUWE GANS
220 '
230 FOR D=0 TO 2:PUT SPRITE D,(0,0),4,7:NEXT D
240 SPRITE ON
250 STRIG(0) ON
260 ON SPRITE GOSUB 700
270 ON STRIG GOSUB 900
280 IF INKEY$<>"" THEN 280
290 DIVE=0:BULL=0
300 X=255:Y=181
310 T=T+1
320 V=RND(1)*6+4
330 H=RND(1)*100+20
400 '
410 'HOOFDLUS
420 '
430 PUT SPRITE 0,(X,H),14,N
440 Q$=INKEY$:IF Q$<>"" THEN W=ASC(Q$)*-(NOT HIT) ELSE W=0
450 IF W=28 THEN A=A+15
460 IF W=29 THEN A=A-15
470 PUT SPRITE 2,(A,184),11+10*HIT,5-HIT
480 X=X-V
490 N=N MOD 4+1
500 FOR E=0 TO 10
510 IF BULL THEN Y=Y-1.5:PUT SPRITE 1,(XA,Y),15,0 ELSE FOR D=0 TO 2:NEXT D
```

```
520 NEXT E
530 IF Y<-4 THEN Y=181:BULL=0:STRIG(0) ON
540 IF ABS(INT(X)-A)<4 THEN SPRITE OFF:LINE(A+5,H+4)-(A+5,192),15:LINE(A+5,H)-
 (A+5,192),4:SPRITE ON:IF RND(1)<.7+.35*HIT THEN IF HIT THEN 600 ELSE HIT=-1
550 IF X>0 AND NOT DIVE THEN 400 ELSE IF T=10 THEN 600 ELSE 200
600 '
610 'EINDE
620 '
630 SCREEN0
640 PRINT G;"GANZEN VAN DE";T;"AFGESCHOTEN"
650 END
700 '
710 'GANS GERAAKT
720 '
730 SPRITE OFF
740 BEEP
750 PUT SPRITE 1,(0,0),4,7
760 D=H:E=1
770 G=G+1
780 SPRITE ON
790 X=X-V:D=D+E
800 PUT SPRITE 0,(X,D),1,D MOD 4+1
810 E=E+E/2
820 ON SPRITE GOSUB 1100
830 IF D<184 THEN FOR U=0 TO 100:NEXT U:GOTO 790
840 DIVE=-1
850 SOUND 6,10
860 SOUND 7,31
870 FOR D=250 TO 0 STEP -1:SOUND 10,D/20:NEXT D
880 IF HIT THEN HIT=0
890 RETURN
900 '
910 'AFVUREN KOGEL
920 '
930 STRIG(0) OFF
940 SOUND 6,15
950 SOUND 7,31
960 SOUND 10,15
970 FOR D=0 TO 5:NEXT D
980 SOUND 10,0
990 BULL=-1
1000 XA=A
1010 RETURN
1100 '
1110 'GANS NEERGESTORT OP GESCHUT
1120 '
1130 SPRITE OFF:BEEP
1140 IF NOT HIT THEN HIT=-1
1150 RETURN
```

Trefwoordenregister

N

O

P

R

S

T

U

V

W

X

Z