BASIC

A.C.J. Groeneveld

handboek voor iedereen

uw MSX computer de baas

BASIC
A.C.J. Groeneveld
handboek voor iedereen

BASIC
handboek voor iedereen

uw MSX computer de baas

BASIC

handboek voor iedereen

uw **MSX** *computer*
de baas

A.C.J. Groeneveld

uitgeverij STARK - TEXEL
postbus 302 - 1794 ZG Oosterend tel. 02223 - 661

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Groeneveld, A.C.J.
BASIC handboek voor iedereen : Uw MSX computer de baas /
A.C.J. Groeneveld. - Oosterend : Stark-Texel
ISBN 90-6398-100-7
SISO 365.3 UDC 800.92
Trefw.: programmeertalen

1e druk 1984
ISBN 90 6398 100 7

INHOUD pag.

1 INLEIDING

MSX is een afkorting van MicroSoft eXtended basic. Met de merknaam MSX heeft MICROSOFT, het toonaangevende software huis op het gebied van microcomputers, een zéér belangrijke en bitter noodzakelijke stap gedaan in de richting van de standaardisering van hobby- en microcomputers en hun taal. Eindelijk is het mogelijk om programmatuur aan te schaffen die op vele merken micro's werkt;een behoefte die al jarenlang bestond.

MSX is meer dan alleen eeen standaard BASIC taal. MSX is een volledig standaard concept voor microcomputers. Niet alleen de taal maar ook de opbouw van het toetsenbord is door MSX standaard voorgeschreven. Daarbij worden standaard chips voorgeschreven voor de processor, de geluidsgenerator, de beeldschermprocessor etcetera. Ook de geheugenindeling ligt vast. Hierdoor wordt mogelijkheid geschapen om behalve programma's ook randapparatuur (ROM-cassettes, floppy disk eenheden, joy sticks e.d.) tussen de verschillende merken microcomputers uit te kunnen wisselen. Hierdoor zal MSX standaard niet alleen op softwaregebied maar ook op hardwaregebied een stroming in de automatiseringsmarkt opwekken die resulteert in een rijk geschakeerd en vriendelijk geprijsd aanbod.

Dit handboek heeft tot doel de steun en toeverlaat te zijn voor de MSX-programmeur. De amateur, maar ook de professionele programmeur vindt in dit handboek een duidelijk, overzichtelijk en nederlands naslagwerk. Het handboek behandelt de MSX standaard, geeft wat bijzonder bruikbare tabelinformatie, behandelt de videochip en de soundchip die MSX voorschrijft, maar gaat vooral zéér diep in op het belangrijkste gebied: het MSX-BASIC. Van dit basic wordt de exacte schrijfwijze en betekenis per kommando behandeld en wordt het één en ander met duidelijke (óók nederlandse) voorbeelden toegelicht. Om ook de nieuwkomer in de computerwereld van dienst te kunnen zijn, worden algemene zaken als konstanten, variabelen en uitdrukkingen breedvoerig behandeld. Doordat daarbij een geadviseerde leervolgorde van de diverse sleutelwoorden is opgenomen, kan dit handboek ook uitstekend dienen als een complete instruktie voor de beginnende programmeur.

In een apart hoofdstuk zijn de bij het MSX disk-basic behorende sleutelwoorden genoemd. Deze bevelen zijn niet standaard maar dienen te worden gezien als een uitbreiding op het MSX-basic bij

gebruik van schijveneenheden. Deze sleutelwoorden kunnen ook pas worden geactiveerd na aansluiting van een schijveneenheid aan de MSX-computer.
Er zijn diverse schijf-uitbreidingen voor de MSX-computer te verkrijgen. Helaas voldoen deze niet allemaal aan de MSX disk-basic standaard.

In een apart verkrijgbaar handboek worden de MSX-disk-basic sleutelwoorden verder behandeld; zij zijn alleen interessant voor de bezitters van een schijven eenheid.

Het handboek behandeld het MSX-basic zoals het in versie 1.0 funktioneert maar zal bij latere versies van MSX-basic zeer bruikbaar blijven.

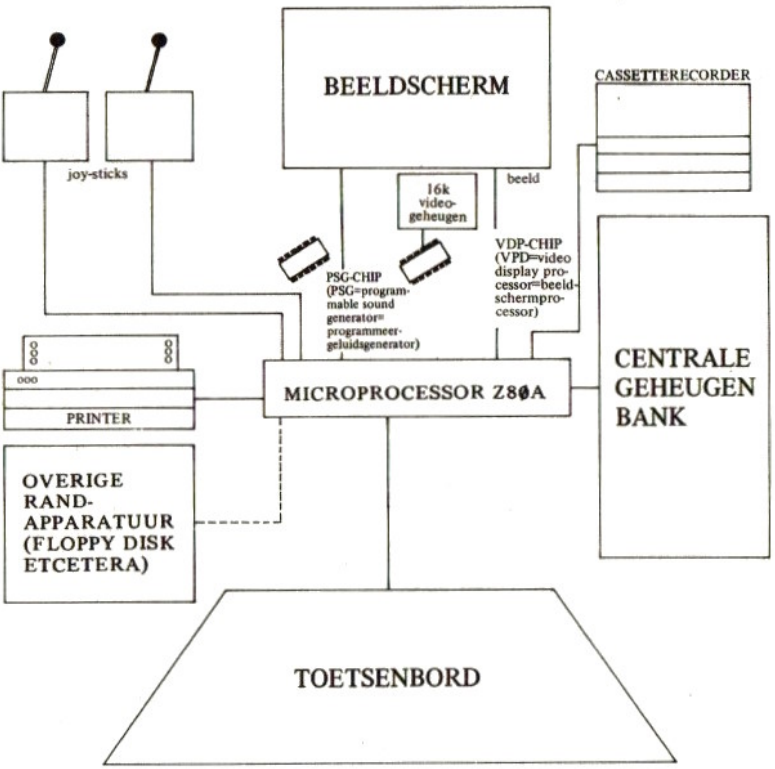
Ik hoop dat dit handboek een welkome ondersteuning voor vele MSX-programmeurs mag blijken te zijn.

januari 1985,
A.C.J. Groeneveld.

Zoals in de inleiding reeds werd opgemerkt, beperkt de MSX standaard zich niet alleen tot de BASIC taal. Ook het hardware-concept wordt duidelijk door deze standaard bepaald.

2.1 De hardware van een MSX-computer

Hieronder volgt een schema van de algemene opbouw van een MSX-computer.



Als standaard microprocessor is binnen de MSX-standaard gekozen voor de Z-80-A microprocessor (of een processor met gelijke specificaties). De Z-80 is de meest verbreide microprocessor en vormt al jaren het hart van vele microcomputers. Alhoewel de Z-80 een 8 bits microprocessor is en door velen als ouderwets wordt betiteld ten opzichte van de 16 bits microprocessoren, voldoet deze chip nog steeds uitstekend op hobby- en semi-professioneel gebied. De verwachting is, dat in de toekomst nog vele nieuwe computermodellen zullen worden ontworpen op basis van de Z-80 microprocessor.

Als geluidsgenerator is gekozen voor de AY-3-8912 geluidsprocessor (of een processor met gelijke specificaties). Deze processor is een computer op zich en regelt alle geluidseffecten van de MSX-computer. Ook voor deze geluidsgenerator geldt dat het een oude bekende is binnen de computerwereld. Echter, door zijn grote aantal mogelijkheden voldoet ook deze processor uitstekend. De AY-3-8912 bezit drie geluidskanalen, één ruisgenerator en een variëteit van effecten. Hierdoor is het bijvoorbeeld goed mogelijk om een driestemmig stuk muziek met slagwerk te programmeren.

Als beeldschermprocessor werd binnen de MSX-standaard gekozen voor de TMS 9929A processor (of een processor met gelijke specificaties). Bijzonder is dat deze zeer geavanceerde grafische processor volgens de standaard dient te zijn uitgerust met een apart stuk werkgeheugen (RAM) van maar liefst 16 kilobytes. Hierdoor kan een hoog oplossend vermogen (fijngetekend beeldscherm) worden bereikt zonder dat meteen een aanzienlijk deel van het toch al zo snel tekort schietende centrale geheugen wordt gereserveerd.

Als standaard magnetisch opslagmedium gaat de MSX-standaard uit van een doodgewone cassetterecorder. Op een normale geluidscassette worden programma's gezet en kunnen ook andere gegevens worden geplaatst. Bijzonder is dat de MSX-standaard uitgaat van een relais dat de recordermotor aan en uit kan schakelen. Dit relais is via het MOTOR-commando ook in MSX-BASIC te besturen; de handige programmeur kan op deze wijze dit relais ook voor andere besturingsdoeleinden gebruiken zoals bijvoorbeeld de besturing van een diaprojector.

Uiteraard voorziet de standaard in de aansluiting van een printer. Deze aansluiting geschiedt via het standaard Centronics parallelle protocol. De printer kan een afdrukeenheid zijn met de mogelijkheid om de standaard MSX grafische symbolen af te drukken maar mag ook een algemene printer zonder deze mogelijkheden zijn.

De joy-sticks vormen onmisbare attributen voor de spelletjes-fanaat! Als zodanig ontbreken deze niet in de MSX-standaard. Programmeurs die spelletjes in MSX-basic willen schrijven, vinden een keur van eenvoudige commando's, waarmee de joy-sticks volledig kunnen worden uitgebuit.

Alhoewel MSX in versie 1.0 nog geen verdere randapparatuur ondersteunt, is aansluiting van overige randapparatuur, met name van floppystations, ook mogelijk. Een hogere MSX-BASIC versie of een speciale BASIC-uitbreiding is noodzakelijk om deze randapparatuur aan te kunnen spreken.

Als beeldscherm kan natuurlijk een mooie en dure kleurenmonitor worden aangesloten, maar een eenvoudige draagbare zwart-wit televisie werkt ook al zeer bevredigend. MSX schrijft zowel een composiet-uitgang (naar de antenne-ingang van het televisietoestel) als een video-uitgang met apart geluidskanaal (voor monitors) voor.

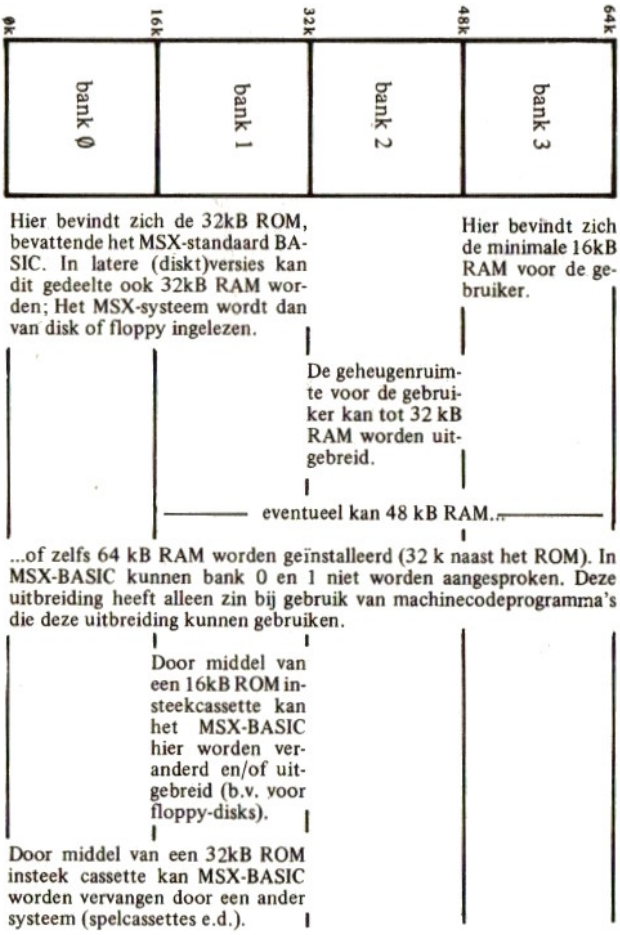
2.2 Het geheugen van een MSX-computer

Het geheugen van een computer heeft een bepaalde grootte, die men uitdrukt in bytes, kilobytes of megabytes.
Een byte is een geheugen-eenheid waarin één enkel karakter kan worden opgeslagen. Om bijvoorbeeld de tekst "MSX-BASIC" in het computergeheugen op te slaan, zijn negen geheugeneenheden ofwel negen bytes benodigd.

Indien men over geheugengrootten spreekt, is het gebruikelijk om deze uit te drukken in kilobytes. Het merkwaardige is, dat in de computerwereld een kilobyte niet 1000, maar 1024 bytes telt. Indien men het over 32 kilobytes of 32 Kb heeft, dan bedoelt men dus een geheugengrootte van 32768 bytes. In dat geheugen kunnen dus 32768 lettertekens (of 32768 machinecode-instructies, zie hoofdstuk 4) worden opgeslagen.

Een megabyte telt weer 1024 kilobytes. Binnen de microcomputerwereld heeft men het meestal pas over megabytes, indien men het over de opslagcapaciteit op magneetschijf of diskette heeft.

De centrale geheugenbank van de MSX-computer bestaat in werkelijkheid uit **vier** geheugenbanken van elk 16 kilobytes. In het volgende schema wordt de mogelijke bezetting van deze geheugenbanken beschreven.

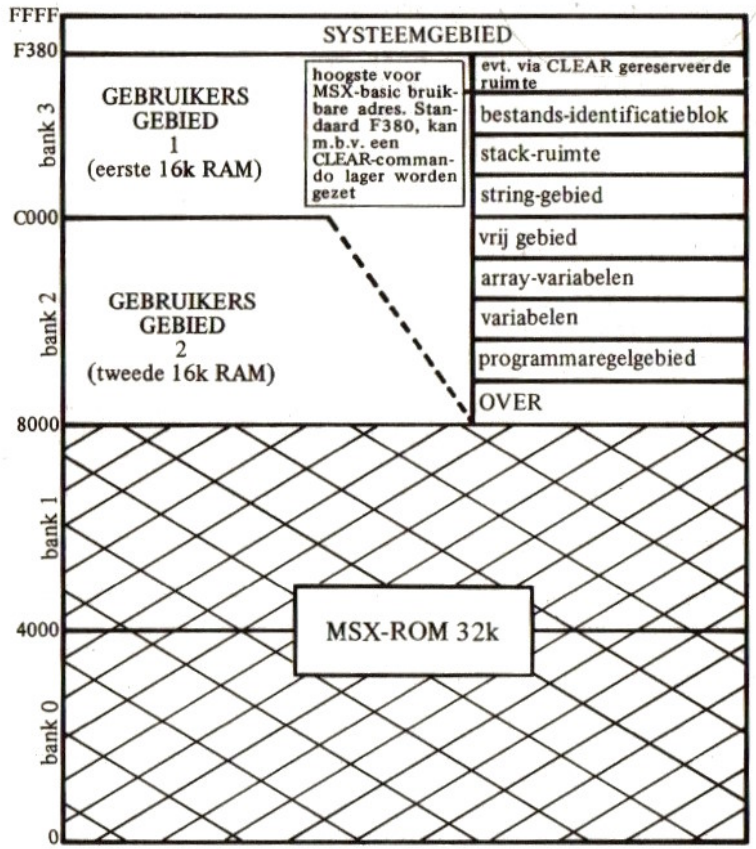


In het meest eenvoudige systeem schrijft de MSX-standaard 32 kilobytes ROM (READ ONLY MEMORY=alleen uitleesbaar geheugen met vaste inhoud) voor waarin het MSX-systeem dient te zijn gecodeerd. Daarbij dient er minimaal 16 kilobytes aan RAM (RANDOM ACCES MEMORY=algemeen toegankelijk en veranderbaar geheugen) in bank 3 aanwezig te zijn. Door middel van uitbreidingen of ROM-cassettes kan de totale geheugenopbouw worden veranderd.

2.3 MSX-BASIC en het RAM

Het MSX-BASIC gebruikt het beschikbare RAM op een wijze die is toegelicht in het volgende schema.

SCHEMA GEHEUGENBANK MSX-BASIC



Het allerbovenste gedeelte is altijd gereserveerd voor het MSX systeem. Dit gedeelte dient als kladblaadje voor interne gegevens, dat het systeem zo nu en dan nodig heeft.

Onder dit systeemgedeelte volgen diverse blokken die elk moment, afhankelijk van het werkende programma, anders van grootte kunnen zijn.

Van boven naar beneden komen we als volgende blok de via CLEAR gereserveerde ruimte tegen. Deze ruimte kan binnen de BASIC-taal worden gecreëerd via het CLEAR sleutelwoord. Deze ruimte kan door de gevorderde programmeur vervolgens worden gebruikt voor het opslaan van routines, geschreven in de Z-80 machinetaal. Deze routines kunnen dan weer vanuit de BASIC-taal worden aangeroepen door middel van het DEFUSR en het USR sleutelwoord. In het normale geval bestaat dit blok niet; onder het systeem volgt dan onmiddellijk het bestandsidentificatieblok.

Het bestandsidentificatieblok bevat gegevens die het MSX-BASIC nodig heeft bij het uitwisselen van gegevens met bijvoorbeeld een cassetterecorder of een floppy disk eenheid.

Het stringgebied bevat de alfanumerieke gegevens (gegevens, bestaande uit letters, cijfers en leestekens) waarmee het betreffende programma manipuleert.

De stack-ruimte (stack=stapel) bevat een aantal gegevens in verband met de besturing van het programma. Later, bij de behandeling van bijvoorbeeld het GOSUB sleutelwoord zal blijken dat het benodigde terugkeeradres een gegeven is dat in deze stack-ruimte wordt opgeslagen.

Onder de stack-ruimte volgt een klein, ongebruikt stukje geheugen.

Hieronder bevinden zich de array-variabelen. De namen, specificaties en numerieke inhoud liggen in dit gebied opgeslagen. De alfanumerieke inhoud van de array-variabelen ligt echter in het stringgebied opgeslagen. Voor de betekenis van variabelen: zie hoofdstuk 6.

Onder de array-variabelen bevinden zich de niet-array-variabelen. Deze variabelen zijn op een dergelijke manier binnen dit gebied opgeslagen.

Als laatste blok volgt het blok met programmaregels. Het feitelijke BASIC-programma ligt in dit blok opgeslagen.

Indien er nog geheugen over is, ligt dat onder het programmaregel-gebied.

2.4 Het toetsenbord van een MSX-computer

Zoals bij de meeste microcomputers het geval is, bezit ook de MSX-computer een toetsenbord met een standaard indeling. In Nederland is het zogenaamde QWERTY-toetsenbord het meest gebruikt.

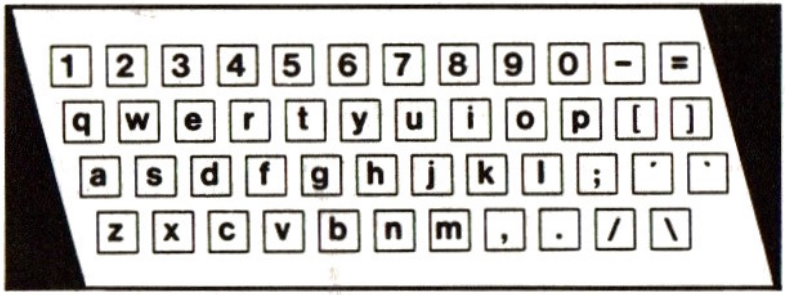
Het toetsenbord valt onder te verdelen in drie soorten toetsen, te weten de karaktertoetsen, de controletoetsen en de functietoetsen.

De karaktertoetsen genereren bij intoetsing de vermelde letters en cijfers; op het eerste gezicht niets bijzonders dus. Echter, in combinatie met de functietoetsen SHIFT, CODE en GRAPH zijn er veel meer tekens te genereren dan op het eerste gezicht duidelijk is. Om deze tekens uit te proberen behoeft men slechts de computer aan te zetten en te wachten op de copyrightmelding, gevolgd door de Ok-melding. Hierna kunnen en mogen alle toetsaanslagen worden uitgeprobeerd. Let u in eerste instantie niet op de vele foutmeldingen die de MSX-computer kan geven als protest op uw door de computer niet begrepen intoetsingen.

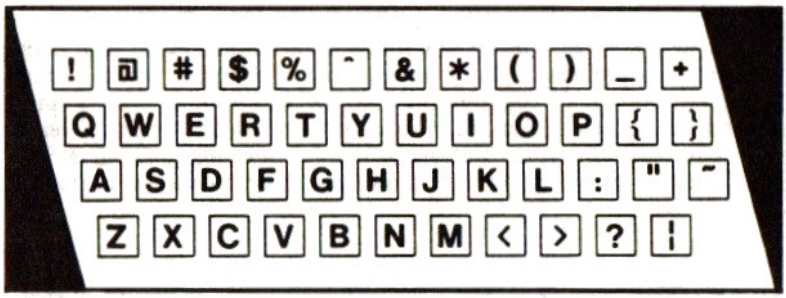
De zes schema's op de volgende twee pagina's geven de karakters weer die via het toetsenbord kunnen worden gegenereerd.

Buiten deze uitgebreide set van karakters kunnen ook acties aan de MSX-computer worden ontlokt. Dit gebeurt via de zogenaamde controletoetsen. De MSX-standaard vereist de aanwezigheid van de volgende controletoetsen:

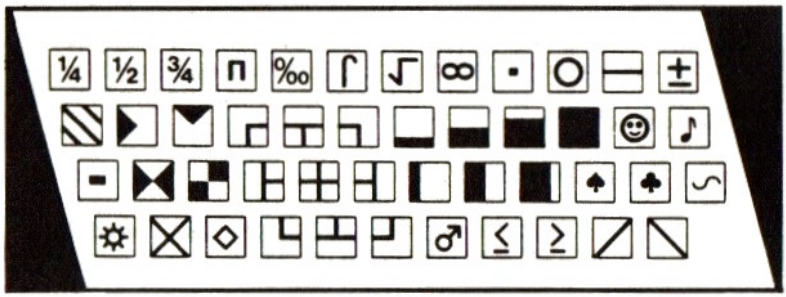
toets	functie
ESC	Kan in programma's worden gebruikt, maar heeft binnen het MSX-basic geen functie.
TAB	Met deze toets kan direct naar een volgende kolom op het beeldscherm worden gesprongen. Heeft over het algemeen niet veel nut. Het beeldscherm is standaard in kolommen van 8 tekens ingedeeld.



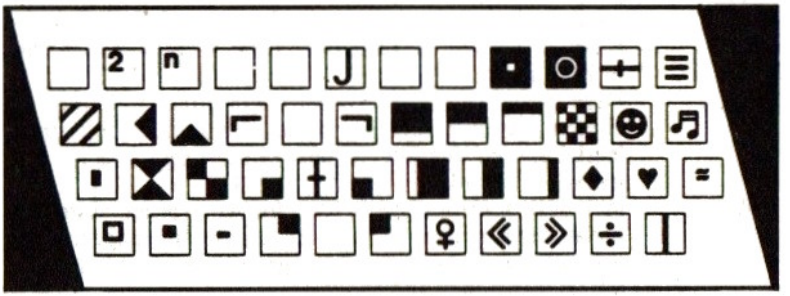
Karakters die normaal ontstaan bij het intoetsen.



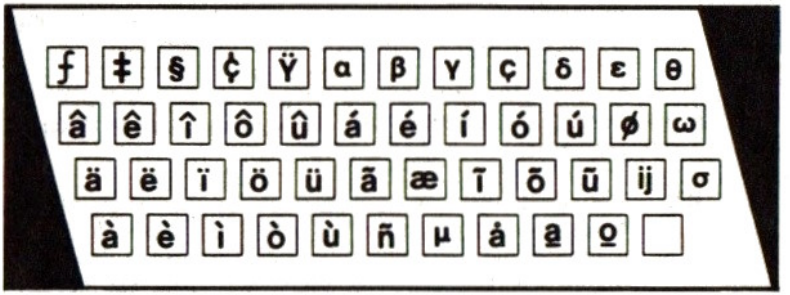
Karakters die ontstaan indien de toetsen tegelijk met de SHIFT-toets worden ingedrukt.



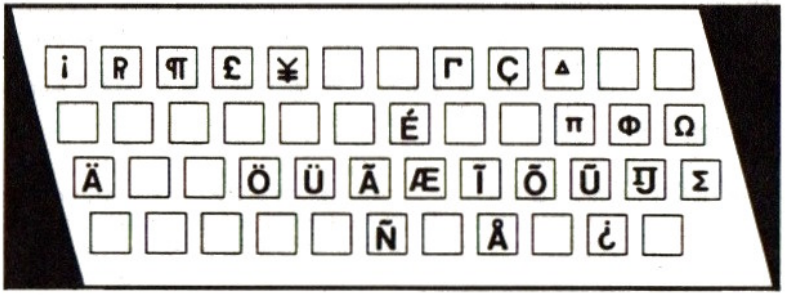
Karakters die ontstaan indien de toetsen tegelijk met de GRAFH-toets worden ingedrukt.



Karakters die ontstaan indien de toetsen tegelijk met de SHIFT- en GRAFH-toets worden ingedrukt.



Karakters die ontstaan indien de toetsen tegelijk met de CODE-toets worden ingedrukt.



Karakters die ontstaan indien de toetsen tegelijk met de SHIFT- en CODE-toets worden ingedrukt.

CTRL Deze toets heeft op zichzelf geen functie. Echter in combinatie met andere toetsen kunnen allerlei controle-functies worden opgeroepen. Deze worden behandeld in hoofdstuk 3.

SHIFT Deze toets heeft op zichzelf geen functie. Echter in combinatie met andere toetsen kunnen hoofdletters, leestekens en speciale tekens worden gegenereerd. Zie hiervoor de eerder geplaatste tabel B.

CAPS Met deze toets kan de SHIFT functie worden vastgezet zodat deze niet steeds behoeft te worden vastgehouden wanneer men bijvoorbeeld een groot stuk tekst in hoofdletters wenst in te tikken.

GRAPH In combinatie met andere toetsen geeft deze toets toegang tot een extra set karakters zoals eerder in tabel C opgenomen.

CODE In combinatie met andere toetsen geeft deze toets toegang tot de karakterset zoals opgenomen in tabel E.

N.B.: Door de controletoetsen SHIFT-GRAPH en SHIFT-CODE te combineren, kunnen de karakters zoals opgenomen in de tabellen D en F worden gegenereerd.

RETURN Met deze toets worden ingaven op het toetsenbord bevestigd. Pas na ingave van RETURN 'weet' de MSX-computer dat de ingave is beëindigd en wordt de ingave op juistheid gecontroleerd. Indien u zo maar wat met het toetsenbord speelt, bijvoorbeeld om de karakterset uit te proberen, merkt u dat de computer pas na ingave van de RETURN-toets een eventuele foutmelding geeft.

SELECT Binnen het MSX-basic heeft deze toets geen speciale functie. Hij kan echter in programma's wel worden gebruikt.

BS of BACKSPACE Met deze toets kan het laatst ingetikte karakter weer worden verwijderd.

CLS/HOME Zonder gebruik te zamen met de SHIFT toets zorgt deze toets ervoor, dat de cursor (het blokje op het scherm dat aangeeft waar de ingave is gebleven) links boven in de beeldschermhoek komt te staan. Met gebruik van de SHIFT toets wordt daarbij ook nog het gehele beeldscherm uitgewist.

INS Met behulp van deze toets kan in een bestaande regel een

karakter of een stuk tekst worden tussengevoegd. Door een intoetsing wordt het tussenvoegen geactiveerd; door nog een intoetsing of door gebruik van één van de pijltoetsen wordt het tussenvoegen weer gedeactiveerd.

DEL Met behulp van deze toets kan één karakter worden verwijderd op elke plaats binnen een regel. De rest van de regel wordt dan aangeschoven.

STOP Met deze toets kan een werkend BASIC-programma worden stilgezet. Een volgende intoetsing zorgt ervoor dat het betreffende programma dan weer verder gaat. In combinatie met de CTRL-toets kan met deze toets een programma worden onderbroken.

PIJLTOETSEN Met deze toetsen kan de cursor in horizontale en vertikale richting worden verplaatst. Door intoetsing van twee toetsen tegelijk kan de cursor ook diagonaal (schuin) worden verplaatst. De pijltoetsten kunnen in een BASIC-programma te zamen als joy-stick dienst doen.

Alle functietoetsen zijn gekenmerkt door een afgekorte, Engelstalige naam. Ter verduidelijking volgt voor elke toets hier een verklaring van de naam:

ESC	ESCape=ontsnappen
TAB	TABulation=tabulatie (kolomindeling)
CTRL	ConTRoL=controle
SHIFT	SHIFT=opschuiven
CAPS	CAPitalS=hoofdletters
GRAPH	GRAPHics=grafische tekens
CODE	CODE=codering
RETURN	RETURN=terugkeren (naar de linkerkant van het beeldscherm, één regel lager)
SELECT	SELECT=selecteren, uitzoeken
BACKSPACE	BACKSPACE='terugspatie' een spatieteken wordt de 'verkeerde' kant uit over het foute karakter gezet.
CLS	CLear Screen=beeldscherm schoonmaken
HOME	HOME=huis. De cursor wordt naar 'huis' teruggestuurd.
INS	INSert=tussenvoegen
DEL	DELete=verwijderen
STOP	STOP=stoppen, stilstaan

De functietoetsen zijn genummerd van 1 tot en met 10. 'Onder' deze

functietoetsen zijn vaste teksten opgenomen die veelvoudig terugkeren. Door middel van de intoetsing van de betreffende functietoets wordt dan de gehele tekst op beeldscherm geplaatst. De functietoetsen kunnen via het MSX-basic eventueel van functie worden veranderd. De functies worden (gedeeltelijk) onder in het beeld zichtbaar gemaakt. Deze regel is eventueel uitschakelbaar; ook dit kan in MSX-basic geschieden.
De functies 6 tot en met 10 worden indien slechts vijf functietoetsen aanwezig zijn, opgeroepen door de functietoetsen te zamen met de SHIFT-toets te gebruiken.

3 DE MSX-EDITOR

De MSX-standaard voorziet in een full screen editor. Een editor is de Engelse naam voor een tekstverzorgend programma. Full screen wil zeggen dat tekst over het gehele beeld kan worden verzorgd.

Om wat met de MSX full screen editor te oefenen, is nog geen enkele kennis van het MSX-basic noodzakelijk. Slechts enkele gegevens zijn in dit stadium belangrijk:

1) Begin een regel tekst altijd eerst met een regelnummer dat groter is dan of gelijk is aan 0. Het regelnummer moet kleiner dan 65530 blijven.
2) Een ingetikte regel mag groter zijn dan de breedte van een beeldschermregel, maar moet kleiner blijven dan 256 tekens.
3) Een regel mag voorlopig elke volgorde van karakters bevatten.
4) Een regel dient altijd te worden beëindigd door de RETURN-toets.
5) Een op een of andere wijze veranderde regel wordt pas definitief opgenomen nadat een RETURN is ingegeven.

Een geldige tekst om in de MSX-editor mee te oefenen, is bijvoorbeeld:

```
1 IK OEFEN WAT MET DE MSX EDITOR
10 HDJS JSJS HSHSHSHS TETETE
123 IK TIK ZOMAAR WAT IN ...
```

maar elke zelf bedachte (nonsens-) tekst is uitstekend.

3.1 LIST

Nadat u een aantal regels op deze wijze heeft ingetikt, kunt u deze regels op volgorde van regelnummer weer opvragen door middel van het LIST-commando. Tikt u het woord LIST (hoofdletters of kleine letters maken hier geen verschil) in, gevolgd door de RETURN-toets. In een vaartje ziet u de regels keurig en in volgorde over het beeldscherm gaan. Maakt u zich niet ongerust over verdwijnende regels op het beeldscherm; deze zijn geregistreerd en kunnen door het list-commando weer worden opgeroepen.
Indien u slechts één regel wilt opvragen, tikt u dan LIST in, gevolgd door het betreffende regelnummer en een RETURN-toets. Een beperkte

hoeveelheid tekst kan worden opgeroepen door het LIST-commando in te toetsen, gevolgd door een eerste en een laatste regelnummer, door middel van een min-teken van elkaar gescheiden.
LIST biedt nog meer mogelijkheden, maar die zijn in het kader van dit hoofdstuk niet belangrijk.

3.2 De pijltoetsen, NEW

Op de boven beschreven wijze kunt u het beeldscherm naar believen met tekst vullen. De cursor, het blokje op het beeldscherm dat aangeeft waar u met intikken gebleven bent, loopt steeds keurig met uw ingave mee.
De cursor kunt u met behulp van de pijltoetsen in de aangegeven richtingen verplaatsen. Zo kunt u de cursor bijvoorbeeld midden in een stuk tekst zetten. Als u daarna wat letters intikt, dan komen die in de plaats van de lettertekens die eerder op die plaats stonden. Wanneer u na de verandering een RETURN-toets ingeeft, dan merkt u dat de cursor aan het begin van de volgende regel gaat staan. De door u aangebrachte wijziging is opgenomen in het programmageheugen. U kunt dit met het LIST-commando natuurlijk nog even controleren.

De pijltoetsen maken het mogelijk om op willekeurige plaatsen op het scherm wijzigingen aan te brengen en deze vervolgens permanent te maken.

Een andere mogelijkheid is het copiëren van regels. Geef bijvoorbeeld eens de tekst NEW in, gevolgd door een RETURN. Controleer daarna met een list of alle regels daadwerkelijk door de computer 'vergeten' zijn (NEW betekent NIEUW).
Tik hierna in:

```
10 DIT IS DE EERSTE REGEL VAN VELE(RETURN)
```

Ga vervolgens met de pijltoetsen naar het regelnummer 10 en maak hier een regelnummer 20 van door alleen de 1 in een 2 te veranderen. Geef vervolgens (uiteraard) een RETURN in en doe daarna een LIST. U ziet dat de regel nu twee maal is opgenomen, eenmaal onder regelnummer 10 en eenmaal onder regelnummer 20.
Nu kan regel 20 bijvoorbeeld worden aangepast door met de pijltoetsen naar de juiste plaats te gaan en de tekst EERSTE te vervangen door de tekst TWEEDE. Na RETURN is ook dezer verandering weer definitief.

Indien een regel niet meer is gewenst, kan deze worden verwijderd door alleen het regelnummer in te geven, gevolgd door een RETURN. Ook deze verwijdering kunt u natuurlijk weer controleren met de LIST-functie.

3.3 Verwijderen van tekens

Laten we nog eens een NEW-commando ingeven (gevolgd door de RETURN) en vervolgens met een SHIFT-CLS (of een CTRL-L) het beeldscherm schoonmaken. We beginnen met een schone lei. Als eerste regel tikken we in:

```
1 DIT IS DAN DE EERSTE REGEL
```

Nu gaan we om te oefenen het woordje DAN uit deze eerste regel verwijderen. Dit doen we door met de pijltoetsen naar de letter D van DAN te gaan en vervolgens vier keer de DEL-toets in te drukken. De DEL-toets dient vier keer te worden ingedrukt omdat ook het spatieteken na DAN moet worden verwijderd.

Op deze wijze kunnen stukken tekst worden verwijderd uit een regel. Na de RETURN is de verkorte regel definitief opgenomen; met een LIST kan dat worden gecontroleerd.

3.4 Invoegen van tekens

Het kan natuurlijk ook zijn dat we juist een stuk tekst willen tussenvoegen in een bestaande regel. Als oefening kunnen we bijvoorbeeld proberen om het woordje DAN weer in de regel uit het laatste voorbeeld terug te zetten.

Hiervoor zetten we met de pijltoetsen de cursor op de letter D van het woordje DE. Tik vervolgens de INS-toets in; de cursor verandert nu in een streepje.
Tik nu het woordje DAN in en vergeet de spatie na DAN niet. Geef vervolgens een RETURN om de regel definitief vast te leggen en controleer of de uitgebreide regel daadwerkelijk is opgenomen met een LIST.

Het zal blijken dat het woordje DAN netjes is tussengevoegd. Pas bij het invoegen op de volgende punten:

1) door de INS-toets nog een keer in te drukken, wordt de invoegfunctie weer opgeheven en wordt het volgende karakter weer gewoon over het oude karakter heen geplaatst.
2) een pijltoets heft de invoegfunctie eveneens op.

3.5 De overige controletoetsen

Het zal duidelijk zijn dat met de MSX-editor het intikken van programmaregels alsmede het corrigeren van deze regels een eenvoudig karwei is. Met een beperkt aantal duidelijke controletoetsen zijn vele mogelijkheden te verwezenlijken, onder andere:

– het intoetsen van een stuk tekst
– het overschrijven van een stuk tekst
– het verwijderen van een stuk tekst
– het tussenvoegen van een stuk tekst.

Behalve de nu bekende controletoetsen (RETURN, INS, DEL en de pijltoetsen) zijn er nog meer functies op te roepen die weleens gemakkelijk zijn. Eén kwamen we er al tegen: de BACKSPACE (BS) voor het corrigeren van het laatste karakter.

De overige functies laten zich slechts oproepen door middel van het intoetsen van de CTRL-toets te zamen met een ander karakter. Eén zo'n functie kwamen we reeds tegen: de CTRL-L (druk tegelijk de CTRL en de L-toets in) voor het schoonmaken van het beeld.

Hieronder volgt een schema met de overige functies van de editor. Om te oefenen kunt u het beste eerst een flink stukje tekst intikken en vervolgens één voor één de functies uitproberen.

CONTROLEFUNCTIES VOOR DE MSX-EDITOR

toets die te zamen met de CTRL-toets moet worden ingetoetst	functie van deze toets
B	de cursor gaat terug naar het begin van het vorige woord.
C	onderbreken van de ingave zonder dat de regel definitief wordt opgenomen.

E	alle tekst op de betreffende regel, rechts van de de cursor, wordt verwijderd.
F	de cursor gaat verder naar het begin van een volgend woord.
G	een kort geluidssignaal is hoorbaar, verder geen functie.
H	heeft dezelfde functie als BACKSPACE (BS).
I	heeft dezelfde functie als de TAB; de cursor verplaatst zich naar de volgende kolom. Het beeldscherm is standaard in kolommen van 16 tekens ingedeeld.
J	de cursor gaat naar de volgende regel. Eventueel wordt de beeldscherminhoud één regel opgeschoven (gebeurt niet met de pijl-naar-beneden-toets).
K	heeft dezelfde functie als HOME.
L	heeft dezelfde functie als CLS.
M	heeft dezelfde functie als RETURN.
N	de cursor gaat helemaal naar het einde van de regel.
R	heeft dezelfde functie als INS.
U	de volledige regel wordt schoongemaakt.
X	heeft dezelfde functie als SELECT (geen).
\	heeft dezelfde functie als pijl naar rechts.
]	heeft dezelfde functie als pijl naar links.
^	heeft dezelfde functie als pijl naar boven.
_	heeft dezelfde functie als pijl omlaag.
DEL	heeft dezelfde functie als DEL zonder CTRL.

Pas op: hoegenaamd alle controlefuncties hebben tot gevolg, dat een eventueel geactiveerde tussenvoeging wordt gede-activeerd. Dit kan men zien aan de cursor, die dan weer de vorm van een blokje (in plaats van een streepje) krijgt.

Het is raadzaam om ervoor te zorgen, enige ervaring met de MSX-editor te verkrijgen alvorens de MSX-studie te vervolgen.

4.1 Computertalen

De Z-80-A microprocessor, het hart van het MSX-computersysteem, 'begrijpt' slechts één taal, de machinetaal. In hele reeksen van getallen, variërend vanaf 0 tot en met 255, dient deze microprocessor te worden duidelijk gemaakt wat van hem wordt verwacht. Om een stukje machinetaal te voorschijn te halen, dient men slechts het volgende korte programma in te toetsen nadat de computer is aangezet en de Ok-melding is verschenen:

```
10 FOR I=1 TO 100
20 PRINT PEEK(I)
30 NEXT I
```

(denk aan de RETURN aan het einde van elke regel)

Na het intikken van het RUN-commando (met daarachter een RETURN) verschijnen op het beeldscherm allemaal getallen tussen 0 en 255. Deze getallen vormen de eerste honderd instructies die in machinetaal in het ROM zijn opgenomen.

Elke machine-instructie heeft een bepaalde taak. Deze taken zijn echter zo elementair dat er al een heel programma dient te worden geschreven alleen om twee getallen met elkaar te vermenigvuldigen. Het programmeren in machinetaal is een langdurig en ingewikkeld werk dat alleen voor gevorderde hobbyisten en specialisten is weggelegd. Zelfs professionele programmeurs zijn meestal niet in staat om een programma in machinetaal te schrijven.

Om er nu voor te zorgen dat de computer vriendelijker is te benaderen zonder dat het noodzakelijk is dat men een specialist is in het moeilijke machinetaal-programmeren, heeft men al vrij lang geleden de zogenaamde HOGERE COMPUTERTALEN bedacht. Deze hogere computertalen stellen iemand die geen specialist is op het gebied van de computeropbouw toch in staat om vrij eenvoudig gebruik te maken van de computer. Zulke hogere computertalen zijn bijvoorbeeld het PASCAL, het ALGOL, het FORTRAN, het COBOL en zeker niet in de laatste plaats het BASIC.

Om er voor te zorgen dat een computer de door de mens bedachte

hogere programmeertaal 'begrijpt', is er een programma nodig dat deze hogere programmeertaal kan vertalen naar de elementaire machinecode die de computer 'begrijpt' en kan uitvoeren. Zo'n programma dat de vertaling van een hogere programmeertaal naar de elementaire machinetaal verricht, noemt men een INTERPRETER (interprete=betekenis geven aan) of in een ander geval een COMPILER (compile=samenstellen).

Het MSX-basic is een hogere programmeertaal die een INTERPRETER nodig heeft voor de vertaling naar machinetaal. Deze interpreter is éénmaal door zéér gespecialiseerde vakmensen van MICROSOFT samengesteld en toen opgeslagen in het niet uitwisbare gedeelte van elke MSX-computer (het ROM). Deze interpreter bestaat uit ongeveer 32000 machinecode-instructies waarvan we in het eerder genoemde kleine programma er honderd lieten zien.

Een enkele MSX-computer heeft de interpreter niet in het ROM staan. Via een eenvoudige handeling kan de MSX-interpreter dan echter van diskette of schijf worden gelezen en in het RAM (wel uitwisbaar geheugen) worden geplaatst.

4.2 BASIC

De naam van de taal BASIC is een afkorting van Beginners All-purpose Symbolic Instruction Code. Zoals de naam doet vermoeden, werd de taal BASIC in eerste instantie ontworpen voor beginners op het gebied van de computerwereld. Sinds de introductie van BASIC is er door diverse software-ontwerpers een keur van zogenaamde dialecten ontworpen zodat het nu niet meer mogelijk is om van een standaard BASIC te spreken. De tegenwoordige BASIC-dialecten zijn vaak zeer uitgebreid en lijken niet meer op het eerste, eenvoudige BASIC waar alle, nu meer dan 300 verschillende, dialecten van af stammen.

Het MSX-basic is één van de vele BASIC-dialecten. Echter door de bijzondere kwaliteit van dit BASIC en de ruime mate waarmee het door diverse computerfabrikanten wordt toegepast, kan het ondertussen wel de standaard BASIC voor micro-computers worden genoemd.

Het MSX-basic bevat meer dan 150 sleutelwoorden (het eerste BASIC bevatte slechts een tiental sleutelwoorden...). Met deze sleutelwoorden kunnen commando's worden gevormd waarmee de MSX-computer wordt opgedragen om te rekenen, te tekenen, geluid te maken, tekst

af te drukken etcetera.

MSX betekent: MicroSoft eXtended basic ofwel 'het uitgebreide basic van Microsoft'. Uit de naamgeving blijkt wel dat de taal niet nieuw is; het normale Microsoft basic is al jarenlang de meest bekende BASIC op de wat professionelere micro-computer.

Echter, met de ontwikkeling van de kleine microcomputer, de zogenaamde huiscomputer of home-computer, groeide de noodzaak om dit BASIC aan te passen aan de moderne mogelijkheden en toe te passen op de kleine microcomputers. Uit deze noodzaak ontstond het uitgebreide Microsoft basic ofwel het MSX-basic; een BASIC waarin ervaring van vele jaren is verwerkt en waarin de meest moderne mogelijkheden zijn ondervangen.

4.3 Commando's

Zoals reeds werd opgemerkt, heeft MSX-basic ongeveer 150 sleutelwoorden waarmee commando's kunnen worden geformuleerd. Indien de formulering van deze commando's juist geschiedt, zal de computer u perfekt gehoorzamen. Indien echter een klein foutje in deze formulering zit, zal de computer u niet begrijpen en reageren met een foutmelding.

Om alvast een beetje de smaak te pakken te krijgen van het programmeren in MSX-basic worden hieronder enkele eenvoudige sleutelwoorden behandeld. Aan het einde van deze paragraaf komen we tot de conclusie dat we het eerste echte computerprogramma in MSX-basic hebben geschreven.

Tik met behulp van de ondertussen vertrouwd geworden MSX-editor eens de volgende regel in:

```
PRINT "HALLO ALLEMAAL"
```

Nadat de RETURN-toets is ingedrukt, zal de computer netjes de tekst "HALLO ALLEMAAL" afdrukken om vervolgens met de Ok-melding aan te geven dat hij klaar is.

We droegen de computer (PRINT betekent AFDRUKKEN) op om een tekst af te drukken; de computer gehoorzaamde ons feilloos.

Probeer ook de volgende regel eens:

```
PRINS "HALLO ALLEMAAL"
```

Iedere BASIC-programmeur begrijpt wat er bedoeld wordt. Echter, de computer, tenslotte maar een dom instrument, begrijpt niet dat het sleutelwoord eigenlijk PRINT had moeten zijn en geeft een syntax error (=fout in schrijfwijze).

Hieruit zal één ding duidelijk zijn:

DE COMPUTER IS EEN DOM INSTRUMENT EN ALS ZODANIG NIET IN STAAT OM FOUTEN TE MAKEN. DE COMPUTER GEHOORZAAMT U FEILLOOS WANNEER U IN STAAT BENT OM DE OPDRACHTEN PRECIES VOLGENS DE VOORSCHRIFTEN IN ZIJN TAAL TE FORMEREN.

We zien dat we met de regel PRINT "HALLO ALLEMAAL" een eerder gestelde regel in verband met de editor hebben overtreden; we hebben voor deze regel namelijk geen regelnummer geplaatst. Echter, geen foutmelding volgde; de computer begreep onze bedoelingen en voerde het commando feilloos uit.

Wanneer we de regel PRINT "HALLO ALLEMAAL" wèl voorzien van een regelnummer, kunnen we deze regel later met het reeds bekende LIST-commando weer zichtbaar maken. Echter, na het intikken van deze regel voert de computer het commando niet meer uit.

Wanneer we het hele kleine programma dat we nu in feite hebben geschreven, willen laten uitvoeren door de computer, dan moeten we het commando RUN gebruiken. Na dit commando zal de tekst "HALLO ALLEMAAL" weer keurig worden afgedrukt. We kunnen dit enige malen herhalen door steeds weer het RUN-commando te gebruiken. Merk op dat het veel gebruikte RUN-commando standaard onder de vijfde functietoets aanwezig is en het LIST-commando onder de vierde toets kan worden gevonden.

We kunnen het nu ontstane programma uitbreiden met nog een regel:

```
10 PRINT "HALLO ALLEMAAL"
20 PRINT "HOE GAAT HET ER MEE"
RUN
```

```
HALLO ALLEMAAL
HOE GAAT HET ER MEE
Ok
```

We zien dat na het RUN-commando de twee opgedragen teksten keurig onder elkaar worden afgedrukt. Blijkbaar werkt de computer de ingetoetste programmaregels op volgorde van regelnummer af totdat alle regels 'op' zijn. We kunnen het programma op deze wijze willekeurig groter maken en steeds weer laten uitvoeren. Een leuk grapje kan men uithalen door het programma als volgt uit te breiden:

```
10 PRINT "HALLO ALLEMAAL"
20 PRINT "HOE GAAT HET ER MEE"
30 GOTO 10
RUN
```

Na het commando RUN blijft de opgedragen tekst achter elkaar worden afgedrukt op het beeldscherm. Het commando GOTO 10 zorgt ervoor, dat wanneer de computer bij regel 30 is aangekomen, hij weer 'teruggaat' naar regel 10 waardoor het programma geen einde kan vinden. Nogmaals blijkt dat de computer een dom instrument is dat de moeilijkste maar ook de domste en meest onzinnige opdrachten zonder protest uitvoert. De enige voorwaarde is, dat de juiste formulering is gebruikt.

Het steeds maar ronddraaiende programma kan door de STOP-toets tijdelijk worden onderbroken; links onder in het beeld verschijnt dan de cursor. Door de STOP-toets een tweede maal aan te raken, gaat het programma weer verder. Definitief kan het programma worden onderbroken door de toetsen CTRL en STOP tegelijkertijd in te drukken. Met een LIST kunnen we dan het programma weer te voorschijn halen.

Vervang nu regel 10 eens door:

```
10       PRINT"HALLO ALLEMAAL"
```

(denk aan de spaties)

en probeer het programma nog eens.
Conclusie: het MSX-basic is niet erg gevoelig voor spaties;het programma werkt nog precies hetzelfde. We hadden dus net zo goed:

```
10PRINT"HALLO ALLEMAAL"
```

(helemaal geen spaties)

kunnen intikken.
Ook wanneer we:

```
10print "HALLO ALLEMAAL"          (kleine letters)
```

intikken, zien we dat het programma nog steeds uitstekend werkt. We kunnen concluderen dat het gebruik van kleine of grote letters bij sleutelwoorden niets uitmaakt. Bij een **LIST** zal zelfs blijken dat de computer het woordje print zelf in hoofdletters heeft veranderd.

Het zal duidelijk zijn dat het programma wel anders gaat werken wanneer we de tekst tussen de aanhalingstekens in kleine letters gaan zetten.

We kunnen vaststellen dat:

- sleutelwoorden met kleine en grote letters mogen worden geschreven.
- sleutelwoorden aan elkaar dienen te worden geschreven.
- er tussen aanhalingstekens precies de tekst dient te worden opgenomen die ook moet worden gebruikt.
- er verder naar believen spatietekens mogen worden gebruikt.

In deze paragraaf kwamen we slechts enkele van de ruim 150 mogelijke sleutelwoorden tegen. In de volgende hoofdstukken worden eerst wat broodnodige begrippen behandeld waarna we in de hoofdstukken 9 en 10 worden geconfronteerd met **alle** bestaande sleutelwoorden met hun preciese schrijfwijzen en betekenissen. Alhoewel de principes van het programmeren vrij eenvoudig zijn, zal het door het grote aantal sleutelwoorden toch een behoorlijke oefening vergen voordat u het MSX-basic volledig beheerst.

5 CONSTANTEN

Om het MSX-basic goed te kunnen beheersen, is begrip nodig van enkele algemene termen. In dit hoofdstuk wordt de term CONSTANTE behandeld.

5.1 Wat is een constante

De definitie van een constante is vrij eenvoudig:

EEN CONSTANTE IS EEN WAARDE DIE NIET
AAN VERANDERING ONDERHEVEN IS.

Wanneer we het over waarden hebben, hebben we het niet alleen over numerieke waarden (waarden die een hoeveelheid aangeven) maar ook over alfanumerieke waarden (tekstwaarden, teksten of STRINGS).

In het volgende voorbeeld zijn enkele constanten genoemd:

123	22.5	JANSSEN
0.124	−100000	DIT IS EEN TEKST
BOEKJE	ABCDEFGHIJ	10203.4

Merk op dat we in de computerwereld in plaats van de decimale komma altijd een punt gebruiken.

In het vorige hoofdstuk zagen we reeds, dat we alfanumerieke constanten door de computer kunnen laten afdrukken. In het reeds eerder genoemde programma:

```
10 PRINT "HALLO ALLEMAAL"
20 PRINT "HOE GAAT HET ERMEE"
30 GOTO 10
```

worden op programmaregel 10 en 20 de alfanumerieke constanten HALLO ALLEMAAL en HOE GAAT HET ERMEE afgedrukt op het beeldscherm. *Alfanumerieke constanten dienen altijd tussen aanhalingstekens (") te worden vermeld in een MSX-basicprogramma.*

Numerieke constanten behoeven *niet* tussen aanhalingstekens te worden

vermeld. Indien numerieke constanten toch tussen aanhalingstekens worden geplaatst, noemen we deze constanten niet meer numeriek maar ALFANUMERIEK. Het volgende programma:

```
NEW                          (eerst oude programmaregels verwijde-
10 PRINT 125                 ren)
20 PRINT "125"
```

zal, wanneer het door middel van het commando RUN in werking wordt gezet, twee maal het getal 125 op het beeldscherm afdrukken. Toch noemen we de constante op regel 10 een numerieke constante en de constante op regel 20 een alfanumerieke constante.

5.2 Rekenen met constanten

We kunnen de computer gebruiken als rekenmachine door hem bijvoorbeeld constanten bij elkaar te laten optellen. Tik bijvoorbeeld eens in:

```
PRINT 123+200
```

De computer zal na het intoetsen van de RETURN-toets onmiddellijk met het antwoord, 323 reageren. Ook moeilijkere sommen maakt de computer zonder problemen. Bijvoorbeeld:

```
NEW
10 PRINT 12.445+123.543
20 PRINT 12/6.5
30 PRINT 123*456             (eerst de oude programma-
40 PRINT 1000-1              regels verwijderen)
```

Dit voorbeeldprogramma zal na het RUN-commando in een oogwenk de uitkomsten van de opgegeven sommen op het beeldscherm laten zien. Merk op dat het sterretje(*) als vermenigvuldigingsteken en de schuine streep(/) als deelteken worden gebruikt.

Het rekenen gaat natuurlijk alleen met NUMERIEKE constanten. Wanneer we

```
PRINT "JAN"-"PIET"
```

intoetsen, dan zal de computer antwoorden met een foutmelding; de

berekening is niet uit te voeren.

Indien we echter de regel:

```
PRINT "HALLO ALLEMAAL"+"HOE GAAT HET
ER MEE"
```

intoetsen, dan zal de computer de uitkomst HALLO ALLEMAAL HOE GAAT HET ERMEE afdrukken. Blijkbaar kan de computer twee alfanumerieke constanten wel *optellen.* Conclusie:

DE COMPUTER KAN REKENEN MET NUMERIEKE CONSTANTEN. ALFANUMERIEKE CONSTANTEN KUNNEN SLECHTS WORDEN OPGETELD; HET RESULTAAT IS DAN DE AANEENSCHAKELING VAN DEZE CONSTANTEN.

Aan de ingewikkeldheid van een opgedragen som zijn hoegenaamd geen grenzen. De opdracht:

```
PRINT 12*(2/3.5)-11*(12345+1.22)/13
```

zal door de computer in een fractie van een seconde worden uitgevoerd. Ook de opdracht:

```
PRINT "ABCDEFGH"+"IJKLMNOPQ"+"RSTUVWXYZ"
```

wordt onmiddellijk uitgevoerd, waarbij de uitkomst wordt gevormd door het alfabet.

5.3 Soorten van constanten

Het MSX-basic kent diverse soorten van constanten. Sommige soorten constanten bieden geen enkele moeilijkheid, andere zijn zeer lastig te begrijpen.

Een eerste onderverdeling van constanten zagen we reeds: de onderverdeling tussen NUMERIEKE en ALFANUMERIEKE constanten.

Van alfanumerieke constanten, die we vaak STRING-CONSTANTEN (string betekent keten) noemen, is er maar één soort. Van numerieke constanten zijn er echter meerdere soorten die we per stuk in de volgende paragrafen zullen behandelen.

5.4 Integere constanten

De integere constante is een eenvoudige vorm van numerieke constanten. Een integere constante dient te voldoen aan de volgende eisen:

- de integere constante mag niet kleiner zijn dan –32768
- de integere constante mag niet groter zijn dan 32767
- de integere constante mag geen decimalen en ook geen decimale punt bevatten

Om duidelijk te laten uitkomen dat een bepaalde constante een integere constante is, moet achter deze constante een procent-teken worden gezet. Enkele voorbeelden:

goede voorbeelden van integere constanten	foute voorbeelden integere constanten
32767%	32768
–32768%	–32769
12%	12.%
123%	–11.33

Een speciale integere constante wordt gevormd door het regelnummer dat voor een programmaregel dient te worden geplaatst. Bij de behandeling van de sleutelwoorden zal deze integere constante steeds weer als type 1 integere constante worden aangeduid. Een type 1 integere constante of regelnummer mag niet kleiner zijn dan 0 en niet groter dan 65529. Verder gelden de normale regels voor een integere constante ook voor een type 1 integere constante.

5.5 Constanten met enkelvoudige precisie

De constante met enkelvoudige precisie heeft een veel grotere vrijheid in het gebruik. De volgende eisen worden aan een constante met enkelvoudige precisie gesteld:

- de minimum waarde voor een constante met enkelvoudige precisie is $-9.99... \times 10^{62}$
- de maximum waarde voor een constante met enkelvoudige precisie is $9.99... \times 10^{62}$
- de grootst mogelijke waarde kleiner dan 0 voor een constante met enkelvoudige precisie is -10^{-64}

– de kleinst mogelijke waarde groter dan 0 voor een constante met enkelvoudige precisie is 10^{-64}
– een constante met enkelvoudige precisie heeft een precisie (significantie) van 6 cijfers

Om het voorgaande te begrijpen dienen, voor sommige lezers ten overvloede, enkele zaken te worden uitgelegd.

De notatie 10^5 staat voor het getal 10 maal 10 maal 10 maal 10 maal 10 = 100000.

In deze notatie geven we in het grondtal (het eerste getal) aan welk getal dient te worden vermenigvuldigd. In de macht (het hoger geplaatste getal) geeft men aan, hoe vaak de vermenigvuldiging dient plaats te vinden. Dus

10^5

– heeft een grondtal, gelijk aan 10
– heeft een macht, gelijk aan 5
– wordt uitgesproken als *'tien tot de vijfde macht'*
– is het zelfde als 100000 (een 1 met *vijf* nullen)

Het zal duidelijk zijn dat het getal 10^{63} (een 1 met drieënzestig nullen erachter!) een bijzonder groot getal is.

Indien de macht gelijk is aan 1 of 0 of kleiner is dan 0, dan wordt het bovenstaande problematisch. Immers, een herhaalde vermenigvuldiging kan dan niet meer worden uitgevoerd. We spreken af:

– een macht gelijk aan 1 mag worden weggelaten. $10^1=10$
– een macht gelijk aan 0 heeft tot gevolg dat de waarde van de betreffende constante altijd gelijk is aan 1.

Dus $10^0=1$ maar ook 2^0 en 12345^0 zijn gelijk aan

$$\frac{1}{\text{deze constante, maar dan met positieve macht}}$$

10^{-20} is dus gelijk aan $1/10^{20} = 1/100000000000000000000 = 0.00000000000000000001$

Het zal duidelijk zijn dat het getal 10^{-64} een bijzonder klein getal is.

Bij wetenschappelijke notaties is het vaak gebruikelijk om de zogenaamde exponentiële notatie toe te passen. Indien een chemicus het over 1000 gram natriumchloride heeft, dan zal hij noteren:

1.0×10^3 g Natriumchloride.

De notatie 1.0×10^3 bevat twee elementen. Het eerste element (1.0) noemen we de mantisse. Het tweede element (10^3) noemen we de exponent.

Indien we de notatie uitwerken, blijkt het volgende:

$1.0 \times 10^3 = 1.0 \times 10 \times 10 \times 10 = 1000$

De chemicus die deze notatie op papier zette, bedoelde net als wij dus gewoon 1000 gram natriumchloride. Echter, hij noteerde meer dan alleen dat:

- met de notatie 1.0 geeft hij een betrouwbaarheid, precisie of significantie aan van de hoeveelheid. Met 1.0 wordt aangegeven dat slechts de eerste twee cijfers van deze notatie betrouwbaar zijn. Het zou net zo goed om 980 gram of om 1032 gram kunnen gaan, maar het is zeker geen 250 of 1080 gram.

- met de notatie 10^3 geeft hij de grootte-orde aan. Het gaat in dit geval om een aantal duizenden (in dit geval één maal duizend) grammen.

Omdat deze exponentiële notatie vooral bij grotere of kleinere getallen het voordeel biedt dat het een korte notatie is (probeer het getal 1.2×10^{60} maar eens gewoon uit te schrijven...) heeft Microsoft deze notatie ook in het MSX-basic mogelijk gemaakt. De notatievorm moest echter natuurlijk enigszins aan de mogelijkheden van de computer worden aangepast.

De constante 1.0×10^3 wordt in MSX-basic genoteerd als 1.0E3 of 1.0D3. Zo wordt 1.243×10^{16} bijvoorbeeld genoteerd als 1.243E16 of 1.1234D16 en wordt 3.14×10^{-33} genoteerd als 3.14E–33 of 3.14D–33.

Achter de mantisse wordt in de MSX-notatie (zoals in veel andere BASIC-talen) eerst een D of een E geplaatst waarna de macht van de exponent wordt opgenomen. Het grondtal van de exponent is bij deze

notatie altijd gelijk aan 10 en wordt in de notatie weggelaten.

Voor de duidelijkheid is het raadzaam om met deze notatie eens wat te experimenteren op de computer. Probeer bijvoorbeeld eens een programma in de volgende vorm te schrijven en kijk eens wat er na een RUN gebeurt. Reken de uitkomsten na.

```
NEW
10 PRINT 1.2E-3
20 PRINT 5.22E-33*1E44/2E0
30 print 123D22/23D21-10
...etcetera
```

Lees nu nog eens de bepalingen voor een constante met enkelvoudige precisie na. Met de hier behandelde stof zullen deze bepalingen nu duidelijk zijn.

Om nu aan te geven, dat een constante een constante met enkelvoudige precisie is,

- moet achter de constante een uitroepteken (!) worden geplaatst of:
- moet het exponentgedeelte, indien aanwezig, de exponentaanduiding E bevatten. De exponentaanduiding D is voor deze constanten verboden.

Het zal duidelijk zijn dat een constante met enkelvoudige precisie niet verplicht in de exponentiële notatie behoeft te zijn genoteerd. Enkele voorbeelden:

goede voorbeelden van constanten met enkelvoudige precisie	foute voorbeelden van constanten met enkelvoudige precisie
1234.5!	12%
1234!	1.234567D33
1.1E28	–12D–33
–32769!	1111111

5.6 Constanten met dubbele precisie

Het enige verschil tussen constanten met dubbele precisie en constanten met enkelvoudige precisie is, dat een constante met dubbele precisie

geen precisie van 6 cijfers maar een precisie van 14 cijfers heeft.

Om aan te geven dat een constante een dubbele precisie heeft,
- moet achter de constante een hekje (#) worden geplaatst, of:
- moet het exponentgedeelte, indien aanwezig, de exponentaanduiding D bevatten. De exponentaanduiding E is voor deze constante *verboden*.

Enkele voorbeelden:

goede voorbeelden van constanten met dubbele precisie	foute voorbeelden van constanten met dubbele precisie
1234.5#	1234.5!
12D22	12E22
123#	123%
12D–60	1!

5.7 Bij twijfel...

Het is niet verplicht om bij een constante een achtervoegsel (#,% of !) te gebruiken. Bij sommige constanten is het dan echter niet precies uit te maken of het nu een integere constante, een constante met enkelvoudige precisie of een constante met dubbele precisie is. Bij de exponentiële notatie bestaat de twijfel nooit; uit de in de exponent genoteerde letter (D of E) is zondermeer het type constante af te leiden.

Indien het MSX-basic het verschil niet precies kan bepalen, dan maakt het om misverstanden te voorkomen, altijd een constante met dubbele precisie van de constante waarover wordt getwijfeld. Dit is een veilige beslissing van het MSX-basic; berekeningen kunnen hierdoor moeilijk verkeerd uitpakken.

5.8 Constanten in andere talstelsels

In MSX-basic is het ook mogelijk om constanten te formuleren in andere talstelsels. We onderscheiden binaire (tweetallige), octale (achttallige) en hexadecimale (zestientallige) constanten.

Om met deze constanten te werken, is kennis van de diverse talstelsel

noodzakelijk. Het werken met deze vormen van constanten is echter voor de meeste programmeurs niet interessant. Slechts in een enkel geval is het plezierig om met binaire, hexadecimale of octale constanten te kunnen werken.

Het is zeker voor de beginnende programmeur maar ook voor de wat gevorderde programmeur die hier geen noodzaak toe ziet, verstandig om de rest van dit hoofdstuk over te slaan.

5.8 Het principe van talstelsel

Het talstelsel waarin wij normaal werken, is het decimale talstelsel. Het decimale talstelsel of tientallige talstelsel is zo genoemd omdat dit talstelsel tien cijfertekens kent (0,1,2...9). Wanneer we in het decimale talstelsel gaan tellen vanaf 0, dan komen we tot de conclusie dat we het tiende getal in twee cijfers moeten gaan noteren. Het honderdste getal dient in drie cijfers te worden genoteerd etcetera. Een decimaal getal kan als volgt worden ontleed:

Bijvoorbeeld het getal 4264:

4	2	6	4
aantal maal grondtal 10^3	aantal maal 10^2	aantal maal 10^1	aantal maal 10^0

Het getal 4264 is op deze wijze gelijk aan

$$\begin{aligned} 4 \times 10 \times 10 \times 10 &= 4000 \\ 2 \times 10 \times 10 &= 200 \\ 6 \times 10 &= 60 \\ 4 &= 4+ \\ &\quad\overline{\quad\quad\quad\quad} \\ &\quad 4264 \end{aligned}$$

Het tweetallige stelsel kent geen tien maar slechts twee cijfertekens. Deze cijfertekens zijn 0 en 1. Een tweetallig of binair getal laat zich als volgt ontleden:

Bijvoorbeeld het binaire getal 11010:

1	1	0	1	0
aantal maal 2^4 grondtal	aantal maal 2^3	aantal maal 2^2	aantal maal 2^1	aantal maal 2^0

Het binaire getal 11010 is dus gelijk aan 1 x 2 x 2 x 2 x 2 = 16
1 x 2 x 2 x 2 = 8
0 x 2 x 2 = 0
1 x 2 = 2
0 = 0+
26

Merk op dat in het binaire stelsel *elke* waarde kan worden genoteerd. Een nadeel is, dat voor de notatie van een relatief klein getal al erg veel posities nodig zijn. Bijvoorbeeld het getal 511 moet binair al als 111111111 worden geschreven; met 9 cijfers maar liefst!

Het tweetallige of binaire stelsel is het stelsel waarin de computer zelf rekent. Wanneer een computergeheugen tot op het laatste element wordt ontleed, dan zal blijken dat de werkelijke opslag van gegevens in dit geheugen tweetallig geschiedt. De geheugenelementen waarin alleen het getal 0 of 1 kan worden opgeslagen, noemt men BITS (afkorting van binary digits=tweetallige cijfers). Om deze reden is het vaak voordelig om in ieder geval het binaire stelsel te beheersen.

Het achttallige of octale stelsel kent voor de notatie van waarden acht cijfertekens, namelijk de cijfers 0,1,2,3,4,5,6 en 7. Een octaal getal laat zich als volgt ontleden:

Bijvoorbeeld het octale getal 7761:

7	7	6	1
aantal maal grondtal 8^3	aantal maal 8^2	aantal maal 8^1	aantal maal 8^0

Het octale getal 7761 is dus gelijk aan $7 \times 8 \times 8 \times 8 = 3584$
$7 \times 8 \times 8 = 448$
$6 \times 8 = 48$
$1 = 1+$
4081

Merk op dat ook in het octale stelsel *elke* waarde kan worden genoteerd.

Het zestientallige of hexadecimale stelsel kent voor de notatie van waarden maar liefst 16 cijfertekens. Omdat in ons normale cijfertekenstelsel slechts tien cijfers voorkomen, moeten er voor notatie van hexadecimale waarden dus zes cijfertekens bij gemaakt worden. Men koos voor de zes tekortschietende cijfers als symbolen de letters A tot en met F. Dus:

het hexadecimale cijfer	heeft decimaal gezien de waarde
0	0
1	1
.	.
.	.
.	.
9	9
A	10
B	11
C	12
D	13
E	14
F	15

Een hexadecimaal getal laat zich als volgt ontleden:

Bijvoorbeeld het getal FE81:

F	E		8	1
aantal maal grondtal 16^3	aantal maal 16^2	aantal maal 16^1	aantal maal 16^0	

Het hexadecimale getal FE81 is dus gelijk aan:

```
15(F) x 16 x 16  x  16 =   61440
14(E) x 16 x 16        =    3584
 8    x 16             =     128
        1               =       1+
                          -------
                          65153
```

Ook in het hexadecimale stelsel kan *elke* waarde worden genoteerd.

Het binaire, het hexadecimale en het octale stelsel zijn de drie belangrijkste stelsels in de computerwereld buiten natuurlijk het decimale stelsel. Alhoewel de theorie van de verschillende talstelsels niet zo erg moeilijk is, is het werken en rekenen met deze talstels vaak een groot probleem.

5.10 Binaire constanten

In het MSX-basic kan een binaire constante worden geformuleerd. De notatie dient dan te worden voorafgegaan door het teken & en de letter B. Probeer bijvoorbeeld eens het volgende programma:

```
NEW
10 PRINT &B10110
20 PRINT &B11110
```

Wanneer dit programma door een RUN wordt geactiveerd, zal het programma keurig worden uitgevoerd; de (decimale) waarde van de betreffende binaire constanten worden keurig afgedrukt.

Een binaire constante dient aan de volgende voorwaarden te voldoen:

- de constante dient door &B te worden voorafgegaan.
- de constante mag alleen de cijfers 0 en 1 bevatten.
- de constante mag maximaal gelijk aan &B1111111111111111 zijn (65535 decimaal).

Pas op, indien de decimale waarde van de binaire constante groter is dan 32767, dan trekt MSX-basic automatisch de waarde 65536 af van de waarde voordat deze wordt afgedrukt. Hierdoor blijft de uitkomst altijd tussen de waarden –32768 en 32767 waardoor de uitkomst altijd een integere waarde is. De meer doorgewinterde programmeur zal hierin het tweecomplementsysteem herkennen.

5.11 Octale constanten

In het MSX-basic kan een octale constante worden geformuleerd. De notatie dient dan te worden voorafgegaan door het teken & en de letter O. Probeer bijvoorbeeld eens het volgende programma:

```
NEW
10 PRINT &0177
20 PRINT &01752
```

(Pas op voor het verschil tussen de letter O en het cijfer Ø)

Wanneer dit programma wordt uitgevoerd, worden keurig de decimale waarden van de vermelde octale constanten afgedrukt.

Een octale constante dient aan de volgende voorwaarden te voldoen:

- de constante dient door &O te worden voorafgegaan.
- de constante mag alleen de cijfertekens 0...7 bevatten.
- de constante mag maximaal gelijk zijn aan &O177777(65535 decimaal).

Pas op, indien de decimale waarde van de binaire constante groter is dan 32767, dan trekt MSX-basic automatisch de waarde 65536 af van de waarde voordat deze wordt afgedrukt. Hierdoor blijft de uitkomst altijd tussen de waarden –32768 en 32767 waardoor de uitkomst altijd een integere waarde is. De meer doorgewinterde programmeur zal hierin het tweecomplementsysteem herkennen.

5.12 Hexadecimale constanten

Ook hexadecimale constanten kunnen binnen het MSX-basic worden gebruikt. De notatie dient dan te worden voorafgegaan door het teken & en de letter H. Probeer bijvoorbeeld eens het volgende programma:

```
NEW
10 PRINT &HFFFF
20 PRINT &H1023
```

Wanneer dit programma wordt uitgevoerd, worden keurig de decimale waarden voor de genoteerde hexadecimale constanten afgedrukt op beeldscherm.

Een hexadecimale constante dient aan de volgende eisen te voldoen:

– de constante dient door &H te worden voorafgegaan.
– de constante mag alleen de cijfertekens 0...9 en A...F bevatten.
– de constante mag maximaal gelijk zijn aan &HFFFF (65535 decimaal).

Pas op, indien de decimale waarde van de binaire constante groter is dan 32767, dan trekt MSX-basic automatisch de waarde 65536 af van de waarde voordat deze wordt afgedrukt. Hierdoor blijft de uitkomst altijd tussen de waarden –32768 en 32767 waardoor de uitkomst altijd een integere waarde is. De meer doorgewinterde programmeur zal hierin het tweecomplementsysteem herkennen.

6.1 Wat is een variabele

Ook de definitie van een variabele is vrij eenvoudig:

> EEN VARIABELE IS EEN WAARDE DIE (IN TEGENSTELLING TOT EEN CONSTANTE) WEL AAN VERANDERINGEN ONDERHEVIG IS.

Een variabele kan dus niet met één vaste aanduiding worden genoteerd. In plaats daarvan dienen we variabelen een naam te geven. Wanneer we bijvoorbeeld intikken:

```
WAARDE=12.5
```

Dan hebben we op dat moment aan de variabele WAARDE de constante 12.5 toegekend. Wanneer we later dan intoetsen:

```
PRINT WAARDE
```

dan zal de computer vervolgens de aan de variabele WAARDE toegekende waarde weer projecteren.

Wanneer we vervolgens intikken:

```
HOEVEELHEID=26
```

dan is op dat moment de waarde 26 aan variabele HOEVEELHEID toegekend. Tikken we nu in:

```
PRINT WAARDE+HOEVEELHEID
```

Dan zal de computer netjes de twee eerder aan de variabelen toegekende WAARDEN en HOEVEELHEID bij elkaar optellen en het resultaat op het beeldscherm laten zien.

We kunnen de waarde van een variabele ook herzien. Indien we intikken:

```
HOEVEELHEID=622.5
```

En daarna:

```
PRINT HOEVEELHEID+WAARDE
```

dan zien we, dat de variabele HOEVEELHEID een andere waarde heeft verkregen. De variabele HOEVEELHEID maar natuurlijk ook alle andere variabelen zijn dus onderhevig aan veranderingen.

Bovenstaande voorbeelden behandelen slechts één van de typen variabelen, namelijk de numerieke variabelen. We kennen echter ook een heel ander soort variabelen, namelijk de alfanumerieke variabelen.

Laten we bijvoorbeeld eens intikken:

```
LET NAAM$="GEERT-JAN"
```

Het woordje LET betekent: laat of laat zijn. LET NAAM$= "GEERT-JAN" betekent dus zoveel als laat variabele NAAM$ gelijk zijn aan GEERT-JAN. Het sleutelwoord LET zouden we ook in onze eerdere voorbeelden kunnen hebben gebruikt maar het mag in deze constructie zonder meer worden weggelaten. We hadden dus ook gewoon:

```
NAAM$="GEERT-JAN"
```

kunnen intikken.

De variabele NAAM$ is een alfanumerieke variabele. Deze heeft dan ook verplicht een dollarteken ($) als laatste teken. Alfanumerieke variabelen kunnen allerlei karakters bevatten. Net als bij alfanumerieke constanten kunnen we alfanumerieke variabelen alleen maar optellen. Probeer bijvoorbeeld eens in te tikken:

```
PRINT NAAM$
```

en

```
PRINT NAAM$+" IS MIJN NAAM."
```

In het laatste voorbeeld werden een alfanumerieke variabele en een alfanumerieke constante bij elkaar opgeteld. Het resultaat bestond uit een aaneenschakeling van deze variabele en constante.

Om het begrip variabele nog wat verder uit te diepen, hebben we de

medewerking van nog een MSX-sleutelwoord nodig, namelijk het sleutelwoord INPUT. Met dit sleutelwoord is het namelijk mogelijk om een variabele een waarde te geven door intoetsing terwijl het programma werkt. Tik bijvoorbeeld het volgende programma eens in:

```
                          (om oude programmaregels te verwij-
NEW                       deren)
10 PRINT "REKENPROGRAMMA"
20 INPUT "HOE HEET U ";NAAM$
30 PRINT "GOEDENDAG "+NAAM$+"HOE IS HET"
40 INPUT "GEEF EEN GETAL IN ";GETAL
50 PRINT GETAL+GETAL;" IS HET DUBBELE
VAN ";GETAL
60 GOTO 40
```

Zo hebben we met relatief weinig sleutelwoorden al een heel programma geschreven. In regel 10 wordt op het beeldscherm vermeld dat het hier om een rekenprogramma gaat. Niets bijzonders, slechts een eenvoudige alfanumerieke constante wordt op het beeldscherm afgedrukt.
In regel 20 komt het nieuwe sleutelwoord, INPUT, aan de beurt.
Wanneer we het programma met een RUN in werking stellen, zien we dat de tekst HOE HEET U netjes op het beeldscherm wordt afgedrukt en dat de computer daarna wacht. De computer verwacht nu namelijk van u, dat u een alfanumerieke constante intoetst en dat u, zoals altijd, deze ingave met een RETURN-toets afsluit. Uw ingave wordt in variabele NAAM$ bewaard.
In regel 30 presenteert de computer de optelling van een alfanumerieke constante, een alfanumerieke variabele en weer een alfanumerieke constante. Hierdoor begroet de computer ons hoffelijk en noemt hij ons zelfs bij de naam; een bewijs dat in NAAM$ inderdaad de eerder gepleegde ingave is geplaatst.
In regel 40 vraagt de computer weer om een ingave. Dit maal dient de ingave een numerieke constante te zijn; dat is te zien aan de naam van de variabele in het INPUT-commando. Deze naam eindigt niet op een dollarteken en dit duidt op een numerieke variabele. Indien we tijdens deze INPUT proberen, een niet geldige numerieke constante in te geven, dan geeft de computer een foutmelding en krijgen we nogmaals de kans om een correcte numerieke constante in te geven.
In regel 50 wordt de optelling van de variabele GETAL bij zichzelf afgedrukt. Op het beeldscherm verschijnt het dubbele van de ingegeven

waarde. Na deze waarde wordt de alfanumerieke constante IS HET DUBBELE VAN afgedrukt, gevolgd door de waarde van de inhoud van numerieke variabele GETAL. Uit deze programmaregel leren we meteen dat we in een PRINT-commando verschillende zaken achter elkaar aan kunnen afdrukken door deze met een punt-komma van elkaar te scheiden.
Tenslotte gaven we de computer op regel 60 het commando om weer naar programmaregel 40 terug te gaan; de computer vraagt weer om een ingave. Wanneer we de computer niet met CTRL-STOP onderbreken, zal hij voortdurend met dit programma bezig blijven.

We zien dat het gebruik van variabelen een nieuwe dimensie geeft aan de computer. Door het gebruik van variabelen kunnen we eenzelfde programma steeds van andere waarden voorzien waardoor steeds nieuwe situaties worden berekend.

6.2 Soorten van variabelen

Net als bij constanten onderscheiden we ook bij variabelen enkele verschillende typen. De eerste onderverdeling, die tussen numerieke en alfanumerieke variabelen, zagen we reeds. Van de stringvariabelen of alfanumerieke variabelen is er slechts één type. Echter, van de numerieke variabelen onderscheiden we verschillende typen die in de volgende paragrafen worden behandeld.

6.3 De integere variabele

Een integere variabele herkennen we aan het procentteken (%) dat achter de variabelenaam is vermeld. Een integere variabele kan alleen een integere waarde bevatten (zie de beschrijving van integere constanten). Wanneer we proberen om een andere dan integere waarde aan een integere variabele toe te kennen, dan wordt de waarde aangepast of volgt een foutmelding. Bij:

```
LET WAARDE%=12.5
```

zal geen foutmeling ontstaan. Echter, wanneer we via een PRINT-opdracht de variabele weer op het beeldscherm laten verschijnen, dan zal blijken dat de variabele de waarde 12 heeft aangenomen; de decimalen werden verwaarloosd en de variabele bleef als zodanig integer. Wanneer we echter intoetsen:

```
LET WAARDE%=100000
```

dan volgt een foutmelding. De maximale integere waarde is 32767; 100000 kan als waarde niet in WAARDE% worden opgenomen.

6.4 De variabele met enkelvoudige precisie

Een variabele met enkelvoudige precisie kunnen we herkennen aan het uitroepteken (!) dat achter de variabelenaam is vermeld. Een variabele met enkelvoudige precisie is aan dezelfde beperkingen gebonden als een constante met enkelvoudige precisie tot zover van toepassing. Wanneer we proberen om een waarde, afwijkend van deze bepalingen, aan een variabele met enkelvoudige precisie toe te kennen dan wordt de waarde aangepast of krijgen we een foutmelding. Wanneer we bijvoorbeeld intoetsen:

```
WAARDE!=12345678
```

dan volgt geen foutmelding. Tikken we echter vervolgens in:

```
PRINT WAARDE!
```

dan presenteert de computer de waarde 12345700. Het zevende en achtste cijfer vallen buiten de precisie van een variabele met enkelvoudige precisie. Wel rondde de computer het zesde cijfer zo af dat de waarde in WAARDE! toch zo dicht mogelijk bij de bedoelde waarde ligt.

Wanneer we intoetsen:

```
WAARDE!=1E99
```

dan geeft de computer een foutmelding; de waarde die we aan WAARDE! wilden toekennen, ligt niet binnen de voor een variabele met enkelvoudige precisie bepaalde grenzen.

6.5 De variabele met dubbele precisie

Een variabele met dubbele precisie kunnen we herkennen aan het hekje (#) dat achter de variabelenaam is geplaatst. Een variabele met dubbele precisie is aan dezelfde beperkingen gebonden als de constante

met dubbele precisie tot zover van toepassing. Wanneer we proberen om een waarde, afwijkend van deze bepalingen, aan een variabele met dubbele precisie toe te kennen dan wordt de waarde aangepast of krijgen we een foutmelding. Wanneer we bijvoorbeeld intikken:

```
WAARDE#=123456789012345
```

dan geeft de computer geen foutmelding. Tikken we echter vervolgens in:

```
PRINT WAARDE#
```

dan presenteert de computer de waarde 1.2345678901235E+15. Behalve dat de computer besloot om dit grote getal in exponentiële vorm te presenteren, werd op het vijftiende cijfer afgerond en worden slechts veertien cijfers afgedrukt.

Indien we intikken:

```
WAARDE#=1D99
```

Dan geeft de computer een foutmelding; de waarde die we aan WAARDE# wilden toekennen, viel buiten de gestelde bepalingen.

6.6 Bij twijfel...

Indien een variabelenaam geen dollarteken, hekje, uitroepteken of procentteken als laatste karakter heeft, dan is het niet meer duidelijk om wat voor soort variabele het hier gaat. De volgende regels gelden dan:

- in het normale geval neemt de computer voor zo'n waarde aan, dat het om een variabele met dubbele precisie gaat.
- indien echter door middel van een DEFINT, DEFSTR, DEFSNG of DEFDBL sleutelwoord een ander type is verbonden aan de eerste letter van de variabelenaam, dan wordt dat type automatisch aangenomen. Zie hiervoor de behandeling van de DEFSTR, DEFINT, DEFSNG en DEFDBL sleutelwoorden in hoofdstuk 9.

6.7 Waarschuwing

Aan de lengte van een variabelenaam is hoegenaamd geen beperking. Een variabele kan zowel W als AANTALGESCOORDEPUNTEN heten. Het is echter raadzaam om korte, duidelijke variabelenamen te kiezen.

Een andere factor komt nog om de hoek kijken. En omdat deze factor al zeer vaak aanleiding heeft gegeven tot dagen zoekwerk naar een vermeende programmafout door radeloze programmeurs (en niet alleen beginnelingen...), zetten we de volgende waarschuwing in grote letters neer:

> HOE LANG DE VARIABELENAAM OOK IS, DE COMPUTER 'KIJKT' ALLEEN NAAR DE EERSTE TWEE LETTERS VAN DIE VARIABELENAAM!!!

Dat bektekent bijvoorbeeld dat de variabele AA en de variabele AANTAL één en dezelfde variabele is. Om dit te bewijzen, kunt u het volgende kleine programma intoetsen:

```
NEW
10 INPUT "GEEF EEN GETAL IN ";AANTAL
20 PRINT "VARIABELE AANTAL=";AANTAL
30 PRINT "EN VARIABELE AA=";AA
40 STOP
```

Dit programma bewijst onomstotelijk dat slechts de eerste twee letters belangrijk zijn. Er is daarom zelfs wel wat voor te zeggen om er een gewoonte van te maken om variabelenamen nooit groter te maken dan twee letters, eventueel gevolgd door een dollarteken, een hekje, een uitroepteken of een procentteken.

6.8 Array-variabelen

Het kan voorkomen dat we meerdere waarden onder één en dezelfde variabelenaam wilen bewaren. In dat geval zullen we behalve de variabelenaam ook dienen aan te geven welke waarde van alle waarden die we hieronder hebben opgeslagen we bedoelen. Een variabele waaronder we meer dan één waarde willen bewaren, kunnen we vergelijken met een tabel. We noemen zo'n variabele een array-variabele of kortweg een array (rij).

Zo'n array dienen we een bepaalde grootte (een bepaald aantal mogelijkheden tot opslag van waarden) toe te kennen. Dit doen we met behulp van het sleutelwoord DIM. We gaan een programma schrijven:

```
NEW
10 DIM TABEL(3)
```

Wanneer we dit programma in werking stellen, dan wordt een variabele, genaamd TABEL, toegewezen. Deze variabele biedt mogelijkheid tot opslag van *vier* waarden, in dit geval numerieke waarden met dubbele precisie. Deze *vier* waarden worden opgeslagen onder:

TABEL(0)	...eerste waarde
TABEL(1)	...tweede waarde
TABEL(2)	...derde waarde
TABEL(3)	...vierde waarde

In het verdere programma kunnen we deze tabel gaan vullen:

```
20 LET TABEL(0)=12
30 LET TABEL(1)=96
40 LET TABEL(3)=-1E22
50 LET TABEL(2)=TABEL(3)*TABEL(1)/TABEL
(0)
60 PRINT TABEL(0),TABEL(1),TABEL(2),TAB
EL(3)
```

Wanneer we dit programma laten werken via een RUN, dan merken we na enige narekening dat de variabele TABEL inderdaad vier afzonderlijke waarden bevat die we kunnen veranderen en waarmee we kunnen rekenen.

In programmaregel 60 zien we overigens, dat bij een PRINT-commando de afzonderlijke gegevens met een komma van elkaar mogen worden gescheiden. We zagen zoiets al in paragraaf 6.1, maar daar gebruikten we een punt-komma. Het gebruik van een komma heeft tot gevolg dat de gegevens ietwat uit elkaar, zo mogelijk op dezelfde regel worden afgedrukt.

We kunnen het programma uitbreiden met het volgende gedeelte:

```
60 INPUT "WELKE TABELWAARDE ";NUMMER
```

```
70 PRINT "WAARDE:";TABEL(NUMMER)
80 GOTO 60
```

(de oude regel 60 vervalt hiermee)

Wanneer we dit programma in werking stellen, vraagt de computer ons op regel 60, welk tabelelement we willen zien. We moeten dan een waarde 0,1,2 of 3 ingeven om geen foutmelding te krijgen. Wanneer we een waarde ingeven, wordt deze in NUMMER opgeslagen. Op programmaregel 70 wordt dan vervolgens het tabelelement met het zojuist ingegeven nummer op het beeldscherm getoond. We zien dat we het bedoelde tabelelement niet alleen met een getal maar ook met een andere variabele kunnen aanduiden (de professionele programmeur spreekt niet over het aanduiden maar over het *adresseren* van een tabelelement).

Een array-variabele kan in meer dan één richting bepaald zijn. Het volgende programmavoorbeeld geeft een voorbeeld van het werken met een tweedimensionale tabel of array. Merk op dat er tevens meerdere commando's op één programmaregel zijn geplaatst. Dit is toegestaan indien deze commando's met een dubbele punt van elkaar zijn gescheiden.

De eerste programmaregel bevat een REM-regel. REM is een afkorting van REMARK (=opmerking). Met deze regel wordt slechts een stukje commentaar in het programma opgenomen dat door de computer wordt genegeerd.

```
NEW
10 REM VOORBEELD TWEE-DIMENSIONALE TABEL
OF ARRAY
20 DIM TB%(2,2)
30 TB%(0,0)=12:TB%(0,1)=345:TB%(0,2)=-22
40 TB%(1,0)=99:TB%(1,1)=871:TB%(1,2)=5
50 TB%(2,0)=TB%(0,0)+TB%(1,0):TB%(2,1)=T
B%(0,1)+TB%(1,1)
60 TB%(2,2)=TB%(0,2)+TB%(1,2)
70 PRINT TB%(0,0),TB%(0,1),TB%(0,2)
80 PRINT TB%(1,0),TB%(1,1),TB%(1,2)
```

```
90 PRINT TB%(2,0),TB%(2,1),TB%(2,2)
100 PRINT "EINDE":STOP
```

Aan het einde van het programma kan men tabel TB% als volgt zien opgebouwd:

eerste adressering

	0	1	2
0	12	99	111 (12+99)
1	345	871	1216 (345+871)
2	–22	5	–17 (–22+5)

Alhoewel tabellen met drie, vier, vijf of meer dimensies moeilijk voorstelbaar zijn, zijn deze binnen het MSX-basic toch mogelijk. Een constructie als:

```
60 DIM TABEL%(3,4,7,8)
70 LET TABEL%(0,1,3,8)=23
```

werkt perfekt en kan in latere, ingewikkelde programma's een uitkomst zijn.

Tot slot van deze paragraaf merken we op, dat tabellen of arrays niet alleen met numerieke maar ook met alfanumerieke variabelen zijn toegestaan. Zo kan in een tabel die met:

```
DIM ADRES$(100,3)
```

is toegewezen, de naam, straat, woonplaats en telefoonnummer van 101 kennissen worden opgeslagen.

6.9 Opslagruimte van variabelen

De computer dient van variabelen de geldende waarden te onthouden.

Dit doet de computer door deze waarden in het geheugen op te slaan. Dit computergeheugen is beperkt. Daarom is het, vooral bij grote programma's, zaak om erop te letten of waarden, opgeslagen in variabelen met dubbele precisie, niet 'passen' in variabelen met enkelvoudige precisie of misschien wel in integere variabelen.
Integere variabelen nemen tussen de verschillende typen relatief de minste geheugenruimte in. Hun mogelijkheden zijn dan ook maar beperkt. Variabelen met enkelvoudige precisie nemen wat meer geheugenruimte in en bieden daarvoor in de plaats veel minder beperkingen. Variabelen met dubbele precisie nemen het meeste ruimte in; hun mogelijkheden zijn dan ook welhaast onbeperkt.

De geheugenruimte van een computer, we zagen dat al eerder, wordt uitgedrukt in een aantal bytes. Een byte is een geheugenelement waarin ruw gezegd één karakter of twee cijfers van een numerieke variabele kunnen worden opgeslagen. Op de manier waarop deze gegevens worden opgeslagen, gaan we hier niet verder in.

De wat gevorderde programmeur vraagt zich op een gegeven moment af, hoeveel geheugenruimte bepaalde variabelen nu kosten. Om deze vraag te beantwoorden, volgt hieronder een schema van typen variabelen met hun benodigde ruimte.

type variabele	ruimte die in ieder geval wordt bezet	per opgeslagen teken extra rekenen	per dimensie extra tellen	per array-element extra tellen
alfan. variabele	6 bytes	1 byte		
alfan. array-variabele	6 bytes	1 byte	2 bytes	3 bytes
num. integere variabele	5 bytes			
num. variabele met enkelvoudige precisie	7 bytes			
num variabele met dubbele precisie	11 bytes			
num. integere array-variabele	6 bytes		2 bytes	2 bytes
num. array-variabele met enkelv. precisie	6 bytes		2 bytes	4 bytes
num. array-variabele met dubbele precisie	6 bytes		2 bytes	8 bytes

Het aantal dimensies van een array is het aantal getallen die in het DIM-statement voor die variabele tussen haakjes worden genoemd. Het aantal array-elementen vindt men door alle getallen die tussen de haakjes van de betreffende variabele in het DIM-statement staan, elk met 1 te verhogen en deze vervolgens met elkaar te vermenigvuldigen. Een DIMA$ (2,3,10) geeft bijvoorbeeld 3x4x11=132 array-elementen. Het aantal dimensies van A$ is in dat geval 3.

7.1 Wat is een uitdrukking

We kunnen in MSX-basic de computer bijvoorbeeld de volgende som laten oplossen:

```
PRINT 2*(WAARDE+5.5)/(2+AANTAL)
```

De variabelen WAARDE en AANTAL dienen dan wel te zijn gevuld met bepaalde waarden.

De opgave (de som) die na het PRINT-commando staat, noemt men in de computerwereld een UITDRUKKING. Vormen van uitdrukkingen zijn bijvoorbeeld ook:

```
2+3              A/B                "JAN"+"KAREL"
25/(3-4*Q)       "HALLO"+NAAM$      12.55+3%
```

Deze uitdrukkingen, numeriek of alfanumeriek, geven aan:

- welke bewerkingen dienen te geschieden (optellen, aftrekken, vermenigvuldigen etcetera)
- op welke variabelen deze bewerkingen dienen te worden toegepast
- op welke constanten deze bewerkingen dienen te worden toegepast
- in welke volgorde de bewerkingen dienen te worden toegepast.

In feite kunnen we een uitdrukking beschouwen als een som die wij de computer opgeven. Zo'n uitdrukking noemen we numeriek indien het resultaat van deze uitdrukking numeriek is. We noemen een uitdrukking alfanumeriek indien het resultaat van deze uitdrukking alfanumeriek is. In een uitdrukking vinden we de volgende elementen:

- variabelen. Deze hebben een waarde die al eerder bepaald zijn.
- constanten. Deze hebben een vastgestelde waarde.
- bewerkingscodes. Deze geven aan welke bewerkingen dienen te geschieden.
- voorrangstekens. Deze worden altijd gevormd door haakjes. Met deze voorrangstekens kan een afwijkende voorrang in bewerking worden vastgelegd.

We kunnen een uitdrukking als volgt definiëren:

EEN UITDRUKKING IS EEN IN PRINCIPE UITVOERBAAR SAMENSTELSEL VAN CONSTANTEN, VARIABELEN, BEWERKINGEN EN VOORRANGSTEKENS.

Op de eerste twee elementen, de constanten en variabelen, zijn we in de voorgaande hoofdstukken reeds diep ingegaan. In de volgende paragrafen gaan we op bewerkingen en voorrang en voorrangstekens in.

7.2 Algebraïsche bewerkingen

Het MSX-basic kent de volgende algebraïsche bewerkingen:

+ optellen. Met het plus-teken geven we aan dat de uitdrukkingen links en recht van dit teken bij elkaar dienen te worden opgeteld. Dit is de enige bewerking die ook op alfanumerieke waarden mag worden toegepast.
– aftrekken. Met het min-teken geven we aan dat de uitdrukkingen links en rechts van dit teken van elkaar moeten worden afgetrokken. Deze bewerking mag alleen op numerieke bewerkingen worden toegepast.
* vermenigvuldigen. Met het sterretje of vermenigvuldigingsteken geven we aan dat de uitdrukkingen rechts en links van dit teken met elkaar dienen te worden vermenigvuldigd. Deze bewerking mag alleen op numerieke uitdrukkingen worden toegepast.
/ delen. Met de deelstreep geven we aan dat de uitdrukkingen links en rechts van dit teken door elkaar moeten worden gedeeld. Deze bewerking mag alleen op numerieke uitdrukkingen worden toegepast.
^ machtsverheffen. Met het 'dakje' geeft men aan, dat de uitdrukking links van dit teken in een macht dient te worden verheven. De uitdrukking rechts van dit teken geeft aan om welke macht het gaat. Deze bewerking mag alleen op numerieke uitdrukkingen worden toegepast.
\ integer delen. Met deze omgedraaide deelstreep geeft men aan dat de uitdrukkingen links en rechts van dit teken eerst dienen te worden gemaakt tot een integere waarde (een gehele waarde, minimaal gelijk aan –32768 en maximaal gelijk aan 32767). Vervolgens worden deze integere waarden op elkaar gedeeld. De uitkomst wordt een tweede maal gemaakt tot

een integere waarde. Deze bewerking mag alleen op numerieke uitdrukkingen worden toegepast.

MOD restbepaling. Met dit sleutelwoord geeft men aan dat de uitdrukkingen links en rechts van dit sleutelwoord eerst dienen te worden gemaakt tot een integere waarde. Daarna dienen deze twee integere waarden door elkaar te worden gedeeld. De restwaarde die bij deling overblijft (er wordt in dit geval niet na de decimale punt doorgedeeld) is de uitkomst van deze bewerking. Deze bewerking mag alleen op numerieke uitdrukkingen worden toegepast.

Met name de laatste twee bewerkingen kunnen wat problemen geven. Hieronder volgen enkele voorbeelden, verwerkt in één programma:

```
NEW
10 INPUT "EERSTE WAARDE ";A
20 INPUT "TWEEDE WAARDE ";B
30 PRINT "OPTELLING ";A+B
40 PRINT "AFTREKKING ";A-B
50 PRINT "VERMENIGVULDIGING ";A*B
60 PRINT "DELING ";A/B
70 PRINT "INTEGERE DELING ";A\B
80 PRINT "REST BIJ DELING ";A MOD B
90 PRINT "MACHT ";A^B
100 GOTO 10
```

Indien we voor A of B een waarde, groter dan 32767 of kleiner dan –32768 ingeven, dan zal programmaregel 70 een foutmelding geven; A of B zijn niet tot integere waarden te maken.
Wanneer we voor A bijvoorbeeld 25.68 ingeven en daarbij voor B bijvoorbeeld 6.99 ingeven, dan geeft programmaregel 70 de uitkomst 4. A wordt namelijk eerst tot een integere waarde (25) gemaakt. Hetzelfde gebeurt met B (6). Deze twee waarden worden op elkaar gedeeld (25/6=4.66...). De uitkomst wordt tenslotte weer integer gemaakt (4).
Programmaregel 80 geeft bij de genoemde waarden voor A en B de uitkomst 1. De deling 25/6 geeft de uitkomst 4 *rest 1*.
Wanneer B te groot wordt ingegeven, veroorzaakt regel 90 al snel een foutmelding omdat de uitkomst veel te groot wordt. Merk op dat ook oneigenlijke machten (gebroken getallen in de macht) zijn toegestaan.

MSX-basic vertoont bij de bewerking \ één foutje. Tik bijvoorbeeld eens in:

```
? -32768\-1
```
(? mag in plaats van PRINT worden gebruikt)

De computer zal als uitkomst de waarde 32768 geven. Deze waarde is niet integer want een integere waarde mag niet groter zijn dan 32767. Ten onrechte wordt hier geen foutmelding gegeven...

7.3 Functionele bewerkingen

Het MSX-basic kent een groot aantal functionele bewerkingen. In de hoofdstukken 9 en 10 zijn deze met N-FUNCTIE of A-FUNCTIE aangeduid.
Een functionele bewerking is een bewerking die kan worden toegepast op een uitdrukking of op een stelsel van uitdrukkingen. Een functionele bewerking heeft altijd de vorm van:

FUNCTIONELE BEWERKING (UITDRUKKING...)

Om het één en ander wat nader toe te lichten, volgen hieronder enkele voorbeelden van functionele bewerkingen.

Tik bijvoorbeeld eens in:

```
PRINT INT(12.9)
```

De computer zal antwoorden met het getal 12. De functionele bewerking INT heeft tot gevolg dat het grootste gehele getal kleiner dan de tussen de haakjes staande uitdrukking wordt berekend.

```
PRINT LEFT$("ABCDEFGHIJKLMNOP",5)
```

De computer antwoordt hier met de uitkomst ABCDE. De functie LEFT$ heeft tot gevolg dat het linkergedeelte van de opgegeven alfanumerieke uitdrukking wordt bepaald. Het aantal tekens is gegeven in de tweede tussen haakjes gegeven uitdrukking, die numeriek is.

```
PRINT SGN(-12.8)
```

De computer antwoordt hier met de uitkomst −1. De functie SGN geeft drie waarden, namelijk 1 (indien de uitdrukking tussen haakjes

positief is), 0 (indien de uitdrukking tussen haakjes gelijk is aan 0) of −1 (indien de uitdrukking tussen haakjes negatief is).

```
PRINT SIN(3.14)
```

Alleen die programmeurs (in spé) die wat van goniometrie afweten, zullen deze functie begrijpen. De functie geeft de sinus van de waarde van de uitdrukking tussen haakjes. SIN gaat ervan uit dat de uitdrukking tussen haakjes een hoek in radialen voorstelt.

In de hoofdstukken 9 en 10 wordt onderscheid gemaakt tussen A-FUNCTIES en N-FUNCTIES. A-FUNCTIES zijn functionele bewerkingen met een alfanumeriek resultaat (zoals LEFT$) en N-FUNCTIES zijn functionele bewerkingen met een numeriek resultaat (zoals INT).

Een voorbeeldprogramma:

```
NEW
10 INPUT "GEEF EEN WAARDE IN ";WAARDE
20 LET A=INT(WAARDE)
30 LET B=SGN(WAARDE)
40 PRINT "INTEGERE WAARDE ";A
50 IF B=-1 THEN PRINT "NEGATIEF"
60 PRINT LEFT$("ABCDEFGHIJKLMNOPQRSTUV
WXYZ",WAARDE)
70 GOTO 10
```

Dit voorbeeldprogramma vraagt eerst een numerieke ingave in WAARDE. Vervolgens worden in programmaregel 20 en 30 twee functies op deze waarde toegepast; de resultaten worden onder de variabelenamen A en B bewaard. Op programmaregel 40 wordt dan de integere waarde op beeldscherm getoond.
Op programmaregel 50 komen we een nieuw sleutelwoord tegen: IF. Deze regel kan gelezen worden als: als variabele B gelijk is aan −1 (dus als een negatieve waarde werd ingegeven), druk dan de tekst NEGATIEF af. Met het IF-sleutelwoord kunnen we dus een voorwaarde stellen. Als aan deze voorwaarde wordt voldaan, wordt het daarna opgenomen commando uitgevoerd. Op dit sleutelwoord komen we verderop nog uitgebreid terug.
Op programmaregel 60 worden de eerste letters van het alfabet afge-

drukt. Het aantal letters is bepaald door de waarde van variabele WAARDE die wij ingaven. Merk op dat programmaregel 60 een fout geeft bij een negatieve waarde.

7.4 Relationele bewerkingen

Het MSX-basic kent een aantal relationele bewerkingen. Een relationele bewerking is een bewerking die uitdrukking geeft aan een relatie.

Tik bijvoorbeeld eens in:

```
PRINT 2=2
```

De computer zal het antwoord –1 geven. Dit antwoord betekent dat de bewering 2=2 inderdaad waar is. De uitdrukking links van het gelijkteken en de uitdrukking rechts van het gelijkteken hebben de relatie, dat ze in waarde gelijk zijn aan elkaar. Tik bijvoorbeeld eens in:

```
PRINT (5+2)=(8-1)
```

Ook hier zal het oordeel van de computer WAAR (–1) luiden; 5+2 is inderdaad gelijk aan 8–1. Tik nu eens in:

```
PRINT 5+3=7
```

Het oordeel van de computer is nu 0 (NIET WAAR). De uitdrukkingen 5+3 en 7 hebben niet de relatie gelijkheid met elkaar.

Het gelijkteken in de hiervoor gegeven voorbeelden is een relationele bewerking. Een relationele bewerking dient altijd plaats te vinden tussen twee numerieke uitdrukkingen of tussen twee alfanumerieke uitdrukkingen. Het resultaat is altijd –1 (WAAR) of 0 (NIET WAAR). Nog een voorbeeld met alfanumerieke uitdrukkingen:

```
PRINT "JAN"="J"+"AN"
```

en

```
PRINT "JAN"="JANNEMAN"
```

De eerste opdracht zal resulteren in het antwoord –1 (WAAR), de tweede in het antwoord 0 (NIET WAAR).

De bewerking 'is gelijk aan' (=) is slechts één van de relationele bewerkingen die het MSX-basic kent. Hieronder volgt een tabel met alle relationele bewerkingen:

RELATIONELE BEWERKING	BETEKENIS
=	is gelijk aan
<	is kleiner dan
>	is groter dan
<> OF ><	is kleiner of groter dan (ofwel: is ongelijk aan)
<= OF =<	is kleiner dan of gelijk aan (ofwel: is ten hoogste gelijk aan)
>= OF =>	is groter dan of gelijk aan (ofwel: is ten minste gelijk aan)
=<> OF =>< OF <=> OF <>= OF >=< OF ><=	is kleiner dan, gelijk aan of groter dan

De laatste relationele bewerking geeft altijd een waar resultaat (−1) en heeft als zodanig geen enkele zin; we zullen deze bewerking verder vergeten.

In principe zijn er slechts drie relationele bewerkingen (<, = en >). De andere relationele bewerkingen zijn combinaties van deze drie basisbewerkingen.

Een voorbeeldprogramma:

```
NEW
10 INPUT "WAARDE ";A
20 INPUT "NOG EEN WAARDE ";B
```

```
30 IF A<B THEN PRINT "DE EERSTE WAARDE
IS KLEINER"
40 IF A=B THEN PRINT "DE TWEE WAARDEN
ZIJN GELIJK"
50 IF A>B THEN PRINT DE EERSTE WAARDE
IS GROTER"
60 IF A<=B THEN PRINT "DE EERSTE WAARDE
IS OF KLEINER OF GELIJK AAN DE TWEEDE"
70 GOTO 10
```

Na enig proberen zal het volgende blijken: indien een uitdrukking de uitkomst −1 heeft (WAAR IS), wordt het commando (of worden de commando's) achter THEN in een IF...THEN constructie uitgevoerd. Is de uitdrukking echter niet waar (de uitkomst is dan 0) dan gebeurt dit niet.

Probeer nu eens het volgende programma:

```
NEW
10 INPUT "WAARDE ";A
20 IF A THEN PRINT "WAAR"
30 GOTO 10
```

en laat dit programma eens uitvoeren. Geef diverse waarden in, negatief, positief en ook eens een 0. De computer zal steeds antwoorden met de tekst "WAAR" tenzij een 0 werd ingegeven. Conclusie:

> *Een relationele bewerking geeft bij uitvoering de uitkomst waar of niet waar. Deze oordelen worden gesymboliseerd in de waarden −1 of 0. In de IF...THEN-constructie kan het al dan niet waar zijn van een uitdrukking worden getest. Indien een uitdrukking als waar wordt bevonden, wordt het programmaregelgedeelte achter het sleutelwoord THEN uitgevoerd; tussen IF en THEN mag elke uitdrukking worden geplaatst. Deze uitdrukking behoeft niet verplicht een relationele bewerking te bevatten. De IF...THEN-constructie beschouwt een uitdrukking als waar indien deze ongelijk is aan nul (zie het laatste voorbeeld, de uitkomst hoeft dus niet verplicht gelijk te zijn aan −1) en als niet waar indien deze wel gelijk is aan nul.*

Merk op dat het gelijkteken binnen MSX-basic een dubbele functie uitoefent.

Eerste functie: LET A=12.34 in een dergelijke constructie vormt het gelijkteken geen relationele bewerking maar geeft het slechts aan dat een variabele dient te worden gelijkgesteld aan een uitdrukking.

Tweede functie: PRINT A=B in een dergelijke constructie vormt het gelijkteken een relationele bewerking.

7.5 Logische bewerkingen

Het MSX-basic biedt verscheidene logische bewerkingen. Logische bewerkingen dienen ondermeer om zeer ingewikkelde afvragingen te verrichten.
De beginnende programmeur kan, zeker in eerste instantie, misschien deze paragraaf beter overslaan.

Een logische bewerking vergelijkt twee uitdrukkingen en doet vervolgens een uitspraak in de zin van WAAR (ongelijk aan 0) of NIET WAAR (gelijk aan 0). Aan de hand van de meest gemakkelijke logische bewerking lichten we dit wat nader toe. Tik bijvoorbeeld eens in:

eerste regel:

```
IF 2+3=5 AND 4+6=10 THEN PRINT "HOERA"
```

tweede regel:

```
IF 2+4=6 AND 6+6=11 THEN PRINT "EEN
VREEMDE ZAAK"
```

derde regel:

```
IF "JAN"<"KAREL" AND 4*4=16 THEN PRINT
"HOERA"
```

De eerste regel resulteert evenals de derde regel in het afdrukken van de tekst HOERA op het beeldscherm. De tweede regel blijft zonder resultaat. Verklaring: De logische bewerking AND onderzoekt de uit-

drukking links en rechts van deze bewerking. Alleen indien beide bewerkingen WAAR zijn (−1 als uitkomst hebben) dan krijgt de totale uitdrukking tussen IF en THEN de beoordeling WAAR (de waarde −1) en wordt het gedeelte achter THEN uitgevoerd. De derde regel geeft een extra moeilijkheid; er wordt een bewering gedaan met twee teksten waarbij wordt beweerd dat de eerste tekst kleiner is dan de tweede tekst. Indien de uitdrukkingen waarop een relationele bewerking wordt toegepast, alfanumeriek zijn, dan worden deze uitdrukkingen alfabetisch uitgewaardeerd. Een tekst die bij een alfabetische rangschikking voor een tweede tekst terecht zou komen, is ook kleiner dan deze tweede tekst.

Over het algemeen kan men stellen dat karakters volgens de MSX-karaktertabel (hoofdstuk 18) worden vergeleken waarbij een karakter met de laagst gecodeerde waarde ook het kleinst is.

De eerste voorbeeldregel zou in normaal Nederlands als volgt kunnen worden vertaald:

Als 2+3 gelijk is aan 5 *en* 4+6 is gelijk aan 10, druk dan de tekst HOERA af.

De logische bewerking AND kan men dus vertalen met het woordje EN.

Het MSX-basic kent diverse logische bewerkingen. Hieronder volgt een tabel met alle logische bewerkingen die in MSX-basic bestaan:

logische bewerking	betekenis ongeveer	uitleg
AND	en	deze logische bewerking wordt tussen twee uitdrukkingen geplaatst. Hij geeft de uitkomst WAAR (ongelijk aan nul) alleen indien de beide uitdrukkingen WAAR (ongelijk aan nul) zijn.
OR	of	deze logische bewerking wordt tussen twee uitdrukkingen geplaatst. Hij geeft de uitkomst WAAR indien minstens één van de twee uitdrukkingen WAAR is.
XOR	exclusief of	deze logische bewerking wordt tussen twee uitdrukkingen geplaatst. Hij geeft

		slechts de uitkomst WAAR indien minstens en hoogstens één van de uitdrukkingen WAAR is.
EQV	gelijkwaardig met	deze logische bewerking wordt tussen twee uitdrukkingen geplaatst. Hij geeft slechts de uitkomst WAAR indien beide uitdrukkingen WAAR zijn of beide uitdrukkingen NIET WAAR zijn (wanneer beide uitdrukkingen dus gelijkwaardig zijn).
IMP	impliceert dat	deze logische bewerking wordt tussen twee uitdrukkingen geplaatst. Hij geeft slechts de uitkomst NIET WAAR indien de eerste uitdrukking waar is maar de tweede niet (wanneer het WAAR zijn van de eerste uitdrukking niet impliceert dat de tweede uitdrukking WAAR is).
NOT	niet	deze logische bewerking wordt vóór een tussen haakjes gestelde uitdrukking geplaatst en draait de waarde van deze uitdrukking om. Indien deze uitdrukking WAAR is, veroorzaakt deze bewerking dus het antwoord NIET WAAR en omgedraaid.

Een voorbeeldprogramma ter afsluiting:

```
NEW
10 REM VOORBEELD LOGISCHE BEWERKINGEN
20 INPUT "HOEVEEL IS 2+3 ";A
30 INPUT "HOEVEEL IS 5+7 ";B
40 IF A=5 AND B=12 THEN PRINT "ALLEBEI
DE SOMMEN GOED"
50 IF A=5 OR B=12 THEN PRINT "EEN OF T
WEE SOMMEN GOED"
60 IF A=5 XOR B=12 THEN PRINT "SLECHTS
```

```
 EEN SOM GOED"
70 IF NOT(A=5 OR B=12) THEN PRINT "ALL
ES FOUT"
80 IF A=5 EQV B=12 THEN PRINT "ALLEBEI
 GOED OF ALLEBEI FOUT"
90 IF NOT (A=5 IMP B=12) THEN PRINT "D
E EERSTE GOED MAAR DE TWEEDE FOUT"
100 GOTO 20
```

Bij het proberen van dit programma moet u eens wat goede en wat foute antwoorden aan de computer geven. Let eens op zijn bevindingen. Indien u beide antwoorden goed gaf, geeft de computer het commentaar BEIDE SOMMEN GOED, EEN OF TWEE SOMMEN GOED en ALLEBEI GOED OF ALLEBEI FOUT. Dit zijn de drie juiste conclusies die via de AND, de OR en de EQV bewerkingen konden worden getrokken.

Het werken met logische bewerkingen vereist een logisch redeneervermogen maar ook een dosis oefening. Alhoewel het gebruik hiervan in eerste instantie niet gemakkelijk is, zal in latere instantie het nut van deze logische bewerkingen zich bewijzen.

7.6 Logische bewerkingen voor gevorderen

Deze paragraaf dient door beginnende programmeurs zeker te worden overgeslagen daar hier erg diep op logische bewerkingen wordt ingegaan. Het is niet aannemelijk dat een beginnende programmeur hier al veel aan heeft.

In de vorige twee paragrafen werd een beperking gehandhaafd die het één en ander wat gemakkelijker maakte. Wanneer we relationele en logische bewerkingen op de keper willen beschouwen, dienen we het volgende vast te stellen.

1) een relationele bewerking heeft de uitkomst −1 (WAAR) of 0 (NIET WAAR) tot gevolg. Deze uitkomst dient te worden beschouwd als een integere waarde die via de tweecomplementmethode wordt gepresenteerd. WAAR of −1 betekent in dat geval: alle bits binnen deze (twee bytes) waarde op 1 en NIET WAAR betekent: alle bits op 0.
2) de logische bewerkingen mogen slechts worden toegepast op uitdrukkingen die een integere waarde tot gevolg hebben of tot

een integere waarde kunnen worden gemaakt.

3) logische bewerkingen oefenen hun bewerking op bit-niveau uit (vandaar de uitkomst –1 en niet 1 bij WAAR).

Het een en ander heeft tot gevolgd dat de geoefende programmeur in MSX-basic op bit-niveau zijn afvragingen kan doen waarbij het in sommige toepassingen mogelijk is om zestien afvragingen in één keer te doen.
Het toepassen van de logische bewerkingen op dit niveau vereist een paar kwaliteiten van de programmeur:

– een perfekte kennis van het binaire talstelsel
– een volledige beheersing van de tweecomplementmethode
– een gedegen kennis van de Boole algebra

Het voert te ver om in dit handboek op deze drie gebieden verder in te gaan; de geïnteresseerde kan zich het beste op de hoogte stellen met de vakliteratuur die op deze onderwerpen bestaat.

Eén tip: oefening kan het best geschieden met een volgende constructie:

```
PRINT BIN$((&B1101) AND (&B1011))
```

In plaats van AND kan natuurlijk elke logische bewerking (behalve NOT) worden geplaatst. Het resultaat van de op bitniveau uitgevoerde logische bewerking wordt binair, dus ook weer bitsgewijs, afgedrukt waardoor een bitsgewijze studie van de werking van logische bewerkingen gemakkelijk mogelijk is. Zie ondermeer de behandeling van BIN$ (een A-FUNCTIE), in hoofdstuk 9.

Merk op dat de binaire constanten van het laatste voorbeeld tussen haakjes moeten. Een foutje in MSX-basic zorgt anders ten onrechte voor een foutmelding (die dan weer opgelost kan worden door de spatie voor AND te verwijderen).

Tot slot volgt voor de gevorderde lezer een waarheidstabel met betrekking tot de logische bewerkingen:

bewerking	eerste argument	tweede argument	resultaat
NOT	0		1
	1		0
AND	0	0	0
	0	1	0
	1	0	0
	1	1	1
OR	0	0	0
	0	1	1
	1	0	1
	1	1	1
XOR	0	0	0
	0	1	1
	1	0	1
	1	1	0
EQV	0	0	1
	0	1	0
	1	0	0
	1	1	1
IMP	0	0	1
	0	1	1
	1	0	0
	1	1	1

7.7 Voorrangsregels

Wanneer in een uitdrukking veel bewerkingen voorkomen, dan zal al snel de twijfel bestaan welke bewerking het eerste dient te geschieden. Bekijken we bijvoorbeeld de uitdrukking in de volgende programmaregel:

```
PRINT 2*3+4
```

De uitkomst zal luiden: 10. Dit wijst erop dat eerst de vermenigvuldiging en pas later de optelling is uitgevoerd. De vermenigvuldiging had blijkbaar voorrang op de optelling. We kunnen deze voorrang wijzigen door het een en ander tussen haakjes te plaatsen:

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

In dit geval wordt eerst de uitdrukking tussen haakjes uitgewerkt en pas daarna de vermenigvuldiging gedaan. De uitkomst luidt dan ook: 14.

Indien de moeilijkheid zich zou beperken tot vermenigvuldigen en optellen, dan zou er niet zoveel aan de hand zijn. Echter, binnen MSX-basic hebben we een keur van algebraïsche, relationele, functionele en logische bewerkingen. In dat woud van bewerkingen is het nuttig om te weten welke bewerkingen voorrang hebben op welke andere bewerkingen in MSX-basic. De voorrangsregels zijn niet zo erg moeilijk; ze komen behoorlijk overeen met de voorrangsregels die algemeen in rekenwerk worden toegepast.

Allereerst noemen we de haakjes. Een gouden regel is dat *haakjes de allerhoogste voorrang* hebben. De uitdrukking die het diepst tussen haakjes staat, wordt altijd het eerst door de computer uitgewerkt. Dit geldt ook voor de haakjes die verplicht zijn bij een functionele bewerking.

Buiten deze haakjes liggen de voorrangsregels als volgt:

voorrangs-volgorde	bewerkingen	opmerkingen
1	functionele bewerkingen	Door het verplichte gebruik van haakjes kunnen onderling geen voorrangsconflicten ontstaan.
2	algebraïsche bewerkingen	eerst ^
		dan *, /, \ en MOD 1)
		dan + en – 1)
3	relationele bewerkingen	1)
4	logische bewerkingen	1)

1) Indien er tussen gelijkwaardige bewerkingen conflictsituaties ontstaan, dan worden deze bewerkingen van links naar rechts in volgorde uitgevoerd.

7.8 Precisie tijdens berekeningen

Bij het uitwerken van uitdrukkingen dient de computer vaak vrij ingewikkelde berekeningen te maken. Alhoewel dit schijnbaar moeiteloos gebeurt, komt voor een berekening vaak veel kijken. Tik eens in:

```
PRINT 2+40\INT(2*5+4.1)
```

Hoegenaamd onmiddellijk produceert de computer het juiste antwoord. Laten we eens nagaan, welke bewerkingen de computer moest doen om tot dit antwoord te komen.

Uit de voorrangsregels volgt, dat de berekening als volgt dient te worden uitgevoerd:

uitdrukking	`2+40\INT(2*5+4.1)`	
stap 1	`2+40\INT(10+4.1)`	
stap 2	`2+40\INT(14.1)`	
stap 3	`2+40\14`	
stap 4	`2+2`	
stap 5	4	(uitkomst)

Tijdens deze uitvoerige berekeningen moeten nogal eens tussenuitkomsten door de computer worden bewaard. Deze tussenuitkomsten worden in zogenaamde systeemvariabelen bewaard en zijn in MSX-basic niet te benaderen. Om onnauwkeurigheden tijdens de berekeningen te voorkomen, worden deze systeemvariabelen altijd toegewezen als variabelen met *dubbele precisie*.

Een uitzondering wordt gevormd door de logische en relationele bewerkingen. Het resultaat hiervan wordt, indien dit als tussenuitkomst dient te worden opgeslagen, altijd als integere systeemvariabele opgeslagen.

8.1 De BNF-notatie

We zagen reeds eerder dat een computer een dom apparaat is. Om de computer te laten doen wat we van hem wensen, is het noodzakelijk om zijn taal te beheersen, in dit geval het MSX-basic.

Juist omdat de computer in feite een dom apparaat is, is hij niet in staat om gegevens te interpreteren, om een betekenis te geven aan begrippen. Elke fout, al is het maar het vergeten van een komma of een verkeerd aangeslagen letter, wordt door de computer dan ook onmiddellijk gemeld; de computer is niet in staat om te onderzoeken wat we eigenlijk bedoelen.

Om deze reden is het noodzakelijk om de computer te voeden met programmagegevens die tot op de laatste letter precies volgens de voorschriften van de computertaal zijn.

Om een computertaal precies volgens de voorschriften te kunnen gebruiken, dienen we deze voorschriften zélf eerst te kennen.

In de hoofdstukken 9 en 10 wordt van elk sleutelwoord de exacte schrijfwijze (syntaxis) gepresenteerd, alsmede een omschrijving van wat het sleutelwoord voor ons kan doen (semantiek).

Om in de preciese *schrijfwijze* van elk sleutelwoord geen twijfel te laten bestaan, is gekozen voor een speciale methode van notatie van de schrijfwijze. Deze speciale notatiemethode noemen we de BNF-notatie, naar de uitvinder van deze notatievorm (BNF=Backus Normal Form). Omdat het MSX-basic één van de meest uitgebreide talen is, schieten de mogelijkheden van de BNF-notatie soms te kort. Daarom zijn bij de normale symbolen uit de BNF-notatie wat extra symbolen toegevoegd.

Om bij de schrijfwijze snel te kunnen bepalen welke de juiste is, is een zekere kennis van de BNF-notatiewijze zeer welkom. Echter, de notatiewijze vereist een logisch denkvermogen dat zeker bij de beginnende programmeur niet altijd aanwezig behoeft te zijn.

Het is verstandig om dit hoofdstuk in ieder geval zo veel mogelijk door

te nemen. Het is daarbij geen probleem dat niet alles wordt begrepen. Wel dienen de voorgaande hoofdstukken grondig te zijn doorgenomen.

Indien later in hoofdstuk 9 en 10 nog problemen bestaan doordat de notatie van de voorgeschreven schrijfwijze niet helemaal duidelijk is, dan kan de laatste twijfel meestal worden weggenomen door de voorbeelden die bij de sleutelwoorden zijn opgenomen, te bestuderen.

8.2 Een eerste kennismaking met de BNF-notatie

In de volgende paragrafen volgt een uitleg van de BNF-notatievorm. Daarbij zijn wat algemene definities (b.v. die van een alfanumerieke constante) in BNF opgenomen. Naar deze algemene specificaties wordt in hoofdstuk 9 en 10 steeds weer verwezen.

Om de BNF in te leiden, volgt in deze paragraaf een eenvoudig en uitgewerkt voorbeeld.

Allereerst beginnen we met de standaard BNF-symbolen.
Deze zijn:

[ ]	het tussen vierkante haken opgenomen gedeelte mag naar wens al of niet worden geplaatst
{ }	het tussen accolades opgenomen gedeelte mag weggelaten worden of één- of meermalen worden geplaatst
< >	het tussen scherpe haken opgenomen gedeelte vormt een beschrijving. Deze beschrijving kan later weer verder worden behandeld
¦	dit teken betekent *of*. Naar keuze mag *of* het linkergedeelte *of* het rechtergedeelte worden opgenomen
::=	dit teken betekent: is per difinitie gelijk aan

Laten we met behulp van deze symbolen eens proberen of we de schrijfwijze van een cijfer kunnen bepalen:

```
<CIJFER>::=0¦1¦2¦3¦4¦5¦6¦7¦8¦9
```

Deze definitie hebben we eenvoudig vastgelegd. We hebben geschreven

dat een cijfer of een 0, of een 1, of een 2 etcetera is. Nu we het begrip cijfer, hoe kinderachtig het ook lijkt, hebben vastgelegd, kunnen we dit begrip verder gebruiken.

Laten we eens proberen om het begrip integere constante vast te leggen. We hoeven ons in dit stadium niet druk te maken over geldende minima of maxima.

We proberen:

```
<INTEGERE CONSTANTE>::={<CIJFER>}
```

Het tussen de accolades geplaatste begrip mag herhaald voorkomen. We hebben met deze eenvoudige definitie dus bepaald dat bijvoorbeeld 0, 125, 3842 etcetera een integere constante is.

Echter, er zit een fout in deze definitie. Het tussen accolades geplaatste begrip mag volgens de omschrijving van de symbolen namelijk ook *weggelaten* worden (=0 keer voorkomen). Een integere constante zou volgens deze definitie dus ook uit *niets* kunnen bestaan!

Aangezien een integere constante minimaal één cijfer dient te bevatten, dienen we onze definitie wat aan te passen.
We proberen:

```
<INTEGERE CONSTANTE>::=<CIJFER>{<CIJFER>}
```

Uit deze definitie volgt dat een integere constante uit één of meer cijfers bestaat. Het eerste probleem is opgelost. Echter, een volgende onvolkomenheid dient zich aan: we weten dat −100 bijvoorbeeld een integere constante is. Volgens onze definitie is dit echter (nog) niet zo. We moeten deze definitie dus nog wat verder aanpassen:

```
INTEGERE CONSTANTE>::=[+|-]<CIJFER>
  {<CIJFER>}
```

Met deze definitie hebben we ook die onvolkomenheid verholpen. Deze laatste definitie bepaalt dat een integere constante als eerste teken een + of een − mag bevatten; het is echter niet verplicht.

Omdat we bij het omschrijven van de syntaxis (schrijfwijze) ons niet hoeven druk te maken om geldende maxima of minima, is deze definitie verder in orde!

Het zal duidelijk zijn dat onze definitie slechts een héél eenvoudige is, in de volgende paragrafen en hoofdstukken komen we veel ingewikkeldere omschrijvingen tegen. Bedenk daarbij echter dat er naar is gestreefd om deze omschrijvingen zo correct mogelijk uit te voeren. De doorgewinterde BNF-lezer kan hier erg veel informatie uit halen; voor de beginnende programmeur is het voldoende dat hij of zij een *behoorlijke indruk* van de schrijfwijze verkrijgt!

8.3 Algemene opmerkingen inzake de notatie van MSX-basic in BNF

Voorafgaande aan de BNF-behandeling en de behandeling van enige algemene specificaties dienen eerst de volgende opmerkingen te worden gemaakt:

- Binnen de in dit hoofdstuk opgenomen notaties zijn geen restricties aangebracht inzake geldende minimale waarden, maximale waarden en maximale lengten. Zie hiervoor de behandeling van de constanten en variabelen in de voorgaande hoofdstukken. Daarbij dient te worden vermeld dat een ⟨kommando⟩ een ⟨rij van kommando's⟩ en een ⟨programmaregel⟩ nooit de lengte van 255 karakters mogen overschrijden.
- Behalve bij de definitie van ⟨letter⟩ mogen in alle in dit hoofdstuk genoemde notaties, kleine letters en hun corresponderende hoofdletters vrij met elkaar worden verwisseld.
- Sleutelwoorden zijn steeds *onderstreept*. Deze sleutelwoorden dienen steeds *aaneensluitend* te worden gebruikt. Buiten deze restrictie is het MSX-basic *spatie-ongevoelig*; op willekeurige plaatsen buiten sleutelwoorden mogen naar believen spaties worden tussengevoegd.
- De behandelde sleutelwoorden zijn *gereserveerd*. Variabelenamen mogen *nooit* gelijk zijn aan een sleutelwoord of een sleutelwoord bevatten. Zie voor een totale opsomming van alle gereserveerde sleutelwoorden hoofdstuk 19.
- Om de schrijfwijze (syntaxis) van MSX-basic zo nauwkeurig mogelijk te omschrijven, moest een onderscheid gemaakt worden tussen numerieke en alfanumerieke variabelen en constanten. Echter, door het dynamische karakter van MSX-basic wat betreft de typering van variabelen (via DEFSTR, DEFINT, DEFSNG en DEFDBL) kunnen sommige definities niet volledig worden uitgedefinieerd, maar moet worden volstaan met een vrij breedvoerige omschrijving.

- De definitie voor ⟨N⟩ (de numerieke uitdrukking) is niet sluitend. In een enkel geval kan een volgens de voorgeschreven schrijfwijze opgebouwde ⟨N⟩ toch een 'type mismatch' foutmelding geven. Deze foutmelding is te wijten aan een niet ver genoeg gaande vaststelling van de volgorde van uitvoering van bewerkingen bij verschillende typen variabelen en/of constanten in MSX-basic.
- Alhoewel de ELSE constructie voor en achter elk commando is toegestaan en zelfs als enig commando is toegestaan, heeft deze constructie slechts zin indien gebruikt achter een IF...THEN-constructie. In elke andere constructie vindt executie van een navolgend gedeelte *nooit* plaats. Daar een syntactische controle pas op het moment van executie plaats vindt, mag achter een ELSE, niet voorkomend in een IF...THEN-constructie, ⟨...⟩ (niet ter zake doende opvolging van karakters) worden opgenomen (zie hoofdstuk 8.4).

8.4 Gebruikte symbolen

De BNF-notatie kent standaard de volgende symbolen:

symbool	betekenis
[ ]	de tussenliggende bepaling mag 0 of 1 maal worden opgenomen (optie)
{ }	de tussenliggende bepaling mag 0 of meer maal worden opgenomen (repetitie of herhaling)
< >	de tussenliggende tekst verwijst naar een andere definitie of bevat een eenduidige omschrijving
: :=	betekent: is per definitie gelijk aan
--	betekent: of. Gekozen dient te worden tussen het links van dit teken of het rechts van dit teken vermelde gedeelte.

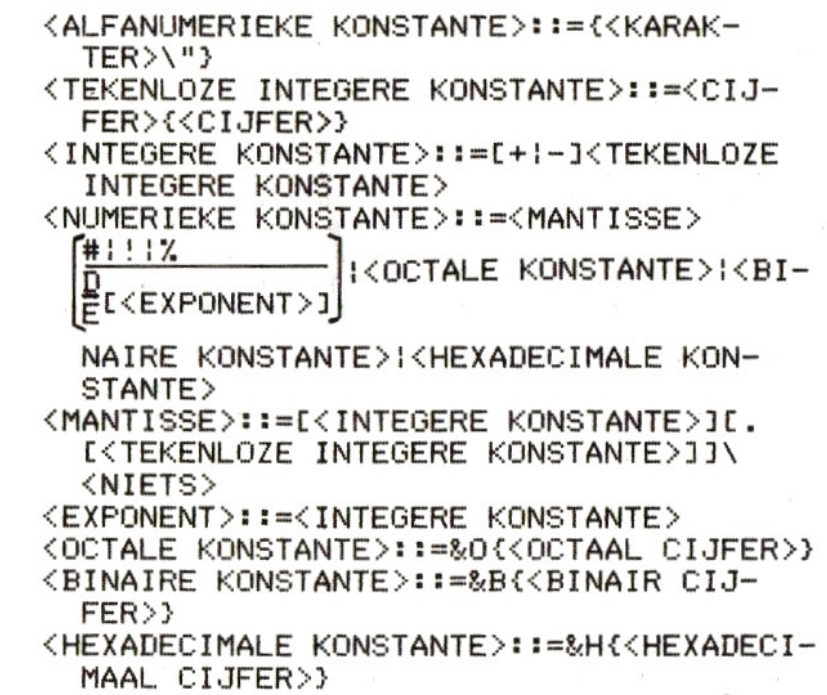
Om het zéér uitgebreide MSX-basic met BNF te kunnen beschrijven, dienen enige aanvullende symbolen te worden gebruikt. Deze zijn:

uitbreiding symbool	betekenis
<...>	(let op de puntjes). Betekent: een willekeurige, niet terzake doende opvolging van karakters. Dus: <...>::={<KARAKTER>} Vanwege de nadruk dat het een en ander niet ter zake doet, is gekozen voor een apart symbool
———	betekent *of*. Gekozen dient te worden tussen het boven dit teken of onder dit teken vermelde gegeven
\	betekent: met uitzondering van, behalve. De voor dit teken opgenomen definitie moet worden *beperkt* met de na dit teken opgenomen definitie
<?>	nog niet vastgelegde definitie of omschrijving
...	logische voortzetting

8.5 Elementaire definities

```
<NIETS>::=
<LETTER>::=A|B|C|...|Z|a|b|c|...|z
<CIJFER>::=0|1|2|...|9
<KARAKTER>::=<ALLE KARAKTERTEKENS, VER-
  MELD IN DE MSX-KARAKTERTABEL BEHALVE
  DE KARAKTERS MET EEN ASCII-WAARDE VAN
  31 OF MINDER DECIMAAL>
<OCTAAL CIJFER>::=0|1|2|3|4|5|6|7
<BINAIR CIJFER>::=0|1
<HEXADECIMAAL CIJFER>::=0|1|2|3|...|9|A|
  B|C|D|E|F|
```

8.6 Definities van konstanten



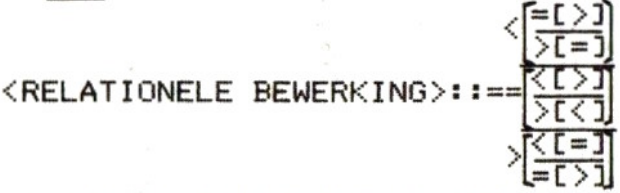
8.7 Definities van variabelen

<NUMERIEKE VARIABELE-NAAM>::=<LETTER>
{<LETTER>|<CIJFER>}[#|!|%]\<LETTER, OPGE-
NOMEN ACHTER EEN UITGEVOERD DEFSTR-
KOMMANDO ZONDER DAT DAARNA EEN DEFINT-,
EEN DEFSNG- OF DEFDBL-KOMMANDO IS UIT-
GEVOERD MET DAARACHTER DEZE LETTER>
{<LETTER>|<CIJFER>}

<ALFANUMERIEKE VARIABELE-NAAM>::=<LET-
TER>{<LETTER>|<CIJFER>}$|<LETTER, OP-
GENOMEN ACHTER EEN UITGEVOERD DEFSTR-
KOMMANDO ZONDER DAT DAARNA EEN DEFINT-,
EEN DEFSNG- OF DEFDBL-KOMMANDO IS UIT-
GEVOERD MET DAARACHTER DEZE LETTER>
{<LETTER>|<CIJFER>}[$]
<NUMERIEKE VARIABELE>::=<NUMERIEKE VARI-
ABELE NAAM>[<INDEXERING>]
<ALFANUMERIEKE VARIABELE>::=<<ALFANUME-
RIEKE VARIABELE-NAAM>[<INDEXERING>]
<INDEXERING>::=(<DIMENSIE>{,<DIMENSIE>})
<DIMENSIE>::=<N>
<VARIABELE-NAAM>::=<ALFANUMERIEKE VARIA-
BELE-NAAM>|<NUMERIEKE VARIABELE-NAAM>
<VARIABELE>::=<ALFANUMERIEKE VARIABELE>|
<NUMERIEKE VARIABELE>

8.8 Definities van numerieke bewerkingen

```
<NUMERIEKE BEWERKING>::=<ALGEBRAISCHE BE-
  WERKING>|<RATIONELE BEWERKING>|<LOGI-
  SCHE BEWERKING>|<NUMERIEKE FUNCTIONELE
  BEWERKING>
<ALGEBRAISCHE BEWERKING>::=+|-|*|/|^|\|
  MOD
```




```
<LOGISCHE BEWERKING>::=NOT|AND|OR|XOR|
  EQV|IMP
<NUMERIEKE FUNKTIONELE BEWERKING>::=<AL-
  LE SLEUTELWOORDEN, BEHORENDE BIJ DE N-
  FUNKTIES ZOALS BEHANDELD IN DE HOOFD-
  STUKKEN 9 EN 10>
```

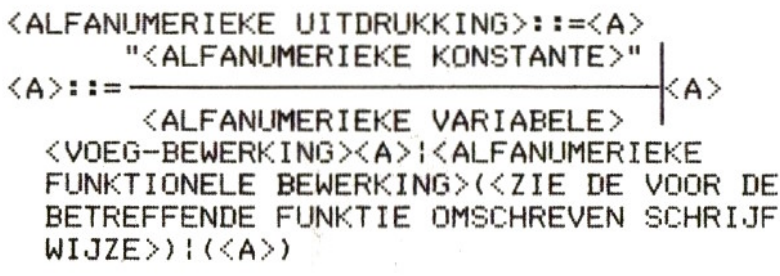
8.9 Definities van alfanumerieke bewerkingen

```
<ALFANUMERIEKE BEWERKING>::=<VOEG-BEWER-
  KING|<RELATIONELE BEWERKING|<ALFANUME-
  RIEKE FUNKTIONELE BEWERKING>
<VOEG-BEWERKING>::=+
<ALFANUMERIEKE FUNKTIONELE BEWERKING>::=
  <ALLE SLEUTELWOORDEN, BEHORENDE BIJ DE
  A-FUNKTIES ZOALS BEHANDELD IN DE HOOFD-
  STUKKEN 9 EN 10>
```

8.10 Definitie van een numerieke uitdrukking

```
<NUMERIEKE UITDRUKKING>::=<N>
      <NUMERIEKE KONSTANTE>
<N>::= <NUMERIEKE VARIABELE>
     <N>!<NUMERIEKE FUNKTIONELE BEWERKING>(<ZIE VOOR DE
     BETREFFENDE FUNKTIE OMSCHREVEN SCHRIJFWIJZE>)!<A>
     <RELATIONELE BEWERKING><A>!(<N>)
```

8.11 Definitie van een alfanumerieke uitdrukking



8.12 Definitie van een uitdrukking

8.13 Overige definities

<TYPE 1 INTEGERE KONSTANTE>::=<INTEGERE KONSTANTE>

<KOMMANDO>::=<ALLE SLEUTELWOORDEN, IN EEN UITWERKING ZOALS ONDER SCHRIJFWIJZE VERMELD, BEHORENDE BIJ DE KOMMANDO'S ZOALS BEHANDELD IN DE HOOFDSTUKKEN 9 EN 10>

<RIJ VAN KOMMANDO'S>::= 

<DIREKTE OPDRACHT 1>::=<RIJ VAN KOMMANDO'S>['<ALFANUMERIEKE KONSTANTE>]

<DIREKTE OPDRACHT 2>::=<RIJ VAN KOMMANDO'S>\<...>"

<DIREKTE OPDRACHT>::=<DIREKTE OPDRACHT 1>|<DIREKTE OPDRACHT 2>

<INDIREKTE OPDRACHT>::=<PROGRAMMAREGEL>

<PROGRAMMAREGEL>::=<REGELNUMMER><DIREKTE OPDRACHT>

<REGELNUMMER>::=<TYPE 1 INTEGERE KONSTANTE>

9 DE SLEUTELWOORDEN VAN MSX-BASIC

In dit hoofdstuk worden de sleutelwoorden van het MSX-basic behandeld tot zover deze in de 1.0-versie door MICROSOFT zijn vrijgegeven.

De sleutelwoorden worden in alfabetische volgorde behandeld.

Indien u de sleutelwoorden één voor één *voor het eerst* wilt leren kennen, is het zeer raadzaam om dit te doen via de aanbevolen leervolgorde. Zie hiervoor hoofdstuk 12.

Indien u een bepaald soort sleutelwoorden nader wenst te bestuderen (bijvoorbeeld alleen de grafische commando's of alleen de geluidscommando's), dan is het zeer raadzaam om dit te doen via hoofdstuk 11, waar de sleutelwoorden op soort zijn gegroepeerd.

Per sleutelwoord worden de volgende gegevens verstrekt:

- de naam van het sleutelwoord
- de moeilijkheidsgraad bij het gebruik van dit sleutelwoord. Deze moeilijkheidsgraad variëert van zeer eenvoudig tot zeer moeilijk en is een indicatie voor de beginnende programmeur die kan helpen bij een eerste selectie van de te bestuderen sleutelwoorden
- de soort. We onderscheiden binnen de sleutelwoorden de A-FUNCTIES (functies met alfanumeriek resultaat), de N-FUNCTIES (functies met numeriek resultaat) en de COMMANDO'S (sleutelwoorden die een aktie aangeven)
- de afkomst. Alle in MSX-basic voorkomende sleutelwoorden zijn afkomstig uit de Engelse taal. Het is vaak veel gemakkelijker om een sleutelwoord met de bijbehorende betekenis te onthouden wanneer men de preciese afkomst kent. Daarom is van elk sleutelwoord de afkomst naar het Nederlands herleid
- de schrijfwijze. De toegestane syntaxis of schrijfwijze is in BNF opgenomen voor elk sleutelwoord. Zie voor uitleg van de BNF-notatie hoofdstuk 8.

– de betekenis. De functie van elk sleutelwoord wordt uitvoerig behandeld

– een voorbeeld. Waar zinvol is een voorbeeld opgenomen. De voorbeelden zijn waar nodig in verband met het gebruik van andere sleutelwoorden gebaseerd op de aanbevolen leervolgorde.

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **N-FUNKTIE**
afkomst ..... **ABS is afkorting van absolute value – absolute waarde**

schrijfwijze

```
ABS(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft van de tussen haakjes vermelde numerieke uitdrukking de ABSOLUTE WAARDE. De absolute waarde van een getal is de waarde van dit getal, ontdaan van enig teken.

Zo is bijvoorbeeld de absolute waarde van:

het getal 12.347	het getal 12.347
en van het getal -133.5	het getal 133.5

Voorbeeld:

```
PRINT ABS(7*(-5))
 35
Ok
PRINT ABS(111)
 111
Ok
```

moeilijkheidsgraad . . **vrij moeilijk, kennis van de ASCII is noodzakelijk**

soort . . . . . **N-FUNKTIE**

afkomst . . . . . **ASC is afkorting van ASCII – american standard code for information interchange – amerikaanse standaard kodering voor informatie-overdracht**

schrijfwijze

```
ASC(<A>)
<A>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft de ASCII-waarde van het eerste teken van de tussen haakjes vermelde alfanumerieke uitdrukking als resultaat.

De ASCII-waarde is de waarde die de computer intern in het geheugen gebruikt om allerlei tekns op te slaan. Deze waarde is nooit kleiner dan 0 en nooit groter dan 255.

Zie hoofdstuk 17 en hoofdstuk 18. In hoofdstuk 17 staat de standaard ASCII-tabel, in hoofdstuk 18 staat de daarvan afgeleide MSX-tabel.

Voorbeeld:

```
NEW
10 LET G$="MSX"
20 PRINT ASC(G$)
RUN
 77
Ok
```

(77 is de ASCII-kodering voor de letter M)

moeilijkheidsgraad . . **vrij moeilijk, kennis van goniometrie is noodzakelijk**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **ATN is afkomstig van arctangent – arctangens**

schrijfwijze

```
ATN(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft de arctangens van de tussen haakjes vermelde numerieke uitdrukking als resultaat. Het resultaat ligt altijd tussen $-\frac{1}{2}\pi$ en $\frac{1}{2}\pi$ en drukt als zodanig een hoek in radialen uit.

Voorbeeld:

```
NEW
10 INPUT "WAARDE ";A
20 PRINT "DE ARCTANGENS VAN";A;"IS";AT
N(A)
RUN
WAARDE ? 22
DE ARCTANGENS VAN 22 IS 1.5253730473733
Ok
```

moeilijkheidsgraad . . . . . **zeer eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **AUTO is afkomstig van automatic – automatisch**

schrijfwijze

```
AUTO[<REGELNUMMER>][,[<STAPGROOTTE>]]
<REGELNUMMER>::=<TYPE 1 INTEGERE KON-
  STANTE>
<STAPGROOTTE>::=<TYPE 1 INTEGERE KON-
  STANTE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE AL-
  GEMENE SPECIFICATIES>
```

betekenis

Dit kommando kan bij invoer van lange programma's erg nuttig zijn. Het verstrekt namelijk automatisch regelnummers; alleen de kommando's behoeven nog maar te worden ingetoetst. Voorbeeld:

```
NEW
AUTO
10 PRINT "AUTO-TEST"
20
```

– Etcetera, de regelnummers worden automatisch gegeven, alleen de kommando's behoeven te worden ingetoetst.

AUTO kan worden onderbroken door CONTROL-STOP of CONTROL-C.

Indien een regel dient te worden overgeslagen, behoeft alleen een RETURN te worden gegeven; een programmaregel met het overgeslagen nummer wordt dan niet in het programma opgenomen.

Indien een programmaregel reeds bestaat, geeft AUTO dat aan door middel van een ster-teken (*). Bijvoorbeeld (uitgaande van het eerste

stukje programma):

```
AUTO
10*PRINT "TWEEDE TEST"
20 REM NOG EEN REGEL
30
```

(Regel 10 was reeds aanwezig en is nu overgetikt.)

Indien regelnummers met sterren worden overgeslagen door alleen RETURN in te toetsen, blijft de oude programmaregel gehandhaafd.

AUTO kan worden vergezeld van een regelnummer en een stapgrootte. Bijvoorbeeld:

```
AUTO
AUTO 100
AUTO 5,2
AUTO ,2
```

Vanaf regel 10 worden regelnummers met stappen van 10 gegeven.
Vanaf regel 100 worden regelnummers met stappen van 10 gegeven.
De regels 5,7,9,11... worden gegeven.
De regels 0,2,4,... worden gegeven.

Het AUTO-kommando mag, alhoewel dat niet erg zinvol is, in een programmaregel worden opgenomen. RENUM (zie de behandeling van dit kommando) beschouwt ten onrechte echter de achter AUTO opgenomen waarden als regelnummers en probeert deze bij het hernummeren ook mee te nemen, hetgeen vreemde resultaten kan opleveren. Merk op dat AUTO standaard onder funktietoets 2 aanwezig is.

moeilijkheidsgraad . . **zeer moeilijk, kennis van de opbouw van het beeldscherm geheugen alsmede de globale verwerking van de beeldschermcontrolechip is vereist**
soort . . . . . **SYSTEEMVARIABELE**
afkomst . . . . . **BASE is basis**

schrijfwijze

```
BASE(<SPECIFICATIE VAN DE GEWENSTE IN-
  FORMATIE>)
<SPECIFICATIE VAN DE GEWENSTE INFORMA-
  TIE>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze systeemvariabele geeft toegang tot de adressen van verschillende tabellen in het video RAM. Het nummer van de gewenste tabel dient te worden opgenomen tussen de haakjes na het sleutelwoord BASE. Dit nummer mag niet kleiner zijn dan 0 en niet groter zijn dan 19 terwijl een eventuele decimale fractie wordt verwaarloosd.

De adrestabellen kunnen worden gebruikt voor verder onderzoek in het video RAM maar kunnen ook worden veranderd. Onoordeelkundige verandering van de tabeladressen leidt tot onvoorspelbare resultaten waarbij de computer zelfs vaak moet worden uit- en aangeschakeld om hem weer aan de gang te krijgen. Een voorbeeld:

```
NEW
10 BASE(5)=0:SCREEN 1
RUN
```

Op het beeldscherm ontstaan vreemde effecten doordat het adres van een tabel werd verminkt. Schakel de computer uit en weer aan om weer normaal te kunnen werken.

Het sleutelwoord LET is voor BASE niet toegestaan.

Zie voor de betekenis van de betreffende tabeladressen met hun bijbehorende tabellen hoofdstuk 16.

SLEUTELWOORD # BEEP

moeilijkheidsgraad ..... **zeer eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **BEEP is piepen**

schrijfwijze

BEEP

betekenis

Dit kommando veroorzaakt een kort piep-signaal, te gebruiken bij bijvoorbeeld een INPUT als attentiesignaal. Voorbeeld:

```
NEW
10 BEEP:INPUT "UW NAAM ";A$
20 PRINT "GOEDENDAG, ";A$
30 STOP
RUN                                (een attentiesignaal klinkt)
UW NAAM ? FREDERIK
GOEDENDAG, FREDERIK
Break in 30
Ok
```

Indien u een televisieaansluiting heeft, komt het geluid via uw televisieluidspreker. U dient het volume van uw toestel dus wel wat open te draaien.

moeilijkheidsgraad . . **vrij moeilijk, vereist kennis van talstelsels en algemene principes van het computergeheugen**
soort . . . . . **A-FUNKTIE**
afkomst . . . . . **BIN$ is afkorting van binary string – binaire (tweetallige) string**

schrijfwijze

```
BIN$(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft de binaire waarde van de tussen haakjes vermelde uitdrukking weer in string-vorm. De waarde van de numerieke uitdrukking dient te liggen tussen – 32769 en 65536. Indien de waarde van de numerieke uitdrukking gebroken is, wordt de grootste gehele waarde, kleiner dan de gebroken waarde als geldende waarde genomen.
Indien de waarde van de numerieke uitdrukking negatief is, geeft BIN$ een binaire representant, samengesteld volgens de tweecomplementmethode. Bijvoorbeeld:

```
PRINT BIN$(-32768)
1000000000000000
Ok
PRINT BIN$(65535)
1111111111111111
Ok
PRINT BIN$(-1)
1111111111111111
Ok
A$=BIN$(1023)
Ok
PRINT A$
1111111111
Ok
```

moeilijkheidsgraad . . **zeer moeilijk, kennis van machinetaal en computergeheugenopbouw is vaak een vereiste**
soort . . . . . **KOMMANDO**
afkomst . . . . . **BLOAD is afkorting van binairy LOAD – binair (tweetallig) laden**

schrijfwijze

```
BLOAD<BESTANDSNAAM> [,R[,<VERSCHUIVING1>]]
                    [,<VERSCHUIVING2>]
<VERSCHUIVING1>::=<N>
<VERSCHUIVING2>::=<N>\R<...>
<BESTANDSNAAM>::=<A>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
<..>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het is noodzakelijk dat eerst de behandeling van BSAVE wordt doorgenomen.

Met BLOAD kan een eerder via BSAVE vastgelegde geheugeninhoud weer in het geheugen worden teruggeladen. De toegestane bestandsnamen worden onder het OPEN-kommando behandeld.

Met alleen BLOAD, gevolgd door een bestandsnaam, wordt de geheugeninhoud op precies dezelfde wijze weer ingelezen als deze eerder werd vastgelegd.

Indien de letter R werd gespecificeerd, dan wordt direkt na het laden vervolgd met uitvoering van het machinekommando in het byte dat eerder met BSAVE werd aangegeven.

Indien een verschuiving werd gespecificeerd, dan worden de eerder met BSAVE opgegeven adressen (eerste byte, laatste byte en startadres) verhoogd met de opgegeven verschuiving voordat de geheugeninhoud wordt geladen. Deze verschuiving mag niet kleiner zijn dan

–32768 en niet groter dan 65535. Een eventuele decimale fraktie wordt verwaarloosd. Indien de verschuiving negatief is, wordt voorafgaand aan de uitvoering de waarde 65536 hierbij opgeteld.

Indien door de opgegeven verschuiving één van de adressen de waarde 65535 overschrijden, dan wordt dit adres met de waarde 65536 verlaagd.

Indien door een opgegeven opschuiving een gedeelte van de geheugeninhoud over de grens van 65535 zou worden ingelezen, dan wordt dit gedeelte verder vanaf geheugenlokatie 0 ingelezen.

Voorbeeld:

```
NEW
10 BLOAD "CAS:MSX"
RUN                    (band terugspoelen en recorder op afspe-
FOUND:MSX              len zetten) Duur bij FOUND:MSX onge-
Ok                     veer 5 minuten.
```

De eerder in het voorbeeld van BSAVE op band gezette ROM-inhoud wordt weer ingelezen. Praktisch heeft dit voorbeeld geen zin daar het ROM-geheugen niet kan worden veranderd.

Het gebruik van BLOAD is alleen zinvol indien de opbouw van het geheugen van een MSX-computer tot in details bekend is. Hiertoe dient specialistische literatuur te worden geraadpleegd; in dit handboek gaan we hierop niet verder in.

moeilijkheidsgraad . . **zeer moeilijk, kennis van machinetaal en computergeheugen opbouw is vereist**

soort . . . . . **KOMMANDO**

afkomst . . . . . **BSAVE is afkorting van binary save – binair (tweetallig) veiligstellen**

schrijfwijze

```
BSAVE<BESTANDSNAAM>,<EERSTE BYTE>,<LAAT-
 STE BYTE>[,<STARTADRES>]
<BESTANDSNAAM>:::=<A>
<EERSTE BYTE>:::=<N>
<LAATSTE BYTE>:::=<N>
<STARTADRES>:::=<N>
<A>:::=<ZIE ALGEMENE SPECIFICATIES>
<N>:::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met het BSAVE-kommando kunnen delen van het computergeheugen direkt op een extern medium worden opgeslagen. De toegestane bestandsnamen worden bij behandeling van het OPEN-kommando verder omschreven.

Welk gedeelte van het geheugen dient te worden vastgelegd, blijkt uit twee waarden die na de bestandsnaam dienen te worden opgenomen: het adres van het eerste vast te leggen byte en het adres van het laatste vast te leggen byte. Deze twee adressen mogen niet kleiner zijn dan –32768 en niet groter zijn dan 65535. Indien een adres negatief is, wordt de waarde 65536 voor uitvoering bij dit adres opgeteld. Een eventuele decimale fractie wordt verwaarloosd.

Een voorbeeld: in het onderstaande programma wordt de inhoud van het volledige ROM, dus in feite het machinekode-programma dat er voor zorgt dat we in MSX-basic kunnen programmeren, op cassetteband vastgelegd:

```
NEW
10 BSAVE "CAS:MSX",0,32767
```

RUN (eerst cassetterecorder op opnemen zetten)
Ok (duur: ongeveer 5 minuten)

Indien een startadres is opgegeven, dan wordt dat mee opgeslagen. Wanneer (zie BLOAD) het geheugendeel dan later met de R-optie wordt geladen, dan vervolgt de computer onmiddellijk na het laden met de uitvoering van het machinekodebevel dat in het byte op het start-adres is gekodeerd.

Dit startadres mag niet kleiner zijn dan –32768 en niet groter dan 65535. Indien het startadres kleiner is dan nul, wordt er voor uitvoering automatisch eerst de waarde 65536 bij opgeteld.

Het gebruik van BSAVE kan alleen zinvol worden gedaan indien de opbouw van het geheugen van een MSX-computer tot in details bekend is. Hiertoe dient specialistische literatuur te worden geraadpleegd; in dit handboek gaan we daar niet verder op in.

moeilijkheidsgraad . . **normaal, hangt af van gebruikte ROM-cassette**
soort . . . . . **KOMMANDO**
afkomst . . . . . **CALL is (aan)roepen**

schrijfwijze

```
CALL<NAAM KOMMANDO>[(<VARIABELE){,<VA-
    RIABELE>})]
<NAAM KOMMANDO>::=<A>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
<VARIABELE>::=<ZIE ALGEMENE SPECIFI-
CATIES>
```

betekenis

Met het CALL-kommando is het mogelijk om niet-MSX-basic kommando's te activeren. Deze kommando's zijn meestal in een apart stuk ROM-geheugen opgeslagen dat apart, bijvoorbeeld via de daartoe dienende sleuf in de computer, dient te zijn aangesloten. De funkties van deze sleutelwoorden alsmede de te specificeren variabelen hangen natuurlijk af van het aangesloten randapparaat en dienen in de bijgaande gebruiksaanwijzing te zijn verklaard.

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **CDBL is afkorting van convert to double precision value – zet over naar dubbele precisie waarde**

schrijfwijze

```
CDBL(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de waarde van de tussen haakjes vermelde numerieke uitdrukking nadat deze tot dubbele precisie is herleid.

Omdat MSX-basic standaard met dubbele precisie rekent, heeft deze funktie geen zin.

moeilijkheidsgraad . . vrij moeilijk, kennis van de ASCII-tabel is noodzakelijk
soort . . . . . A-FUNKTIE
afkomst . . . . . CHR$ is afkorting van character string – karakter string

schrijfwijze

```
CHR$(<ASCII-KODE>)
<ASCII-KODE>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de letter die korrespondeert met de gegeven ASCII-kode. De ASCII-kode is de interne kode die de computer gebruikt om allerlei tekens in het geheugen op te slaan en moet minimaal gelijk zijn aan nul en maximaal gelijk zijn aan 255. Indien de waarde van de tussen haakjes vermelde uitdrukking een gebroken getal is, wordt de hoogste gehele waarde kleiner dan deze gebroken waarde als geldige waarde genomen.

Zie voor de ASCII-kodering hoofdstuk 17 en voor de hiervan afgeleide MSX-tabel hoofdstuk 18.

Voorbeeld:

```
NEW
10 FOR I=65 TO 90
20 PRINT CHR$(I);
30 NEXT I
RUN
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ok
```

(de ASCII-kodes 65 ... 90 corresponderen met de letters A ... Z).

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **CINT is afkorting van convert to integer value – zet over naar integere waarde**

schrijfwijze

```
CINT(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de waarde van de tussen haakjes vermelde numerieke uitdrukking nadat deze geconverteerd is naar een integere waarde. Hiertoe wordt een decimale fractie verwaarloosd. Indien de resulterende waarde groter is dan 32767 of kleiner is dan –32768, volgt een foutmelding; de waarde kan dan niet tot een integere waarde worden herleid. Voorbeeld:

```
NEW
10 PRINT CINT(12.4)
20 PRINT CINT(4/3)
30 PRINT CINT(120000)
RUN
 12
 1
Overflow in 30
Ok
```

SLEUTELWOORD # CIRCLE

moeilijkheidsgraad ..... **normaal**
soort ..... **KOMMANDO**
afkomst ..... **CIRCLE is cirkel**

schrijfwijze

```
CIRCLE<LOCATIE>,<STRAAL>[,[<KLEUR>][,
   [<AANVANGSHOEK>][,[<EINDHOEK>][,<AF-
   PLATTING>]]]]\CIRCLE<...>,{,}
   <LOKATIE>::=[STEP](<HORIZONTAAL>,<VER-
   TIKAAL>)
<HORIZONTAAL>::=<N>
<VERTIKAAL>::=<N>
<STRAAL>::=<N>
<KLEUR>::=<N>
<AANVANGSHOEK>::=<N>
<EINDHOEK>::=<N>
<AFPLATTING>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
<...>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met dit kommando kunnen cirkels, ellipsen, cirkeldelen en ellipsdelen worden getekend. Voor goed begrip van dit kommando is het vereist dat de behandeling van het PSET-kommando terdege is bestudeerd.

In de meest eenvoudige vorm kunnen we een cirkel laten tekenen door de locatie van het middelpunt van de cirkel en een straal te specificeren. Bijvoorbeeld:

```
NEW
10 SCREEN 2
20 CIRCLE (111,111),50
30 GOTO 30
RUN
```

Een cirkel wordt getekend met het middelpunt op (111,111) en een straal van 50 puntjes. In feite wordt geen zuivere cirkel maar een ellips getekend. Dit is het gevolg van het feit dat de puntjes vertikaal 'dichter op elkaar' zitten dan horizontaal.

De straal dient groter dan of gelijk aan nul te zijn. Daarbij mag de straal niet groter zijn dan 32767. Indien de straal bestaat uit een gebroken waarde, dan wordt deze waarde eerst ontdaan van de decimale fractie.
Een foutje in MSX-basic draagt er zorg voor dan een negatieve straal, mits groter dan of gelijk aan −32768, wel wordt geaccepteerd. De computer tekent in zo'n geval de cirkel echter niet; in plaats hiervan blijft de computer op het CIRCLE-kommando 'hangen' en kan *alleen met CONTROL-STOP* worden onderbroken.

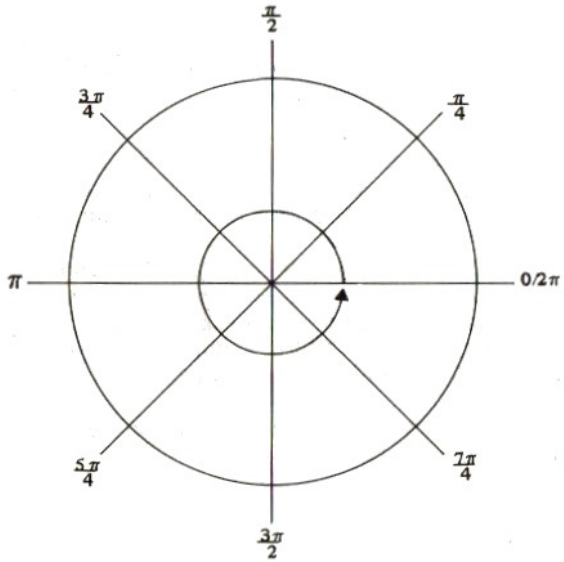
In plaats van de actieve voorgrondkleur kunnen we een andere voorgrondkleur specificeren. Bijvoorbeeld:

```
NEW
10 COLOR ,0,0
20 SCREEN 2
30 CIRCLE (111,111),50,12
40 GOTO 40
RUN
```

Een donkergroene cirkel wordt op een zwarte achtergrond getekend.

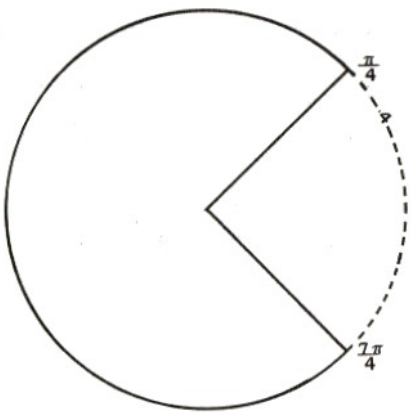
Een aanvangs- en een eindhoek kan worden gespecificeerd. De cirkel wordt dan gedeeltelijk getekend. Om precies te begrijpen welk deel van de cirkel gaat worden getekend, is een klein stukje goniometrie noodzakelijk.

De begin- en eindhoek dienen in radialen te worden opgegeven. De betekenis van de beginhoek en de eindhoek moge blijken uit het volgende schema:



$\pi = 3.141592653589793\ldots$

Om het volgende cirkeldeel te tekenen:



dient een beginhoek van $\frac{\pi}{4}$ en een eindhoek van $\frac{7\pi}{4}$ te worden opgenomen. $\frac{\pi}{4}$ is ongeveer gelijk aan 0.7854 en $\frac{7\pi}{4}$ is ongeveer gelijk aan 5.4978. Om dit cirkeldeel van een bepaalde cirkel te tekenen, kunnen we dus programmeren:

```
NEW
10 SCREEN 2
20 CIRCLE (111,111),50,,0.7854,5.4978
30 GOTO 30
RUN
```

Het betreffende cirkeldeel wordt getekend.

Indien we verbindingen naar het middelpunt willen tekenen, dienen we de begin- en eindhoek negatief op te geven. Bijvoorbeeld:

```
20 CIRCLE (111,111),50,,-0.7854,5.4978
30 GOTO 30
RUN
```

Het cirkeldeel verschijnt; de verbindingen met het middelpunt zijn aangebracht.

Een kleine fout in MSX-basic komt nu aan het licht; als u de tekening goed bekijkt zal blijken dat de verbinding naar het middelpunt soms uit twee vlak naast elkaar lopende lijnen bestaat.

Probeer het bovenstaande voorbeeld ook eens met een SCREEN 1 kommando; precies dezelfde cirkel wordt, zij het in grovere punten, getekend.

Tenslotte kunnen we ook nog een afplattingsfaktor definiëren. Door deze faktor kunnen we in plaats van een cirkel een ellips tekenen. Indien de afplattingsfaktor groter is dan de waarde 1, dan wordt in vertikale richting de straal op de juiste maat gehouden en wordt de straal in horizontale richting gedeeld door deze afplattingsfaktor. Het resultaat is dus een vertikaal gerichte ellips.
Indien de afplattingsfaktor kleiner is dan de waarde 1, dan wordt de straal in horizontale richting op de juiste maat gehouden en wordt de vertikale straal vermenigvuldigd met de afplattingsfaktor. Het resultaat is dus een horizontaal gerichte ellips. Bijvoorbeeld:

```
NEW
10 SCREEN 2
20 FOR P=.2 TO 5 STEP .2
30 CIRCLE (111,111),50,,,,P
40 NEXT P
50 GOTO 50
RUN
```

Op het beeldscherm verschijnt een samenspel van ellipsen.

N.B.: De afplattingsfaktor moet altijd groter zijn dan nul.

De cirkel behoeft niet volledig binnen het beeldschermbereik te liggen; hij mag zelfs volledig buiten het beeldschermbereik worden getekend. Bijvoorbeeld:

```
NEW
10 SCREEN 2
20 CIRCLE (-111,-111),200
30 GOTO 30
RUN
```

Een klein gedeelte van de cirkel, het gedeelte dat binnen het beeldschermbereik valt, wordt afgebeeld.

Uit het volgende voorbeeld blijkt dat indien een ellips of cirkel(deel) wordt getekend, de computer aanneemt dat het laatst getekende punt wordt gevormd door het middelpunt:

```
NEW
10 SCREEN 2
20 CIRCLE (111,111),50
30 LINE -STEP(100,100)
40 GOTO 40
RUN
```

Een cirkel met een vanuit het middelpunt naar rechtsonder lopende lijn is het resultaat. Deze lijn vangt aan in het laatst getekende punt; in dit geval het middelpunt (alhoewel dit niet daadwerkelijk is getekend).

Tot slot volgt hier een combinatievoorbeeld:

```
NEW
10 SCREEN 2
20 X=RND(1)*512-256
30 Y=RND(1)*384-192
40 R=RND(1)*128
50 S=RND(1)*-6.28
60 E=RND(1)*-6.28
70 C=RND(1)*15+1
80 P=.01+10*RND(1)
90 CIRCLE (X,Y),R,C,S,E,P
100 GOTO 20
RUN
```

Het beeldscherm wordt volgeplaatst met cirkel- en ellipssegmenten in diverse kleuren. Vele figuren vallen geheel of gedeeltelijk buiten het beeldscherm. Merk op dat een kleine fout in MSX-basic veroorzaakt dat geheel boven in het beeld regelmatig een 'afkappingslijn' wordt getekend van een gedeeltelijk buiten het beeld vallende cirkel.

SLEUTELWOORD # CLEAR

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **CLEAR is schoonmaken**

schrijfwijze

```
CLEAR[<STRINGRUIMTE>[,<BOVENGRENS GEHEU-
  GEN>]]
<STRINGRUIMTE>::=<N>
<BOVENGRENS GEHEUGEN>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Eenvoudige betekenis

Met het enkele CLEAR-kommando worden de volgende acties ondernomen:

- alle eventueel nog openstaande kanalen worden gesloten (zelfde funktie als het CLOSE-kommando; zie aldaar)
- alle variabelen worden gewist
- alle array-variabelen worden van hun bijbehorende dimensies ontdaan (het DIM-kommando wordt als het ware voor elke variabele ongedaan gemaakt)

Bijvoorbeeld:

```
NEW
10 LET A$="DIT IS EEN TEST"
20 DIM B(20,20)
30 LET B(19,19)=22
40 CLEAR
50 PRINT A$;B(19,19)
RUN
Subscript out of range in 50
Ok
```

Uit bovenstaand programma blijkt dat de dimensionering voor variabele B is tenietgedaan. Hierdoor volgt een foutmelding. Tevens blijkt dat variabele A$ leeg is na uitvoering van CLEAR.

Gevorderd gebruik

Achter het CLEAR-kommando kunnen twee gegevens worden gespecificeerd. Om deze gegevens te vergelijken is het raadzaam om de geheugenindelingstabellen van hoofdstuk 1.3 nog eens te bestuderen.
Met het eerste gegeven kunnen we de maximaal door MSX te gebruiken string-ruimte bepalen. Indien we deze bepaling niet stellen, wordt een standaard grootte van 200 karakters aangenomen. Het onderstaande programma loopt bij een zojuist opgestarte MSX-computer fout:

```
NEW
10 FOR I=1 TO 255:A$=A$+"*":NEXT
RUN
Out of string space in 10
Ok
```

Deze foutmelding volgt uit het feit dat er slechts een string-gebied van 200 karakters is toegewezen. Na intoetsen van

```
CLEAR 1000:RUN
```

zal het programma nu wel tot een goed einde komen; er werd een string-gebied van 1000 karakters toegewezen.

Belangrijk is het om te weten dat MSX-basic bij het uitwerken van alfanumerieke uitdrukkingen een bepaalde werkruimte nodig heeft. Deze werkruimte is net zo groot als de grootste stringlengte die tijdens de uitwerking ontstaat. Hierdoor kan de bovengenoemde foutmelding eerder dan verwacht optreden. In het laatste voorbeeld trad deze reeds op bij de stringlengte van 100 tekens voor A$.

Met het tweede gegeven achter CLEAR kunnen we het laatste door MSX-basic te gebruiken geheugenadres bepalen. Hierdoor (zie geheugentabel) kan een vrije geheugenruimte worden gecreëerd waarin de specialist of zéér ver gevorderde amateur zijn of haar machinekodeprogrammatuur kan opslaan. Bijvoorbeeld:

```
CLEAR ,50000
```

bepaalt dat het MSX-basic niet verder

mag gaan in geheugengebruik dan geheugenadres 50000

`CLEAR 300,50000` bepaalt hetzelfde maar daarbij wordt de stringruimte op 300 karakters bepaald.

SLEUTELWOORD # CLOAD

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **CLOAD is afkorting van cassette load – laden van cassetteband**

schrijfwijze

```
CLOAD[?][<BESTANDSNAAM>][,<...>]
<BESTANDSNAAM>::=<A>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
<...>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Zie eerst de behandeling van het CSAVE-kommando

Met het CLOAD-kommando kunnen programma's die met CSAVE op cassetteband zijn vastgelegd, weer worden ingelezen in het computergeheugen. Indien het vraagteken is opgenomen, wordt een programma niet van cassette geladen doch wordt slechts gecontroleerd of een programma op band overeen komt met het in het computergeheugen opgeslagen programma.
Indien geen programmanaam wordt gespecificeerd, wordt het eerst op de cassetteband voorkomende programma ingelezen. Indien wel een programmanaam is gespecificeerd, dan wordt de band afgezocht naar het juiste programma en wordt alleen dit programma in behandeling genomen. Bijvoorbeeld:

```
NEW
10 REM **********************
20 REM TESTPROGRAMMA
30 REM **********************
40 STOP
```

(nu eerst een cassetteband plaatsen en de cassetterecorder op opnemen zetten)

```
CSAVE "TEST",2
```
(met een schrijfsnelheid van 2400 baud wordt het programma op cassetteband geschreven)

(nu de cassetterecorder stoppen)

```
CLOAD? "TEST"
```

(nu de cassette terugspoelen en de recorder op afspelen zetten)

```
FOUND:TEST
Ok
```

(het programma is *vergeleken* en goed bevonden. Indien de vergelijking fout gaat, wordt de foutmelding Verify error gegeven)

(nu de computer uitschakelen en weer inschakelen; het programma is gegarandeerd uit het geheugen verdwenen.)

```
CLOAD
```

(nu de band terugspoelen en de recorder op afspelen zetten)

```
FOUND:TEST
Ok
LIST
10 REM **********************
20 REM TESTPROGRAMMA
30 REM **********************
40 STOP
Ok
```

(het programma is weer ingelezen)

Merk op dat bij de laatste keer geen programmanaam werd opgegeven; het eerste op de band voorkomende programma werd geladen.

Het is zéér raadzaam om *altijd* een met CSAVE vastgelegd programma te controleren op juistheid met een CLOAD?-kommando, direkt daarna uitgevoerd. De cassetteband is geen erg betrouwbaar opslagmedium; een eventuele 'drop-out' op een cassetteband kan vele uren programmeerwerk te niet doen!

Indien een programma dient te worden ingelezen terwijl we niet zeker weten of het het eerste programma op de band is, dienen we een programmanaam op te geven teneinde zekerheid te hebben dat het

juiste programma wordt geladen. Stel voor dat op een cassetteband twee programma's TEST1 en TEST2 achter elkaar staan en we TEST2 willen laden. We kunnen dan de band terugspelen, de recorder op afspelen zetten en dan ingeven:

```
CLOAD "TEST2"
SKIP:TEST1
FOUND:TEST2
Ok
```

Bij SKIP (= overslaan) vermeldt de computer dat hij een programma wel heeft gevonden maar niet inleest. Bij FOUND (gevonden) geeft de computer aan dat hij inleest.

MSX schrijft geen controle voor op de integriteit (betrouwbaarheid) van ingelezen gegevens. Een verkeerd ingesteld volume of een kwalitatief slechte apparatuur kan leiden tot verminkte programma's bij inlezen zonder dat daarvan tijdens het inlezen melding wordt gemaakt. Zet het volume bij afspelen altijd op ongeveer 80 pct. en kies bij de CSAVE voor de laagste schrijfsnelheid indien u de apparatuur niet geheel vertrouwt.

Een CLOAD behoeft geen snelheidsindicatie te bevatten zoals een CSAVE. De CLOAD bepaalt de snelheid waarmee werd geschreven automatisch en leest ook weer op deze snelheid in.

Het CLOAD-kommando, opgenomen in een programmaregel, heeft tot gevolg dat het betreffende programma eerst wordt ingelezen en dat daarna de Ok-melding verschijnt (het programma loopt dus niet door).

N.B.: Het schrijven naar of laden van band van een groot programma kan tot verscheidene minuten duren.

Merk op dat het CLOAD-kommando standaard onder funktietoets 7 aanwezig is.

SLEUTELWOORD # CLOSE

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **CLOSE is sluiten/dichtmaken**

schrijfwijze

```
CLOSE[[#]<KANAAL>{,[#]<KANAAL>}]
<KANAAL>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Dit kommando wordt bij het OPEN-kommando reeds behandeld; zie aldaar.

SLEUTELWOORD CLS

moeilijkheidsgraad . . . . . **zeer eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **CLS is afkorting van clear screen – scherm schoonmaken**

schrijfwijze

CLS

betekenis

Uitvoering van dit kommando resulteert in het geheel schoonvegen van het beeldscherm. Ook indien een grafisch beeldscherm geactiveerd is, zal dit volledig worden schoongemaakt. De grafische pointer wordt dan niet veranderd.
De cursor wordt na het schoonmaken van het beeldscherm op de linker bovenpositie van het beeldscherm geplaatst. Voorbeeld:

```
NEW
10 INPUT "HOE HEET U ";NAAM$
20 CLS
30 PRINT "HALLO ";NAAM$
40 STOP
```

– Bij het uitvoeren van dit programma vraagt de computer eerst om uw naam. Daarna wordt het beeldscherm schoongemaakt en wordt de tekst HALLO, gevolgd door uw naam, afgedrukt.

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **COLOR is kleur**

schrijfwijze

```
COLOR[<VOORGROND>][,[<ACHTERGROND>][,
  <RAND>]]
<VOORGROND>::=<N>
<ACHTERGROND>::=<N>
<RAND>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met dit kommando kan de kleurinstelling van het beeldscherm worden bepaald. Een voorgrondkleur (de kleur van de letters of tekening), een achtergrondkleur (de kleur van het scherm) en een randkleur (de kleur van de boven- en onderrand bij een grafische instelling van het beeldscherm) kan worden gespecificeerd.

De volgende kleuren kunnen worden gekodeerd:

kleurnummer	kleur of effect
0	doorschijnend
1	zwart
2	groen
3	lichtgroen
4	donkerblauw
5	lichtblauw
6	donkerrood
7	cyaan (een soort blauw)
8	rood
9	lichtrood
10	donkergeel
11	lichtgeel
12	donkergroen
13	magenta (een soort paars)
14	grijs
15	wit

Kleurkode 0 (transparant) resulteert in zwart indien toegepast als achtergrondkleur of randkleur. Toegepast als voorgrondkleur heeft kleurkode 0 tot gevolg dat de diverse afbeeldingen (tekst of tekeningen) niet zichtbaar zijn.

`COLOR 3,1,2` de voorgrondkleur werd lichtgroen gekozen. De achtergrond is zwart en de rand (bij grafisch gebruik) is groen

`COLOR ,,14` alleen de randkleur is naar grijs veranderd

`COLOR 0` de voorgrond is nu doorschijnend en dus onzichtbaar

`COLOR ,0,0` de achtergrond en rand zijn zwart gekleurd; de voorgrondkleur is ongewijzigd

`COLOR 15,4,4` de standaard kleurinstelling (wit op donkerblauw, rand ook donkerblauw). Deze funktie is standaard onder funktietoets 6 aanwezig.

Merk op dat het COLOR sleutelwoord onder funktietoets nummer 1 aanwezig is.

Indien de numerieke uitdrukkingen die de kleuren aangeven, gebroken waarden bevatten, worden deze waarden ontdaan van de decimale fractie.
De kleurwaarden mogen niet kleiner zijn dan 0 en niet groter dan 15.

Tenslotte merken we op dat een COLOR-kommando vóór een SCREEN-kommando dient plaats te vinden teneinde de bedoelde achtergrondkleur en randkleur te realiseren. Indien een COLOR na een SCREEN-kommando wordt uitgevoerd, blijven de oude achtergrond- en randkleur actief.

moeilijkheidsgraad ..... **zeer eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **CONT is afkorting van continue – ga door**

schrijfwijze

```
CONT<...>
<...>:::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met CONT kunnen we een via het STOP-kommando of via de CONTROL-STOP afgebroken programma weer hervatten. Voorbeeld:

```
NEW
10 PRINT "REGEL 1"
20 STOP
30 PRINT "REGEL 2"
40 END
RUN
REGEL 1
Break in 20
CONT
REGEL 2
Ok
```

Indien een programma door middel van een STOP werd onderbroken, begint de hervatting van het programma na CONT bij de volgende programmaregel. Indien het programma echter door CONTROL-STOP werd onderbroken, dan begint de hervatting bij het kommando waarin het programma werd onderbroken. Voorbeeld:

```
NEW
10 GOTO 10
RUN
Break in 10
CONT
```

– Het programma staat in een zogenaamde lus. CONTROL-BREAK wordt nu ingetoetst.
– De oneindige lus wordt weer hervat...

Indien na onderbreking reeds wijzigingen in het programma werden aangebracht, zal de computer melden: Can't CONTINUE. Het is voor de computer dan niet mogelijk om de uitvoering te hervatten.

Merk op dat CONT standaard onder funktietoets 8 aanwezig is.

moeilijkheidsgraad . . **vrij moeilijk, kennis van goniometrie is noodzakelijk**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **COS is afkorting van cosine – cosinus**

schrijfwijze

```
COS(<HOEK>)
<HOEK>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de cosinus van de waarde van de tussen haakjes vermelde uitdrukking. Deze waarde wordt beschouwd als de uitdrukking van een hoek in radialen. Het resultaat ligt uiteraard altijd tussen –1 en 1.

Alhoewel elke waarde voor de hoek is toegestaan, is het in verband met de precisie van berekenen raadzaam om niet te grote hoekmaten te gebruiken.

Voorbeeld:

```
NEW
10 FOR I=0 TO 90
20 LET HOEK=I/180*3.1415926535
30 PRINT I;":";COS(HOEK)
40 NEXT I
RUN
```

(een cosinustabel verschijnt op het beeldscherm. Merk op dat op regel 20 de hoek van graden naar radialen wordt geconverteerd)

SLEUTELWOORD # CSAVE

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **CSAVE is afkorting van cassette save – stel veilig op cassetteband**

schrijfwijze

```
CSAVE<BESTANDSNAAM>[,<SCHRIJFSNELHEID>]
<BESTANDSNAAM>::=<A>
<SCHRIJFSNELHEID>::=<N>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met het CSAVE-kommando kunnen programma's op cassetteband worden vastgelegd. Dit vastleggen gebeurt gekodeerd; het programma wordt op cassetteband vastgelegd in de kodes zoals het zich in het geheugen bevindt; dit in tegenstelling tot het SAVE-kommando waarin het programma in tekstuele vorm wordt vastgelegd.

Programma's, vastgelegd met CSAVE, kunnen met een RUN-kommando niet automatisch worden opgestart; hiertoe dient gebruik te worden gemaakt van het SAVE-kommando.

Met het CSAVE-kommando dient de programmanaam op cassette te worden bepaald. Deze naam MOET minimaal één teken en MAG maximaal 6 tekens bevatten. Een teveel aan tekens wordt verwaarloosd.

Als tweede gegeven bij de CSAVE kan eventueel een schrijfsnelheid worden gespecificeerd. De numerieke uitdrukking die de schrijfsnelheid bepaalt, mag gelijk zijn aan 1 of 2. Indien deze numerieke uitdrukking een gebroken waarde bevat, dan wordt deze herleid naar de grootste gehele waarde kleiner dan deze gebroken waarden. De snelheidsaanduiding heeft de volgende betekenis:

- er wordt met 1200 baud geschreven, hetgeen wil zeggen dat er 1200 bits (ongeveer 150 karakters) per seconde worden overgedragen op de band
- er wordt met 2400 baud geschreven; de dubbele snelheid dus.

Indien geen schrijfsnelheid wordt gespecificeerd, wordt automatisch een snelheid van 1200 baud aangenomen of de snelheid die bij het SCREEN-kommando werd bepaald (zie de behandeling van SCREEN).

Indien men over een cassetterecorder van goede kwaliteit beschikt, en daarbij kwalitatief goede cassettebanden gebruikt, kan voor de hoogste snelheid worden gekozen. Echter, wanneer de kwaliteiten wat lager liggen, is het zéér raadzaam om de laagste snelheid te kiezen.

MSX schrijft geen lees/schrijfcontrole voor; een slechte kwaliteit van apparatuur kan veroorzaken dat er fouten optreden tijdens het vastleggen.

voorbeeld

Zie het voorbeeld bij CLOAD

SLEUTELWOORD CSNG

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **CSNG is afkorting van convert to single precision value – zet over naar enkelvoudige precisie waarde**

schrijfwijze

```
CSNG(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de waarde van de tussen haakjes vermelde numerieke uitdrukking nadat deze is herleid naar enkelvoudige precisie. Bijvoorbeeld:

```
NEW
10 PRINT 1/3
20 PRINT CSNG(1/3)
RUN
 .33333333333333
 .333333
Ok
```

moeilijkheidsgraad . . . . . **zeer eenvoudig**
soort . . . . . **SYSTEEMVARIABELE**
afkomst . . . . . **CSRLIN is afkorting van cursorline – cursor regel**

schrijfwijze

CSRLIN

betekenis

Deze systeemvariabele geeft als resultaat het regelnummer waarop de cursor zich op het moment bevindt. Voorbeeld:

```
NEW
10 FOR I=0 TO 5:LOCATE ,I
20 PRINT CSRLIN:NEXT:STOP
RUN
```

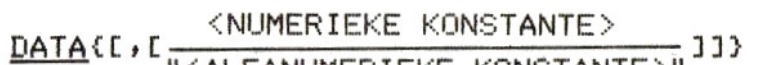
(In de linkerbovenhoek van het beeld verschijnen onder elkaar de cijfers 0 tot en met 5.)

De systeemvariabele CSRLIN kan geen waarde worden toegekend; de waarde kan alleen worden gebruikt.

SLEUTELWOORD # DATA

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **DATA is gegevens**

schrijfwijze



betekenis

Behalve het sleutelwoord DATA worden ook de sleutelwoorden READ en RESTORE hier behandeld.

Met het DATA-sleutelwoord kunnen we vaste gegevens in een programma opnemen. Deze gegevens mogen bestaan uit numerieke konstanten of alfanumerieke konstanten. Bijvoorbeeld:

```
10 DATA 1.2,3.4,123.6,-12.33,"MAANDAG",
"PIETJE"
```

De diverse gegevens dienen door middel van een komma van elkaar te worden gescheiden.

De op deze wijze in een programma opgenomen gegevens kunnen door middel van een READ-kommando in een variabele worden ingelezen. Bijvoorbeeld:

```
NEW
10 DATA "MAANDAG","DINSDAG","WOENSDAG"
20 DATA "DONDERDAG","VRIJDAG","ZATERDAG"
30 DATA "ZONDAG"
40 FOR I=1 TO 7:READ DAG$
50 PRINT DAG$:NEXT I:STOP
RUN
MAANDAG
DINSDAG
```

```
WOENSDAG
DONDERDAG
VRIJDAG
ZATERDAG
ZONDAG
Break in 50
Ok
```

In het laatste voorbeeld worden de alfanumerieke konstanten MAANDAG...VRIJDAG achtereenvolgens in de variabele DAG$ ingelezen en vervolgens afgedrukt op het beeldscherm. Bij een READ 'onthoudt' de computer waar hij is gebleven met inlezen. Is MAANDAG reeds ingelezen, dan zal de volgende keer dat een READ wordt ingelezen, automatisch DINSDAG worden ingelezen. Het maakt hiervoor niet uit wáár precies de konstanten in een DATA-commando zijn genoemd; de computer begint altijd bij de eerste konstante en werkt deze konstante in volgorde van voorkomen af. Bijvoorbeeld:

```
NEW
10 DATA 33,44,55,66
20 READ A:PRINT A:GOTO 20
30 DATA 77,88,99
RUN
 33
 44
 55
 66
 77
 88
 99
Out of DATA in 20
Ok
```

We zien dat alle genoemde konstanten worden ingelezen en afgedrukt. Het maakt niet uit of de gegevens in een DATA-kommando voor of na de leesopdracht zijn genoemd. Pas als er geen volgende gegevens meer kunnen worden gevonden, geeft de computer een foutmelding.

Met een RESTORE-kommando kunnen we de computer opdracht

geven, bij de volgende READ-opdracht weer bij een bepaald gegeven te beginnen met inlezen. Bijvoorbeeld:

```
NEW
10 DATA 1,2,3,4,5
20 RESTORE 20
30 DATA 111,222,333
40 READ A,B,C
50 PRINT A;B;C
RUN
 111   222   333
Ok
```

Met de RESTORE 20 opdracht gaven we in bovenstaand voorbeeld aan dat het inlezen op of na programmaregel 20 diende te geschieden. Alleen een RESTORE-opdracht (dus zonder regelnummer) heeft tot gevolg dat de computer weer bij het allereerste DATA-gegeven begint. Bijvoorbeeld:

```
NEW
10 READ A:IF A=0 THEN RESTORE:GOTO 10
20 PRINT A;:READ A$:PRINT A$:GOTO 10
1000 DATA 1,"ZONDAG",2,"MAANDAG",
3,"DINSDAG"
1010 DATA 4,"WOENSDAG",5,"DONDERDAG",6
1020 DATA "VRIJDAG",7,"ZATERDAG",0
RUN
 1 ZONDAG
 2 MAANDAG
 3 DINSDAG
 4 WOENSDAG
 5 DONDERDAG
 6 VRIJDAG
 7 ZATERDAG
 1 MAANDAG
 2 DINS...
```

(etcetera, de dagen van de week blijven

op het beeldscherm voorbijkomen. Onderbreken met CONTROL-STOP)

In bovenstaand voorbeeld wordt steeds een dagnummer en een dagnaam ingelezen; in variabele A en A$. Indien op regel 10 echter voor variabele A een nul-waarde wordt ingelezen, wordt een RESTORE uitgevoerd waarna opnieuw de READ wordt uitgevoerd. Na deze RESTORE wordt op regel 10 weer de waarde 1 voor variabele A ingelezen en wordt op regel 20 de waarde ZONDAG voor variabele A$ ingelezen.

SLEUTELWOORD # DEF FN

moeilijkheidsgraad . . **normaal, enige elementaire kennis van funktie-theorie is een voordeel**

soort . . . . . **KOMMANDO**

afkomst . . . . . **DEF FN is afkorting van define function – definieer funktie/bewerking**

schrijfwijze

```
DEFFN<ALFANUMERIEKE VARIABELE-NAAM>[(<VA-
  RIABELE-NAAM>{,<VARIABELE-NAAM>}]=<A>!
  DEFFN<NUMERIEKE VARIABELE-NAAM>[(<VA-
  RIABELE-NAAM{,<VARIABELE-NAAM>}]=<N>
<ALFANUMERIEKE VARIABELE-NAAM>::=<ZIE AL-
  GEMENE SPECIFICATIES>
<NUMERIEKE VARIABELE-NAAM>::=<ZIE ALGEME-
  NE SPECIFICATIES>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

MSX-basic bevat vele ingebouwde funkties. SQR (worteltrekken), INT (integere waarde bepalen) en SIN (sinus berekenen) zijn enkele van deze funkties.
Het kan echter zijn dat men in een programma een specifieke funktie nodig heeft die helaas niet in MSX-basic bestaat. In dat geval kan deze funktie door middel van DEF FN zelf worden ontworpen.

Allereerst dient deze te ontwerpen funktie een naam te hebben. We geven de funktie de naam van een alfanumerieke variabele wanneer de uitkomst van deze funktie alfanumeriek gaat worden en we geven de funktie de naam van een numerieke variabele indien de uitkomst van deze funktie een numerieke wordt.

Tussen haakjes komen dan variabelenamen te staan. Deze variabelenamen zijn slechts namen en vormen *geen echte variabelen*. Zij dienen slechts om aan te geven waar later bij het uitvoeren van de funktie de op dat moment ingevulde variabelen moeten komen te staan.

We ontwerpen als eerste een funktie met een numeriek resultaat. Deze funktie noemen we HYP. Hij rekent de lengte van de schuine zijde van een rechthoekige driehoek uit wanneer de lengten van de twee overige zijden gegeven zijn. De funktie luidt:

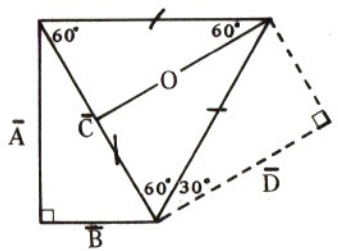
```
NEW
10 DEF FN HYP(A,B)=SQR(A*A+B*B)
```

Later kunnen we alleen met het woordje HYP deze funktie weer oproepen. Bijvoorbeeld:

```
20 PRINT FN HYP(12,13)
RUN
 17.691806012953
Ok
```

De ene funktie mag de andere funktie gebruiken. Zo kunnen we een volgende funktie ontwerpen die, uitgaande van de lengte van de schuine zijde van de rechthoekige driehoek, de oppervlakte berekent van de gelijkbenige driehoek met deze lengte. Zie de constructietekening. De funktie luidt:

```
20 DEF FN OPP(A,B)=SQR(3)*
FN HYP(A,B)^2/4
30 PRINT FN OPP(12,13)
RUN
 135.53297569224
Ok
```



$$\overline{C}=\sqrt{(A^2+B^2)}$$
$$\overline{D}=C\cdot\cos(30^\circ)=2\sqrt{3}\cdot C$$
$$O=\frac{1}{4}\sqrt{3}\ \overline{C}^2$$

Uiteindelijk zouden we nog een derde funktie in het leven kunnen roe-

pen die dezelfde berekening uitvoert, uitgaande van een *gelijkbenige* driehoek. Er behoeft dan slechts één zijde te worden opgegeven:

```
30 DEF FN OVL(A)=FN OPP(A,A)
40 PRINT FN OVL(12)
RUN
 124.70765814495
Ok
```

Een éénvoudige funktie die erg nuttig kan zijn, is bijvoorbeeld een afrondfunktie op twee cijfers (geldbedragen):

```
NEW
10 DEF FN AFR(X)=FIX(100*X)/100
20 PRINT FN AFR(100*RND(1)-50)
30 GOTO 20
RUN
```

Op het beeldscherm worden willekeurige getallen afgedrukt. Zij zijn allemaal op twee cijfers na de komma afgerond.

Een voorbeeld van een alfanumerieke funktie: we definieren een funktie die uit een gegeven string een tussen haakjes geplaatst gedeelte isoleert:

```
NEW
10 DEF FN HAA$(A$)=MID$(A$,INSTR(A$,
"(")+1,INSTR(A$,")")-INSTR(A$,"(")-1)
20 PRINT FN HAA$("HIJ ZEI (MET LUIDE
STEM) HOERA!!!")
30 Q$="DEZE TAAL (BASIC) IS NIET
MOEILIJK."
40 PRINT FN HAA$(Q$)
RUN
MET LUIDE STEM
BASIC
Ok
```

Moeilijke funkties kunnen als zodanig één keer worden uitgedacht en opgenomen en voortaan in het programma eenvoudig worden toegepast.

SLEUTELWOORD # DEFINT

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **DEFINT is afkorting van define integer variables – definieer integere variabelen**

schrijfwijze

```
DEFINT<LETTER>[-<LETTER>]{,<LETTER>
   [-<LETTER>]}
<LETTER>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Dit kommando wordt onder DEFSTR behandeld. Zie aldaar.

SLEUTELWOORD DEFDBL

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **DEFDBL is afkorting van define double precision variables – definieer variabelen met dubbele precisie**

schrijfwijze

```
DEFDBL<LETTER>[-<LETTER>]{,<LETTER>
   [-<LETTER>]}
<LETTER>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Dit kommando wordt onder DEFSTR behandeld. Zie aldaar.

SLEUTELWOORD # DEFSNG

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **DEFSGN is afkorting van define single precision variables – definieer variabelen met enkelvoudige precisie**

schrijfwijze

```
DEFSNG<LETTER>[-<LETTER>]{,<LETTER>
   [-<LETTER>]}
<LETTER>:::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Dit kommando wordt onder DEFSTR behandeld. Zie aldaar.

SLEUTELWOORD # DEFSTR

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **DEFSTR is afkorting van define string variables – definieer string-variabelen**

schrijfwijze

```
DEFSTR<LETTER>[-<LETTER>]{,<LETTER>
   [-<LETTER>]}
<LETTER>:::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Voor een goed begrip van onderstaande funkties is het raadzaam, de hoofdstukken 5 en 6 nog eens door te nemen.

In MSX-basic wordt een variabele, indien de naam niet één van de achtervoegsels $, %, ! of # heeft, automatisch gemaakt tot een variabele met dubbele precisie. Indien we bijvoorbeeld programmeren:

```
NEW
10 LET A=1/3
20 PRINT A
RUN
 .33333333333333
Ok
```

Dan zien we dat voor de variabele A automatisch een numerieke variabele werd gekozen met een precisie van 14 cijfers; een dubbele precisie dus. Als het ware werd automatisch voor het achtervoegsel # gekozen.

Dit automatisme kunnen we sturen door gebruik van de DEFSTR, de DEFINT, de DEFSNG en de DEFDBL. Achter het betreffende sleutelwoord dienen letterreeksen te worden aangegeven. Na het uitvoeren van één van de genoemde kommando's wordt voortaan voor een variabele met een beginletter, genoemd in een letterreeks, een ander variabeletype gehanteerd (indien het type niet via een achtervoegsel is bepaald).

De sleutelwoorden hebben het volgende effekt:

DEFSTR	als standaard variabele-type wordt het type alfanumerieke variabele gehanteerd
DEFINT	als standaard variabele-type wordt het type integere variabele gehanteerd
DEFSNG	als standaard variabele-type wordt het type variabele met enkelvoudige precisie gehanteerd
DEFDBL	als standaard variabele-type wordt het type variabele met dubbele precisie gehanteerd

De letterreeksen die achter een DEFSTR, DEFINT, DEFSNG of DEFDBL worden opgenomen, kunnen uit twee soorten bestaan, namelijk de enkelvoudige opsomming en de reeksaanduiding. Bij de enkelvoudige opsomming worden alle geldende letters, gescheiden door een komma, opgegeven. Bij de reeksaanduiding wordt de eerste geldende letter, een min-teken en de laatste geldende letter opgegeven. Bijvoorbeeld:

DEFSTR A, C, F	alle variabelen waarvan de naam begint met een A, een C of een F en geen achtervoegsel hebben ($, %, ! of #), worden **automatisch** beschouwd als alfanumerieke variabelen (krijgen automatisch een $ als achtervoegsel)
DEFSNG C–Q, Z	alle variabelen waarvan de naam begint met een C, D, E...Q of met een Z en daarbij geen achtervoegsel hebben, worden automatisch beschouwd als variabelen met enkelvoudige precisie (krijgen als het ware automatisch een ! als achtervoegsel)

Voorbeeld:

```
NEW
10 DEFSTR A-C
20 A="ALFANU"
30 B="MERIEKE V"
40 C="ARIABELEN"
50 PRINT A;B;C
55 PRINT A$;B$;C$
```

```
60 A%=32000:PRINT A%:STOP
RUN
ALFANUMERIEKE VARIABELEN
ALFANUMERIEKE VARIABELEN
 32000
Break in 60
Ok
```

We zien dat de variabelen A, B, C, indien niet met achtervoegsel gebruikt, na de DEFSTR A–C automatisch als alfanumerieke variabelen worden beschouwd. Op programmaregel 60 zien we echter dat wanneer een achtervoegsel wordt gebruikt, dit achtervoegsel geldend is. Ook zien we dat het achtervoegsel (regel 55) eventueel wel mag worden gebruikt.

Merk op dat een reeksaanduiding waarbij de tweede letter kleiner is dan de eerste letter (bijvoorbeeld een DEFDBL C–A), ten onrechte een syntax error (fout in schrijfwijze) veroorzaakt. Deze foutmelding is niet de juiste; een kleine fout in MSX-basic.

moeilijkheidsgraad . . **zeer moeilijk, kennis van machinetaal en of ROM-inhoud is noodzakelijk**
soort . . . . . **KOMMANDO**
afkomst . . . . . **DEFUSR is afkorting van define user function – definieer een door de gebruiker ontworpen (niet basic) funktie**

schrijfwijze

```
DEFUSR[<CIJFER>]=<GEHEUGENADRES>
<CIJFER>::=<ZIE ALGEMENE SPECIFICATIES>
<GEHEUGENADRES>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met dit kommando kan een funktie worden geactiveerd die eerder in machinekode in het computergeheugen werd opgenomen. Maximaal 10 van deze funkties kunnen worden gedefinieerd. Deze funkties kunnen genummerd worden vanaf 0 tot en met 9; het cijfer direkt achter het DEF USR-sleutelwoord. Indien dit cijfer wordt weggelaten, wordt een 0 aangenomen. Na het gelijkteken dient een geheugenadres te worden opgenomen. Dit adres geeft aan op welk byte in het geheugen het eerste uit te voeren machinekommando staat voor de betreffende funktie. Het geheugenadres mag niet kleiner zijn dan –32768 en niet groter dan 65535. Indien de waarde van het geheugenadres kleiner is dan 0, wordt er ter berekening van het juiste adres de waarde 65536 bij opgeteld.

Nadat de funktie op deze wijze is gedefinieerd, kan hij met het sleutelwoord USR weer worden opgeroepen. Na USR dient het nummer van de funktie te staan (0 ... 9). Indien dit nummer wordt weggelaten, wordt een 0 aangenomen. Vervolgens dient tussen haakjes een uitdrukking te worden opgenomen. Deze uitdrukking bevat al dan niet een waarde die naar het betreffende stuk machinekode-programma moet worden doorgespeeld maar moet altijd worden opgenomen. Indien deze waarde verder niet van belang is, kan het beste een cijfer 0 worden opgenomen.

USR vormt een funktie net zoals bijvoorbeeld de INT en de ABS

funktie. Als zodanig kan ook deze funktie worden opgenomen in een uitdrukking of worden afgedrukt met een PRINT-kommando e.d. Steeds als de USR-funktie op welke wijze dan ook wordt aangesproken, wordt het stuk machinekodeprogrammatuur op het aangegeven geheugenadres uitgevoerd. MSX-basic gaat er dan van uit dat deze machinekodeprogrammatuur uiteindelijk resulteert in een waarde die in een uitdrukking verder kan worden verwerkt.

Een voorbeeld: Het volgende programma start een machine-routine op geheugenadres 0. Op dit adres start de computer altijd op wanneer de stroom wordt in geschakeld en hier bevindt zich dan ook de opstart-routine. Het effect van dit programma is, dat de machine opnieuw wordt opgestart als ware de stroom even uitgeschakeld geweest. Ook de copyright-melding verschijnt weer op scherm. Een eventueel programma is verdwenen:

```
NEW
10 DEF USR5=0
20 PRINT USR5(0)
RUN
```

Het samenstellen van een stuk machinetaal-programmatuur vereist een zeer uitgebreide kennis van zaken in verband met de Z80 microprocessor. Ook dient de samenstelling van de MSX-computer alsmede de samenstelling van het computergeheugen te worden beheerst. Hiertoe dient specialistische literatuur te worden geraadpleegd; in dit handboek wordt hierop niet verder ingegaan.

SLEUTELWOORD # DELETE

moeilijkheidsgraad ..... **zeer eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **DELETE is verwijderen**

schrijfwijze

```
DELETE[<REGELNUMMER>][-<REGELNUMMER>]
<REGELNUMMER>::=<TYPE 1 INTEGERE KONSTAN-
  TE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE ALGE-
  MENE SPECIFICATIES>
```

betekenis

Met dit kommando kunt u:

- Een regel uit het programma verwijderen. Geef in dat geval DELETE in, gevolgd door het regelnummer of een punt voor het laatst behandelde regelnummer.
- Een eerste gedeelte van een programma verwijderen. Geef hiertoe DELETE in, gevolgd door een minteken en het laatste te verwijderen regelnummer.
- Een laatste gedeelte van een programma verwijderen. Geef hiertoe DELETE in, gevolgd door het eerste te verwijderen regelnummer en een minteken.
- Een middengedeelte van een programma verwijderen. Geef hiertoe DELETE in, gevolgd door het eerste te verwijderen regelnummer, een minteken en het laaste te verwijderen regelnummer.

Voorbeelden:

```
DELETE 60
```

regel 10 wordt verwijderd

```
DELETE 60-120
```

alle regels vanaf 60 tot en met 120 worden verwijderd

```
DELETE -90
```

alle regels tot en met 90 worden verwijderd

```
DELETE .
```

de laatste behandelde regel wordt verwijderd

De aangegeven regelnummers dienen bestaande regelnummers te zijn.

Indien DELETE in een programmaregel is opgenomen, worden de daarbij vermelde regelnummers meehernummerd (zie de behandeling van RENUM).

moeilijkheidsgraad . . **normaal, zeer elementaire kennis van matrix-berekeningen is een voordeel**
soort . . . . . **KOMMANDO**
afkomst . . . . . **DIM is afkorting van dimension – dimensie**

schrijfwijze

```
DIM<RIJ VAN ARRAY-VARIABELEN>
<RIJ VAN ARRAY-VARABELEN>::=<VARIABELE
  NAAM>[(<DIMENSIE>{,<DIMENSIE>})][,<RIJ
  VAN ARRAY-VARIABELEN>]
<DIMENSIE>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
<VARIABELE NAAM>::=<ZIE ALGEMENE SPECIFI-
  CATIES>
```

betekenis

Met het DIM-kommando kunnen we zogenaamde array-variabelen toewijzen. Een array-variabele is te vergelijken met een tabel waarin we meerdere waarden kunnen opslaan.
Zowel het aantal dimensies als het aantal elementen per dimensie is praktisch onbeperkt. Theoretisch is het maximum aantal dimensies gelijk aan 255 en het maximum aantal elementen per dimensie gelijk aan 65535. Indien tussen haakjes een numerieke uitdrukking voorkomt met een gebroken waarde, dan wordt de grootste gehele waarde kleiner dan deze gebroken waarde als geldige waarde genomen.
Een array kan één-dimensionaal (een rij van waarden), twee-dimensionaal (een matrix van waarden), drie-dimensionaal (een ruimtelijke matrix) maar ook méér-dimensionaal zijn. Alhoewel het werken met meerdere dimensies een abstract denkvermogen vereist, kan dit toch wel eens erg gemakkelijk zijn.

Voorbeeld:

```
NEW
10 DIM A(3,3):LET T=0
20 FOR I=0 TO 3:FOR J=0 TO 3
```

```
30 LET A(I,J)=T:LET T=T+1
40 NEXT J,I
50 INPUT "RIJ ";R:IF NOT(R) THEN STOP
60 INPUT "KOLOM ";K
70 PRINT "DE WAARDE OP";R;",";K
80 PRINT "IS";A(R,K):GOTO 60
RUN
RIJ ? 2
KOLOM ? 2
DE WAARDE OP 2 , 2
IS 10
RIJ ? 3
KOLOM ? 1
DE WAARDE OP 3 , 1
IS 13
RIJ ? 0
Break in 50
Ok
```

Indien een variabele-naam in een dim-kommando wordt genoemd zonder dimensiebepaling, dan wordt deze variabele slechts toegewezen en op nul gesteld.

Merk op dat meer arrays in een dim-kommando kunnen worden gedimensioneerd. Bijvoorbeeld:

```
10 DIM A(3,2),B$(2),C(1,2,5),B%(100,5)
```

Ook alfanumerieke tabellen kunnen worden aangelegd. Elk element in een alfanumerieke tabel mag een andere lengte hebben. Bijvoorbeeld:

```
NEW
10 DIM NAAM$(100)
20 INPUT "NUMMER ";NUMMER%
30 IF NUMMER%<0 OR NUMMER%>100 GOTO 20
40 PRINT "OUDE NAAM: ";NAAM$(NUMMER%)
50 INPUT "NIEUWE NAAM ";NAAM$(NUMMER%)
60 GOTO 20
```

```
RUN
NUMMER ? 1
OUDE NAAM:
NIEUWE NAAM ? FERDINAND
NUMMER ? 55
OUDE NAAM:
NIEUWE NAAM ? KAREL DE GROTE
NUMMER ? 1
OUDE NAAM: FERDINAND
NIEUWE NAAM ? FREDERIK
NUMMER ? 55
OUDE NAAM: KAREL DE GROTE
NIEUWE NAAM ?
```

SLEUTELWOORD # DRAW

moeilijkheidsgraad . . . . . **vrij moeilijk**
soort . . . . . **KOMMANDO**
afkomst . . . . . **DRAW is tekenen**

schrijfwijze

```
DRAW<TEKENAANWIJZING>
<TEKENAANWIJZING>::=<A>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Voor goed begrip van dit kommando is het noodzakelijk dat de PSET terdege is bestudeerd.

Met het DRAW-kommando kunnen tekeningen op het beeldscherm worden gemaakt. Achter het DRAW-kommando dient dan een alfanumerieke uitdrukking te worden opgenomen die de teken-aanwijzingen voor de computer bevatten. Deze alfanumerieke uitdrukking dient te zijn opgesteld volgens de GML (Graphics Macro Language) richtlijnen. GML kan als een eenvoudige, aparte taal worden gezien die door Microsoft is ontwikkeld en alleen een grafische toepassing heeft.

Binnen GML kennen we volgende, éénletterige kommando's:

M...,... de M staat voor MOVE. De getallen, achter MOVE opgenomen, geven aan naar welk punt moet worden gegaan op het beeldscherm. De geadresseerde beeldschermlocatie mag zich buiten het scherm bevinden. Vanuit het laatst getekende punt wordt een lijn getrokken naar het aangegeven punt. Bijvoorbeeld:

```
NEW
10 SCREEN 2
20 DRAW "M100,100M200,100M20,30M22,33"
30 GOTO 30
RUN
```

Een lijnenspel wordt getekend.

Indien voor de achter M opgenomen getallen een plus- of minteken staat, wordt de verplaatsing relatief beschouwd ten opzichte van het laatst getekende punt. Een M+100,100 heeft dan bijvoorbeeld tot gevolg dat een lijn vanuit het laatst getekende punt wordt getrokken naar een punt dat op de vertikale locatie 100 ligt en op een horizontale locatie, 100 beeldschermpunten verder naar rechts dan die van het laatst getekende punt.

Voorbeeld:

```
NEW
10 SCREEN 2
20 DRAW "M100,100M+11,0M0,-11M-11,0M0,
+11"
30 GOTO 30
RUN
```

Eerst wordt een lijn vanuit het laatst getekende punt naar (100,100) getrokken. Vervolgens wordt een vierkantje getekend door achtereenvolgens een lijn van 11 beeldpunten naar rechts, naar beneden, naar links en weer naar boven te tekenen.

U... de U staat voor UP. Vanuit het laatst getekende punt wordt een lijn naar boven getrokken ter lengte van een aantal aangegeven beeldschermpunten.

R... de R staat voor RIGHT. Vanuit het laatst getekende punt wordt een lijn naar rechts getrokken ter lengte van het aantal aangegeven beeldschermpunten.

D... de D staat voor DOWN. Vanuit het laatst getekende punt wordt een lijn naar beneden getrokken ter lengte van het aantal aangegeven beeldschermpunten.

L... de L staat voor LEFT. Vanuit het laatst getekende punt wordt een lijn naar links getrokken ter lengte van het aantal aangegeven beeldschermpunten.

Bijvoorbeeld:

```
NEW
10 SCREEN 2
20 DRAW "M100,100U20R20D20L20"
```

```
30 GOTO 30
RUN
```

Vanuit het laatst getekende punt wordt eerst een lijn naar (100,100) getrokken. Daarna wordt vanuit dit punt een vierkantje getekend; twintig beeldpunten naar boven, naar rechts, naar beneden en weer naar links.

E... vanuit het laatst getekende punt wordt een lijn getrokken naar een punt, het aantal aangegeven beeldschermpunten hoger en verder naar rechts gelegen. Er wordt dus een lijn rechts schuin naar boven getekend.

F... vanuit het laatst getekende punt wordt een lijn getrokken naar een punt, het aantal aangegeven beeldschermpunten lager en verder naar rechts gelegen. Er wordt dus een lijn rechts schuin naar beneden getekend.

G... vanuit het laatst getekende punt wordt een lijn getrokken naar een punt, het aantal aangegeven beeldschermpunten lager en verder naar links gelegen. Er wordt dus een lijn links schuin naar beneden getekend.

H... vanuit het laatst getekende punt wordt een lijn getrokken naar een punt, het aantal aangegeven beeldschermpunten hoger en verder naar links gelegen. Er wordt dus een lijn links schuin naar boven getekend.

Voorbeeld:

```
NEW
10 SCREEN 2
20 DRAW "M100,100E10F10G10H10"
30 GOTO 30
RUN
```

Vanuit het laatst getekende punt wordt eerst een lijn naar (100,100) getekend. Daarna wordt vanuit dit punt een ruitfiguur getekend; tien beeldpunten naar rechtsboven, tien beeldpunten naar rechtsonder, tien beeldpunten naar linksonder en tien beeldpunten naar linksboven.

B... de B staat voor BLANK. De eerst volgende tekenopdracht

wordt wel uitgevoerd maar de lijn wordt niet getrokken. In feite wordt met behulp van deze funktie een ander punt aangewezen als zijnde de laatst getekende. Bijvoorbeeld:

```
NEW
10 SCREEN 2
20 DRAW "BM100,100E10F10G10H10"
30 GOTO 30
RUN
```

Net als in het laatste voorbeeld wordt ook hier een ruitfiguur getekend. Echter, de eerste lijn (M100,100) die vanuit het laatst getekende punt naar (100,100) zou moeten lopen, wordt niet getekend.

N... de eerstvolgende tekenopdracht wordt niet van begin- naar eindpunt maar van eind- naar beginpunt uitgevoerd. Hierdoor blijft het laatst getekende punt vóór deze opdracht ook het laatst getekende punt ná deze opdracht. Bijvoorbeeld:

```
NEW
10 SCREEN 2
20 DRAW "BM100,100NU50ND50NL50NR50NE50NF
50NG50NH50"
30 GOTO 30
RUN
```

Steeds wordt vanuit punt (100,100) een lijn getrokken naar diverse richtingen waardoor een sterfiguur ontstaat.

A... de letter A staat voor ANGLE. Een hoek van 0 graden (A0), 90 graden (A1), 180 graden (A2) of 270 graden (A3) kan worden gespecificeerd. Alle volgende tekenopdrachten (tot een volgend A-kommando) worden het aantal aangegeven graden naar links gedraaid. Bijvoorbeeld:

```
NEW
10 LINE INPUT "HOEK:";H$
20 SCREEN 2
30 DRAW "A0BM100,100A"+H$+"D20R20U20L20
E10F10"
```

```
40 GOTO 40
RUN
HOEK:
```

Een hoek (0, 1, 2 of 3) kan worden ingegeven. Een eenvoudig tekeningetje van een huis wordt, gedraaid volgens de aangegeven hoek, getekend. RUN dit programma diverse malen met verschillende hoeken.

C... de C staat voor COLOR. Een voorgrondkleur (C0... C15) kan worden opgegeven. Indien geen kleur wordt gespecificeerd, wordt de op dat moment actieve voorgrondkleur als actieve kleur gebruikt. Een eenmaal gespecificeerde kleur blijft geldig totdat binnen het DRAW-kommando een volgende kleur wordt gespecificeerd. Bijvoorbeeld:

```
15 LINE INPUT "KLEUR:";C$
25 DRAW "C"+C$
RUN
HOEK:
```

Bij het vorige voorbeeld werden twee programmaregels toegevoegd. Nu kan ook een kleurkode worden ingegeven waarna het tekeningetje in de gewenste kleur wordt uitgevoerd.

S... de S staat voor SCALE. Een schaal kan worden opgegeven. Het achter de S opgenomen getal, gedeeld door vier, vormt het aantal werkelijk getekende puntjes ten opzichte van één geprogrammeerd puntje. Indien geen schaalfaktor wordt opgegeven, of een S0 of een S4 wordt opgegeven, komt een getekend puntje precies overeen met een geprogrammeerd puntje. Een S1 heeft tot gevolg dat een tekening vier maal zo klein wordt uitgevoerd terwijl een S16 juist tot gevolg heeft dat een tekening vier maal zo groot wordt uitgevoerd. Het M-kommando met absolute adressering (dus zonder gebruik van een plus- of minteken) wordt door het S-kommando als enige niet beïnvloed. Het S-kommando blijft geldig totdat een nieuw S-kommando wordt gegeven. Bijvoorbeeld:

```
16 LINE INPUT "SCHAAL:";S$
26 DRAW "S"+S$
```

```
RUN
HOEK:
```

Alleen regelnummers 16 en 26 werden aan het vorige voorbeeld toegevoegd. Nu kan ook een schaal worden opgegeven. Afhankelijk van de opgegeven schaal wordt het tekeningetje nu groter of kleiner afgebeeld.

X...; de tussen de X en de puntkomma vermelde alfanumerieke variabele wordt tijdens het tekenen tussengevoegd. Hierdoor kan een tekening die is gekodeerd in een aparte variabele, diverse malen worden herhaald. Bijvoorbeeld:

```
NEW
10 SCREEN 2
20 Q$="U10R10D10L10E10H5G5F10"
30 DRAW "S4C15A0BM99,99XQ$;"
40 GOTO 40
RUN
```

Een eenvoudig tekeningetje zoals in Q$ gekodeerd, wordt in regel 30 getekend nadat schaal, kleur, hoek en beginpunt zijn voorbereid.

=...; de tussen het gelijkteken en de puntkomma vermelde numerieke variabele wordt tijdens het tekenen als waarde ingevoegd. Hierdoor behoeft een string met een tekenopdracht niet moeizaam met behulp van onder meer het STR$-sleutelwoord worden opgebouwd maar kan onmiddellijk vanuit een numerieke waarde een afmeting of locatie worden bepaald. Bijvoorbeeld:

```
NEW
10 INPUT "GROOTTE ";GR%:GT%=GR%/2
20 SCREEN 2
30 DRAW "S4C15A0BM99,99U=GR%;R=GR%;D=GR%;
L=GR%;"
40 DRAW "E=GR%;H=GT%;G=GT%;F=GR%;"
50 GOTO 50
RUN
GROOTTE ?
```

Een grootte in aantal beeldschermpuntjes kan worden ingegeven. Een eenvoudig tekeningetje wordt op de aangegeven grootte gemaakt.

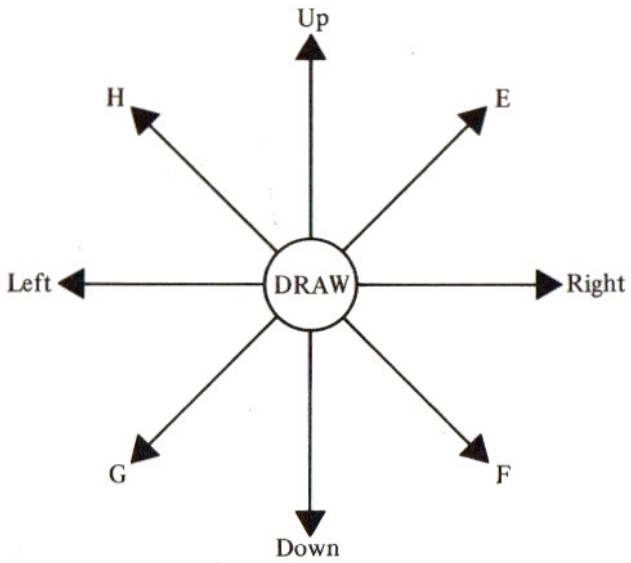
Algemene opmerkingen:

- DRAW gedraagt zich aan de randen van het beeldscherm niet geheel correct. Indien een lijn moet worden getrokken naar een buiten het beeld liggend punt, dan is het effect dat de lijn in werkelijkheid wordt getrokken naar de loodrechte projektie van dat niet bestaande punt op de betreffende beeldschermrand.
- Boven in het beeld verschijnen afkaplijnen wanneer delen van een tekening die boven buiten het beeld vallen, worden getekend.

Maxima en minima:

- Voor numerieke variabelen die met behulp van het gebruik van een gelijkteken in een DRAW-string worden opgenomen geldt dat deze variabelen geheel in waarde dienen te zijn. Een eventueel aanwezige decimale fractie wordt genegeerd.
- Voor de in verband met het U, E, R, F, D, G, L, H en M gebruikte konstanten of variabelen gelden de volgende bepalingen:
  variabele: minimaal –32768, maximaal 32787
  konstante: minimaal –65535, maximaal 65535
  deze waarden worden modulo 32768 behandeld, dat wil zeggen: indien een waarde groter is dan of gelijk is aan 32768 dan wordt 32768 van deze waarde afgetrokken en wanneer een waarde kleiner is dan of gelijk aan –32768 dan wordt 32768 bij deze waarde opgeteld.
- In verband met het S-kommando geldt een minimum van 0 en een maximum van 255 voor de betreffende konstante of variabele.
- In verband met het A-kommando geldt een minimum van 0 en een maximum van 3 voor de betreffende konstante of variabelen.
- In verband met het C-kommando geldt een minimum van 0 en een maximum van 15 voor de betreffende variabele of konstante.

Wanneer men het DRAW-kommando veelvuldig wil gaan gebruiken, is het gemakkelijk om een samenvatting van de uitgebreide mogelijkheden bij de hand te hebben. Om deze reden is op de volgende blad zijde een funktie-overzicht van het DRAW-kommando opgenomen Het is misschien een goed advies om deze kaart over te tekenen of te kopiëren om apart bij de computer te kunnen gebruiken.



kode	betek.	funktie
M...,...	Move	ga naar het aangegeven punt en trek een lijn
B	Blank	voer het volgende kommando uit zonder een lijn te trekken
N	–	voer het volgende kommando in omgekeerde volgorde uit, ofwel vanaf het eindpunt naar het beginpunt
A.	Angle	teken de rest onder een hoek van 0 (A0), 90 (A1), 180 (A2) of 270 (A3) graden
C..	Color	teken de rest in de gekodeerde kleur
S...	Scale	teken de rest op de aangegeven schaal/4
X...;	–	voeg de vermelde stringvariabele in
=...;	–	voeg de waarde van de aangegeven numerieke variabele in
+/–	–	relatieve adressering

SLEUTELWOORD # END

moeilijkheidsgraad ..... **zeer eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **END is einde**

schrijfwijze

END

betekenis

Met het END-kommando geven we aan dat het programma is beeindigd. Meestal is het opnemen van een END-kommando slechts een formaliteit. Voorbeeld:

```
NEW
10 PRINT "DIT IS HET EINDE"
20 END
RUN
DIT IS HET EINDE
Ok
```

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **EOF is afkorting van end of file – einde van het bestand**

schrijfwijze

EOF(<KANAAL>)
<KANAAL>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>

betekenis

Dti kommando wordt bij het OPEN-kommando reeds behandeld; zie aldaar.

SLEUTELWOORD # ERASE

moeilijkheidsgraad . . normaal, zeer elementaire kennis van matrix-berekeningen is een voordeel
soort . . . . . KOMMANDO
afkomst . . . . . ERASE is uitwissen

schrijfwijze

```
ERASE<VARIABELE NAAM>{,<VARIABELE NAAM>}
<VARIABELE NAAM>::=<ZIE ALGEMENE SPECIFI-
  CATIES>
```

betekenis

Het is noodzakelijk om eerst het DIM-kommando door te nemen.

Dit kommando heft een DIM-kommando voor de genoemde variabelen op. Voorwaarde is dat het betreffende DIM-kommando wel werd uitgevoerd.

Array's kunnen een enorm deel van het computergeheugen opeisen. Het is dus zaak om array's die niet meer worden gebruikt, op te heffen.

Een voorbeeld:

```
NEW
10 PRINT FRE(0)
20 DIM A(20,20)
30 PRINT FRE(0)
40 ERASE A
50 PRINT FRE(0)
60 LET A(20,20)=12
RUN
 28471
 24933
 28471
Subscript out of range in 60
Ok
```

Uit dit voorbeeld blijkt, dat de A-array een ruimte van 28471–24933= 3538 posities (bytes) inneemt. Na de ERASE is deze ruimte weer vrijgekomen. Programmaregel 60 veroorzaakt een fout; een bewijs dat de variabele A als array niet meer bekend is.

moeilijkheidsgraad . . . . . **vrij eenvoudig**
soort . . . . . **SYSTEEMVARIABELE**
afkomst . . . . . **ERL is afkorting van error linenumber – regelnummer fout**

schrijfwijze

ERL

betekenis

Deze systeemvariabele wordt behandeld onder ON ERROR GOTO; zie aldaar.

moeilijkheidsgraad . . . . . **vrij eenvoudig**
soort . . . . . **SYSTEEMVARIABELE**
afkomst . . . . . **ERR is afkorting van errornumber – foutnummer**

schrijfwijze

ERR

betekenis

Deze systeemvariabele wordt onder ON ERROR GOTO behandeld; zie aldaar.

SLEUTELWOORD # ERROR

moeilijkheidsgraad . . . . . **vrij eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **ERROR is fout**

schrijfwijze

```
ERROR<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Dit kommando wordt onder ON ERROR GOTO behandeld; zie aldaar.

moeilijkheidsgraad . . vrij moeilijk, kennis van logarithmische funkties is vereist
soort . . . . . N-FUNKTIE
afkomst . . . . . EXP is afkorting van exponent – macht

schrijfwijze

```
EXP(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de waarde *e* (2,7182818284...), verheven tot de macht die gevormd wordt door de waarde van de tussen haakjes vermelde uitdrukking.

Voorbeeld:

```
NEW
10 INPUT "WAARDE ";A
20 PRINT "E TOT DE MACHT A=";EXP(A)
30 GOTO 10
RUN
WAARDE ? 1
E TOT DE MACHT A= 2.7182818284588
WAARDE ? 200
E TOT DE MACHT A=
Overflow in 20
Ok
```

Merk op dat de maximale waarde van de numerieke uitdrukking ligt bij 145,06286085862 en de minimale waarde bij –75453,410912322. Deze laatste minimale waarde zou niet mogen bestaan maar is te wijten aan een klein foutje in MSX-basic.

SLEUTELWOORD # FIX

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **FIX is vastleggen, fixeren**

schrijfwijze

```
FIX(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het resultaat van deze funktie is een waarde, gelijk aan de waarde van de tussen haakjes vermelde numerieke uitdrukking maar dan ontdaan van decimale cijfers. FIX geeft dus altijd een geheel getal.

Voorbeeld:

```
NEW
10 LET A=12.5:LET B=-12.5
20 PRINT FIX(A);FIX(B)
RUN
 12 -12
Ok
```

moeilijkheidsgraad . . normaal, enige elementaire kennis van de funktietheorie is een voordeel
soort . . . . . N-FUNKTIE of A-FUNKTIE
afkomst . . . . . FN is afkorting van function – funktie/bewerking

schrijfwijze

```
N: FN<NUMERIEKE VARIABELE NAAM>[(<VARIA-
     BELE>{,<VARIABELE>})]
A: FN<ALFANUMERIEKE VARIABELE NAAM>[(<VA-
     RIABELE>{,<VARIABELE>})]
<ALFANUMERIEKE VARIABELE NAAM>::=<ZIE AL-
  GEMENE SPECIFICATIES>
<NUMERIEKE VARIABELE NAAM>::=<ZIE ALGEME-
  NE SPECIFICATIES>
<VARIABELE>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie wordt bij DEF FN behandeld; zie aldaar.

SLEUTELWOORD # FOR-TO-STEP

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **FOR-TO-STEP is voor-tot-stapgrootte**

schrijfwijze

```
FOR<NUMERIEKE VARIABELE>=<STARTWAARDE>TO
  <EINDWAARDE>[STEP<STAPWAARDE>]
<STARTWAARDE>::=<N>
<EINDWAARDE>::=<N>
<STAPWAARDE>::=<N>
<N>::<ZIE ALGEMENE SPECIFICATIES>
<NUMERIEKE VARIABELE>::=<ZIE ALGEMENE
  SPECIFICATIES>
```

betekenis

De FOR...TO...STEP combinatie is in samenwerking met het NEXT-kommando één van de krachtigste besturingsmogelijkheden van de computer.

Allereerst wijst FOR de startwaarde toe aan de genoemde variabele zoals een LET-kommando dat zou doen. Vervolgens vervolgt het programma op gebruikelijke wijze.
Wanneer nu vervolgens een bijbehorend NEXT-kommando wordt tegengekomen, dan wordt de betreffende variabele met de stapwaarde verhoogd. Hierna wordt gecontroleerd of de betreffende variabele ondertussen de eindwaarde niet is gepasseerd. Wanneer dit nog niet zo blijkt te zijn, wordt hervat met het kommando dat direkt op de FOR...TO...STEP volgt. Is de variabele deze eindwaarde echter wel gepasseerd, dan wordt het programma na de NEXT voortgezet.

Voorbeeld:

```
NEW
10 FOR I=1 TO 4 STEP 1.5
20 PRINT I
30 NEXT I
```

```
40 PRINT "KLAAR"
RUN
 1
 2.5
 4
KLAAR
Ok
```

De stapwaarde mag ook negatief zijn. Bijvoorbeeld:

```
NEW
10 FOR A=10 TO 0 STEP -1
20 PRINT A;
30 NEXT A
RUN
 10  9  8  7  6  5  4  3  2  1  0
Ok
```

Indien STEP wordt weggelaten, wordt automatisch een stapwaarde 1 aangenomen. Bijvoorbeeld:

```
NEW
10 FOR P=2 TO 5:PRINT P*P:NEXT:STOP
RUN
 4
 9
 16
 25
Break in 10
Ok
```

Uit het vorige voorbeeld blijkt dat bij de NEXT niet noodzakelijk een variabele behoeft te worden vermeld.

Het geheel vanaf FOR...TO...STEP tot en met NEXT noemt men wel de for-next loop (loop = lus). Twee of meer for-next loops mogen binnen elkaar plaatsvinden. Bijvoorbeeld:

```
NEW
10 FOR A=1 TO 3
20 FOR B=1 TO 3
30 PRINT A*B;
40 NEXT B:PRINT
50 NEXT A
RUN
 1  2  3
 2  4  6
 3  6  9
Ok
```

Kruisende loops zijn echter verboden.

GOED

```
NEW
10 FOR A=1 TO 5
20 FOR B=1 TO 5
.
.     (rest programma)
.
90 NEXT B
100 NEXT A
```

FOUT

```
NEW                    verboden
10 FOR A=1 TO 5
20 FOR B=1 TO 5
.
.
.
90 NEXT A
100 NEXT B
```

Twee NEXT-kommando's direkt ná elkaar mogen door één kommando worden vervangen. Voorwaarde is dan wel dat het betreffende NEXT-kommando beide variabelen in de juiste volgorde noemt. Regel 90 en 100 van het bovenstaande goede voorbeeld mogen dus vervangen worden door de ene programmaregel:

```
90 NEXT B,A
```

In tegenstelling tot de meeste andere BASIC-dialecten voert MSX-basic een for-next loop altijd minimaal één keer uit, hoe de startwaarde en de eindwaarde zich ook verhouden.

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **N-FUNKTIE**
afkomst ..... **FRE is afkorting van free – vrij**

schrijfwijze

```
FRE(<U>)
<U>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat een getal dat de nog vrije geheugenruimte uitdrukt. Met de FRE-funktie kan

- de vrije ruimte in het geheugen, met uitzondering van het stringgeheugen, worden bepaald. Hiertoe dient de tussen haakjes op te nemen uitdrukking numeriek te zijn
- de vrije ruimte in het string-gebied worden bepaald. Hiertoe dient de tussen haakjes te vermelden uitdrukking alfanumeriek te zijn.

Bijvoorbeeld:

(zet eerst de computer uit en weer aan)

```
PRINT FRE(0)
 28815
Ok
PRINT FRE("")
 200
Ok
```

Uit het bovenstaande blijkt dat de computer een vrij geheugenruimtegebied heeft van 28815 tekens en dat in het vrije string-gebied nog plaats is voor 200 karakters.

SLEUTELWOORD # GOSUB

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **GOSUB is afkorting van go to subroutine – ga naar onderprogramma (subroutine)**

schrijfwijze

```
GOSUB<REGELNUMMER>
<REGELNUMMER>::=<TYPE 1 INTEGERE KONSTAN-
  TE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE ALGE-
  MENE SPECIFICATIES>
```

betekenis

In eerste instantie werkt de GOSUB hetzelfde als de GOTO; het programma gaat verder met het genoemde regelnummer. Echter, bij GOSUB doet de computer nog meer. De computer 'onthoudt' namelijk het punt *waarvandaan* wordt gesprongen, het zogenaamde terugkeeradres in een stukje geheugen, de stack (stapel) genaamd. Wanneer later een RETURN-kommando wordt gegeven (RETURN = keer terug), dan haalt de computer dit regelnummer weer van die stack en gaat door met het kommando dat direkt komt na de GOSUB (dus één stap verder dan wat de stack aangeeft).

Het voordeel van een GOSUB is, dat gedeelten van programmatuur meerdere malen kunnen worden gebruikt op diverse plaatsen in het programma. Bijvoorbeeld:

```
NEW
10 INPUT "HOEVEEL IS 2+3 ";U
20 IF U<>5 THEN GOSUB 500:GOTO 10
30 INPUT "EN HOEVEEL IS 5+6 ";U
40 IF U<>11 THEN GOSUB 500:GOTO 10
50 PRINT "GOEDZO, KLAAR!":STOP
500 REM SUBROUTINE FOUTMELDING
510 PRINT "NEE HOOR, DAT IS NIET"
520 PRINT "GOED. NOG EEN KEER!"
530 RETURN
```

```
RUN
HOEVEEL IS 2+3 ? 4
NEE HOOR, DAT IS NIET
GOED. NOG EEN KEER!
HOEVEEL IS 2+3 ? 5
EN HOEVEEL IS 5+6 ? 12
NEE HOOR, DAT IS NIET
GOED. NOG EEN KEER!
EN HOEVEEL IS 5+6 ? 11
GOEDZO, KLAAR!
Break in 50
Ok
```

We zien dat het programmadeel vanaf regel 500 een keer vanuit regel 20 en een keer vanuit regel 40 werd aangeroepen. Het programmadeel waarin de foutmelding wordt afgedrukt, behoeft door de GOSUB-mogelijkheid maar één maal in het programma te worden opgenomen.

Een programmagedeelte dat via een GOSUB wordt aangeroepen, noemt men meestal een onderprogramma of een *subroutine*.

Een subroutine mag weer een andere subroutine aanroepen. In dat geval wordt het te onthouden terugkeeradres bovenop de stack geplaatst. Dit heeft tot gevolg dat een RETURN altijd een terugkeer veroorzaakt naar het gedeelte na de laatste GOSUB.
Voorbeeld:

```
NEW
10 GOSUB 100:PRINT "TERUG
  IN HOOFDPROGRAMMA"
20 STOP
100 GOSUB 500:PRINT "TERUG
  IN EERSTE SUBROUTINE"
110 RETURN
500 PRINT "TWEEDE SUBROUTI-
  NE":RETURN
RUN
TWEEDE SUBROUTINE
TERUG IN EERSTE SUBROUTINE
```

```
TERUG IN HOOFDPROGRAMMA
Break in 20
Ok
```

Ga na dat de regelnummers waar de computer achtereenvolgens mee bezig is, de nummers 10, 100, 500, 100, 110, 10 en 20 zijn.

Een goede BASIC-programmeur stelt zijn programma uit vele subroutines samen. In elke subroutine verzorgt hij of zij dan een compleet afgerond geheel. Uiteindelijk stelt deze programmeur dan een HOOFDPROGRAMMA samen waarin al deze subroutines met GOSUB-kommando's worden aangesproken. Wanneer het hoofdprogramma, dat bijna alleen uit GOSUB's bestaat, daarbij ook nog met REM-kommando's goed van kommentaar is voorzien, ontstaat een programma dat ook bij grote afmetingen voor iedereen nog zeer goed te volgen is. Een voorbeeld (niet intikken, dat heeft geen zin) van hoe zo'n hoofdprogramma er dan uit zou kunnen zien:

```
NEW
10 REM ************************
20 REM HOOFDPROGRAMMA ADRESSEN
30 REM ************************
40 GOSUB 1000'BEELDSCHERM SCHOON & KOP
50 GOSUB 2000'PROGRAMMAKEUZE
60 REM KEUZE 1...AANMAKEN ADRESSEN
70 IF KEUZE=1 THEN GOSUB 3000:GOTO 40
80 REM KEUZE 2...ADRESSEN OP BEELD
90 IF KEUZE=2 THEN GOSUB 4000:GOTO 40
100 REM KEUZE 3...ADRESSEN OP PAPIER
110 IF KEUZE=3 THEN GOSUB 5000:GOTO 40
120 REM KEUZE 4...EINDE PROGRAMMA
130 PRINT "EINDE PROGRAMMA":BEEP:STOP
140 REM *****EINDE HOOFDPROGRAMMA*****
1000 REM SUBROUTINE 1, BEELD SCHOON/KOP
.
.
(etcetera)
```

Indien men op deze wijze programmeert, is men struktureel bezig. Bij

deze gestruktureerde vorm van programmeren is de GOSUB-RETURN combinatie onontbeerlijk.

Indien men ten onrechte niet met RETURN uit een subroutine terugkeert, kunnen na verloop van tijd vreemde situaties ontstaan. Elke keer dat een GOSUB wordt uitgevoerd, wordt namelijk een nieuw terugkeeradres op de stack bijgeplaatst. Hierdoor wordt er steeds een iets groter stukje geheugen door de computer bezet. Uiteindelijk kan het overblijvende geheugendeel zo klein blijken te zijn dat een foutmelding volgt. Een eenvoudig voorbeeld van dit verkeerde GOSUB-gebruik:

```
NEW
10 GOSUB 10
RUN
Out of memory in 10
Ok
```

Het bovenstaande programma vult continu de stack aan zonder dat er weer terugkeeradressen door RETURN worden afgehaald. Na ongeveer 4 seconden volgt een foutmelding; het hele geheugen is opgebruikt aan stack-ruimte.

Het is een gouden regel dat een subroutine ALTIJD met een GOSUB moet worden aangesproken en ALTIJD met een RETURN moet worden verlaten.

Indien tijdens het samenstellen van een programma de situatie ontstaat waarbij een normale RETURN niet meer kan worden gebruikt, kan een RETURN, gevolgd door een regelnummer, worden gebruikt. Dit *bijzonder weinig fraaie* gebruik van RETURN werkt als een GOTO met dit verschil dat een terugkeeradres van de stack wordt gehaald zonder dat er iets mee wordt gedaan; het geheugen wordt niet overbelast terwijl eventueel volgende RETURN-kommando's goed blijven werken (naar de juiste aanroeplocaties terugkeren). Voorbeeld:

```
NEW
10 GOSUB 100:STOP
100 INPUT "WAARDE TUSSEN 1 EN 10 ";A
110IF A=INT(A) AND A>0 AND A<11 THEN RE
TURN
```

```
120 RETURN 10
RUN
WAARDE TUSSEN 1 EN 10 ? 2.5
WAARDE TUSSEN 1 EN 10 ? 3
Break in 10
Ok
```

(via een RETURN 10 wordt een verbetering geforceerd)

SLEUTELWOORD # GOTO

moeilijkheidsgraad ..... **zeer eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **GOTO is samentrekking van go to – ga naar**

schrijfwijze

```
GOTO<REGELNUMMER>
<REGELNUMMER>::=<TYPE 1 INTEGERE KONSTAN-
  TE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE ALGE-
  MENE SPECIFICATIES>
```

betekenis

Met GOTO kan de programmaloop worden veranderd. Het regelnummer dat achter GOTO wordt vermeld, is het regelnummer waar de computer verder gaat. Een GOTO mag naar een eerder- of verdergelegen regelnummer verwijzen. Ook mag GOTO naar zichzelf verwijzen.

Voorbeeld:

```
NEW
10 GOTO 10
RUN
```

– Het programma staat in een eeuwigdurende lus en kan slechts met een CONTROL-STOP worden onderbroken.

```
NEW
10 REM DIT PROGRAMMA BLIJFT UW NAAM
20 REM CONTINU AFDRUKKEN.
30 INPUT "HOE HEET U ";NAAM$
40 PRINT NAAM$;"-";
50 GOTO 40
RUN
HOE HEET U ? KAREL
KAREL-KAREL-KAREL-KA...
```

```
REL-KAREL-KAREL-KA...
...
```

– Het gehele beeldscherm wordt vol gezet, onderbreken met CONTROL-STOP.

Merk op dat GOTO standaard onder funktie-toets 3 aanwezig is.

moeilijkheidsgraad .. **vrij moeilijk, vereist kennis van talstelsels en algemene principes van het computergeheugen**
soort ..... **A-FUNKTIE**
afkomst ..... **HEX$ is afkorting van hexadecimal string – hexadecimale (zestientallige) string**

schrijfwijze

```
HEX$(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft de hexadecimale waarde van de tussen haakjes vermelde uitdrukking weer in string-vorm. De waarde van de numerieke uitdrukking dient te liggen tussen –32769 en 65536. Indien de waarde van de numerieke uitdrukking gebroken is, wordt de grootste gehele waarde, kleiner dan de gebroken waarde als geldende waarde genomen.
Indien de waarde van de numerieke uitdrukking negatief is, geeft HEX$ een hexadecimale representant, samengesteld volgens de tweecomplementmethode. Bijvoorbeeld:

```
PRINT HEX$(-32768)
8000
Ok
PRINT HEX$(65535)
FFFF
Ok
PRINT HEX$(-1)
FFFF
Ok
PRINT HEX$(1023)
3FF
Ok
```

SLEUTELWOORD IF-THEN/GOTO-ELSE

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **IF-THEN/GOTO-ELSE is als-dan/ga naar - anders (goto – samentrekking van go to)**

schrijfwijze

```
      THEN    <DIREKTE OPDRACHT>
IF<N>GOTO
      GO TO   <REGELNUMMER>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
<REGELNUMMER>::=<TYPE 1 INTEGERE KON-
  STANTE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE
  ALGEMENE SPECIFICATIES>
<DIREKTE OPDRACHT>::=<ZIE ALGEMENE
  SPECIFICATIES>
```

betekenis

Voor goed begrip van IF...THEN/GOTO...ELSE dienen de hoofdstukken 5, 6 en 7 terdege te zijn doorgenomen.

Deze kombinatie van sleutelwoorden biedt de mogelijkheid om numerieke of alfanumerieke relaties te onderzoeken en daarop beslissingen te nemen. Alleen indien de tussen IF en THEN/GOTO opgenomen numerieke uitdrukking WAAR is (of ongelijk aan nul is), dan worden de kommando's na THEN uitgevoerd of wordt met het regelnummer achter GOTO vervolgd.

Voorbeeld:

```
NEW
10 INPUT "HOEVEEL IS 2 MAAL 5 ";UITKOMST
20 IF UITKOMST=10 GOTO 100
30 IF UITKOMST<10 THEN PRINT "TE LAAG":
GOTO 10
40 PRINT "TE HOOG":GOTO 10
```

```
100 PRINT "HOERA,GOED"
110 STOP
RUN
HOEVEEL IS 2 MAAL 5 ? 9
TE LAAG
HOEVEEL IS 2 MAAL 5 ? 112.5
TE HOOG
HOEVEEL IS 2 MAAL 5 ? 10
HOERA,GOED
Break in 110
Ok
```

Ook ingewikkelder zaken kunnen worden afgevraagd, bijvoorbeeld met behulp van de logische bewerkingen:

```
NEW
10 INPUT "DE LUCHT IS ";KLEUR$
20 INPUT "EN HET GRAS RUIKT ";GEUR$
30 IF KLEUR$="BLAUW" AND GEUR$="FRIS"
THEN PRINT "GOEDZO":STOP
40 PRINT "NOG EEN KEER PROBEREN":GOTO 10
RUN
DE LUCHT IS ? GEEL
EN HET GRAS RUIKT ? FRIS
NOG EEN KEER PROBEREN
DE LUCHT IS ? BLAUW
EN HET GRAS RUIKT ? FRIS
GOEDZO
Break in 30
Ok
```

Met de ELSE-optie kan worden gespecificeerd welke akties dienen te worden ondernomen indien de uitdrukking tussen IF en THEN/GOTO NIET WAAR is. Bijvoorbeeld:

```
NEW
10 INPUT "HOE HEET U ";NAAM$
20 IF LEN(NAAM$)=5 THEN PRINT "UW NAAM
```

```
IS PRECIES 5 LETTERS LANG" ELSE PRINT "
HALLO ";NAAM$;" HOE GAAT HET ERMEE ?":G
OTO 10
RUN
HOE HEET U ? JAN
HALLO JAN HOE GAAT HET ERMEE
HOE HEET U ? FRITS
UW NAAM IS PRECIES 5 LETTERS LANG
Ok
```

Merk op dat regel 30 en 40 uit het voorlaatste voorbeeld kunnen worden vervangen door één regel:

```
30 IF KLEUR$="BLAUW" AND GEUR$="FRIS"
THEN PRINT "GOEDZO":STOP ELSE PRINT
"NOG EEN KEER PROBEREN":GOTO 10
```

SLEUTELWOORD # INKEY$

moeilijkheidsgraad ..... **zeer eenvoudig**
soort ..... **SYSTEEMVARIABELE**
afkomst ..... **INKEY$ is afkorting van input key string – (vrij) ingetoetst karakter**

schrijfwijze

INKEY$

betekenis

Deze funktie geeft als resultaat het karakter van de laatst ingegeven toets. Indien niets werd ingegeven, geeft INKEY$ een lege waarde als resultaat. Bijvoorbeeld:

```
NEW
10 LET A$=INKEY$:IF A$="" THEN 10
20 PRINT "U GAF ";A$;" IN."
30 GOTO 10
RUN
```

(De computer wacht. Zodra u een toets indrukt, meldt hij dat waarna hij weer wacht. Programma onderbreken met CONTROL-STOP).

N.B.: Zodra INKEY$ één maal is gebruikt, is het betreffende karakter verdwenen. Daarom moet INKEY$ steeds eerder in een alfanumerieke variabele worden overgebracht (zie regel 10) en daarna pas worden afgevraagd.

Op de CONTROL-BREAK-intoetsing na, worden alle toetsaanslagen bij het gebruik van INKEY$ geregistreerd. Voor controletoetsen wordt ook een karakter gegeven. Om dit karakter te kunnen bestuderen, kunnen we de ASCII-waarde van dit karakter als volgt afvragen:

```
NEW
10 LET A$=INKEY$:IF K$="" THEN 10
20 PRINT "ASCII-WAARDE:";ASC(A$)
30 GOTO 10
RUN
```

```
ASCII-WAARDE:  13
ASCII-WAARDE:  27
ASCII-WAARDE:  24
```
(de RETURN-toets werd ingedrukt)
(de ESC-toets werd ingedrukt)
(de SELECT-toets werd ingedrukt)

(etcetera; probeer hoofdletters, kleine letters, controletoetsen en ook eens de funktietoetsen. Controleer het een en ander met hoofdstuk 17 en 18.)

moeilijkheidsgraad . . **zeer moeilijk, kennis van de funkties van de poorten van het computersysteem is vereist**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **INP is afkorting van input – invoer, ingave**

schrijfwijze

```
INP(<POORT>)
<POORT>::=<N>
<N>::=<ZIE NADERE SPECIFICATIES>
```

betekenis

Een poortnummer dient te worden gespecificeerd. De waarde van het byte dat op het moment van aanspreken op de poort-buffer staat, vormt het resultaat van deze funktie.

Het poortnummer mag niet kleiner zijn dan –32768 en niet groter dan 65535. Een eventuele decimale factor wordt verwaarloosd. Indien het poortnummer negatief is, wordt 65536 bij dit nummer opgeteld voordat de betreffende poort wordt aangesproken.

Bijvoorbeeld:

```
NEW
10 PRINT INP(12)
RUN
 255
Ok
```

Voor een goed gebruik van INP is een gedetailleerde kennis van de hardware-opbouw van een MSX-computer noodzakelijk. Daarbij dienen de diverse componenten van een MSX-computer ook software-technisch te worden beheerst. Hiertoe dient specialistische literatuur te worden geraadpleegd; in dit handboek wordt hierop niet verder ingegaan.

moeilijkheidsgraad . . . **normaal, in de eerste betekenis vrij eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **INPUT is invoer/ingave**

schrijfwijze

```
        #<KANAAL>,
INPUT ------------------------<VARIABELE>
        ["<SATELLIETTEKST>";]
  {,<VARIABELE>}
<SATELLIETTEKST>::=<ALFANUMERIEKE KON-
  STANTE>
<ALFANUMERIEKE KONSTANTE>::=<ZIE ALGEME
  NE SPECIFICATIES>
<KANAAL>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
<VARIABELE>::=<ZIE ALGEMENE SPECIFICA-
  TIES>
```

betekenis

Met het INPUT-kommando kunnen we gegevens tijdens de programmaloop aan de computer worden toegevoerd.

eenvoudig gebruik

Met het INPUT-kommando kunnen gegevens via het toetsenbord worden ingevoerd. Deze worden dan in een variabele opgeslagen. Voorbeeld:

```
NEW
10 INPUT A
20 PRINT A
30 STOP
RUN
? 12.34          — De computer geeft een vraagteken. Een
 12.34             waarde dient te worden ingetoetst.
```

```
Break in 30
Ok
```

De in te voeren gegevens kunnen numeriek of alfanumeriek zijn. Bij invoer van alfanumerieke gegevens dient het aanhalingsteken als eerste karakter te worden vermeden omdat MSX-basic versie 1.0 hierop niet correct reageert. Voorbeeld:

```
NEW
10 INPUT A$
20 PRINT "HALLO ";A$
RUN
? FRANS
HALLO FRANS
Ok
```

Indien meerdere gegevens dienen te worden ingevoerd, kan dat in één INPUT-kommando gebeuren. De variabelen achter INPUT dienen dan met een komma van elkaar te zijn gescheiden. Voorbeeld:

```
NEW
10 INPUT A,B,C
20 PRINT A,B,C
30 GOTO 10
RUN
? 1,2,3.5
 1   2   3.5
? 22
?? 33
?? 44
 22   33   44
? 1,2,3,4
?Extra ignored
 1   2   3
?
```

– De gegevens mogen in één keer achter elkaar worden ingegeven, gescheiden door een komma.
– De gegevens mogen ook elk met een enter worden afgesloten. Met een dubbel vraagteken vraagt de computer dan een volgende ingave totdat alle variabelen zijn gevuld.
– Indien te veel waarden worden ingevoerd geeft de computer eerst een melding dat het teveel aan ingevoerde gegevens wordt overgeslagen.

De komma is het scheidingsteken bij INPUT. Daarom kan een komma als *teken* via de input nooit in een stringvariabele worden overgeplaatst.

Een INPUT-kommando mag worden voorzien van een begeleidende tekst. Bijvoorbeeld:

```
NEW
10 INPUT "HOE HEET U ";NAAM$
20 PRINT "HALLO ";NAAM$
30 STOP
RUN
HOE HEET U ? GERARD
HALLO GERARD
Break in 30
Ok
```

Indien voor MSX-basic onbegrijpelijke ingaven worden gedaan, vraagt de computer u om een herhaalde ingave. Bijvoorbeeld:

```
NEW
10 INPUT "GEEF WAARDE IN ";A
20 PRINT "DANK U":STOP
RUN
GEEF WAARDE IN ? GERARD
?Redo from start
GEEF WAARDE IN ? 12.3
DANK U
Break in 20
Ok
```

Indien bij een INPUT slechts een RETURN-toets wordt ingegeven, dan behoudt of behouden de genoemde variabele(n) de vorige waarde(n). Voorbeeld:

```
NEW
10 INPUT A:PRINT A:GOTO 10
RUN
? 11.4
 11.4
? 12.5
 12.5
```

– Alleen een RETURN wordt ingegeven.

```
?
 12.5
?
```
– En de oude waarde bleef behouden.

Gevorderd gebruik

Zie hiervoor ook de beschrijving van het PRINT-kommando, onderdeel gevorderd gebruik: schrijven naar randapparatuur alsmede het OPEN- en CLOSE-kommando.

Indien een kanaalnummer is gespecificeerd, kan de aanvoer van gegevens vanaf een ander apparaat dan alleen het toetsenbord plaatsvinden; bijvoorbeeld vanaf een cassetterecorder. Voorwaarde is dat de gegevens op precies dezelfde manier als via het toetsenbord worden aangevoerd. Dit betekent dat de diverse gegevens die op één regel staan, ook bijvoorbeeld door een komma dienen te worden gescheiden. In feite kan men stellen dat de gegevens naar het betreffende randapparaat (b.v. een cassetterecorder) dient te PRINTen zoals het INPUT-kommando deze later weer vereist. Een voorbeeld (zie ook het voorbeeld onder PRINT):

eerste programma:

```
NEW
10 OPEN "CAS:TEST" FOR OUTPUT AS 1
20 PRINT#1,"DEZE GEGEVENS WORDEN"
30 PRINT#1,"STRAKS WEER OPGEHAALD"
40 PRINT#1,123;",";-12.33
```

– Tussen de twee numerieke gegevens op regelnummer 40 een komma, omdat zij straks door één INPUT weer worden opgehaald.

– Nu de cassetterecorder op opnemen zetten.

```
RUN
```
– De cassetterecorder begint te lopen.

```
Ok
```
– De cassetterecorder stopt weer; de gegevens zijn geschreven.

– Cassetterecorder nu eventueel uitzetten.

tweede programma:

```
NEW
10 OPEN "CAS:TEST" FOR INPUT AS 4
20 INPUT#4,A$:PRINT A$:INPUT#4,A$:
PRINT A$
30 INPUT#4,A,B:PRINT A;B:CLOSE:STOP
RUN
```

– De cassetteband terugspoelen en de recorder op afspelen zetten.

```
DEZE GEGEVENS WORDEN
STRAKS WEER OPGEHAALD
 123 -12.33
Break in 30
Ok
```

Indien u dit voorbeeld uitprobeert, zorg er dan voor dat het volume van de cassetterecorder op ongeveer 80 procent staat. De MSX-standaard schrijft geen controles voor op de integriteit van gegevens. Afhankelijk van de kwaliteit en instelling van de cassetterecorder kunnen gegevensverminkingen optreden.

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **A-FUNKTIE**
afkomst ..... **INPUT$ is afkorting van input string – ingave string**

schrijfwijze

```
INPUT$(<AANTAL TEKENS>[,<KANAAL>])
<AANTAL TEKENS>:::=<N>
<KANAAL>:::=<N>
<N>:::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de ingevoerde tekens nadat het vermelde aantal is bereikt. Deze ingevoerde tekens kunnen van een randapparaat komen (cassetterecorder of disk) maar ook van het toetsenbord.

Eenvoudig gebruik

Indien de in te voeren tekens van het toetsenbord dienen te komen, kan het kanaalnummer worden weggelaten. Bijvoorbeeld:

```
NEW
10 PRINT INPUT$(5)
RUN
TEST.
Ok
```

(nu kunnen tekens worden ingegeven; deze worden in eerste instantie niet op het scherm weergegeven)

Pas nadat er vijf tekens zijn ingegeven, verschijnen deze tekens op het beeldscherm en is het programma ten einde.

Gevorderd gebruik

Zie ook het OPEN en CLOSE-kommando.
Indien een kanaalnummer werd gespecificeerd, kan de invoer van tekens ook via dit kanaal plaatsvinden. Bijvoorbeeld:

```
NEW
10 OPEN "CAS:TEST" FOR OUTPUT AS 1
20 PRINT 1,"DIT IS EEN TEST"
30 CLOSE
```

(plaats nu een cassetteband in de recorder en zet deze op opnemen)

```
RUN
Ok
```

(na enige tijd is de band geschreven)

(zet nu de recorder uit)

(tweede programma)

```
NEW
10 OPEN "CAS:TEST" FOR INPUT AS 1
20 PRINT INPUT$(5,1)
30 PRINT INPUT$(3,1)
40 CLOSE
RUN
```

(spoel nu de band terug en zet de recorder op afspelen)

```
DIT I
S E
Ok
```

(eerst worden de eerste 5 en later de eerste 3 ingevoerde karakters afgedrukt op het beeldscherm)

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **INSTR is afkorting van in string – binnen string**

schrijfwijze

```
INSTR([<STARTPOSITIE>,]<TE ONDERZOEKEN
  STRING>,<ZOEKSTRING>)
<STARTPOSITIE>::=<N>
<ZOEKSTRING>::=<A>
<TE ONDERZOEKEN STRING>::=<A>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat een positiewaarde binnen de te onderzoeken string van de op te zoeken string. Bijvoorbeeld:

```
PRINT INSTR("LIESJE LEERDE LOTJE
LOPEN","LEERDE")
 8
Ok
```

Bovenstaande voorbeeld laat zien hoe het woord LEERDE binnen de zin LIESJE LEERDE LOTJE LOPEN kan worden opgezocht. Het resultaat is de waarde 8; de positie van de eerste letter van de gevonden string.

Wanneer de te onderzoeken string niet de zoekstring bevat, levert de INSTR-funktie de waarde 0 op. Bijvoorbeeld:

```
PRINT INSTR("IN EEN GROEN GROEN GROEN",
"BLAUW")
 0
Ok
```

Wanneer slechts een gedeelte van de te onderzoeken string dient te

worden onderzocht, dient een startpositie te worden opgegeven die niet kleiner mag zijn dan de waarde 1 en niet groter mag zijn dan 255. Een eventuele decimale fractie wordt verwaarloosd. Bijvoorbeeld:

```
PRINT INSTR(3,"DE KLOK EN DE KLEPEL",
"DE")
 12
Ok
```

Indien in dit voorbeeld de startpositie (3) niet was opgenomen, zou het resultaat niet 12 maar 1 zijn (het eerste woord is al meteen gelijk aan DE).

Een laatste voorbeeld:

```
NEW
10 LINE INPUT "ZIN:";ZIN$
20 LET ST=1
30 PRINT "HET WOORDJE -DE- KOMT VOOR"
40 PRINT "OP POSITIE";
50 LET PS=INSTR(ST,ZIN$,"DE")
60 IF PS=0 THEN PRINT:GOTO 50
70 PRINT PS;:LET ST=PS+1:GOTO 50
RUN
ZIN:ALS DE KAT VAN HUIS IS, DANSEN DE
MUIZEN OP TAFEL
HET WOORDJE -DE- KOMT VOOR
OP POSITIE 5  32
ZIN:DE DEERNEN DEELDEN DE DEENSE DEKEN
HET WOORDJE -DE- KOMT VOOR
OP POSITIE 1  4  12  16  20  23  30
ZIN:
```

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **N-FUNKTIE**
afkomst ..... **INT is afkorting van integer – gehele waarde**

schrijfwijze

```
INT(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het resultaat van deze funktie is de grootste gehele waarde, kleiner dan of gelijk aan de waarde van de tussen haakjes vermelde numerieke uitdrukking. Om het verschil met FIX aan te geven, zijn beide in het voorbeeld verwerkt:

```
NEW
10 LET A=12.5:LET B=-12.5
20 PRINT INT(A);FIX(A)
30 PRINT INT(B);FIX(B)
RUN
 12  12
-13 -12
Ok
```

SLEUTELWOORD # INTERVAL

moeilijkheidsgraad ..... **vrij moeilijk**
soort ..... **KOMMANDO**
afkomst ..... **INTERVAL is pauze**

schrijfwijze

```
          ON
INTERVAL  OFF
          STOP
```

betekenis

Voorafgaand aan de behandeling van dit kommando dient het ON KEY GOSUB kommando grondig te zijn doorgenomen.

Behalve dat we in MSX-basic een programma tijdelijk kunnen onderbreken bij het intoetsen van een funktietoets, kunnen we het programma ook periodiek laten onderbreken op tijdsduur.

Wanneer we een mogelijke onderbreking van het programma op tijdsduur willen activeren, gebruiken we het kommando:

INTERVAL ON

Deactivering van deze onderbreking geschiedt met behulp van INTERVAL OFF.
Indien we een onderbreking willen laten 'onthouden' door de computer zonder dat er direkt actie op volgt, gebruiken we INTERVAL STOP.
Een eventueel geregistreerde wens tot onderbreking wordt dan bij een INTERVAL ON onmiddellijk gehonoreerd.

Een INTERVAL STOP heeft geen zin indien deze niet werd voorafgegaan door een INTERVAL ON.

Voor een zinvol voorbeeld: zie ON INTERVAL GOSUB.

INTERVAL vindt zijn toepassing bij uitstek in spelprogramma's en educatieve programma's (tijdsbewaking).

moeilijkheidsgraad . . **normaal, wat lastig door de vele uiteenlopende mogelijkheden**
soort . . . . . **KOMMANDO**
afkomst . . . . . **KEY is toets**

schrijfwijze

```
KEY  LIST
     (<FUNKTIETOETSNUMMER>)STOP
     <FUNKTIETOETSNUMMER>,<UIT TE
     VOEREN FUNKTIE>\KEY(<...>
     [(<FUNKTIETOETSNUMMER>)]ON
                             OFF
<FUNKTIETOETSNUMMER>::=<N>
<UIT TE VOEREN FUNKTIE>::=<A>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
<...>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het KEY-sleutelwoord heeft vele verschillende betekenissen. Hieronder volgt een beschrijving per betekenis:

Eerste betekenis: het uitlijsten van de inhoud van de funktietoetsen

Door ingave via KEY LIST verschijnen onder elkaar op het beeldscherm de funkties die onder de funktietoetsen liggen opgeslagen. Eventuele controlekarakters (zoals de in CHR$(13) gekodeerde RETURN toets, zie de derde betekenis) worden als spaties afgedrukt.
Wanneer we KEY LIST op een zojuist aangeschakelde MSX-computer gebruiken, krijgen we de volgende lijst:

```
KEY LIST
color
auto
goto
list
```

```
run
color 15,4,4
cload"
cont
list.
 run
Ok
```

Wanneer we de funktietoetsen zelf indrukken, verschijnen de afgebeelde funkties daadwerkelijk. Zo heeft funktietoets F10 tot gevolg dat het beeldscherm eerst wordt schoongemaakt (controlekarakter vóór run; zie de afgebeelde spatie) waarna het programma wordt opgestart.

Tweede betekenis; het ophouden van een onderbreking via een funktietoets

Deze betekenis, te activeren met de sleutelwoorden KEY en STOP, wordt bij de behandeling van ON KEY GOSUB behandeld; zie aldaar.

Derde betekenis; het veranderen van de funktie van een funktietoets

Via het KEY sleutelwoord, gevolgd door een numerieke uitdrukking die niet met een haakje begint, gevolgd door een komma en een alfanumerieke uitdrukking, kan de funktie van een funktietoets worden veranderd. Bijvoorbeeld:

```
KEY 2,"TRON"
Ok
```

Na het intikken van dit kommando heeft funktietoets 2 de funktie TRON verkregen. Wanneer we hierna deze funktietoets indrukken, verschijnt de tekst TRON. Er dient nog wel een RETURN te worden ingegeven om het kommando te bevestigen.

De RETURN-toets correspondeert met een ASCII-kode 13. Deze toets kan als karakter dus worden meegegeven in een funktie. Bijvoorbeeld:

```
KEY 2,"TRON"+CHR$(13)
Ok
```

Wanneer we nu funktietoets 2 indrukken, verschijnt wederom het TRON-kommando maar wordt daarbij ook meteen een return gegeven.

Direkt na de funktietoets verschijnt de Ok-melding; het TRON-kommando is uitgevoerd.

Wanneer we een funktie willen creëren die een TRON en vervolgens een RUN activeert, kunnen we dat bijvoorbeeld als volgt bewerkstelligen:

```
KEY 2,"TRON"+CHR$(13)+"RUN"+CHR$(13)
Ok
```

Het ligt voor de hand dat we funktietoets 1 als volgt inrichten:

```
KEY 1,"TROFF"+CHR$(13)+"RUN"+CHR$(13)
Ok
```

Indien we funktietoets 3 de funktie 'maak het beeld schoon en doe een LIST' willen geven, kan dat als volgt:

```
KEY 3,"CLS"+CHR$(13)+"LIST"+CHR$(13)
Ok
```

of

```
KEY 3,CHR$(12)+"LIST"+CHR$(13)
Ok
```

ASCII-kode 12 heeft tot effect dat het beeld wordt schoongemaakt.

De numerieke uitdrukking voor de komma geeft het funktietoetsnummer aan en dient een gehele, positieve waarde te bevatten, kleiner dan 11. Indien de numerieke uitdrukking een gebroken waarde bevat, wordt de decimale fractie verwaarloosd.

Vierde betekenis; het aan- of uitzetten van de onderbrekingsmogelijkheid door middel van een funktietoets

Deze betekenis, die met behulp van de sleutelwoorden KEY,ON en OFF kan worden geactiveerd, wordt onder ON KEY GOSUB behandeld; zie aldaar.

Vijfde betekenis; het aan- of uitzetten van de funktietoetsregel

De MSX-computer heeft standaard de beeldschermregel (in de alfanumerieke toestand) gereserveerd voor het presenteren van (een gedeelte van) de funkties van de funktietoetsen. Zolang deze funkties daar worden geprojekteerd, kan die onderste regel niet worden gebruikt door het programma.

Merk op dat de eerste vijf funkties zijn afgebeeld. Door de SHIFT-toets in te drukken, verschijnt onder in het beeld de tweede set van vijf funkties.

Door eenvoudigweg het kommando

```
KEY OFF
```

in te toetsen, kunnen we de onderste regel leeg maken en vrij maken voor gebruik in een programma. Door het

```
KEY ON
```

kommando halen we de funktietoetsregel weer te voorschijn en wordt deze regel weer verboden terrein voor het programma.

moeilijkheidsgraad ..... **normaal**
soort ..... **A-FUNKTIE**
afkomst ..... **LEFT$ is afkorting van left part of string – linker gedeelte van de string**

schrijfwijze

```
LEFT$(<BASISSTRING>,<AANTAL TEKENS>)
<BASISSTRING>::=<A>
<AANTAL TEKENS>::=<N>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft van een alfanumerieke uitdrukking het linkergedeelte als resultaat. Hoe groot dit linkergedeelte is, wordt bepaald door het opgegeven aantal tekens. Bijvoorbeeld:

```
PRINT LEFT$("MSX-BASIC",4)
MSX-
Ok
```

Het aantal tekens moet groter dan of gelijk aan nul zijn en mag de waarde 255 niet overschrijden. Een eventuele decimale fractie wordt verwaarloosd.

Nog een voorbeeld:

```
NEW
10 LINE INPUT "HOE HEET U ?:";NAAM$
20 INPUT "HOEVEEL LETTERS HEEFT UW
VOORNAAM";AANTAL
30 PRINT "HALLO ";LEFT$(NAAM$,AANTAL);
"!"
RUN
HOE HEET U ?:JOHAN KARDINAAL
HOEVEEL LETTERS HEEFT UW VOORNAAM ? 5
HALLO JOHAN!
Ok
```

moeilijkheidsgraad . . . . . **zeer eenvoudig**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **LEN is afkorting van length –lengte**

schrijfwijze

```
LEN(<TE METEN STRING>)
<TE METEN STRING>::=<A>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de lengte van de te meten string. Spaties en niet afdrukbare karakters die in de te meten string (een alfanumerieke uitdrukking) voorkomen, worden meegeteld.

Voorbeeld:

```
NEW
10 INPUT "HOE HEET U ";NAAM$
20 PRINT "UW NAAM IS";LEN(NAAM$);"LETTER
S LANG"
RUN
HOE HEET U ? FERDINAND
UW NAAM IS 9 LETTERS LANG
Ok
```

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **LET is laat**

schrijfwijze

```
[LET]<NUMERIEKE VARIABELE>=<N>|[LET]<ALFA-
  NUMERIEKE VARIABELE>=<A>
<ALFANUMERIEKE VARIABELE>::=<ZIE ALGEMENE
  SPECIFICATIES>
<NUMERIEKE VARIABELE>::=<ZIE ALGEMENE
  SPECIFICATIES>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met het LET-kommando kan aan een variabele een bepaalde waarde worden toegekend. Indien een variabele nog niet eerder werd genoemd, wordt deze automatisch gecreëerd.

Bijvoorbeeld:

```
NEW
10 LET A=4
20 LET A$="TEST"
30 PRINT A;A$
RUN
 4 TEST
Ok
```

Het sleutelwoord LET mag volledig worden weggelaten. De volgende twee programmaregels zijn dan ook volledig identiek:

```
20 LET A$="TEST"
20 A$="TEST"
```

Indien een variabele zowel rechts als links van het eerste gelijk-teken voorkomt, dan wordt eerst de uitdrukking rechts van het eerste gelijk-teken uitgerekend en wordt daarna pas het eindresultaat aan de betreffende variabele toegekend. Voorbeeld:

```
NEW
10 INPUT "WAARDE ";WAARDE
20 LET WAARDE=WAARDE+WAARDE+WAARDE
30 PRINT "DRIE MAAL DEZE WAARDE IS ";WAARDE
40 END
RUN
WAARDE 12.5
DRIE MAAL DEZE WAARDE IS 37.5
Ok
```

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **LINE is lijn**

schrijfwijze

```
LINE[@][<LOCATIE>]-<LOCATIE>[,[<KLEUR>][,
  B¦,BF]\LINE<...>,
<LOCATIE>::=[STEP(<HORIZONTAAL>,<VERTI-
  KAAL>)]
<HORIZONTAAL>::=<N>
<VERTIKAAL>::=<N>
<KLEUR>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
<...>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Voor goed begrip van dit kommando is het noodzakelijk dat eerst de PSET-behandeling volledig is doorgenomen.

Met dit kommando kunnen volledige lijnen of rechthoeken op het beeldscherm worden getekend. Een lijn wordt getrokken vanuit de eerste locatie naar de tweede in het LINE-kommando aangegeven locatie. Indien de eerste locatie niet wordt opgenomen, dan wordt het laatste getekende punt als beginlocatie genomen. Beide locaties mogen zich buiten het beeldscherm bevinden. Bijvoorbeeld:

```
NEW
10 SCREEN 2
20 LINE (10,10)-(100,100)
30 GOTO 30
RUN
```

(een lijn wordt getrokken)

```
20 LINE (10,10)-STEP(90,90)
RUN
```

(alleen regel 20 werd veranderd. Dezelfde lijn wordt getrokken)

```
15 PSET (-10,-10)
20 LINE -STEP(100,100)
RUN
```

Regel 15 werd toegevoegd en regel 20 werd veranderd. Een lijn wordt getrokken vanaf het denkbeeldige punt (−10,−10) naar (90,90).

```
15
20 LINE@(-100,-100)-(500,500)
RUN
```

Regel 15 werd weer verwijderd en regel 20 werd veranderd. Een lijn wordt tussen de twee denkbeeldige punten getrokken.

De toevoeging van het karakter @ is toegestaan. Deze toevoeging dient om LINE te onderscheiden van LINE INPUT en heeft verder geen funktie. We zullen deze toevoeging in alle voorbeelden verder weglaten.

Indien bij het LINE-kommando geen kleur wordt gespecificeerd, wordt de voorgrondkleur die op dat moment actief is, genomen als tekenkleur. Uiteraard kan een andere voorgrondkleur worden geactiveerd. Bijvoorbeeld:

```
NEW
10 COLOR 15,4,4
20 SCREEN 2
30 LINE (1,1)-(233,120),6
40 GOTO 40
RUN
```

Een donkerrode streep wordt op een donkerblauwe achtergrond getekend. Merk op dat het COLOR-kommando VOOR het SCREEN-kommando werd gegeven om de juiste achtergrondkleur en randkleur te realiseren.

Indien de letter B (van BLOCK) wordt opgenomen, wordt er plotseling geen lijn meer getrokken maar wordt de rechthoek getekend waarvan de twee aangegeven punten respectievelijk de linker bovenhoek en

de rechter onderhoek vormen. Bijvoorbeeld:

```
NEW
10 SCREEN 2
20 LINE (20,20)-(200,200),12,B
30 GOTO 30
RUN
```

Een donkergroene rechthoek wordt getekend.

```
20 LINE (20,20)-(200,200),,B
```

Alleen programmaregel 20 werd gewijzigd. De rechthoek wordt nu in de geactiveerde voorgrondkleur getekend.

Indien een rechthoek gedeeltelijk of geheel buiten het beeldschermbereik valt, worden de buiten het beeld vallende lijnen van de rechthoek toch getekend. Dit gebeurt aan de corresponderende uiterste rand van het beeldscherm. Dit is onder meer in het bovenstaande voorbeeld het geval.

Indien de letters BF (block filled) worden opgenomen, wordt de rechthoek ook nog ingekleurd. Bijvoorbeeld:

```
20 LINE (20,20)-(200,200),12,BF
```

Alleen regel 20 werd vervangen. De rechthoek uit het voorlaatste voorbeeld wordt nu ook donkergroen ingekleurd.

Ter afsluiting een combinatievoorbeeld:

```
NEW
10 COLOR ,0,0
20 SCREEN 2
30 PSET (0,0)
40 FOR I=0 TO 191 STEP 10
50 LINE -STEP(10,10),RND(1)*15,BF
60 NEXT I
70 GOTO 30
RUN
```

SLEUTELWOORD # LINE INPUT

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **LINE INPUT is regel ingave/invoer**

schrijfwijze

```
                    ["<SATELLIETTEKST>";]
LINE INPUT ----------------------------- <ALFANUME-
                    #<KANAAL>,
  RIEKE VARIABELE>
<SATELLIETTEKST>::=<ALFANUMERIEKE KON-
  STANTE>
<ALFANUMERIEKE KONSTANTE>::=<ZIE ALGEME-
  NE SPECIFICATIES>
<ALFANUMERIEKE VARIABELE>::=<ZIE ALGEME-
  NE SPECIFICATIES>
<KANAAL>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Eenvoudig gebruik

Zie ook de beschrijving van het INPUT-kommando.
De LINE INPUT is een kommando dat erg veel lijkt op de INPUT. Ook met het LINE INPUT-kommando kunnen tijdens de programmaloop gegevens aan de computer worden toegevoerd. De in te voeren gegevens moeten alfanumeriek zijn terwijl per LINE INPUT-kommando slechts één alfanumerieke variabele met gegevens kan worden gevuld. Bijvoorbeeld:

```
NEW
10 LINE INPUT "GEEF UW NAAM IN:";NAAM$
20 PRINT "HALLO, ";NAAM$
30 STOP
RUN
NAAM:FREDERIK
```

```
HALLO, FREDERIK
Break in 30
Ok
```

In tegenstelling tot de INPUT presenteert de LINE INPUT bij de ingave geen vraagteken. Ook in tegenstelling tot de INPUT is, dat bij de LINE INPUT ook komma's en aanhalingstekens mogen worden ingegeven.

Indien in plaats van een ingave bij een LINE INPUT alleen een RETURN-toets wordt ingegeven, wordt een lege ingave aangenomen; de oude inhoud van de input-variabele wordt in dat geval (in tegenstelling tot hetgeen bij de INPUT het geval is) niet behouden.

Gevorderd gebruik

Zie hiervoor ook de beschrijving van het PRINT-kommando, onderdeel gevorderd gebruik: schrijven naar randapparatuur alsmede het OPEN en CLOSE-kommando.

Indien een kanaalnummer is gespecificeerd, kan de invoer van gegevens vanaf een ander apparaat dan het toetsenbord plaatsvinden; bijvoorbeeld vanaf een cassetterecorder. Voorwaarde is, dat de gegevens regelgewijs op cassette zijn opgeslagen. Men kan stellen dat de gegevens van het betreffende randapparaat dienen te worden aangevoerd zoals deze ook vanaf het toetsenbord worden ingevoerd. Voorbeeld:

Eerste programma, schrijven van enkele gegevens naar cassetteband:

```
NEW
10 OPEN "CAS:TEST" FOR OUTPUT AS 1
20 LINE INPUT "GEGEVENS:";A$
30 IF A$="" THEN 100
40 PRINT #1,A$
50 GOTO 20
100 CLOSE:STOP
```

(nu eerst een cassetteband in de cassetterecorder plaatsen en de recorder op opnemen zetten)

```
RUN
```

(band loopt even en stopt weer)

```
GEGEVENS:DIT IS EEN STUK TEST-TEKST
GEGEVENS:DAT STRAKS WEER WORDT INGELEZEN
GEGEVENS:TEST,TEST,TEST,TEST.
GEGEVENS:
Break in 100
Ok
```

(hier wordt alleen een RETURN ingegeven; de band loopt even en stopt weer)

(nu de cassetteband terugspoelen)

Tweede programma, lezen van gegevens van cassetteband:

```
NEW
10 OPEN "CAS:TEST" FOR INPUT AS 1
20 LINE INPUT #1,A$
30 PRINT A$:IF EOF(1) THEN CLOSE:STOP
40 GOTO 20
RUN
```

(cassetterecorder nu op afspelen zetten)

```
DIT IS EEN STUK TEST-TEKST
DAT STRAKS WEER WORDT INGELEZEN
TEST,TEST,TEST,TEST.
Break in 30
Ok
```

(de band stopt)

Indien u dit laatste voorbeeld uitprobeert, denk er dan wel aan, het volume van de cassetterecorder op ongeveer 80% te zetten. De MSX-standaard schrijft geen controles voor op het juiste inlezen van gegevens. Afhankelijk van de instelling en de kwaliteit van de cassetterecorder kunnen gegevens verminkt worden ingelezen.

Zie voor de verklaring van het gebruik van het sleutelwoord FOF de verklaring onder dit sleutelwoord.

SLEUTELWOORD # LIST

moeilijkheidsgraad . . . . . **zeer eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **LIST is lijst maken**

schrijfwijze

```
LIST[<REGELNUMMER>][-[<REGELNUMMER>]]
<REGELNUMMER>::=<TYPE 1 INTEGERE KONSTAN-
   TE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE ALGE-
   MENE SPECIFICATIES>
```

betekenis

Met dit kommando kunt u op het beeldscherm:

- Het volledige programma laten lijsten. Geef in dat geval slechts LIST in.
- Het eerste gedeelte van uw programma laten lijsten. Geef in dat geval LIST in, gevolgd door een minteken en het laatste te lijsten regelnummer.
- Het laatste gedeelte van uw programma laten lijsten. Geeft u in dat geval LIST in, gevolgd door het eerste te lijsten regelnummer en een minteken.
- Een middengedeelte van uw programma laten lijsten. Geeft u in dat geval LIST in, gevolgd door het eerste te lijsten regelnummer, een minteken en het tweede te lijsten regelnummer
- één enkel regelnummer laten lijsten. Geef in dat geval LIST in, gevolgd door het betreffende regelnummer of een punt voor het laatste behandelde regelnummer.

Voorbeelden:

```
LIST
```

lijst het gehele programma

```
LIST -100
```

lijst alle regels tot en met regel 100

```
LIST 100-
```

lijst alle regels vanaf regel 100

```
LIST 100-200
```

lijst alle regels vanaf 100 tot en met 200

```
LIST 100
```

lijst alleen regel 100

```
LIST .
```

lijst het laatst behandelde regelnummer

De aangegeven regelnummers behoeven niet noodzakelijkerwijs aanwezig te zijn.

LIST zit in twee verschillende uitvoeringen standaard onder de funktie-

moeilijkheidsgraad . . . . . **zeer eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **LLIST is afkorting van lineprinter list – lijst maken op afdrukeenheid (printer)**

schrijfwijze

```
LLIST[<REGELNUMMER>][-[<REGELNUMMER>]]
<REGELNUMMER>::=<TYPE 1 INTEGERE KONSTAN-
  TE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE ALGE-
  MENE SPECIFICATIES>
```

betekenis

Het LLIST-kommando werkt precies als het LIST-kommando met dit verschil, dat de programmaregels niet op het beeldscherm maar op de printer worden afgedrukt. Zie verder de behandeling van LIST.

SLEUTELWOORD # LOAD

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **LOAD is laden**

schrijfwijze

```
LOAD<BESTANDSNAAM>[,R]
<BESTANDSNAAM>::=<A>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het is noodzakelijk dat eerst de behandeling van SAVE wordt doorgenomen.

Programma's die met een SAVE zijn vastgelegd, kunnen met een LOAD weer in de computer worden teruggelezen.

Eenvoudig gebruik

een voorbeeld op cassetterecorder:

```
NEW
10 REM DIT IS EEN TESTPROGRAMMA
20 PRINT "TEST"
30 GOTO 20
```

(cassetterecorder op opnemen zetten)

```
SAVE "CAS:TEST"
Ok
NEW
LOAD "CAS:TEST"
```

(CAS: staat voor cassetterecorder; het programma heet TEST)

(terugspoelen en afspelen)

```
FOUND:TEST
Ok
```

```
LIST
10 REM DIT IS EEN TESTPROGRAMMA
20 PRINT "TEST"
30 GOTO 20
Ok
```

Met de achtervoeging ,*R* kan het programma direkt worden opgestart:

```
NEW
10 LOAD "CAS:TEST",R
RUN
```

(terugspoelen en afspelen)

Na verloop van tijd is het eerste testprogramma geladen en begint de computer onmiddellijk TEST af te drukken.

Merk op dat wanneer het LOAD-kommando in een programmaregel is opgenomen, de mededelingen FOUND: (of SKIP:) achterwege blijven.

MSX schrijft geen controle voor op de integriteit van de ingelezen gegevens. Een verkeerd ingesteld volume of kwalitatief slechte apparatuur kan leiden tot verminkte programma's bij het inlezen zonder dat daar melding van wordt gemaakt. Zet het volume bij afspelen altijd op ongeveer 80 pct.

N.B.: Het lezen van of schrijven naar cassetteband kan, wanneer het een groot programma betreft, wel tot enkele minuten duren.

Gevorderd gebruik

Wanneer de toevoeging ,*R* wordt gebruikt, dan worden kanalen opengelaten en het programma onmiddellijk opgestart. Voorbeeld:

Eerst gaan we een programma schrijven en SAVEn:

```
NEW
10 PRINT #1,"DIT IS EEN TEST"
20 GOTO 10
```

(cassetterecorder op opnemen)

```
SAVE "CAS:TEST"
Ok
```

Vervolgens schrijven we een tweede programma dat het alfanumerieke beeldscherm opent en vervolgens ons eerdere programma activeert:

```
NEW
10 OPEN "CRT:" AS 1
20 LOAD "CAS:TEST",R
RUN
```

(terugspoelen en afspelen)

```
DIT IS EEN TEST
DIT IS EEN TEST
DIT IS...
etcetera
```

Uit dit voorbeeld blijkt:

- kanaal 1 bleef open staan
- programma TEST werd na laden onmiddellijk uitgevoerd.

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **LOCATE is plaatsen**

schrijfwijze

```
LOCATE[<TEKENPOSITIE>][,[<REGELPOSITIE>]
[,<CURSOR>]]\LOCATE[<...>,]
<TEKENPOSITIE>::=<N>
<REGELPOSITIE>::=<N>
<CURSOR>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
<...>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met deze funktie kan de cursor gepositioneerd worden op een bepaalde plaats op het beeldscherm. Voorbeeld:

```
NEW
10 FOR X=0 TO 24:FOR Y=0 TO 20
20 LOCATE X,Y:PRINT "*"
30 NEXT Y,X:STOP
RUN
```

(In de linkerbovenhoek van het beeldscherm worden van boven naar beneden sterren afgedrukt.)
Merk op dat positie 0,0 wordt gevormd door de linkerbovenhoek van het scherm.

Eén van de twee beeldscherm-coördinaten mag worden weggelaten. Bijvoorbeeld:

```
NEW
10 LOCATE 20:PRINT "HALLO";
20 LOCATE ,0:PRINT "HIER BEN IK"
RUN
```

(De tekst HALLO wordt 20 posities uit de kantlijn afgedrukt.) Op de eerste regel van het beeldscherm wordt, vier posities verder uit de kantlijn, de tekst HIER BEN IK afgedrukt.

Met LOCATE kan ook de cursor zichtbaar of onzichtbaar worden gemaakt tijdens het projekteren. Het derde element achter LOCATE dient dan een 0 (onzichtbaar) of een positieve waarde, ongelijk aan 0 maar kleiner dan 256 (zichtbaar) te bevatten.

Indien alleen de cursor dient te worden veranderd, dient een LOCATE „0 (onzichtbaar) of bijvoorbeeld een LOCATE „1 (zichtbaar) te worden gebruikt.

De aangestuurde cursorposities mogen de beeldschermgrenzen natuurlijk niet overschrijden. De betreffende beeldschermmaten worden bij SCREEN behandeld.

Nog wat voorbeelden:

LOCATE 10,20,0	de cursor wordt op positie 10 (horizontaal), 20 (vertikaal) geplaatst en de cursor wordt onzichtbaar
LOCATE ,5,1	de corsor verplaatst zich naar de vijfde regel en wordt zichtbaar
LOCATE 11,,0	de cursor verplaatst zich naar de elfde positie horizontaal en wordt onzichtbaar.

Tijdens een INPUT of een LINE INPUT is de cursor altijd zichtbaar, hoe ook ingesteld. LOCATE heeft geen zin wanneer een grafisch scherm actief is. Zie hiervoor SCREEN.

moeilijkheidsgraad .. **vrij moeilijk, kennis van logarithmische funkties is vereist**
soort ..... **N-FUNKTIE**
afkomst ..... **LOG is afkorting van natural logarithm – natuurlijke logarithmen**

schrijfwijze

```
LOG(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de natuurlijke logarithme van de waarde van de tussen haakjes genoemde numerieke uitdrukking. Bijvoorbeeld:

```
NEW
10 INPUT "WAARDE ";A
20 PRINT "NAT. LOG.=";LOG(A)
30 GOTO 10
RUN
WAARDE ? 12
NAT. LOG.= 2.4849066497879
WAARDE ? 2.7182818285
NAT. LOG.= 1.000000000015
WAARDE ? 0
NAT. LOG.=
Illegal function call in 20
Ok
```

Uiteraard dient de tussen haakjes vermelde waarde positief te zijn.

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **N-FUNKTIE**
afkomst ..... **LPOS is afkorting van line printer headposition – positie van de kop van de afdrukeenheid (printer)**

schrijfwijze

```
LPOS(<U>)
<U>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de positie van de printercursor. De printercursor is niet zichtbaar, maar men moet zich deze voorstellen als iets dat de huidige positie op de printer aangeeft. Merk op dat de LPOS- en de POS-funktie veel met elkaar gemeen hebben.
De tussen haakjes op te nemen uitdrukking is een dummy-uitdrukking; in de praktijk kan het beste gewoon het cijfer 0 worden ingevuld. Bijvoorbeeld:

(voor dit voorbeeld is een aangesloten printer noodzakelijk)

```
NEW
10 LPRINT
20 PRINT LPOS(0)
30 LPRINT "DIT IS EEN TEST";
40 PRINT LPOS(0)
50 LPRINT "JE"
60 PRINT LPOS(0)
RUN
 0
 15
 0                    (op de printer verschijnt: DIT IS EEN TESTJE)
Ok
```

moeilijkheidsgraad . . **eenvoudig maar in complexe samenstellingen vrij moeilijk**
soort . . . . . **KOMMANDO**
afkomst . . . . . **LPRINT is samtrekking van lineprinter en print afdrukken op afdrukeenheid (printer)**

schrijfwijze

```
LPRINT[{,|;}<P>{,|;[<P>]}][USING<USING
STRING>;<U>{,|;<U>}]
<P>::=<U>|TAB(<POSITIE>)|SPC(<AANTAL
  SPATIES>)
<POSITIE>::=<N>
<AANTAL SPATIES>::=<N>
<USING STRING>::=<A>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
<U>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

De LPRINT werkt hoegenaamd volkomen identiek aan de PRINT. Het enige verschil is, dat de afdruk niet op het beeldscherm, maar op een eventueel aangesloten afdrukeenheid (printer) geschiedt.

Zie voor een verdere behandeling de behandeling onder PRINT.

SLEUTELWOORD # MAXFILES

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **SYSTEEMVARIABELE**
afkomst . . . . . **MAXFILES is afkorting van maximum number of files – maximaal aantal bestanden**

schrijfwijze

```
MAXFILES=<MAXIMUM AANTAL BESTANDEN>
<MAXIMUM AANTAL BESTANDEN>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Een bestand (file) is een verzameling van bij elkaar behorende gegevens. Deze gegevens kunnen in het computergeheugen staan maar kunnen ook op cassetteband of op een floppy disk staan.
Een bestand op cassette of floppy disk kan met MSX-basic op een aparte wijze worden benaderd.

Het aantal bestanden dat tegelijkertijd kan worden benaderd, wordt door MSX-basic bij het opstarten van de computer standaard op de waarde 1 gezet. Meestal is dit ook genoeg; het zal niet vaak voorkomen dat er meer dan één bestand tegelijkertijd wordt aangeroepen tenzij men in het bezit is van een schijveneenheid.

Wanneer men het aantal tegelijk te openen bestanden wil vergroten, dan kan dat door de systeemvariabele MAXFILES op een waarde te stellen, niet kleiner dan nul en niet groter dan 15. Een eventuele decimale fractie wordt verwaarloosd.

Elke extra mogelijkheid om een bestand tegelijkertijd te benaderen, kost wat geheugenruimte omdat de computer een gedeelte hiervan voor het lezen en schrijven van en naar het betreffende bestand dient te reserveren. Deze stukjes voor bestandsbenadering gereserveerd geheugen noemt men buffers. Per buffer is de grootte in MSX-basic 267 bytes (karakters).
Het is dus zaak om de systeemvariabele MAXFILES zo klein mogelijk te houden om zoveel mogelijk geheugen vrij te houden. Voorbeeld:

```
NEW
```

```
PRINT FRE(0)
 28815
Ok
MAXFILES=0
Ok
PRINT FRE(0)
 29082
Ok
MAXFILES=15
Ok
PRINT FRE(0)
 25077
Ok
```

We zien dat alleen door MAXFILES aan een bepaalde waarde gelijk te stellen, aanzienlijke stukken geheugen al dan niet worden gereserveerd.

MAXFILES heeft nog een bijverschijnsel: MAXFILES voert automatisch ook een CLEAR uit. Bijvoorbeeld:

```
NEW
10 LET A=12.5
20 LET A$="TEST"
30 MAXFILES=0
40 PRINT A;A$
RUN
 0
Ok
```

Na MAXFILES zijn alle variabelen schoongemaakt.

Het sleutelwoord LET mag niet voor MAXFILES worden gebruikt. MAXFILES mag alleen maar aan een bepaalde waarde worden gelijkgesteld en kan niet op een andere wijze worden gebruikt. Bijvoorbeeld:

```
NEW
10 MAXFILES=5
20 PRINT MAXFILES
RUN
Syntax error in 20
Ok
```

SLEUTELWOORD # MERGE

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **MERGE is samenvoegen**

schrijfwijze

```
MERGE<BESTANDSNAAM>
<BESTANDSNAAM>::=<A>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het is noodzakelijk dat eerst de LOAD- en SAVE-behandeling is doorgenomen.

Programma's of programma-onderdelen die met een SAVE zijn vastgelegd, kunnen met een MERGE weer worden opgehaald. MERGE werkt op twee punten na identiek aan LOAD:

- LOAD maakt eerst het oude programmageheugen schoon, MERGE doet dat niet
- MERGE staat de ,*R* toevoeging niet toe.

Het eerste punt is zeer belangrijk. Doordat oude programmaregels niet worden verwijderd, kunnen verschillende programma's met elkaar worden *gecombineerd*. Bijvoorbeeld:

```
NEW
10 REM EERSTE PROGRAMMA
20 PRINT "TEST DEEL 1"
30 STOP
SAVE "CAS:TEST1"                (eerst cassetterecorder op
Ok                               opnemen zetten)
NEW
11 REM TWEEDE PROGRAMMA
30 PRINT "TEST DEEL 2"
40 STOP
SAVE "CAS:TEST2"
```

```
NEW
LOAD "CAS:TEST1"
FOUND:TEST1
Ok
MERGE "CAS:TEST2"
FOUND:TEST2
Ok
LIST
10 REM EERSTE PROGRAMMA
11 REM TWEEDE PROGRAMMA
20 PRINT "TEST DEEL 1"
30 PRINT "TEST DEEL 2"
40 STOP
Ok
```

(cassetterecorder terugspoelen en op afspelen zetten)

In bovenstaand voorbeeld werden twee onafhankelijke programma's, TEST1 en TEST2, apart op cassetteband gezet. Later werden deze twee in één programma samengevoegd. We zien:

- bij een MERGE wordt een programma bij een reeds in de computer aanwezig programma gevoegd
- indien een regelnummer al aanwezig is, wordt dit verwijderd ten behoeve van de in te lezen regel.

Men kan zich de MERGE-opdracht het beste voorstellen als ware het dat tijdens het uitvoeren van deze opdracht, het programma dat door MERGE wordt aangesproken, heel snel wordt ingetoetst (vanaf de cassetteband).

Door de MERGE wordt het zinvol, grote programma's in gedeelten te ontwerpen en pas later, nadat de onderdelen zijn uitgetest, samen te voegen tot één geheel. Men zal dan al snel tot de conclusie komen dat sommige programma-onderdelen vaak voor meer dan één programma kunnen worden gebruikt.

SLEUTELWOORD # MID$

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **MID$ is afkorting van middle part of string – midden gedeelte van de string**

schrijfwijze

```
MID$(<TE WIJZIGEN STRING>,<EERSTE TE-
  KEN>[,<AANTAL TEKENS>])=<A>
<EERSTE TEKEN>::=<N>
<AANTAL TEKENS>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
<TE WIJZIGEN STRING>::=<ALFANUMERIEKE
  VARIABELE>
<ALFANUMERIEKE VARIABELE>::=<ZIE ALGE-
  MENE SPECIFICATIES>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met dit kommando (niet te verwarren met de MID$-*funktie*) kunnen delen van een alfanumerieke variabele worden vervangen. Welk gedeelte wordt vervangen, wordt bepaald door het opgegeven eerste teken en het aantal tekens. Waardoor wordt vervangen, wordt bepaald door de na het gelijkteken opgenomen alfanumerieke uitdrukking. Bijvoorbeeld:

```
NEW
10 LET A$="???-BASIC"
20 MID$(A$,1,3)="MSX"
30 PRINT A$
RUN
MSX-BASIC
Ok
```

Het opgegeven eerste teken mag niet kleiner zijn dan 1 en niet groter zijn dan de lengte van de te wijzigen string. Hieruit volgt dat alleen een reeds bestaande alfanumerieke variabele binnen het MID$-kommando

mag worden genoemd.
Het aantal tekens mag niet kleiner zijn dan nul en niet groter dan 255.
Decimale fracties worden verwaarloosd.

Indien het aantal tekens wordt weggelaten, dan wordt een gedeelte vervangen tot aan de lengte van de te wijzigen string of totdat de alfanumerieke uitdrukking waarmee moet worden vervangen, 'op' is.
Bijvoorbeeld:

```
NEW
10 LET A$="DIT IS EEN TEST"
20 MID$(A$,5)="WAS EEN TEST"
30 PRINT A$
40 A$=A$+"T"
50 PRINT A$
60 MID$(A$,5)="IS"
70 PRINT A$
80 MID$(A$,7,1)=STRING$(32," ")
90 PRINT A$
RUN
DIT WAS EEN TES
DIT WAS EEN TEST
DIT ISS EEN TEST
DIT IS  EEN TEST
Ok
```

De eerste regel laat zien dat er maximaal vervangen wordt tot aan de lengte van de te wijzigen alfanumerieke variabele.
De derde regel na RUN laat zien dat er vervangen wordt totdat de alfanumerieke uitdrukking waarmee moet worden vervangen, 'op' is.

N.B.: Het sleutelwoord LET mag niet voor MID$ worden gebruikt.

SLEUTELWOORD # MID$

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **A-FUNKTIE**
afkomst . . . . . **MID$ is afkorting van middle part of string – midden gedeelte van de string**

schrijfwijze

```
MID$(<BASISSTRING>,<EERSTE TEKEN>[,<AAN-
  TAL TEKENS>])
<BASISSTRING>::=<A>
<EERSTE TEKEN>::=<N>
<AANTAL TEKENS>::=<N>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat een middengedeelte van een alfanumerieke uitdrukking. Welk middengedeelte wordt gegeven als resultaat, hangt af van het gespecificeerde eerste teken en het gespecificeerde aantal tekens. Bijvoorbeeld:

```
PRINT MID$("MSX-BASIC-COMPUTER",5,5)
BASIC
Ok
```

Het aantal tekens mag niet kleiner zijn dan nul en niet groter zijn dan 255. Het opgegeven eerste teken mag niet kleiner zijn dan 1 en niet groter dan 255. Eventuele decimale fracties worden verwaarloosd.

Indien het aantal tekens niet wordt opgegeven, dan wordt vanaf het eerste teken de rest van de waarde van de alfanumerieke uitdrukking als resultaat gegeven. Bijvoorbeeld:

```
NEW
10 LET A$="DIT IS EEN TEST"
20 LET B$=LEFT$(A$,3)
30 LET C$=MID$(A$,5,2)
```

```
40 LET D$=MID$(A$,8,3)
50 LET E$=MID$(A$,12)
60 PRINT C$;" ";B$;" ";D$;" ";E$;"?"
RUN
IS DIT EEN TEST?
Ok
```

N.B.: De funktie MID$ en het kommando MID$ dienen niet met elkaar te worden verward.

SLEUTELWOORD # MOTOR

moeilijkheidsgraad . . . . . **zeer eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **MOTOR is motor**

schrijfwijze

MOTOR [ON / OFF]

betekenis

Alle cassetterecorderbesturende opdrachten in MSX kunnen de cassetterecordermotor ook aan- of uitschakelen. Hiertoe is een speciale REMOTE-aansluiting (afstandsbediening) voorgeschreven onder MSX Bij een CSAVE-kommando en een CLOAD-kommando zien we, da de cassetterecorder door de computer automatisch wordt ingeschakeld totdat de opdracht is voltooid. Na de opdracht wordt de cassetterecordermotor weer uitgeschakeld.

De cassetterecordermotor kan ook apart worden in- en uitgeschakeld zonder dat er direkt geladen of geschreven wordt. In dat geval dienen we het MOTOR-kommando te gebruiken. Wanneer we bijvoorbeeld de cassette in de recorder terug willen spoelen, kunnen we het afstandsbedieningsstekkertje uit de recorder halen en dan daadwerkelijk terugspoelen. Gemakkelijker is het om eerst

MOTOR ON

in te toetsen en dan terug te spoelen; de cassetterecorder is ingeschakeld. Om de cassetterecorder weer uit te schakelen, geven we in:

MOTOR OFF

In een programma kan het MOTOR-kommando worden gebruikt bij begeleide bediening van de cassetterecorder, bijvoorbeeld:

```
NEW
10 PRINT "ZET DE RECORDER
OP TERUGSPOELEN"
```

```
20 PRINT "EN GEEF EEN TOETS IN"
30 LET K$=INKEY$:IF K$="" THEN 30
40 MOTOR ON
50 PRINT "GEEF EEN TOETS IN ALS
DE BAND"
60 PRINT "IS TERUGGESPOELD"
70 LET K$=INKEY$:IF K$="" THEN 70
80 MOTOR OFF
90 PRINT "ZET NU DE RECORDER OP
OPNEMEN"
100 PRINT "EN GEEF EEN TOETS IN"
110 LET K$=INKEY$:IF K$="" THEN 110
120 REM LAAT DE BAND EEN PAAR SECONDEN
130 REM LOPEN VOOR AANLOOPTAPE
140 PRINT "EEN OGENBLIK"
150 MOTOR ON:FOR I=1 TO 5000:NEXT
I:MOTOR OFF
160 PRINT "PROGRAMMA WORDT GESCHREVEN"
170 CSAVE "TEST1",2
180 PRINT "PROGRAMMA GESCHREVEN"
190 PRINT "ZET DE RECORDER AF"
200 END
```

In het bovenstaande programma, dat als subroutine eventueel in een totaalprogramma zou kunnen worden opgenomen, wordt het op band zetten van een programma voor een groot deel begeleid. Eventueel zou aansluitend op dit voorbeeld op dezelfde wijze nog een controle met CLOAD? kunnen worden ingebouwd.

Het MOTOR-kommando kent drie verschijningen:

MOTOR ON	de cassetterecorder wordt ingeschakeld
MOTOR OFF	de cassetterecorder wordt uitgeschakeld
MOTOR	de cassetterecorder wordt ingeschakeld indien deze uitgeschakeld was en wordt uitgeschakeld indien deze ingeschakeld was.

Het MOTOR-kommando bedient een relais dat de stroomtoevoer naar de cassetterecordermotor al dan niet onderbreekt. Dit relais is licht

belastbaar en kan voor allerlei doeleinden door de handige programmeur worden aangewend. Het automatisch besturen van een diaprojektor is bijvoorbeeld een vrij spectaculaire en in bepaalde toepassingen erg bruikbare mogelijkheid.

SLEUTELWOORD # NEW

moeilijkheidsgraad . . . . . **zeer eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **NEW is nieuw**

schrijfwijze

NEW

betekenis

Het NEW-kommando heeft tot gevolg dat het programmageheugen geheel wordt schoongewist. Ook variabelen worden verwijderd en alle openstaande kanalen worden gesloten. Na een NEW komt de computer altijd met de Ok-melding, ook wanneer de NEW in een regel werd opgenomen.

Voorbeeld:

```
NEW
10 PRINT "HALLO ALLEMAAL"
30 NEW
RUN
HALLO ALLEMAAL
Ok
LIST
Ok                                (programma is verdwenen)
```

SLEUTELWOORD NEXT

moeilijkheidsgraad ..... **normaal**
soort ..... **KOMMANDO**
afkomst ..... **NEXT is volgende**

schrijfwijze

```
NEXT[<NUMERIEKE VARIABELE>{,<NUMERIEKE
  VARIABELE>}
<NUMERIEKE VARIABELE>::=<ZIE ALGEMENE
  SPECIFICATIES>
```

betekenis

Dit kommando wordt volledig bij FOR...TO...STEP behandeld.
Zie aldaar.

moeilijkheidsgraad . . **vrij moeilijk, vereist kennis van talstelsels en algemene principes van het computergeheugen**
soort . . . . . **A-FUNKTIE**
afkomst . . . . . **OCT$ is afkorting van octal string – octale (achttallige) string**

schrijfwijze

```
OCT$(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft de octale waarde van de tussen haakjes vermelde uitdrukking weer in string-vorm. De waarde van de numerieke uitdrukking dient te liggen tussen –32769 en 65536. Indien de waarde van de numerieke uitdrukking gebroken is, wordt de grootste gehele waarde, kleiner dan de gebroken waarde als geldende waarde genomen.
Indien de waarde van de numerieke uitdrukking negatief is, geeft OCT$ een octale representant, samengesteld volgens de tweecomplementmethode. Bijvoorbeeld:

```
PRINT OCT$(-32768)
100000
Ok
PRINT OCT$(65535)
177777
Ok
PRINT OCT$(-1)
177777
Ok
PRINT OCT$(1023)
1777
Ok
```

SLEUTELWOORD # ON ERROR GOTO

moeilijkheidsgraad . . . . . **vrij eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **ON ERROR GOTO is ga bij een fout naar (goto – samentrekking van go to)**

schrijfwijze

```
ON ERROR GOTO[<REGELNUMMER>]
<REGELNUMMER>::=<TYPE 1 INTEGERE KONSTAN-
  TE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE ALGE-
  MENE SPECIFICATIES>
```

betekenis

Met dit kommando kan een fout die in het programma optreedt, worden gedetecteerd en kan een bepaalde actie worden ondernomen.

Wanneer de computer de ON ERROR GOTO tegenkomt, wordt geen actie ondernomen. De computer 'weet' echter vanaf dat moment dat wanneer een fout optreedt, hij met het vermelde regelnummer dient verder te gaan.

Bijvoorbeeld:

```
NEW
10 ON ERROR GOTO 1000
20 INPUT "WAARDE ";A
30 PRINT 1/A:GOTO 20
1000 PRINT "FOUT ONTDEKT":STOP
RUN
WAARDE ? 1
 1
WAARDE ? 2
 .5
WAARDE ? 0
FOUT ONTDEKT
Break in 1000
Ok
```

Doordat de laatste ingave een nul werd ingegeven, moet op regel 30 door nul worden gedeeld. Dit resulteert in een fout waardoor de computer regel 1000 en verder ging uitvoeren.
De regel waarop de foutmelding plaatsvond en het nummer van de fout zijn in de systeemvariabelen ERL en ERR terug te vinden. Bijvoorbeeld:

```
NEW
10 ON ERROR GOTO 100
20 LET A$=12
30 STOP
500 PRINT "FOUT NUMMER";ERR
510 PRINT "OP REGEL";ERL
520 STOP
RUN
FOUT NUMMER 13
OP REGEL 20
Break in 520
Ok
```

Op regel 20 werd een Type mismatch foutmelding (fout 13, zie hoofdstuk 13 en 14) geforceerd. Hierdoor werd verder gegaan met programmaregel 500. Het foutnummer en de regel waarop de fout optrad, worden vermeld.

Vanuit de ERROR-ROUTINE (het programmaonderdeel waarin de foutmeldingen worden behandeld) kan het programma eventueel weer worden verdergestart met een RESUME of een RESUME 0. Het effect hiervan is dat het kommando waarin de fout werd ontdekt, nogmaals wordt uitgevoerd. De error-routine moet dan natuurlijk een zodanige actie hebben ondernomen dat de fout niet meer optreedt. Bijvoorbeeld:

```
NEW
10 ON  ERROR GOTO 200
20 A=0:B=1/A
30 STOP
200 A=1:RESUME
RUN
Break in 30
Ok
```

Dit weinig zinvolle programma wordt pas interessant wanneer we eerst een TRON ingeven en dan pas een RUN. Het zal dan blijken dat de programmaregels in de volgende volgorde worden uitgevoerd:

10 de error-routine wordt bepaald
20 A wordt op 0 gezet. B wordt op 1/A gesteld. Dit geeft een foutmelding in verband met het delen door nul. Door deze foutmelding gaat het programma verder met regel:
200 de variabele A wordt op 1 gezet en het programma wordt vervolgd
20 wederom wordt B op 1/A gesteld. A heeft nu echter de waarde 1 waardoor geen foutmelding meer optreedt
30 het programma is ten einde.

Wanneer een kommando dat een fout veroorzaakt, dient te worden overgeslagen, kan een RESUME NEXT worden gebruikt, bijvoorbeeld:

```
NEW
10 ON ERROR GOTO 1000
20 FOR I=10 TO -10 STEP -1
30 PRINT 1/I
40 NEXT I:STOP
1000 IF ERR=11 AND ERL=30 THEN RESUME
NEXT
1010 STOP
RUN
```

Er worden 20 getallen op het scherm geprojekteerd die allemaal het resultaat zijn van de deling op regel 30. Eén van de delingen die op regel 30 wordt uitgevoerd, is de deling 1/0, namelijk wanneer I gelijk is aan nul. In dat geval wordt in de error-routine een RESUME NEXT gegeven; het fout veroorzakende kommando wordt overgeslagen.
Merk op dat in de error-routine eerst wordt gecontroleerd (in regel 1000) of inderdaad de verwachte fout op het verwachte regelnummer is opgetreden.

Indien vanuit de foutmelding dient te worden besloten dat hervatting van het programma op een geheel andere plaats dient plaats te vinden, kan een RESUME worden opgenomen, gevolgd door het regelnummer waarmee moet worden hervat. Regel 1000 uit het laatste voorbeeld had dus mogen luiden:

```
1000 IF ERR=11 AND ERL=30 THEN RESUME 40
```

Wanneer de foutafvanging moet worden afgezet, dus wanneer niet langer van de computer wordt verlangd dat bij een fout een speciale actie wordt ondernomen, kan de foutafvanging worden uitgeschakeld door de ON ERROR GOTO zonder regelnummer of met regelnummer 0 op te nemen. Bijvoorbeeld:

```
NEW
10 ON ERROR GOTO 200'FOUTAFVANGING AAN
20 LET A$=12'MAAK OPZETTELIJK EEN FOUT
30 ON ERROR GOTO'FOUTAFVANGING WEER UIT
40 LET A$=12'EN NOG EENS DE ZELFDE FOUT
50 STOP
200 RESUME NEXT'FOUT OVERSLAAN
RUN
Type mismatch in 40
Ok
```

(De fout op regel 20 werd afgevangen; die op regel 40 niet meer.)

Wanneer de ON ERROR GOTO zonder regelnummer of met regelnummer 0 IN de error-routine wordt geplaatst, heeft dat tot effect dat de laatste fout alsnog wordt afgedrukt op het beeldscherm en het programma alsnog wordt onderbroken. Bijvoorbeeld:

```
NEW
10 ON ERROR GOTO 1000
20 LET B=1/0
30 LET A$=12
40 STOP
1000 IF ERL=30 THEN ON ERROR GOTO 0
1010 RESUME NEXT
RUN
Type mismatch in 30
Ok
```

De fout op regel 20 wordt door de error-routine ontdekt en overgeslagen. De fout op regel 30 resulteert echter in een uitvoering van een ON ERROR GOTO 0 waardoor alsnog de foutmelding wordt gegeven en het programma wordt onderbroken.

In een ingewikkeld programma kan ook de error-routine ingewikkelde vormen aannemen. Om de error-routine uit te testen op goed funktioneren, moeten we in staat zijn om fouten te forceren. In de voorbeelden deden we dit al enkele malen, bijvoorbeeld met een LET A$ = 12 (geeft fout 13, Type mismatch). Veel eenvoudiger kan een fout worden geforceerd met het ERROR-kommando. Achter dit kommando behoeven we slechts het foutnummer (uit hoofdstuk 13 of 14) op te nemen. Bij uitvoering van dit kommando forceert de computer de opgegeven fout. Bijvoorbeeld:

```
NEW
10 INPUT "WELKE FOUT ";F
20 ERROR F
RUN
WELKE FOUT ? 5
Illegal funktion call in 20
Ok
RUN
WELKE FOUT ? 20
Verify error in 20
Ok
RUN
WELKE FOUT ? 99
Unprintable error in 20
Ok
```

Merk op dat de fout niet slechts wordt aangegeven maar ook daadwerkelijk wordt gesimuleerd.

Enkele opmerkingen:

- Indien een fout ontstaat IN een foutroutine, dan wordt deze fout niet meer afgevangen maar gewoon gemeld waarna het programma onderbreekt.

- Er zit een kleine fout in MSX-basic waardoor de fout-afvanging, eenmaal geactiveerd, actief blijft ook al is het programma reeds onderbroken. Hierdoor kan het voorkomen dat een programma plotseling vanuit de error-routine weer actief wordt wanneer bij het intypen van een stuk programma een fout wordt gemaakt.

Het één en ander kan verwarrende resultaten opleveren. Geef ter voorkoming een ON ERROR GOTO 0 in nadat een programma met foutafvanging is onderbroken.
Een voorbeeld van een mogelijk verwarrende situatie:

```
NEW
10 ON ERROR GOTO 100
20 STOP
100 RESUME NEXT
RUN
Break in 20
Ok
NU KAN IK
Ok
ALLES INGEVEN
Ok
DE COMPUTER GEEFT GEEN
Ok
FOUTEN MEER...
Ok
```

Nadat het bovenstaande programma eenmaal is uitgevoerd, is de computer niet meer in staat om een foutmelding te geven. Steeds bij een optredende fout wordt regel 100 geactiveerd. De RESUME NEXT geeft aan dat de regel met de fout gewoon moet worden overgeslagen; geen foutmelding verschijnt.

RENUM (zie de behandeling van dit kommando) probeert ook de numerieke uitdrukking na ERL= te hernummeren. Wanneer deze numerieke uitdrukking bestaat uit één enkele konstante, geeft dat geen problemen. Echter, wanneer de numerieke uitdrukking complexer is, resulteert RENUM in weinig zeggende foutmeldingen of (en dit is kwalijker) helemaal geen foutmeldingen. Bijvoorbeeld:

```
NEW
10 ON ERROR GOTO 1000
20 ERROR 5:STOP
1000 IF ERL=20 THEN RESUME NEXT
1010 ON ERROR GOTO
```

```
RENUM 1
Ok
LIST
1 ON ERROR GOTO 21
11 ERROR 5:STOP
21 IF ERL=11 THEN RESUME NEXT
31 ON ERROR GOTO
Ok
```

In het bovenstaande voorbeeld ging ook het hernummeren van ERL goed. Indien we echter de volgende verandering in het programma aanbrengen:

```
21 LET A=10
22 IF ERL=21-A THEN RESUME NEXT
RENUM
Ok
```

dan volgt geen foutmelding. Het programma ziet er echter als volgt uit:

```
LIST
10 ON ERROR GOTO 40
20 ERROR 5:STOP
30 LET A=10
40 IF ERL=20-A THEN RESUME NEXT
50 ON ERROR GOTO
Ok
```

Regel 40 is door het onjuiste ingrijpen van RENUM nutteloos geworden (het voorkomen van een fout op regel 20 wordt niet meer afgevraagd); een foutmelding werd in dit geval niet gegeven omdat bij het hernummeren een regel met regelnummer 21 kon worden gevonden.

Over het algemeen kan men stellen dat men moet voorkomen, de systeemvariabele ERL anders te gebruiken dan in een *direkte afvraging*.

SLEUTELWOORD # ON-GOSUB

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **ON-GOSUB is samentrekking van on-go to subroutine – ga bij... naar onderprogramma (subroutine)**

schrijfwijze

```
ON<RANGWAARDE>GOSUB<REGELNUMMER>(,<REGEL-
  NUMMER>)
<REGELNUMMER>::=<TYPE 1 INTEGERE KONSTAN-
  TE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE ALGE-
  MENE SPECIFICATIES>
<RANGWAARDE>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het is noodzakelijk om eerste de behandeling van ON...GOTO goed door te nemen alsmede die van het GOSUB-kommando.

Dit kommando werkt ongeveer hetzelfde als het ON...GOTO-kommando met dit verschil dat er subroutines worden aangesproken. Deze subroutines of onderprogramma's eindigen allemaal met het RETURN-kommando. Na het RETURN-kommando wordt verdergegaan met het kommando na de ON...GOSUB tenzij in het RETURN-kommando iets anders is aangegeven.

Bijvoorbeeld:

```
NEW
10 FOR I=1 TO 4
20 ON I GOSUB 100,200,,300
30 NEXT I:FOR I=1 TO 1E20:NEXT I
100 SCREEN 2:LINE (0,0)-(100,100):RETURN
200 CIRCLE (100,100),50:RETURN
300 LINE (10,10)-(60,40),,B:RETURN
```

RUN

Door elkaar worden een lijn (eerste subroutine), een cirkel (tweede subroutine) en een rechthoek (derde subroutine) getekend. Hierna staat de computer vast in regel 30.

SLEUTELWOORD # ON-GOTO

moeilijkheidsgraad ..... **vrij eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **ON-GOTO is ga bij... naar**

schrijfwijze

```
ON<RANGWAARDE>GOTO<REGELNUMMER>{,<REGEL-
  NUMMER>}
<REGELNUMMER>::=<TYPE 1 INTEGERE KONSTAN-
  TE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE ALGE-
  MENE SPECIFICATIES>
<RANGWAARDE>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het is noodzakelijk dat eerste het GOTO-kommando is doorgenomen.

Met dit kommando is het mogelijk om afhankelijk van de waarde een numerieke uitdrukking naar diverse locaties in het programma te springen. Hiertoe worden de volgende acties bij een ON...GOTO ondernomen:

- allereerst wordt de numerieke uitdrukking uitgewerkt. Het resultaat wordt ontdaan van een eventuele decimale fractie.
- vervolgens wordt gecontroleerd of deze waarde groter is dan of gelijk is aan nul. Bij een negatieve waarde volgt een foutmelding.
- daarna wordt gecontroleerd of de berekende waarde niet groter is dan 255. Indien dit wel het geval is, volgt een foutmelding.
- de berekende waarde geeft aan, naar welk regelnummer moet worden gesprongen. Zo wordt bij de waarde 1 naar het eerste regelnummer gesprongen en bij de waarde 2 naar het tweede regelnummer enzovoorts. Indien geen regelnummer voor de betreffende waarde is gespecificeerd, wordt het programma vervolgd met het kommando na de ON...GOTO.

Bijvoorbeeld:

```
NEW
10 REM PROGRAMMAKEUZE
20 SCREEN 0:PRINT "PROGRAMMAKEUZE"
:PRINT:PRINT
30 PRINT "1...TEKEN EEN LIJN"
40 PRINT "2...TEKEN EEN CIRKEL"
50 PRINT "4...TEKEN EEN RECHTHOEK"
60 INPUT "KEUZE ";KEUZE
70 ON KEUZE GOTO 100,200,,300
80 PRINT "ALLEEN EEN 1,2 OF 4 INGEVEN":
GOTO 60
90 K$=INKEY$:IF K$="" THEN 90 ELSE 20
100 REM LIJN
110 SCREEN 2:LINE (0,0)-(200,100):
GOTO 90
200 REM CIRKEL
210 SCREEN 2:CIRCLE (100,100),50:
GOTO 90
300 REM RECHTHOEK
310 SCREEN 2:LINE (20,20)-(100,100),,B:
GOTO 90
RUN
```

Op het beeldscherm verschijnt een keuzemenu. Naar keuze kunnen een lijn, een cirkel of een rechthoek worden getekend. Nadat de tekening klaar is, behoeft slechts een toets te worden ingedrukt om weer het keuzemenu te verkrijgen.

De dubbele komma in regel 70 laat zien dat regelnummers kunnen worden overgeslagen. Pas bij de waarde 4 voor variabele KEUZE wordt naar regel 300 gesprongen.

SLEUTELWOORD ON INTERVAL GOSUB

moeilijkheidsgraad . . . . . **vrij moeilijk**
soort . . . . . **KOMMANDO**
afkomst . . . . . **ON INTERVAL GOSUB is samentrekking van on interval go to subroutine – bij pauze ga naar onderprogramma (subroutine)**

schrijfwijze

```
ON INTERVAL=<TIJDSDUUR>GOSUB[<REGELNUM-
  MER>]
<TIJDSDUUR>:::=<N>
<REGELNUMMER>:::=<TYPE 1 INTEGERE KONSTAN-
  TE>
<N>:::=<ZIE ALGEMENE SPECIFICATIES>
<TYPE 1 INTEGERE KONSTANTE>:::=<ZIE ALGE-
  MENE SPECIFICATIES>
```

betekenis

Voorafgaand aan de behandeling van dit kommando dient het ON KEY GOSUB kommando en het INTERVAL-kommando grondig te zijn doorgenomen.

Met dit kommando kunnen we de computer opdragen, de programmaloop met regelmatige tussenpozen te onderbreken. Men kan de ON INTERVAL GOSUB vergelijken met een soort ON KEY GOSUB waarbij de computer met regelmatige tussenpozen ZELF de funktietoets indrukt. Als zodanig gelden de algemene voorwaarden met betrekking tot ON KEY GOSUB ook voor ON INTERVAL GOSUB.

De tijdspanne die tussen twee onderbrekingen dient te verlopen, dient achter het gelijkteken te worden vermeld. Deze numerieke uitdrukking dient een waarde te vertegenwoordigen, groter dan –32769 en kleiner dan 65536 en stelt het aantal vijftigsten seconden voor dat de betreffende tijdspanne lang dient te zijn. Indien de waarde van de numerieke uitdrukking kleiner is dan 0, wordt er 65536 door de computer bij opgeteld waarna de tijdspanne is bepaald. De kenner herkent hierin de tweecomplementmethode.
Een eventuele decimale fractie wordt verwaarloosd.

Voorbeeld:

```
10 INTERVAL ON:LET Q=0:CLS
20 ON INTERVAL=50 GOSUB 1000
30 GOTO 30
1000 LOCATE 0,0:LET Q=Q+1:PRINT Q:BEEP
1010 RETURN
RUN
```

Elke seconde wordt variabele Q verhoogd en linksboven in het scherm afgedrukt. Bovendien klinkt een attentie-signaal.

Opmerkingen:

- Tijdens de uitvoering van de onderbrekings-subroutine wordt automatisch door de computer een INTERVAL STOP uitgevoerd. Op het moment dat de RETURN uit het onderbrekingsgedeelte wordt uitgevoerd, wordt weer een INTERVAL ON uitgevoerd tenzij in de routine zelf reeds een INTERVAL STOP of INTERVAL OFF werd uitgevoerd.

- Tijdens het uitvoeren van de ERROR-routine wordt automatisch een INTERVAL OFF uitgevoerd. Direkt bij de RESUME wordt dan weer een INTERVAL ON uitgevoerd tenzij de ERROR-routine zelf een INTERVAL STOP of INTERVAL OFF uitvoerde.

- Wanneer een programma ten einde is, wordt automatisch een INTERVAL OFF uitgevoerd.

- Een kommando wordt altijd in het geheel uitgevoerd alvorens onderbreking plaatsvindt.

- De RETURN in de onderbrekings-subroutine mag elke voor RETURN toegestane vorm aannemen.

ON INTERVAL GOSUB vindt zijn toepassing bij uitstek binnen spelprogramma's en educatieve programma's (tijdsbewaking).

SLEUTELWOORD

ON KEY GOSUB

moeilijkheidsgraad . . . . . **vrij moeilijk**
soort . . . . . **KOMMANDO**
afkomst . . . . . **ON KEY GOSUB is afkorting van on key go to subroutine – bij toets ga dan naar onderprogramma (subroutine)**

schrijfwijze

```
ON KEY GOSUB{,}[<REGELNUMMER>]{,{,}<RE-
  GELNUMMER>}
<REGELNUMMER>::=<TYPE 1 INTEGERE KON-
  STANTE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE AL-
  GEMENE SPECIFICATIES>
```

betekenis

De ON ERROR GOTO dient voorafgaand aan deze behandeling te zijn doorgenomen alsmede de GOSUB en de KEY.

Met dit kommando, dat in principe wel wat lijkt op de ON ERROR GOTO, kan worden bewerkstelligd dat een programma tijdelijk wordt onderbroken, in welk stadium dan ook, bij het intoetsen van een funktietoets. Deze onderbreking bestaat hier uit, dat een subroutine wordt aangesproken zodra de funktietoets in kwestie wordt ingetoetst. Hervatting van het programma kan aan het einde van deze subroutine met een RETURN plaatsvinden. Na deze return gaat het programma, tenzij anders aangegeven, door op de plaats waar het eerder werd onderbroken (vergelijk de RESUME bij ON ERROR GOTO).

Om ervoor te zorgen dat een programma door middel van een funktietoets kan worden onderbroken, dienen we de betreffende funktietoets eerst te activeren. Zonder deze activering kan een onderbreking via deze funktietoets nooit plaatsvinden. Het activeren van de funktietoets gaat als volgt:

```
KEY (numerieke uitdrukking) ON
```

De numerieke uitdrukking dient in waarde positief, geheel en kleiner dan 11 te zijn. Een eventuele decimale fractie wordt verwaarloosd.

Wanneer we bijvoorbeeld funktietoets 1 en 4 willen activeren, programmeren we:

```
NEW
10 KEY (1) ON:KEY(4) ON
```

De haakjes zijn verplicht; hierdoor wordt de mogelijkheid tot verwarring met een andere KEY-betekenis (zie behandeling van KEY, derde betekenis) voorkomen.

Vervolgens dienen we aan te geven, welke subroutines dienen te worden aangesproken wanneer deze funktietoetsen worden ingetoetst. Hiervoor gebruiken we het ON KEY GOSUB kommando:

```
20 ON KEY GOSUB 1000,,,1200
```

Merk op dat achter ON KEY GOSUB maximaal 10 regelnummers mogen worden vermeld. Deze regelnummers verwijzen naar de subroutines die worden aangesproken bij intoetsing van funktietoets 1, 2, 3... Vergelijk deze funktie voor het gemak met de ON...GOSUB waarbij de (verboden) variabele KEY de waarde van de funktietoets heeft.

Wanneer voor een funktietoets geen regelnummer is gespecificeerd, dan maakt het niet uit of deze funktietoets al dan niet is geactiveerd; een onderbreking zal in dat geval nooit plaatsvinden.

Belangrijk is het om te beseffen dat (ongeveer zoals bij de ON ERROR GOTO) de ON KEY GOSUB niet wordt uitgevoerd. Het enige dat de computer na uitvoering van ON KEY GOSUB 'weet' is, naar welk regelnummer moet worden gesprongen bij intoetsing van een funktietoets. *Daadwerkelijke* uitvoering vindt eventueel pas plaats op het moment van intoetsing van de funktietoets.

We maken ons voorbeeldprogramma af. Voor de duidelijkheid herhalen we de eerste twee regels ook nog even:

```
NEW
10 KEY(1) ON:KEY(4) ON
20 ON KEY GOSUB 1000,,,1200
30 LET A$="GEEN FUNKTIETOETS INGEDRUKT"
40 PRINT A$
50 GOTO 40
```

```
1000 LET A$="FUNKTIETOETS 1"
1010 RETURN
1200 LET A$="FUNKTIETOETS 4"
1210 RETURN
RUN
```

Het beeld wordt volgeschreven met de tekst GEEN FUNKTIETOETS INGEDRUKT. Wanneer funktietoets 1 wordt ingetoetst, wordt plotseling de tekst FUNKTIETOETS 1 afgedrukt. Wanneer funktietoets 4 wordt ingetoetst, verschijnt continu de tekst FUNKTIETOETS 4. De overige funktietoetsen hebben geen effect.

Ondanks dat de computer in de eeuwigdurende lus (loop) 40-50-40 verkeert, heeft het aanraken van één van de geprogrammeerde funktietoetsen tot gevolg dat het programma wordt onderbroken en de desbetreffende subroutine wordt geactiveerd. In dit geval veranderen deze subroutines de inhoud van variabele A$. De RETURN aan het einde van deze subroutines zorgt ervoor, dat het programma de 'draad' weer oppakt waar het eerder met deze subroutine werd onderbroken.

Het zal duidelijk zijn dat ondermeer in spelprogramma's dit kommando enorme mogelijkheden biedt.

Wanneer de mogelijkheid tot onderbreken van het programma dient te worden opgeheven voor een bepaalde funktietoets, dan kan dat met

```
KEY (numerieke uitdrukking) OFF
```

Gebruik van dit kommando is hetzelfde als de KEY...ON
Wanneer we bijvoorbeeld in het eerdere voorbeeld programmeren:

```
35 KEY (4) OFF
```

Dan zal funktietoets 4 na uitvoering van de programmaregel 35 niet meer het effect hebben dat er ten behoeve van subroutine 1200 het programma wordt onderbroken.

Alle funktietoetsen kunnen in één keer worden gedeactiveerd door het kommando:

```
ON KEY GOSUB ,,,,,,,,,
```
(9 komma's)

Wanneer het in een bepaald gedeelte niet wenselijk is dat het programma wordt onderbroken maar een intoetsing van de betreffende funktietoets wel dient te worden onthouden, dan kan dat worden geregeld met behulp van het kommando:

```
KEY (numerieke uitdrukking) STOP
```

Gebruik dit kommando hetzelfde als de KEY...ON.

Een eventuele intoetsing van de aangegeven funktietoets wordt door de computer 'onthouden' zonder dat het programma wordt onderbroken. Echter, op het moment dat de betreffende funktietoets weer wordt geactiveerd (KEY...ON) wordt de door de computer 'onthouden' intoetsing onmiddellijk gehonoreerd met een onderbreking ten behoeve van de aangesproken subroutine. Bijvoorbeeld:

```
NEW
10 ON KEY GOSUB ,,,,,,,,,1000
20 KEY(10) STOP
30 PRINT "DEZE REGEL WORDT";
40 PRINT " ALTIJD AANEENGESLOTEN
AFGEDRUKT"
50 KEY(10) ON:GOTO 20
1000 PRINT:RETURN
RUN
```

Continu wordt de tekst DEZE REGEL ... AFGEDRUKT steeds op één beeldschermregel afgedrukt. Door het intoetsen van funktietoets 10 kan wel een extra regelopschuiving worden geforceerd (routine 1000) maar wordt de regel nooit over twee beeldschermregels afgedrukt; een bewijs dat de KEY...STOP er voor zorgt dat nooit tussen regel 20 en 30 kan worden onderbroken. Door regel 20 te vervangen door bijvoorbeeld een REM-kommando kan (de KEY...STOP is nu weg) nu plotseling wèl een regelopschuiving tussen DEZE REGEL WORDT en ALTIJD AANEENGESLOTEN AFGEDRUKT worden geforceerd met funktietoets 10.

Opmerkingen:

– Tijdens het uitvoeren van een subroutine die door de ON KEY GOSUB werd aangeroepen, wordt voor de betreffende funktie-

toets automatisch een KEY...STOP uitgevoerd. Op het moment dat de RETURN wordt uitgevoerd, wordt automatisch een KEY...ON voor de betreffende funktietoets uitgevoerd, tenzij in de betreffende subroutine een KEY...STOP of KEY...OFF voor de betreffende subroutine werd uitgevoerd. Dit is om te voorkomen dat er merkwaardige fouten ontstaan doordat binnen de subroutine dezelfde routine een volgende maal wordt aangeroepen.

- Tijdens het uitvoeren van de ERROR-routine wordt voor elke actieve funktietoets automatisch een KEY...OFF uitgevoerd. Bij het verlaten van de ERROR-routine worden de betreffende funktietoetsen wederom geactiveerd tenzij in de ERROR-routine een KEY...STOP of KEY...OFF werd uitgevoerd.

- Wanneer het programma ten einde is, wordt automatisch voor elke funktietoets een KEY...OFF uitgevoerd.

- Een KEY...STOP heeft geen zin wanneer ervoor niet een KEY...ON voor dezelfde funktietoets werd uitgevoerd.

- Een kommando wordt altijd *volledig* afgewerkt voordat een onderbreking plaatsvindt.

- De RETURN in een onderbrekings-subroutine mag elke voor RETURN toegestane vorm hebben.

moeilijkheidsgraad . . . . . **moeilijk**
soort . . . . . **KOMMANDO**
afkomst . . . . . **ON SPRITE GOSUB is afkorting van on sprite go to subroutine – bij (botsing van) sprites ga naar onderprogramma (subroutine). Sprite is geest**

schrijfwijze

```
ON SPRITE GOSUB[<REGELNUMMER>]
<REGELNUMMER>::=<TYPE 1 INTEGERE KONSTAN-
  TE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE ALGE-
  MENE SPECIFICATIES>
```

betekenis

Voorafgaande aan deze behandeling dient het ON KEY GOSUB- en het SPRITE-kommando te zijn doorgenomen.

ON SPRITE GOSUB geeft de mogelijkheid om het programma tijdelijk te onderbreken wanneer twee sprites met elkaar in botsing komen. Achter ON SPRITE GOSUB dient een regelnummer te zijn vermeld. Dit regelnummer is het nummer van de eerste regel van de subroutine die in geval van een sprite-botsing dient te worden geactiveerd. Bij uitvoering van ON SPRITE GOSUB 'onthoudt' de computer slechts naar welk regelnummer bij een botsing dient te worden gesprongen. De daadwerkelijke sprong vindt pas bij botsing plaats. Een voorbeeld:

```
NEW
10 ON SPRITE GOSUB 100
20 SCREEN 2,1
30 SPRITE$(0)=STRING$(8,255)
40 PUT SPRITE 0,(100,100),1,0
50 SPRITE ON
60 FOR I=1 TO 200 STEP 2
70 PUT SPRITE 1,(110,Y),1,0
80 NEXT Y
90 GOTO 50
100 PUT SPRITE 0,,15
```

```
110 BEEP
120 PUT SPRITE 0,,1
130 SPRITE OFF
140 RETURN
RUN
```

Op het beeldscherm zijn twee zwarte sprites te zien, één stilstaande (40) en één vertikaal bewegende (60-80). Wanneer een botsing tussen deze twee sprites plaatsvindt, dan klinkt een attentiesignaal (110) en flitst de stilstaande sprite even wit op (100 en 120). Vervolgens wordt de mogelijkheid tot het onderbreken van het programma in verband met een botsing uitgezet (130) en wordt de for-next-lus afgemaakt. Aan het einde van de for-next-lus wordt de mogelijkheid tot het onderbreken van het programma voor een sprite-botsing weer aangezet (50).

Opmerkingen:

– Tijdens het uitvoeren van de onderbrekings-subroutine wordt automatisch door de computer een SPRITE STOP uitgevoerd. Op het moment dat de RETURN uit het onderbrekingsgedeelte wordt uitgevoerd, wordt weer een SPRITE ON uitgevoerd tenzij in de subroutine zelf een SPRITE STOP of een SPRITE OFF werd uitgevoerd.
De automatische SPRITE STOP heeft weinig nut. Omdat een botsing van sprites ook een tweede keer in de subroutine wordt opgemerkt, 'onthoudt' de computer meestal meteen een tweede botsing waardoor de RETURN uit de subroutine onmiddellijk resulteert in een nieuw aanspreken van de subroutine. Haal in het bovenstaande voorbeeld regel 130 er maar eens uit en probeer het programma nog eens. Het programma zal blijven 'hangen' in de onderbrekingssubroutine.
Om te voorkomen dat een dergelijke fout optreedt, kunnen twee acties worden ondernomen:

– of één van de botsing-veroorzakende sprites wordt in het allereerste kommando onmiddellijk op een plaats op beeldscherm gezet waar geen botsing meer wordt veroorzaakt

–of in de subroutine wordt een SPRITE OFF uitgevoerd die pas weer met een SPRITE ON teniet wordt gedaan nadat één van de botsing-veroorzakende sprites zo is verplaatst dat er geen sprake van een botsing meer is.

– Tijdens het uitvoeren van de ERROR-routine wordt automatisch een SPRITE OFF uitgevoerd. Bij de RESUME wordt dan weer een SPRITE ON uitgevoerd tenzij de ERROR-routine zelf een SPRITE STOP of een SPRITE OFF uitvoerde.

– Wanneer een programma ten einde is, wordt automatisch een SPRITE OFF uitgevoerd.

– Een kommando wordt altijd in het geheel uitgevoerd voordat een onderbreking plaatsvindt.

– De RETURN in de onderbrekingsroutine mag elke voor RETURN toegestane vorm aannemen.

– Alleen twee zich geheel of gedeeltelijk in het zichtbare schermgebied bevindende sprites kunnen een botsingsonderbreking veroorzaken. Deze sprites dienen zich dan beslist niet te bevinden achter een transparant waarop een sprite met y-coördinaat 208 is geplaatst (208 is een waarde voor de y-coördinaat waardoor alle op achterliggende transparanten geplaatste sprites onzichtbaar worden). Ook dienen zij niet geheel te zijn weggevallen in verband met het feit dat er meer dan vier sprites zich op dezelfde horizontale lijn bevinden.

SLEUTELWOORD # ON STOP GOSUB

moeilijkheidsgraad . . . . . **vrij moeilijk**
soort . . . . . **KOMMANDO**
afkomst . . . . . **ON STOP GOSUB is afkorting van on stop go to subroutine – bij stop ga naar onderprogramma (subroutine)**

schrijfwijze

```
ON STOP GOSUB[<REGELNUMMER>]
<REGELNUMMER>::=<TYPE 1 INTEGERE VARI-
  ABELE>
<TYPE 1 INTEGERE VARIABELE>::=<ZIE AL-
  GEMENE SPECIFICATIES>
```

betekenis

Voorafgaand aan de behandeling van ON STOP GOSUB dient de werking van ON KEY GOSUB en STOP (tweede gedeelte) grondig te zijn doorgenomen.

De ON STOP GOSUB heeft hoegenaamd dezelfde funktie als de ON KEY GOSUB. De verschillen zijn:

- er is slechts één funktietoets die wordt afgevangen, namelijk de CONROL-STOP (CTRL en STOP tegelijk ingedrukt)
- achter ON STOP GOSUB behoeft dus ook maar één regelnummer te worden vermeld
- er behoeft geen funktietoetsnummer te worden gegeven. Dus:

activeren van de STOP-toets	STOP ON	(vgl. KEY...ON)
deactiveren van de STOP-toets	STOP OFF	(vgl. KEY...OFF)
ophouden van de gevraagde onderbreking	STOP STOP	(vgl. KEY...STOP)

De ON STOP GOSUB biedt de mogelijkheid om een programma ononderbreekbaar te maken; de normale funktie van CONTROL-STOP, namelijk het onderbreken van een programma, kan worden gewijzigd. Ga na dat het volgende voorbeeld ononderbreekbaar is:

```
NEW
10 ON STOP GOSUB 1000:STOP ON
20 PRINT "ONONDERBREEKBAAR-";
30 GOTO 20
1000 RETURN
RUN
```

Het beeldscherm wordt met de tekst ONONDERBREEKBAAR– volgeplaatst; het intoetsen van CONTROL-BREAK helpt niet. Er is maar één manier om dit programma te onderbreken: de computer eerst uit en dan weer aan zetten.

Opmerkingen:

- Tijdens het uitvoeren van de door ON STOP GOSUB opgeroepen subroutine wordt door de computer automatisch een STOP STOP uitgevoerd. Op het moment dat de RETURN wordt uitgevoerd, wordt dan weer automatisch een STOP ON uitgevoerd tenzij in de betreffende subroutine een STOP STOP of een STOP OFF werd uitgevoerd.

- Tijdens het uitvoeren van de ERROR-routine wordt de STOP-toets gedeactiveerd. Activering vindt bij uitvoering van de RESUME plaats tenzij de ERROR-routine een STOP STOP of een STOP OFF bevatte die werd uitgevoerd.

- Wanneer het programma ten einde is, wordt automatisch een STOP OFF uitgevoerd.

- Een kommando wordt altijd volledig afgewerkt voordat een onderbreking plaatsvindt.

- De RETURN in een onderbrekings-subroutine mag elke voor RETURN toegestane vorm hebben.

SLEUTELWOORD # ON STRIG GOSUB

moeilijkheidsgraad . . . . . **vrij moeilijk**
soort . . . . . **KOMMANDO**
afkomst . . . . . **STRIG is samentrekking van shot en trigger (schot en trekker) vrij vertaald: ON STRIG GOTO is bij overhalen van trekker ga naar...**

schrijfwijze

```
ON STRIG GOSUB{,}[<REGELNUMMER>]{,{,}<RE-
  GELNUMMER>}
<REGELNUMMER>::=<TYPE 1 INTEGERE KONSTAN-
  TE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE ALGE-
  MENE SPECIFICATIES>
```

betekenis

Voorafgaand aan de behandeling van dit kommando dienen ON KEY GOSUB en STRIG grondig te zijn doorgenomen.

ON STRIG GOSUB heeft dezelfde funktie als ON KEY GOSUB maar dan met betrekking tot de spatiebalk en de vuurknoppen van de joystick als funktietoetsen. Voorbeeld:

```
NEW
10 STRIG(0) ON
20 ON STRIG GOSUB 1000
30 GOTO 30
1000 PRINT "PANG!!!":RETURN
RUN
```

Steeds wanneer de spatiebalk wordt ingedrukt, wordt de tekst PANG!!! afgedrukt op het beeldscherm. Blijkbaar wordt de oneindige loop op regel 30 dan even onderbroken ten behoeve van de subroutine op regel 1000.

Nog een voorbeeld:

```
NEW
10 FOR I=1 TO 4:STRIG(I) ON:NEXT I
20 ON STRIG GOSUB ,1000,2000,3000,4000
30 GOTO 30
1000 PRINT "JOY-STICK 1 VUURKNOP 1":
RETURN
2000 PRINT "JOY-STICK 2 VUURKNOP 1":
RETURN
3000 PRINT "JOY-STICK 1 VUURKNOP 2":
RETURN
4000 PRINT "JOY-STICK 2 VUURKNOP 2":
RETURN
RUN
```

Dit voorbeeld is alleen zinvol indien er minimaal één joy-stick is aangesloten. Steeds wanneer een vuurknop wordt ingedrukt, wordt dit door de betreffende subroutine gemeld. De spatiebalk is ditmaal niet geactiveerd.

Opmerkingen:

- Tijdens de uitvoering van de door ON STRIG GOSUB opgeroepen subroutine wordt door de computer automatisch een STRIG... STOP uitgevoerd voor de betreffende vuurknop. Op het moment dat de RETURN wordt uitgevoerd, wordt weer automatisch een STRIG...ON uitgevoerd tenzij de betreffende subroutine de vuurknop deactiveerde met STRIG...STOP of STRIG...OFF.

- Tijdens het uitvoeren van de ERROR-routine worden alle vuurknoppen gedeactiveerd. Activering vindt pas bij de RESUME weer plaats tenzij een vuurknop in de ERROR-routine met een STRIG...OFF of STRIG...STOP werd gedeactiveerd.

- Wanneer het programma ten einde is, wordt automatisch een STRIG...OFF voor elke vuurknop uitgevoerd.

- Een kommando wordt altijd in zijn geheel afgerond voordat onderbreking plaatsvindt.

- De RETURN in een onderbrekings-subroutine mag elke voor RETURN toegestane vorm hebben.

SLEUTELWOORD # OPEN

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **OPEN is openstellen/openmaken**

schrijfwijze

```
OPEN<BESTANDSNAAM>[FOR INPUT ] AS[#]
                   OUTPUT
  <KANAAL>
<BESTANDSNAAM>:::=<A>
<KANAAL>:::=<N>
<A>:::=<ZIE ALGEMENE SPECIFICATIES>
<N>:::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het is noodzakelijk dat eerst MAXFILES wordt bestudeerd.

Om een bestand, een verzameling van bij elkaar behorende gegevens, te kunnen gebruiken of inrichten, dient het eerst te worden geopend. Dit openen geschiedt met het OPEN-kommando en dient ondermeer om het bestand een KANAALNUMMER toe te kennen waardoor het gemakkelijk is te benaderen.

Als eerste gegeven na het OPEN-sleutelwoord dient de naam van het betreffende bestand te worden opgegeven. Deze naam bestaat uit twee gedeelten, van elkaar gescheiden door een puntkomma.
Het eerste gedeelte van de bestandsnaam geeft aan, op welk randapparaat het bestand staat of moet komen te staan. Het tweede gedeelte geeft aan hoe het betreffende bestand moet worden genoemd.

Als tweede gegeven mag worden gespecificeerd op welke wijze het betreffende bestand moet worden gebruikt. FOR INPUT wil zeggen dat gegevens naar de computer toe dienen te worden ingevoerd vanuit het bestand en FOR OUTPUT wil zeggen dat gegevens vanuit de computer dienen te worden uitgevoerd naar het betreffende bestand.
FOR INPUT en FOR OUTPUT behoeven niet te worden opgenomen wanneer het zondermeer duidelijk is op welke wijze het bestand dient te worden benaderd.

Als derde gegeven dient het kanaalnummer te worden aangegeven waaronder het eenmaal geopende bestand verder kan worden benaderd. Dit kanaalnummer is minimaal gelijk aan 1 en maximaal gelijk aan de waarde die de systeemvariabele MAXFILES (standaard 1) is gegeven. Een eventuele decimale fractie wordt verwaarloosd.

Allereerst volgt hieronder een tabel waarin diverse voorwaarden zijn opgenomen in verband met het OPEN-kommando:

randapparaat	betekenis	benadering	verplicht?	bestandsnaam verplicht?	kommentaar
CAS:	cassette-recorder	for input for output	ja	nee*	een cassetterecorder moet zijn aangesloten
GRP:	grafisch scherm	for output	nee	nee, zinloos	een grafisch scherm moet zijn geactiveerd
CRT:	alfanumeriek scherm	for output	nee	nee, zinloos	een alfanumeriek scherm moet zijn geactiveerd
LPT:	printer	for output	nee	nee, zinloos	een printer moet zijn aangesloten
(niets)	cassette-recorder	zie CAS:	zie CAS:	ja	zie CAS:

* indien geen bestandsnaam wordt opgegeven:
– bij for output wordt een bestand zonder naam aangemaakt
– bij for input wordt het eerste bestand dat op de band wordt tegengekomen, geopend

Per randapparaat worden de diverse mogelijke vormen van het OPEN-kommando nu nader behandeld:

Cassetterecorder

Om met MSX-basic optimaal van de cassetterecorder gebruik te maken is het noodzakelijk dat de cassetterecorder met drie kabeltjes is verbonden aan de computer, te weten:
– de microfoonkabel. Via deze kabel worden gegevens vanuit de computer naar de cassetterecorder gebracht
– de oortelefoonkabel. Via deze kabel worden gegevens vanuit de cassetterecorder naar de computer gebracht
– de afstandsbedieningskabel. Via deze kabel wordt de cassetterecorder door de computer aan- en uit gezet.

Enkele opmerkingen in verband met de cassetterecorder in koppeling aan de computer:

- wanneer gegevens naar de cassetterecorder worden gestuurd vanuit de computer, dan dient de cassetterecorder op opnemen te staan;
- wanneer gegevens vanuit de cassetterecorder naar de computer toe worden gestuurd, dan dient de cassetterecorder op afspelen te staan;
- indien gegevens vanuit de computer naar een randapparaat worden gestuurd dan spreekt men van het SCHRIJVEN van gegevens;
- indien gegevens vanuit een randapparaat naar de computer toe worden gestuurd dan spreekt men van het LEZEN van gegevens;
- bij het afspelen van de cassetterecorder dient het volume op ongeveer 80% te staan
- het MSX-basic schrijft geen controlemaatregelen voor bij het teruglezen van gegevens. Slecht materiaal kan ervoor zorg dragen dat gegevens verminkt worden ingelezen zonder dat daar door de computer melding van wordt gemaakt. Zorg dat u altijd cassettebanden van een goed merk gebruikt. Belangrijk is het om te weten of de cassettebanden geschikt zijn voor de betreffende recorder. Gebruik bij een eenvoudige recorder, tenzij anders aangegeven, altijd een ferro-kwaliteit cassettebandje (standaardkwaliteit). Het gebruik van chroomdioxidebanden en dergelijke op een niet daarvoor geschikt apparaat geeft meestal slechts negatieve resultaten. Indien u dat kunt, maak dan zeer regelmatig de opnamekop en de weergavekop van uw cassetterecorder met een wattenstaafje en met zuivere alkohol (geen spiritus) schoon
- indien u een cassetteband wilt heen- of terugspoelen dan kan dat meestal niet doordat de computer de cassetterecorder heeft uitgeschakeld. Trek in dat geval de afstandsbediening aan de kant van de cassetterecorder even los of gebruik het MOTOR-kommando (zie aldaar).

Het schrijven van gegevens naar de cassetterecorder

Met het SAVE, CSAVE en BSAVE-kommando kunnen (programma-) gegevens naar de cassetterecorder worden geschreven. Zie de behandeling van deze kommando's. Indien men andere dan programma- of machinegegevens naar de cassetterecorder wilt schrijven, dan dient eerst een bestand te worden toegewezen en wel als volgt:

```
NEW
10 OPEN "CAS:TEST" FOR OUTPUT AS 1
```

Bij uitvoering van dit kommando wordt op de cassetterecorder (CAS:) een bestand toegewezen, genaamd TEST. CAS: mogen we in dit geval eventueel weglaten terwijl we voor TEST elke andere naam mogen invullen die niet groter is dan 6 letters. Een eventueel teveel aan letters wordt gewoon genegeerd.

In de volgende programmaregels kunnen vervolgens diverse gegevens naar dit bestand toe worden geschreven. Hiertoe dient men gebruik te maken van het PRINT-kommando. Een eenvoudig voorbeeld wordt hier gegeven; zie voor het gebruik van het PRINT-kommando verder de behandeling onder PRINT.

```
20 LINE INPUT "GEGEVENS:";A$
30 IF A$="" THEN GOTO 100
40 PRINT #1,A$
50 GOTO 20
```

Met dit programmagedeelte kunnen de op regel 20 ingegeven gegevens naar cassette worden geschreven. Het werkelijke schrijven gebeurt op regel 40. Merk op dat op regel 40 alleen maar het kanaalnummer behoeft te worden genoemd; na het OPEN-kommando weet de computer alleen door dit kanaalnummer naar welk randapparaat de gegevens dienen te worden geschreven.
Op regel 30 wordt afgevraagd of er misschien niets werd ingegeven. In dat geval dient het programma te worden verlaten. Voordat het programma wordt beëindigd, dient eerst het betreffende bestand dat zojuist werd geopend, weer te worden afgesloten. Dit gebeurt via een CLOSE-kommando. Indien slechts een CLOSE wordt gegeven, dan worden alle kanalen achter elkaar afgesloten. Indien slechts één of enkele kanalen dienen te worden afgesloten, dan kan dat door achter het CLOSE-kommando de betreffende kanalen op te nemen, van elkaar gescheiden door een komma. In ons voorbeeld programmeren we:

```
100 CLOSE
110 STOP
RUN                                    (enige tijd verstrijkt)
GEGEVENS:DEZE GEGEVENS WORDEN STRAKS
```

```
GEGEVENS:WEER DOOR DE COMPUTER VAN
GEGEVENS:CASSETTEBAND TERRUGGELEZEN
GEGEVENS:TEST 0123456789
GEGEVENS:                    (alleen de RETURN-toets wordt ingegeven)
Break in 110                 (enige tijd verstrijkt)
Ok
```

Vergeet niet om, voordat het programma wordt gestart, de cassetteband te plaatsen, op de juiste positie te spelen en om de cassetterecorder op opnemen te zetten.

Nadat het programma is uitgevoerd (en er mogen natuurlijk andere gegevens worden ingevoerd), is een bestand op de cassetteband aangemaakt onder de naam TEST en met de ingegeven gegevens daarin geschreven.

Indien achter CAS: geen bestandsnaam werd opgegeven, dan werd een bestand zonder naam op de cassetteband aangemaakt.

Het lezen van gegevens van de cassetterecorder

Met het LOAD, CLOAD en BLOAD-kommando kunnen (programma-) gegevens van de cassetteband worden gelezen. Zie hiervoor de betreffende behandelingen. Indien men andere dan programma- of machinegegevens vanuit de cassetterecorder in wil lezen, dan dienen deze gegevens zoals in het vorige gedeelte behandeld als bestand op cassetteband te zijn gezet. Eén uitzondering vormt een programma dat door SAVE op cassette is gezet. Dit bestand kan met OPEN worden geopend geopend en vervolgens worden ingelezen.

In het volgende voorbeeld gaan we de gegevens die zojuist in bestand TEST zijn opgeslagen, weer teruglezen. Hiertoe dienen we het bestand eerst te openen:

```
NEW
10 OPEN "CAS:TEST" FOR INPUT AS 1
```

Bij uitvoering van dit kommando mag CAS: eventueel worden weggelaten. Indien CAS: niet wordt weggelaten, mag de bestandsnaam (TEST) worden weggelaten; in dat geval wordt het eerste op de cassetteband voorkomende bestand in behandeling genomen.

Vervolgens programmeren we:

```
20 LINE INPUT #1,A$
30 PRINT A$
40 IF EOF(1) THEN CLOSE:STOP
50 GOTO 20
RUN                                    (enige tijd verstrijkt)
DEZE GEGEVENS WORDEN STRAKS
WEER DOOR DE COMPUTER VAN
CASSETTEBAND TERRUGGELEZEN
TEST 0123456789
Break in 40
Ok
```

Vergeet niet om vooraf aan dit programma eerst de cassetteband naar het begin van het bestand terug te spoelen (gebruik de bandteller) en de cassetterecorder op afspelen te zetten. Zet het volume op ongeveer 80%.

We zien één nieuw sleutelwoord in regel 40, namelijk het sleutelwoord EOF. De funktie EOF dient te worden gevolgd door een numerieke uitdrukking die een kanaalnummer bevat. Een decimale fractie wordt eventueel verwaarloosd. De funktie geeft als resultaat een 0 indien het bestand nog niet ten einde is en een −1 indien het laatste gegeven van het op dat kanaalnummer geopende bestand zojuist werd ingelezen.

Om bestanden optimaal te kunnen gebruiken op cassette, is het na deze behandeling noodzakelijk om de gevorderde behandelingen van PRINT, INPUT en LINE INPUT door te nemen tot zover zij betrekking hebben op het lezen en schrijven van gegevens van of naar randapparatuur.

Grafisch scherm

Een geactiveerd grafisch scherm wordt door MSX-basic ook als randapparaat beschouwd. Een bestandsnaam mag worden gespecificeerd bij het OPEN-kommando maar is zinloos.
De naar het grafische scherm geschreven gegevens worden vanaf het laatst getekende punt op het beeldscherm geplaatst. Het laatst getekende punt vormt de linker bovenhoek van de 8 bij 8 puntenmatrix waarin het eerste karakter kan worden geplaatst. Een voorbeeld:

```
NEW
10 SCREEN 2                    (probeer ook eens SCREEN 3)
20 OPEN "GRP:" AS 1
30 PSET (0,0)
40 PRINT #1,"DIT IS EEN TEST"
50 CIRCLE (100,100),50
60 GOTO 60
RUN
```

De tekst DIT IS EEN TEST wordt op het grafische beeldscherm geplaatst tesamen met de tekening van een cirkel.

Alfanumeriek scherm

Een geactiveerd alfanumeriek scherm wordt door MSX-basic eveneens als apart randapparaat beschouwd. Het printen op een alfanumeriek scherm via een kanaal heeft precies hetzelfde effect als het normale printen op een alfanumeriek beeldscherm. Bijvoorbeeld:

```
NEW
10 OPEN "CRT:" AS 1
20 LOCATE 0,0
30 PRINT #1,"DIT IS EEN TEST"
RUN
```

de tekst DIT IS EEN TEKST wordt linksboven in beeld geplaatst. Regel 10 kan vervallen wanneer regel 30 als volgt wordt veranderd:

```
30 PRINT "DIT IS EEN TEST"
```

Toch heeft het openen van het alfanumeriek beeldscherm zin; hierop komen we later terug.

Printer

Ook de printer is een randapparaat. Met het LPRINT-kommando kunnen we gegevens naar de printer sturen; zie de betreffende behandeling.
Door de printer te openen op een kanaal, kunnen we hetzelfde effect bereiken. Bijvoorbeeld:

```
NEW
10 OPEN "LPT:" AS 1
20 PRINT #1,"DIT IS EEN TEST"
RUN
```

Indien een printer is aangesloten, verschijnt de tekst DIT IS EEN TEST op de printer. Indien regel 10 wordt verwijderd, kan regel 20 worden vervangen door:

```
20 LPRINT "DIT IS EEN TEST"
```

Toch heeft het apart openen van de printer in sommige gevallen nut. Bijvoorbeeld:

```
NEW
10 MAXFILES=2
20 OPEN "CAS:TEST" FOR INPUT AS 1
30 INPUT "PRINTER OF BEELDSCHERM
(P/B)";A$
40 IF A$="P" THEN OPEN "LPT:" AS 2:
GOTO 70
50 IF A$="B" THEN OPEN "CRT:" AS 2:
GOTO 70
60 PRINT "FOUTE INGAVE":GOTO 30
70 LINE INPUT #1,A$:PRINT #2,A$
80 IF EOF(1) THEN CLOSE:STOP ELSE
GOTO 70
RUN                                    (enige tijd verstrijkt)
PRINTER OF BEELDSCHERM (P/B)? B
DEZE GEGEVENS WORDEN STRAKS
WEER DOOR DE COMPUTER VAN
CASSETTEBAND TERRUGGELEZEN
TEST 0123456789
Break in 80
Ok
```

Als invoerbestand werd het bestand TEST zoals dat in het voorbeeld bij het schrijven naar cassetterecorder werd gegeven, gebruikt. Vergeet niet om voorafgaand aan dit programma de cassetterecorder op af-

spelen te zetten nadat de cassette tot op het juiste punt is teruggespoeld.

Op regel 10 wordt bepaald dat er twee verschillende bestanden naast elkaar worden geopend. Op regel 20 wordt dan in ieder geval het bestand TEST van cassetteband geopend. Op regel 30 wordt de keuze tussen het afdrukken van de gegevens op beeldscherm of printer mogelijk gemaakt. Afhankelijk van de ingave op deze regel wordt de printer of het alfanumerieke beeldscherm geopend. Vervolgens worden over kanaal 2 de vanuit de cassetterecorder ingelezen gegevens afgedrukt, hetzij op beeldscherm, hetzij op de printer.

Merk op dat een dergelijke keuze veel moeilijker is te programmeren indien we het PRINT-kommando zonder kanaalnummer en het LPRINT-kommando naast elkaar zouden gebruiken. In dit geval (en in vele andere denkbare gevallen) is het zinvol om een alfanumeriek scherm of een printer apart te openen.

moeilijkheidsgraad . . **vrij moeilijk, kennis van de funkties van de poorten van het computersysteem is vereist**
soort . . . . . **KOMMANDO**
afkomst . . . . . **OUT is uit**

schrijfwijze

```
OUT<POORT>,<TE VERZENDEN WAARDE>
<POORT>::=<N>
<TE VERZENDEN WAARDE>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Een byte-waarde kan op een poortbuffer worden geplaatst.
Het poortnummer mag niet kleiner zijn dan –32768 en niet groter dan 65535. Een eventuele decimale fractie wordt verwaarloosd. Indien het poortnummer negatief is, wordt 65536 bij dit nummer opgeteld alvorens deze poort wordt aangesproken.
De via de poort te verzenden waarde dient in één byte te passen en mag als zodanig niet kleiner zijn dan 0 en niet groter dan 255, terwijl een decimale fractie wordt verwaarloosd.

Ondeskundig gebruik van dit kommando kan leiden tot vreemde effecten of het 'ophangen' van het systeem. In dat geval is er maar één remedie: computer uitzetten en daarna weer aanzetten.

Bijvoorbeeld:

```
OUT 168,255
```

De computer staat vast; uitzetten en weer aanschakelen helpt.

Voor een goed gebruik van OUT is een gedetailleerde kennis van de hardware-opbouw van een MSX-computer noodzakelijk. Daarbij dienen de diverse componenten van een MSX-computer ook softwaretechnisch te worden beheerst. Hiertoe dient specialistische literatuur te worden geraadpleegd; in dit handboek wordt hierop niet verder ingegaan.

moeilijkheidsgraad ..... **vrij moeilijk**
soort ..... **FUNKTIE**
afkomst ..... **PAD is blocknote**

schrijfwijze

```
PAD(<PAD NUMMER EN SPECIFICATIE VAN DE
  GEWENSTE INFORMATIE>)
<PAD NUMMER EN SPECIFICATIE VAN DE GE-
  WENSTE INFORMATIE>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Als wat minder bekend randapparaat kan aan de MSX-computer een aanraakpaneel (pad) worden aangesloten. Met dit aanraakpaneel kunnen tekeningen worden gedigitaliseerd (in cijferkodering omgezet) voor de computer. In wat professionelere vorm noemt men zo'n apparaat meestal een digitizer.

Een aanraakpaneel bestaat uit een paneel dat met een speciale stift kan worden aangeraakt. Daarbij is, meestal op de tekenstift aangebracht, een schakelaar (drukknop) aanwezig. Door de stift op bepaalde posities neer te zetten en eventueel de schakelaar in te drukken, kan een tekening worden overgetrokken, worden gekodeerd en naar de computer worden geseind.

In de funktie PAD kunnen we de toestand van het aanraakpaneel afvragen en wel als volgt:

`PAD(0) of PAD(4)` geeft resultaat 0 indien de tekenstift niet op het paneel wordt gedrukt en geeft resultaat −1 wanneer de stift wél op het tekenpaneel wordt gedrukt

`PAD(1) of PAD(5)` geeft de x-coördinaat (de horizontale puntpositie) van de plaats waar het paneel door de stift wordt aangeraakt

`PAD(2) of PAD(6)` geeft de y-coördinaat (de vertikale punt-

positie) van de plaats waar het paneel wordt aangeraakt

`PAD(3) of PAD(7)` geeft resultaat 0 indien de schakelaar niet wordt ingedrukt en geeft resultaat −1 indien de schakelaar wél wordt ingedrukt.

Het volgende voorbeeld is alleen zinvol indien een aanraakpaneel aan de computer is aangesloten. De penbeweging wordt op het scherm gevolgd terwijl een lijn wordt getrokken. Indien de schakelaar wordt ingedrukt, wordt een cirkel op die plaats getekend.

```
NEW
10 SCREEN 2:CLS
20 IF PAD(0) THEN LET X=PAD(1):Y=PAD(2)
ELSE LET M%=0:GOTO 20
30 IF M% THEN LINE -(X,Y) ELSE PSET
(X,Y):M%=-1
40 IF PAD(3) THEN CIRCLE (X,Y),20
50 GOTO 20
RUN
```

In het gegeven voorbeeld dient het aanraakpaneel aangesloten te zijn aan ingang A. Indien een aansluiting aan ingang B is gerealiseerd, dienen in plaats van PAD(0)...PAD(3) de funkties PAD(4)...PAD(7) te worden gebruikt.

SLEUTELWOORD # PAINT

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **PAINT is schilderen**

schrijfwijze

```
PAINT<LOCATIE>[,[<SCHILDERKLEUR>][,<RAND-
  KLEUR>]]\PAINT[<...>,]
<LOCATIE>::=[STEP](<HORIZONTAAL>,<VERTI-
  KAAL>)
<HORIZONTAAL>::=<N>
<VERTIKAAL>::=<N>
<SCHILDERKLEUR>::=<N>
<RANDKLEUR>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met dit kommando kunnen binnen een tekening op scherm gehele vlakken worden ingekleurd. Door slechts een punt aan te geven, eventueel een schilderkleur te kiezen en eventueel een randkleur te kiezen, wordt het betreffende oppervlak ingekleurd.

Voor goed begrip van dit kommando is het noodzakelijk om de behandeling van PSET grondig door te nemen.

De wijze waarop PAINT werkt, kunnen we het beste vergelijken met de wijze waarop we zelf een tekening inkleuren.
Wanneer we een oppervlak gaan inkleuren is het noodzakelijk om eerst het juiste kleurpotlood te kiezen, om de juiste kleur te bepalen. Vervolgens dienen we te bepalen waar we het kleurpotlood op papier gaan neerzetten. Als laatste moeten we beslissen tot aan welke lijnen we gaan inkleuren.
Met het PAINT-kommando geven we het punt waar het kleurpotlood moet worden neergezet aan met de betreffende locatie. Met de schilderkleurkodering bepalen we de kleur. Met de randkleurkodering bepalen we de kleur van de lijnen waarbinnen de computer met kleuren dient te blijven.

Indien de schilderkleur wordt weggelaten, wordt aangenomen dat de schilderkleur gelijk is aan de op dat moment actieve voorgrondkleur. Indien de randkleur wordt weggelaten, wordt aangenomen dat ook deze kleur gelijk is aan de op dat moment actieve voorgrondkleur.

Indien de hoge resolutie is geactiveerd (SCREEN 2) dan is het overbodig om een randkleur te specificeren; de schilderkleur wordt altijd als randkleur genomen.

Enige voorbeelden:

```
NEW
10 SCREEN 2
20 CIRCLE (111,111),75
30 CIRCLE (200,200),100
40 PAINT (160,160)
50 GOTO 50
RUN
```

Twee cirkels worden getekend. Het snij-oppervlak wordt ingekleurd.

```
10 SCREEN 3
40 PAINT (160,160),12
RUN
```

Alleen regels 10 en 40 werden veranderd. In lage resolutie worden de twee cirkels nu ingekleurd. Het snij-oppervlak wordt donkergroen gekleurd.

```
20 CIRCLE (111,111),75,12
30 CIRCLE (200,200),100,12
40 PAINT (160,160),4,12
RUN
```

Programmaregels 20, 30 en 40 werden vervangen. Twee groene cirkels worden getekend. Het snij-oppervlak wordt donkerblauw ingekleurd.

```
40 PAINT (160,160),,12
RUN
```

Alleen regel 40 werd vervangen. Het snij-oppervlak wordt met de op dat moment actieve voorgrondkleur ingekleurd.

Merk op dat het met PAINT voldoende is om ergens binnen het in te kleuren vlak een punt te prikken.

```
20 CIRCLE (111,111),75,0
30 CIRCLE (200,200),100,0
40 PAINT (160,160),12,0
RUN
```

De regels 20, 30 en 40 werden vervangen. De cirkels werden in de transparantkleur (onzichtbaar) getekend. Daarna werd het snijvlak ingekleurd. Alleen het ingekleurde snijvlak is zichtbaar. Dit laatste voorbeeld is uitstekend om te zien hoe de transparantkleur (kleurkode 0) funktioneel kan worden gebruikt.

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **PDL is afkorting van paddle – schoep**

schrijfwijze

```
PDL(<PADDLE NUMMER>)
<PADDLE NUMMER>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met deze funktie kan de waarde van een peddel worden afgevraagd. Een peddel bestaat uit een regelknop en meestal een vuurknop en kan aan één van de uitgangen van de computer worden aangesloten.

De numerieke uitdrukking tussen haakjes dient een waarde te hebben, gelijk aan 1, 2, 3, 4..., 12. Indien de tussen haakjes vermelde waarde gebroken is, wordt deze herleid naar de grootste gehele waarde, kleiner dan deze gebroken waarde.

De waarde tussen haakjes heeft de volgende betekenis:

PDL (oneven waarde) de eerste peddel wordt beschouwd

PDL (even waarde) de tweede peddel wordt beschouwd

De funktie heeft een waarde tot gevolg die minimaal gelijk is aan 0, maximaal gelijk is aan 255 en geheel is. De waarde 0 correspondeert met een geheel dichtgedraaide peddel, de waarde 255 correspondeert met een geheel opengedraaide peddel. Elke tussenliggende waarde correspondeert met een bepaalde stand van de regelknop.

Voorbeeld: (alleen zinvol met een aangesloten peddel)

```
NEW
10 LET A=PDL(1)/255*23:IF A=AA THEN 10
20 CLS:LOCATE ,A:PRINT "====";:LET AA=A
GOTO 10
RUN
```

moeilijkheidsgraad . . zeer moeilijk, kennis van de opbouw van het computergeheugen is vereist
soort . . . . . N-FUNKTIE
afkomst . . . . . PEEK is gluren, kijken

schrijfwijze

```
PEEK(<GEHEUGENADRES>)
<GEHEUGENADRES>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met deze funktie kan de direkte inhoud van het computergeheugen worden opgevraagd. Achter het sleutelwoord PEEK dient hiertoe tussen haakjes het betreffende geheugenadres te worden opgenomen. Als resultaat wordt dan de inhoud van het betreffende byte numeriek weergegeven. Bijvoorbeeld:

```
NEW
10 FOR I=1 TO 100
20 PRINT PEEK(I)
30 NEXT I
RUN
```

Van de eerste honderd adressen wordt de inhoud numeriek op beeldscherm afgedrukt.

Het met PEEK toegankelijke geheugen wordt gevormd door:

- 0 ... 32767 — 32 kilobyte ROM waarin het MSX-basic is gekodeerd
- 32768 ... 65535 — 32 kilobyte RAM, het voor MSX-basic beschikbare geheugen.

Het geheugenadres dat achter PEEK dient te worden gespecificeerd, mag niet kleiner zijn dan –32768 en niet groter dan 65535. Een eventuele decimale fractie wordt verwaarloosd. Indien het geheugenadres

negatief is, wordt de waarde 65536 bij dit adres opgeteld voordat het geheugen wordt geadresseerd.

Het volgende programma tracht het ROM-geheugen karaktergewijs op het beeldscherm af te drukken. De geduldige toeschouwer zal tussen een wirwar van tekens regelmatig bekende delen herkennen en hele karaktertabellen zien voorbijkomen:

```
NEW
10 FOR I=0 TO 32767
20 PRINT CHR$(PEEK(I));
30 NEXT I
RUN
```

Voor een goed gebruik van PEEK is gedetailleerde kennis van de opbouw van het geheugen van een MSX-computer noodzakelijk. Hiertoe dient specialistische literatuur te worden geraadpleegd; in dit handboek wordt hierop verder niet ingegaan.

SLEUTELWOORD # PLAY

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **PLAY is spelen, muziek maken**

schrijfwijze

```
PLAY(<STEMNUMMER>)
<STEMNUMMER>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met de funktie PLAY kan worden afgevraagd of de toongenerator al klaar is met spelen. De toongenerator is een apart computertje binnen de MSX-computer en kan als zodanig nog bezig zijn met het afwerken van een stukje muziek terwijl de Ok-melding al is gegeven of terwijl het programma reeds verder gaat.
Tussen de haakjes dient het nummer van de stem in een numerieke uitdrukking te worden gegeven. Een eventuele decimale fractie wordt verwaarloosd. Het stemnummer mag gelijk zijn aan 0, 1, 2 of 3. Indien een 1, 2 of 3 wordt gebruikt, dan wordt de corresponderende stem afgevraagd. Indien stemnummer 0 wordt opgegeven, dan wordt *elke* stem afgevraagd.

Play geeft de waarde 0 indien de betreffende stem niet actief is en de waarde –1 indien de betreffende stem wel actief is. Bijvoorbeeld:

```
NEW
10 PLAY "CDEFGFEDCDEFGFEDC"
20 PRINT "KLAAR"
RUN
KLAAR
Ok                                  (de muziek speelt echter nog)

15 IF PLAY(0) THEN 15               (regel wordt bijgevoegd)
RUN
KLAAR
Ok                          (de tekst KLAAR wordt pas gegeven
                                nadat de muziek is uitgespeeld)
```

moeilijkheidsgraad . . **vrij moeilijk, een elementaire kennis van de muziektheorie is vereist**
soort . . . . . **KOMMANDO**
afkomst . . . . . **PLAY is spelen, muziek maken**

schrijfwijze

```
PLAY[<EERSTE STEM>][,[<TWEEDE STEM>][,
   <DERDE STEM>]]
<EERSTE STEM>::=<A>
<TWEEDE STEM>::=<A>
<DERDE STEM>::=<A>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
<...>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met het PLAY-kommando kunnen we op eenvoudige wijze een maximaal driestemmig stuk muziek vanaf een muziekmanuscript programmeren in de computer. Afhankelijk van het aantal stemmen dienen hiertoe achter het PLAY-kommando één, twee of drie alfanumerieke uitdrukkingen te zijn opgenomen.
Een geluids*effect*, bijvoorbeeld als begeleiding bij spelletjes, kan beter met het SOUND-kommando worden geprogrammeerd; zie aldaar.
De alfanumerieke uitdrukkingen dienen te zijn opgebouwd zoals de MML regels dat voorschrijven. MML (Music Macro Language) is te beschouwen als een aparte taal binnen MSX-basic, ontworpen door Digital Research, één van de grotere wereld-software producenten.

MML gaat uit van de volgende toonhoogte-symbolen:

C,D,E,F,G,A,B — deze letters stellen de namen van de noten voor van het te programmeren stuk muziek.

+, – — met een plus- of een min-teken achter de naam van de noot geeft men aan of deze halve toon moet worden verhoogd of verlaagd. Een plusteken komt dus overeen met een kruis en een minteken komt overeen met een mol. Dubbelmollen en dubbelkruizen zijn niet toegestaan.

Voorbeeld: de chromatische toonladder:

```
PLAY "CC+DD+EFF+GG+AA+B"
Ok
```

Achtereenvolgens worden de C, de Cis, de D, de Dis, de E, de F, de Fis, de G, de Gis, de A, de Ais en de B gespeeld door de computer.

Technisch heeft de volgende ingave exact hetzelfde effect:

```
PLAY "CD-DE-EFG-GA-AB-B"
Ok
```

Exact dezelfde klanken, voorstellende de C, de Des, de D, de Es, de E, de F, de Ges, de G, de As, de A, de Bes en de B, worden gespeeld.

Een zéér eenvoudig stukje driestemmige muziek:

```
NEW
10 PLAY "CDEFGABC","EFGABCDE","GABCDEFG"
RUN
```

Een zevental verschillende drieklanken wordt door de computer geproduceerd.

Door een getal (een integere konstante groter dan 0 en kleiner dan 65) achter de betreffende noot te plaatsen, bepaalt men de lengte van de noot. Een 1 staat voor een gehele noot, een 2 staat voor een halve noot, een 3 staat voor een derde noot etcetera.
In het voorbeeld worden enkele verschillende noten gespeeld; een hele, een halve, een kwart, een achtste, een zestiende en een tweeendertigste noot:

```
PLAY "C1D2E4F8G16A32"
Ok
```

Als standaard nootlengte wordt een kwartnoot aangenomen. Deze standaardlengte kan worden gewijzigd door het L-kommando. Met een L, gevolgd door een nootlengte (1 ... 64) wordt de standaard nootlengte voor de betreffende stem veranderd. Indien alleen de letter L wordt gebruikt, wordt een L4 aangenomen.

Het volgende voorbeeld resulteert in een snelle toonladder; de standaard nootlengte werd op een zestiende noot gezet:

```
PLAY "L16CDEFCAB"
Ok
```

Een rust wordt met het symbool R aangegeven, gevolgd door de duur van de rust (1 ... 64). Indien geen duur wordt opgegeven, wordt een kwartrust uitgevoerd. Voorbeeld: altijd is kortjakje ziek, eerste regel, tweestemmig:

```
PLAY "M6000S1L4CCGGAAGRL8FFFFL4
EEDDCR","M6000S1L4RREEFFERL8DD
DDL4CCGGER"
Ok
```

Op de M600S1 kodering die in beide stemmen werd gebruikt, gaan we later in. Voorlopig is het van belang om slechts te weten, dat deze toevoeging ervoor zorgt dat de tonen afzonderlijk hoorbaar zijn.

Met het O-kommando (pas op: de letter O en niet het cijfer 0!) kunnen we het oktaaf sturen. Een oktaaf loopt altijd van C tot B. Indien geen oktaaf wordt gespecificeerd, wordt een vierde oktaaf aangenoment; het oktaaf waarin de centrale C zich als basis bevindt. Het volgende voorbeeld geeft de grote terts toonladder van C, klimmend over drie oktaven:

```
PLAY "O3CDEFGABO4CDEFGABO5CDEFGAB"
Ok
```

Achter het O-kommando dient dus een cijfer (1 ... 8) te worden opgenomen dat aangeeft in welk oktaaf de betreffende noot dient te worden gespeeld. Het volgende voorbeeld laat de laagst en hoogst mogelijke toon binnen deze notatie horen:

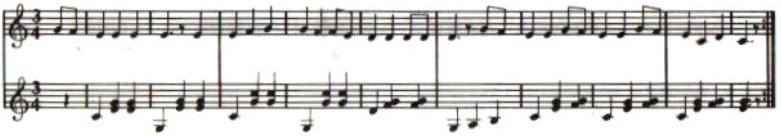
```
PLAY "O1CO8B"
Ok
```

Indien achter de letter O geen cijfer wordt opgenomen, wordt een O4 uitgevoerd.
Indien achter een noot een punt wordt geplaatst, wordt deze analoog aan de normale muzieknotatie met de halve lengte verlengd. Een twee-

de punt resulteert op dezelfde wijze in een tweede verlenging die de helft in lengte duurt van de eerste verlenging enzovoorts.

Voordat we op de wat meer technische kommando's van MML overgaan, volgt eerst een combinatievoorbeeld in de vorm van een eenvoudig driestemmig kinderliedje:

GOEDENAVOND SPEELMAN
MSX-ARRANGEMENT ♩=120



```
10 PLAY "V15T120","S1M9999T120",
"S1T120"
20 PLAY "L8O4GFL4EEEE.8RE","O4L4RCEEO3
GO4EE","O4L4RRGGRGG"
30 PLAY "EFGGFL8FEL4DDL8DD","CGGO3GO4G
GDFF","RO5CCRCCO4RGG"
40 PLAY "L4D.R8L8GFL4EEL8GFL4EEL8GF","
O3GABO4CEFCEF","RRRRO4GGRGG"
50 PLAY "L4ECDC.R8","CEFE.R8","RGGG.R8"
60 GOTO 20
RUN
```

Op de funktie van regel 10 gaan we later pas verder in.

Met het V-kommando kan per stem het volume worden bepaald. Achter de letter V dient dan een integere konstante te worden opgenomen, niet kleiner dan nul en niet groter dan 15. 15 geeft het hoogste volume, 0 geeft geen volume. In het voorgaande voorbeeld zien we in regel 10 ondermeer de volume-instelling van de eerste stem. Omdat deze het beste gehoord dient te worden, wordt deze op het hardst (V15) gezet. Probeer het voorbeeld ook eens met V0 ... V14.
Wanneer niet eerder een volume werd gespecificeerd, staat het volume standaard op V10.

Met het T-kommando kan het tempo van de muziek per stem worden bepaald. Achter de letter T dient dan een integere konstante, niet kleiner dan 32 en niet groter dan 255 te worden opgenomen. Dit getal stelt het aantal kwartnoten voor dat per minuut dient te worden gespeeld. Standaard staat het tempo ingesteld op 120 kwartnoten per seconde (T120). Het laatste voorbeeld kan door verandering van de T120 in regel 10 naar T255 (drie maal, één maal voor elke stem) tot twee maal zo snel worden gespeeld door de computer. Indien alleen de letter T wordt gebruikt, dan wordt een T120 uitgevoerd.

Met het N-kommando kunnen toonhoogten op een andere wijze worden bepaald. Het MSX-basic staat de aansturing van 96 verschillende tonen toe, genummerd van 1 tot en met 96. Elke volgende toon ligt weer een halve toon hoger . De 36e toon komt overeen met de centrale C. Met het N-kommando kan een toon worden gespeeld niet door door de muzikale notatie te gebruiken maar door de juiste toon uit de rij van 96 te kiezen. Het nummer van deze toon dient dan achter de letter N te worden geplaatst.
Het volgende voorbeeld toont twee kommando's die precies dezelfde uitwerking hebben:

```
PLAY "O4L4CDEFGFEDC"
Ok
PLAY "N36N38N40N41N43N41N40N38N36"
Ok
```

Een 0 achter de letter N resulteert in een rustpauze.

Met de letter X, gevolgd door een alfanumerieke variabele en een puntkomma, kan een alfanumerieke variabele worden ingelast. Hierdoor vormt ondermeer de maximale lengte van een alfanumerieke variabele (255 tekens) geen barrière meer. Een voorbeeld:

```
NEW
10 LET A$="CDEFGAB"
20 PLAY "O1XA$;O2XA$;O3XA$;O4XA$;
O5XA$;O6XA$;O7XA$;O8XA$;"
RUN
Ok
```

De grote terts toonladder wordt in alle mogelijke oktaven gespeeld.

Met het gelijkteken (=), gevolgd door een numerieke variabele en een puntkomma, kan de waarde van een numerieke variabele worden tussengevoegd. De variabele wordt pas tussengevoegd nadat een eventuele decimale fractie is verwaarloosd. Indien de waarde niet ligt binnen de toegestane waarden, volgt een foutmelding.
Het voorbeeld speelt alle mogelijke toonhoogten:

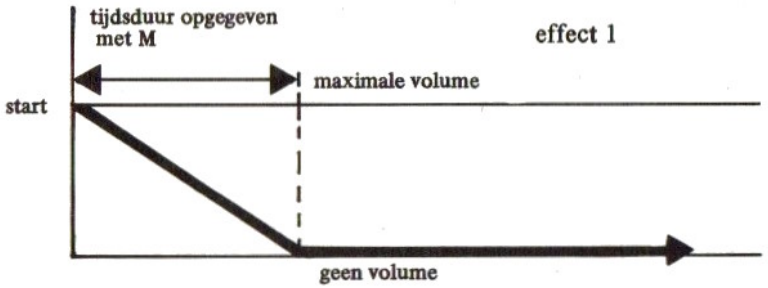
```
NEW
10 FOR I=1 TO 96
20 PLAY "N=I;"
30 NEXT
RUN
Ok
```

Met de letter S kan men één van de acht mogelijke volume-effecten selekteren. Het gebruik van de letter V naast de letter S heeft geen zin; een volume-effect heeft altijd een vast verloop.

Indien gebruik wordt gemaakt van een volume-effect, dan is het M-kommando daarbij onmisbaar. Met het M-kommando kan men de tijdseenheid specificeren waarbinnen een geluidseffect dient te zijn afgewikkeld. Hiertoe dient achter de letter M een gehele konstante te worden opgenomen, niet kleiner dan –65535 en niet groter dan 65535. Indien de tijdseenheid niet via een konstante maar via een numerieke variabele wordt gespecificeerd met gebruikmaking van het gelijkteken en de puntkomma, dan mag de waarde van de gebruikte variabele niet kleiner zijn dan –32768 en niet groter dan 32767.
Bij een negatieve waarde wordt voor uitvoering eerst de waarde 65536 opgeteld. De kenner herkent hierin de tweecomplementmethode.

De duur in seconden van deze tijdseenheid rekent men uit door de achter het kommando M gespecificeerde waarde te delen door de waarde 6965.

Een voorbeeld: volume-effect nummer 1 heeft tot gevolg dat het volume op de navolgende wijze verloopt:



Een toon begint dus op vol volume en sterft vervolgens weg. De tijd die verstrijkt totdat de toon volledig is weggestorven, wordt via het M-kommando bepaald. We laten nu een toon in één seconde wegsterven waardoor we een soort nagalm-effekt krijgen:

```
PLAY "M695S1T32A"
Ok
```

Vervolgens laten we de toon in een tiende seconde wegsterven zodat een tokkel-effect ontstaat:

```
PLAY "M697S1A"
Ok
```

Volume-effect 8 is een continue herhaling van effect 1. Steeds wanneer de toon is uitgestorven, wordt deze weer op vol volume gebracht. We laten een toon tien maal per seconde uitsterven en weer op volume komen:

```
PLAY "M697S8A"
Ok
```

Een snel tokkeleffect ontstaat. We kunnen ook 6965 maal per seconde de toon laten uitsterven en weer op volume komen:

```
PLAY "M1S8A"
Ok
```

Door deze zeer snelle afwisseling ontstaat een schel geluidseffect.

Door veel met het M- en het S-kommando te combineren, kunnen allerlei geluidseffecten worden verkregen.

In de volgende tabel staan alle geluidseffecten die in MSX-basic mogelijk zijn, bij elkaar genoemd.

Indien geen tijdsduur door middel van het M-kommando werd gespecificeerd, dan wordt een tijdsduur M255 aangenomen.
Indien een M-kommando zonder achtervoegsel wordt gebruikt, dan wordt de standaard waarde (255) aangenomen.
Indien een S-kommando zonder achtervoegsel wordt gebruikt, dan wordt een S0 uitgevoerd.
Een S-kommando kan worden opgeheven door gebruik van het V-kommando. Voor alle drie de stemmen kan maar één effect (s) en één tijdsduur (M) worden gespecificeerd. Steeds worden de laatst gespecificeerde waarden als geldend aangenomen voor alle stemmen waarvoor een effect werd aangezet.

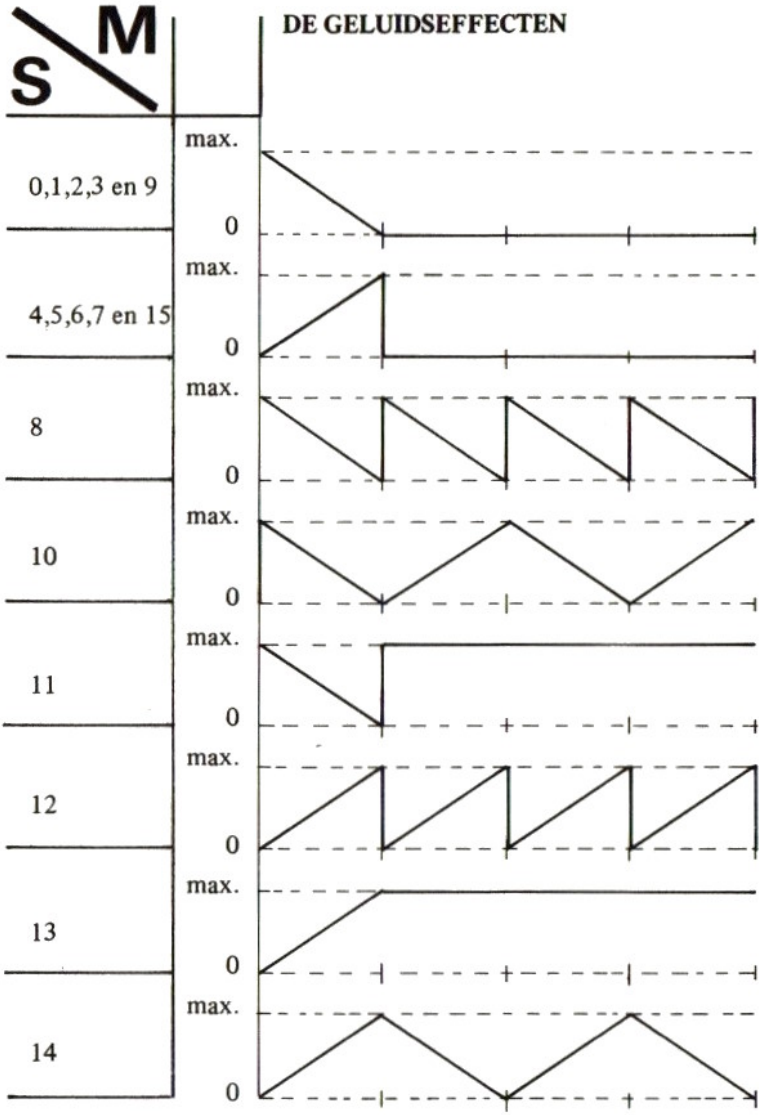
Enkele opmerkingen in verband met het PLAY-kommando:

- Door gebruik van het BEEP-kommando worden alle waarden weer op de standaardwaarde teruggebracht.

- Het maximale uitvoeringstempo is tien tonen per seconde. Indien een snellere passage wordt geprogrammeerd, blijft de uitvoering achter. Indien twee- of driestemmig wordt uitgevoerd, kan hierdoor een ongelijke uitvoering van de drie stemmen (een desynchronisatie) onstaat.

- Doordat besturingskommando's (zoals O, L, T en V) een kleine uitvoeringstijds inbeslag nemen, kan een desynchronisatie tussen de geprogrammeerde stemmen ontstaan bij meerstemmige programmering. Deze desynchronisatie kan worden opgelost door tussenvoeging van R64-kommando's (vierenzestigste rusten) op gehoor in de 'te snelle' stem. Ook verdient het aanbeveling om de stemmen per PLAY-kommando niet te lang te maken. Per PLAY-kommando vindt namelijk een algehele synchronisatie plaats; de stemmen worden bij elk PLAY-kommando altijd gelijk gestart.

- Indien geen effecten worden gebruikt, is de toonscheiding tussen twee gelijke tonen niet hoorbaar. Dit kan worden opgelost door


S
M
DE GELUIDSEFFECTEN
0,1,2,3 en 9
4,5,6,7 en 15
8
10
11
12
13
14
max.
0

een R64 tussen de twee noten in te lassen. Bijvoorbeeld:

```
PLAY "V10L4T120"
Ok
PLAY "AA"                    (een toon is hoorbaar)
Ok
PLAY "AR64A"          :onen zijn hoorbaar)
Ok
```

Pas met deze methode echter wel op voor desynchronisatie-effecten.

Op de volgende pagina is een kort overzicht van de kommando's van MML opgenomen. Het is raadzaam om dit overzicht over te schrijven of te kopiëren zodat het als hulpmiddel kan fungeren wanneer men veel met de geluidsgenerator wil werken.

CDEFGAB.. toonhoogten. Achter de naam van de noot kan de tijdsduur (1 ... 64) worden opgenomen. Indien geen tijdsduur wordt opgenomen, wordt de standaard tijdsduur genomen

\+ kruis; halve noot verhoging

– mol; halve toon verlaging

L.. veranderen standaard tijdsduur voor een noot. L4 is standaard. De tijdsduur dient achter de letter L te worden opgenomen (1 ... 64). Alleen de letter L is gelijk aan een L4

R.. rust. Achter de letter R dient de duur van de rust (1 ... 64) te worden opgenomen. Alleen de letter R is gelijk aan een kwart rust

O. oktaaf. Een oktaafnummer (1 ... 8) kan worden uitgekozen. Een oktaaf loopt altijd van C naar B. O4 is het standaard oktaaf en heeft als eerst noot de centrale C. Indien achter de O geen cijfer wordt opgenomen, wordt een O4 uitgevoerd

=...; tussenvoeging van de waarde van een numerieke variabele

X...; tussenvoeging van een alfanumerieke variabele

S.. selectie van een volume-effect. Achter S dient het effectnummer (0 ... 15) te worden opgegeven. Indien geen effectnummer wordt opgegeven, wordt effectnummer 0 aangenomen

M..... tijdsduur van de effect-ontwikkeling. Vermenigvuldig de gewenste ontwikkelingstijd in seconden met 6965 en rond de verkregen waarde af. Deze waarde dient achter de M te worden opgenomen (0 ... 65535). Bij M=...; dient de genoemde variabele een integere waarde te bezitten (–32768 ... 32767). Het M-kommando telt bij uitvoering bij een negatieve waarde dan 65536 op

N.. de toonhoogte dient na de letter N (0 ... 96) te worden opgenomen. N0 is gelijk aan een rust en N36 is gelijk aan de centrale C. Elke volgende toon is weer een halve toon hoger

T... tempo. Achter de letter T dient het aantal kwartnoten per minuut te worden opgenomen. Indien achter de letter T geen waarde wordt opgenomen, wordt een T120 uitgevoerd.

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **N-FUNKTIE**
afkomst ..... **POINT is punt**

schrijfwijze

```
POINT<LOCATIE>
<LOCATIE>:::=[STEP](<HORIZONTAAL>,<VERTI-
  KAAL>)
<HORIZONTAAL>:::=<N>
<VERTIKAAL>:::=<N>
<N>:::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Voor goed begrip van deze funktie is het noodzakelijk dat eerste de PSET-behandeling volledig is doorgenomen.

Deze funktie geeft als resultaat de kode van de kleur die het gelocaliseerde puntje heeft. Indien het punt zich buiten het beeld bevindt, geeft deze funktie de waarde –1.

Bijvoorbeeld:

```
NEW
10 COLOR 15,0,0:SCREEN 3
20 FOR I=10 TO 100 STEP 8
30 FOR J=10 TO 100 STEP 8
40 PSET (I,J),-15*(POINT(I,J)=0)
50 NEXT J,I
60 GOTO 20
RUN
```

Op beeldscherm verschijnt een raster van punten die steeds weer verschijnen en verdwijnen. In regel 40 wordt de kleur van het betreffende puntje op 15 gesteld indien deze eerder 0 was en op 0 gesteld indien deze eerder 15 was.

De POINT-funktie geeft geen zinvolle resultaten wanneer een alfanumeriek beeldscherm is geactiveerd.

moeilijkheidsgraad . . **zeer moeilijk, kennis van de opbouw van het computergeheugen is vereist**
soort . . . . . **KOMMANDO**
afkomst . . . . . **POKE is wegstoppen**

schrijfwijze

```
POKE<GEHEUGENADRES>,<GEHEUGENWAARDE>
<GEHEUGENADRES>::=<N>
<GEHEUGENWAARDE>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met het POKE-kommando kunnen geheugenadressen van andere waarden worden voorzien. Hiertoe dient een geheugenadres te worden opgegeven, gevolgd door een komma en de waarde die het byte op de betreffende geheugenplaats dient aan te nemen.
Het geheugenadres mag niet kleiner zijn dan –32768 en niet groter dan 65535. Een eventuele decimale fractie wordt verwaarloosd. Indien een geheugenadres negatief is, wordt voordat het geheugen wordt aangesproken eerst de waarde 65536 bij dit adres opgeteld.
De geheugenwaarde dient in één byte te passen en mag derhalve niet kleiner zijn dan 0 en niet groter zijn dan 255 terwijl een eventuele decimale fractie wordt verwaarloosd.

Het met POKE toegankelijke geheugen wordt gevormd door:

–	0 ... 32767	32 kilobyte ROM waarin het MSX-basic is gekodeerd
–	32768 ... 65535	32 kilobyte RAM, het voor MSX-basic beschikbare geheugen.

Het veranderen van geheugenwaarden in de eerste 32 kilobyte geheugen heeft geen zin; dit gedeelte van het geheugen is ROM (Read Only Memory = alleen leesbaar geheugen) en heeft een vaste, onveranderde waarde. Bijvoorbeeld:

```
NEW
10 PRINT PEEK(32)
```

```
20 POKE 32,0
30 PRINT PEEK(32)
RUN
 192
 192
Ok
```

Op regel 20 werd een vergeetse poging gedaan om een stukje ROM te wijzigen.

In het tweede stuk geheugen van 32 kilobyte kunnen wel wijzigingen worden aangebracht; dit gedeelte van het geheugen bestaat uit RAM (Random Access Memory = vrij toegankelijk geheugen). Wijzigingen in dit geheugen dienen met grote terughoudendheid en met goede kennis van zaken te worden aangebracht; onoordeelkundig gebruik van POKE leidt bijna altijd tot het 'ophangen' van het computersysteem. Uit- en weer inschakelen van de computer is dan de enige remedie.

Bijvoorbeeld:

```
NEW
10 FOR I=32768 TO 65535
20 POKE I,0
30 NEXT I
RUN
Ok
LIST
Ok
```

Dit programma wist zichzelf uit; na uitvoering zijn de programmaregels verdwenen.

```
NEW
10 FOR I=65535 TO 32768 STEP -1
20 POKE I,0
30 NEXT I
RUN
```

Na korte tijd start de computer opnieuw op; de geheugenwijziging had blijkbaar nogal destructieve gevolgen.

Voor een goed gebruik van POKE is gedetailleerde kennis van de opbouw van het geheugen van een MSX-computer noodzakelijk. Hiertoe dient specialistische literatuur te worden geraadpleegd; in dit handboek wordt hierop verder niet ingegaan.

moeilijkheidsgraad . . . . . **zeer eenvoudig**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **POS is afkorting van position –positie**

schrijfwijze

```
POS(<U>)
<U>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de positie waarop de cursor zich op het moment in een regel bevindt. De uitdrukking tussen haakjes is een dummy-uitdrukking; in de praktijk kan het beste gewoon een nul worden gebruikt. Voorbeeld:

```
NEW
10 FOR I=0 TO 10 STEP 2
20 LOCATE I:PRINT POS(0)
30 NEXT I
RUN
 0
   2
     4
       6
         8
           10
Ok
```

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **PRESET is afkorting van point resetting–punt terugzetten (verwijderen)**

schrijfwijze

```
PRESET<LOCATIE>[,<KLEUR>]
<LOCATIE>::=[STEP](<HORIZONTAAL>,<VERTI-
  KAAL>)
<HORIZONTAAL>::=<N>
<VERTIKAAL>::=<N>
<KLEUR>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES)
```

betekenis

PRESET werkt bijna volledig identiek aan PSET. Het enige verschil is, dat bij het weglaten van de kleurkode niet de voorgrondkleur maar de achtergrondkleur wordt gekozen als kleur waarin het puntje wordt uitgevoerd.

moeilijkheidsgraad . . **eenvoudig maar in compexe samenstellingen tot vrij moeilijk**
soort . . . . . **KOMMANDO**
afkomst . . . . . **PRINT is afdrukken**

schrijfwijze

```
PRINT
  ?   [#<KANAAL>,][{,|;}<P>{;|,}[<P>]}][
USING<USING STRING>;<U>;<U>{;|,}<U>}<KA-
NAAL>:::=<N>
<P>:::=<U>|TAB(<POSITIE>)SPC(<AANTAL
SPATIES>)
<POSITIE>:::=<N>
<AANTAL SPATIES>:::=<N>
<USING STRING>:::=<A>
<N>:::=<ZIE ALGEMENE SPECIFICATIES>
<A>:::=<ZIE ALGEMENE SPECIFICATIES>
<U>:::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het PRINT-kommando is één van de meest belangrijke maar ook ingewikkelde kommando's. Het kommando is op veel zeer verschillende manieren te gebruiken.

eenvoudig gebruik

Met het PRINT-kommando kan op beeldscherm een regel worden opgeschoven. Voorbeeld:

```
PRINT:PRINT:PRINT:PRINT
```

(deze regels worden overgeslagen)

```
Ok
```

Met het PRINT-kommando kunnen konstanten of de inhoud van vari-

abelen worden afgedrukt op het beeldscherm. Voorbeeld:

```
LET A=2.3:LET A$="TEST"
Ok
PRINT "DIT IS EEN ";A$;A
DIT IS EEN TEST 2.3
Ok
```

Indien de verschillende af te drukken uitdrukkingen van elkaar gescheiden zijn met een punt-komma, worden deze uitdrukkingen tegen elkaar aan afgedrukt. Merk op dat voor een positieve waarde altijd een spatie wordt afgedrukt. Deze spatie kan worden beschouwd als het teken (plus-teken) van deze waarde. Ná een numerieke waarde wordt ook altijd een extra spatie afgedrukt.
Indien de verschillende uitdrukkingen van elkaar gescheiden worden door een komma, wordt elke uitdrukking een kolom verder afgedrukt. Het beeldscherm is hiertoe ingedeeld in kolommen van veertien tekens. Bijvoorbeeld:

```
PRINT "EEN","TEST"
EEN            TEST
Ok
```

Een TAB-funktie biedt de mogelijkheid om zelf tabulaties (kolomindelingen) door te voeren. Tussen de haakjes van de TAB-funktie dient de positie te worden vermeld waarop het afdrukken op het beeldscherm dient te beginnen. Bijvoorbeeld:

```
NEW
10 PRINT "EEN";TAB(20);"TWEE"
20 PRINT "APPELS";TAB(20);"PEREN"
RUN
EEN                  TWEE
APPELS               PEREN
Ok
```

Een SPC-funktie biedt de mogelijkheid om tussen twee af te drukken uitdrukkingen een vast aantal spaties te plaatsen. Bijvoorbeeld:

```
NEW
10 PRINT "EEN";SPC(20);"TWEE"
20 PRINT "APPELS";SPC(20);"PEREN"
RUN
EEN                          TWEE
APPELS                          PEREN
Ok
```

Indien de tussen de haakjes van TAB of SPC vermelde waarde gebroken is, zal de hoogste gehele waarde kleiner dan deze gebroken waarde geldig zijn.

Wanneer een PRINT-kommando wordt afgesloten met een puntkomma, dan zal een volgende printopdracht achter het laatst afgedrukte gedeelte doorgaan en dus niet op een volgende regel beginnen. Bijvoorbeeld:

```
NEW
10 PRINT "HALLO ";
20 PRINT "ALLEMAAL"
RUN
HALLO ALLEMAAL
Ok
```

Wanneer een PRINT-kommando wordt afgesloten met een komma, dan zal een volgende printopdracht niet op de volgende regel maar in een volgende kolom doorgaan. Bijvoorbeeld:

```
NEW
10 PRINT "EEN",
20 PRINT "TWEE"
RUN
EEN             TWEE
Ok
```

N.B.: MSX-basic versie 1.0 maakt in het PRINT-bevel in sommige constructies een fout. Een enkele maal wordt een gegeven toch onbedoeld op een volgende regel gezet. De tweede kolom laat zich met een komma niet vinden; in plaats daarvan wordt op de volgende regel verder gegaan.

gevorderd gebruik: USING

Indien een PRINT-kommando wordt geconstrueerd met daarin het USING-kodewoord, dan kan direkt na dat USING-kodewoord een formaat (of print-masker) worden gespecificeerd. De af te drukken gegevens dienen dan naar dit formaat te worden gevormd.
De diverse karakters die in een formaatspecificatie (using-string) mogen worden gebruikt, worden hieronder per stuk in een voorbeeld behandeld.

Het "!"-teken

Met het "!"-teken kan worden aangegeven dat slechts één karakter dient te worden afgedrukt. Voorbeeld:

```
PRINT USING "!";"MSX"
M
Ok
```

Het "\"-teken

Met het "\"-teken kan worden aangegeven dat slechts een bepaald aantal karakters dient te worden afgedrukt. Hiertoe dienen in de using-string twee "\"-tekens te worden opgenomen met daartussenin het aantal af te drukken tekens minus 2 **aan spaties**. Voorbeeld:

```
NEW
10 LET A$="ABCDEFGHIJKLMNOP"
20 PRINT USING "\    \";A$
RUN
ABCDE
Ok
```

Zoals ook voor de nog te behandelen tekens geldt, mogen de diverse speciale tekens ook bij elkaar in één using-string worden gebruikt. Daarbij mogen ook niet-speciale karakters in een using-string worden gebruikt; deze worden dan gewoon afgedrukt. Bijvoorbeeld:

```
NEW
10 LET U$="\    \/!"
20 PRINT USING U$;"ABCDEFGH";"919191"
```

```
RUN
ABCDE/9
Ok
```

Het "&"-teken

Met het "&"-teken kan worden aangegeven dat een alfanumerieke uitdrukking onverkort dient te worden afgedrukt. Voorbeeld:

```
10 INPUT "WOORD ";A$
20 PRINT USING "DE EERSTE LETTER VAN & I
S !";A$;A$
RUN
WOORD ? KRAAI                    (dit wordt ingegeven)
DE EERSTE LETTER VAN KRAAI IS K
Ok
```

Het "#"-teken

Met het "#"-teken geeft men aan dat op die plaats een cijfer van de uitkomst van een numerieke uitdrukking mag worden geplaatst. Met een veelheid van "#"-tekens kan men precies bepalen waar de uitkomst van een numerieke uitdrukking moet komen te staan en hoeveel posities deze mag hebben. Bijvoorbeeld:

```
10 FOR I=7 TO 12:PRINT USING "##";I
20 NEXT I
RUN
 7
 8
 9
10
11
12
Ok
```

Merk op dat in bovenstaand voorbeeld de getallen netjes **rechts** gericht onder elkaar staan. Zonder het gebruik van de USING-mogelijkheid zouden ze links gericht onder elkaar staan.
Indien de using-string te klein is, wordt het betreffende numerieke antwoord normaal afgedrukt maar met een voorlopend "%"-teken

als waarschuwing. Bijvoorbeeld:

```
PRINT USING "### ##";100;100
100 %100                          (de tweede "using" is te klein)
Ok
```

Indien nodig, wordt het af te drukken getal door dit USING-gebruik afgerond. Bijvoorbeeld:

```
NEW
10 PRINT USING "##";1.4;1.5
 1 2
Ok
```

Uit het voorgaande voorbeeld blijkt ook dat wanneer de using-string is "opgebruikt", hij weer van voor af aan opnieuw wordt gebruikt. In dit geval wordt hierdoor twee maal hetzelfde patroon gebruikt voor het afdrukken van de getallen 1.4 en 1.5. Voor een negatieve waarde dient een extra positie te worden gecreëerd in verband met het te plaatsen "–"-teken. Bijvoorbeeld:

```
PRINT USING "##";10;-10
10%-10                        (voor –10 is de using-string te klein).
```

De punt

Met de decimale punt kan het numerieke print-masker met decimale posities worden gebruikt. Ook bij dit gebruik wordt eventueel weer afgerond. Voorbeeld:

```
PRINT USING "###.##";11.2;12.125
 11.20 12.13
Ok
```

Merk op dat posities na de decimale punt eventueel netjes met nullen wordt aangevuld.

Het "+"-teken

Het "+-teken mag vóór of na een numeriek print-masker worden geplaatst. Dit heeft tot gevolg dat het teken van het af te drukken getal voor of na dit getal wordt vermeld. Bijvoorbeeld:

```
NEW
10 LET A=22.3:LET B=-111
20 PRINT USING "+#####.##";A;B
30 PRINT USING "#####.##+";A;B
RUN
 +22.30 -111.00
 22.30+ 111.00-
Ok
```

Het "–"-teken

Het "–"-teken heeft bijna dezelfde uitwerking als het "+"-teken. Het verschil is dat in plaats van een "+"-teken een spatie word afgedrukt in het uiteindelijke resultaat en dat het "–"-teken alleen rechts van een print-masker het gewenste effect heeft. Bijvoorbeeld:

```
PRINT USING "###.##-";1;-1
 1.00     1.00-
Ok
```

Het "**"-teken

Een dubbele ster, geplaatst voor een numeriek print-masker, heeft tot gevolg dat het print-masker met twee numerieke plaatsingsmogelijkheden voor cijfers wordt uitgebreid en dat daarbij alle spaties die door gebruik van dit print-masker voor het af te drukken getal zouden ontstaan, worden vervangen door sterren. Bijvoorbeeld:

```
PRINT USING "****.##";2
***2.00
Ok
PRINT USING "** ";2;22
*2 22
Ok
```

Uit het laatste voorbeeld blijkt dat de dubbele ster ook alleen mag worden gebruikt.

Het "$$"-teken

Een dubbel dollarteken, geplaatst voor een numeriek print-masker,

heeft tot gevolg dat er één numerieke plaatsmogelijkheid extra wordt gecreëerd voor een cijfer en dat vóór het af te drukken getal een dollarteken wordt geplaatst. Bijvoorbeeld:

```
PRINT USING "$$#.##";2.5
 $2.50
Ok
```

Ook het "$$"-teken mag eventueel alleenstaand worden gebruikt.

Het "**$"-teken

Het gebruik van twee sterren en één dollarteken heeft tot gevolg dat de "$$" en de "**" worden gecombineerd. Het resultaat is twee extra plaatsingsmogelijkheden voor cijfers, een dollarteken voor het af te drukken getal en een opvulling van de linker spaties met sterren. Bijvoorbeeld:

```
PRINT USING "**$#.##";2.5
**$2.50
Ok
```

Ook het "**$"-teken mag alleenstaand worden gebruikt.

De komma

Een komma, opgenomen in een numeriek print-masker, niet als eerste teken en links van de eventueel aanwezige decimale punt heeft tot gevolg dat de duizendtallen met een komma worden afgescheiden in het af te drukken getal. Eventueel mogen meerdere komma's op de beschreven wijze worden opgenomen; zij resulteren alle in extra plaatsingsmogelijkheden voor cijfers. Voorbeeld:

```
NEW
10 PRINT USING "####,.##";1000
20 PRINT USING "####.##,";1000
RUN
1,000.00
1000.00,
Ok
```

Merk op dat op programmaregel 20 de komma niet het beschreven effect heeft; de komma staat **na** de decimale punt.

Het "^^^^"-teken

Met het teken "^^^^" geeft men aan, dat een exponentiële vorm dient te worden gehanteerd bij het afdrukken. De "**", de "$$", de "**$" en het gebruik van de komma zijn in combinatie met "^^^^" wel toegestaan maar geven zinloze resultaten. Het gebruik van een "+" of een "–" **ACHTER** het numerieke print-masker is ook toegestaan maar geeft eveneens een zinloos resultaat.
"^^^^" dient direkt na het numerieke print-masker te worden opgenomen. Bijvoorbeeld:

```
PRINT USING "#.##^^^^";20000
0.20E+05
Ok
```

Gevorderd gebruik: schrijven naar randapparatuur.

Zie hiervoor ook de beschrijving van het **OPEN** en **CLOSE**-kommando.

Indien de kanaal-optie van het PRINT-kommando wordt gebruikt, kan men de gegevens in plaats van naar het beeldscherm ook naar een andere plaats zenden. Dit kan bijvoorbeeld een cassetteband of het grafische scherm zijn. Ook kunnen de gegevens op deze wijze naar een printer (afdrukeenheid) worden verzonden alhoewel het LPRINT-kommando daar veel gemakkelijker voor is.

Indien de gegevens later met een INPUT weer dienen te worden gelezen (geldt natuurlijk niet voor de printer en het (grafische) beeldscherm), dan dient men er voor te zorgen dat de diverse gegevens die op één printregel staan door een komma van elkaar zijn gescheiden. Zie verder hiervoor het INPUT-kommando. Voorbeeld: gegevens schrijven naar cassetteband.

```
NEW
10 OPEN "CAS:TEST" FOR OUTPUT AS 1
20 PRINT #1,"DEZE GEGEVENS WORDEN"
30 PRINT #1,"STRAKS WEER OPGEHAALD"
40 PRINT #1,123;",";-12.33
50 CLOSE
```

(nu eerst de cassetterecorder op opnemen zetten)

```
RUN                    (de cassetterecorder begint te lopen)
Ok     (de cassetterecorder stopt weer; de gegevens zijn op cassette-
                                                band geschreven)
```

(N.B. : op dit voorbeeld komen we bij de behandeling van het INPUT-kommando nader terug)

Een tweede voorbeeld: schrijven naar het grafische scherm.

```
NEW
10 SCREEN 3
20 OPEN "GRP:" AS 1
30 PRINT #1,"MSX";
40 GOTO 30
RUN
```

(het beeldscherm wordt met grote letters MSX volgezet)

SLEUTELWOORD **PSET**

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **PSET is point setting – punt zetten**

schrijfwijze

```
PSET<LOCATIE>[,<KLEUR>]
<LOCATIE>:::=[STEP](<HORIZONTAAL>,<VERTI-
  KAAL>)
<HORIZONTAAL>:::=<N>
<VERTIKAAL>:::=<N>
<KLEUR>:::=<N>
<N>:::=<ZIE ALGEMENE SPECIFICATIES>
```

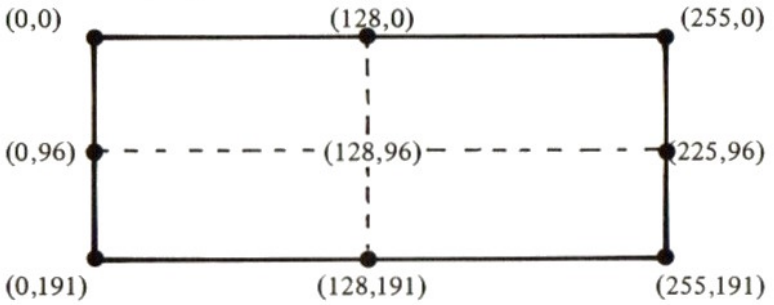
betekenis

Voor een goed begrip van dit kommando is het raadzaam, eerst het SCHREEN- en het COLOR-kommando goed door te nemen.

Met dit kommando kan een puntje op een grafisch beeldscherm worden gezet. Dit puntje kan in een bepaalde kleur worden uitgevoerd; indien geen kleur wordt gegeven, wordt de voorgrondkleur die op dat moment actief is, als kleur genomen.

De plaats op het beeldscherm kan als volgt worden bepaald:

Hoog oplossend vermogen: de linker bovenhoek van het beeldscherm krijgt positie 0,0. De rechter onderhoek krijgt positie 255,191. Tussen deze twee grenzen kan elk puntje worden aangesproken. Een schema ter verduidelijking:



Laag oplossend vermogen: de plaats op het beeldscherm wordt precies hetzelfde bepaald. Echter, op de puntjes die op het beeldscherm worden geplaatst, zijn bij een laag oplossend vermogen vier maal zo groot als bij een hoog oplossend vermogen in beide richtingen. Hierdoor geven bij voorbeeld de locaties (0,0) tot en met (3,3) bij een laag oplossend vermogen dezelfde locatie voor een puntje aan. Op een beeldscherm met laag oplossend vermogen moet dus de linker bovenhoek van het te plaatsen puntje worden aangegeven.

Voorbeelden: (allemaal te onderbreken met CONTROL-STOP)

```
NEW
10 SCREEN 2
20 PSET (10,10)
30 PSET (30,30)
40 PSET (10,30)
50 PSET (30,10)
60 GOTO 60
RUN
```

Op het beeldscherm verschijnen vier puntjes als hoekpunten van een vierkant. Regel 60 voorkomt dat ongewenst weer een alfanumeriek scherm wordt geactiveerd en de tekening verdwijnt.

```
10 SCREEN 3
RUN
```

Alleen programmaregel 10 werd veranderd. Dezelfde punten worden afgebeeld; de punten zijn echter in beide richtingen vier maal zo groot.

```
NEW
10 SCREEN 2
20 FOR I=10 TO 30
30 FOR J=10 TO 30
40 PSET (I,J)
50 NEXT J,I
60 GOTO 60
RUN
```

Het vierkant waarvan in de eerdere voorbeelden alleen de hoekpunten werden afgebeeld, verschijnt nu in ingekleurde vorm doordat elk puntje binnen dit vierkant wordt geplaatst.

```
10 SCREEN 3
40 PSET (4*I,4*J)
RUN
```

Alleen de programmaregels 10 en 40 worden veranderd. Hetzelfde vierkant maar dan zestien maal vergroot (vier maal in elke richting) wordt afgebeeld. Merk op dat in programmaregel 40 steeds de locatie van de linker bovenhoek van het te plaatsen puntje wordt aangegeven.

```
10 SCREEN 2
RUN
```

Alleen programmaregel 10 werd veranderd. Het resultaat is dat nu een raster van puntjes wordt afgedrukt. Elk puntje geeft de linkerbovenhoek aan van een puntje dat zou zijn geplaatst bij een SCREEN 3-inrichting (voorlaatste voorbeeld).

```
10 SCREEN 3
40 PSET (4*I,4*J),15*RND(1)
RUN
```

Alleen programmaregels 10 en 40 werden veranderd. De punten zoals in het voorlaatste voorbeeld worden in willekeurige kleuren uitgevoerd.

```
10 SCREEN 2
RUN
```

Alleen programmaregel 10 werd veranderd. De punten zoals in het voorlaatste voorbeeld worden weer afgebeeld maar nu in diverse kleuren. De kleuren zijn niet helemaal willekeurig; in dit voorbeeld zijn de kleuren van de punten horizontaal twee aan twee gelijk.

De hoogste resolutie (oplossend vermogen) brengt met zich mee dat voor elk puntje niet elke kleur kan worden geselekteerd. De kleur wordt per groep van puntjes bepaald waarbij de laatst vastgestelde kleur geldig is. Kleuren worden per groep van acht naast elkaar liggende puntjes bepaald. De groepering is enigszins onduidelijk en is alleen

goed te begrijpen wanneer men de exacte opbouw van het video-geheugen weet. Het is raadzaam om in de hoge resolutie kleuren met mate te gebruiken en de diverse kleuringen op enige afstand van elkaar te gebruiken. Misschien is het zelfs wel verstandig om, teneinde problemen te voorkomen, slechts twee kleuren (voorgrond en achtergrond) te gebruiken indien dat enigszins mogelijk is.

De locatie op scherm kan ook met behulp van het STEP sleutelwoord worden bepaald. In dat geval spreekt men over een relatieve locatie; de beeldschermlocatie wordt gegeven ten opzichte van het laatste getekende punt. Bijvoorbeeld:

```
NEW
10 SCREEN 2
20 PSET (0,0)
30 FOR I=1 TO 20
40 PSET STEP (10,10)
50 NEXT I
60 GOTO 60
RUN
```

Op één lijn, steeds 10 puntjes horizontaal en vertikaal verderop, verschijnen puntjes op het beeldscherm. Programmaregel 20 zet het eerste puntje in de linker bovenhoek; programmaregel 40 zet vervolgens steeds (10,10) verderop een nieuw puntje neer. Steeds wordt een volgende locatie bepaald door de aangegeven stapgrootten in horizontale en vertikale richting bij de vorige locatie op te tellen.

```
NEW
10 SCREEN 2
20 PSET (10,10)
30 PSET STEP (20,20)
40 PSET STEP (-20,0)
50 PSET STEP (20,-20)
60 GOTO 10
RUN
```

Dit programma geeft precies hetzelfde resultaat als het eerste voor beeld; het geeft de vier hoekpunten van een vierkant aan.

De tussen haakjes vermelde locaties dienen integere waarden te bevatten. Indien één van deze uitdrukkingen een gebroken waarde als uitkomt heeft, wordt deze waarde ontdaan van een decimale fractie (de decimalen worden verwaarloosd. Indien de uiteindelijke waarde van een uitdrukking dan kleiner is dan –32768 of groter is dan 32767, volgt een foutmelding.

Het is opmerkelijk dat er puntjes buiten het beeldscherm kunnen worden gezet. Deze mogelijkheid heeft een reële betekenis. Hierop wordt in de diverse beschrijvingen van de grafische kommando's verder ingegaan.

SLEUTELWOORD

PUT SPRITE

moeilijkheidsgraad . . . . . **moeilijk**
soort . . . . . **KOMMANDO**
afkomst . . . . . **PUT SPRITE is plaats 'geest'**

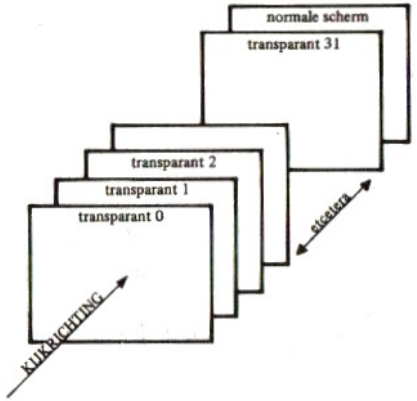
schrijfwijze

```
PUT SPRITE<TRANSPARANT NUMMER>,[<LOCA-
  TIE>][,[<KLEUR>][,<SPRITE NUMMER>]]\
  PUT SPRITE<...>,{,}
<LOCATIE>::=[STEP](<HORIZONTAAL>,<VER-
  TIKAAL>)
<TRANSPARANT NUMMER>::=<UITDRUKKING>
<KLEUR>::=<UITDRUKKING>
<SPRITE NUMMER>::=<UITDRUKKING>
<HORIZONTAAL>::=<UITDRUKKING>
<VERTIKAAL>::=<UITDRUKKING>
<UITDRUKKING>::=<ZIE ALGEMENE SPECIFICA-
  TIES>
<...>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Alvorens dit kommando te bestuderen, is het een noodzaak om SPRITE$, COLOR, SCREEN en PSET grondig te hebben bestudeerd.

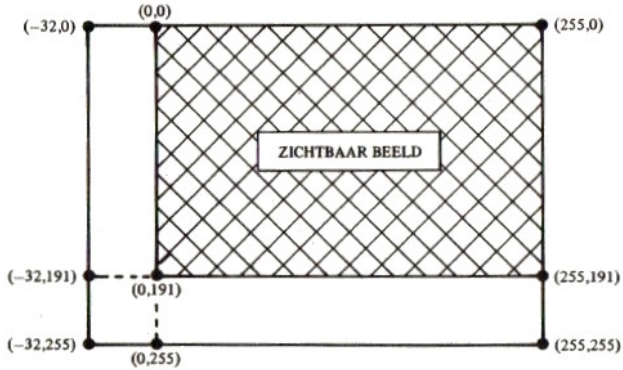
Het beeldscherm van een MSX-computer kan men zich als volgt voorstellen:



Het beeldscherm bestaat in feite uit een normaal (achtergrond)scherm met daarvoor 32 transparante schermen. Tot nu toe gebruikten we deze transparanten niet maar tekenden we slechts op het achtergrondscherm.
Met het PUT SPRITE kommando kan een sprite op een willekeurige plaats in een willekeurige kleur op een willekeurig transparant worden afgebeeld. De volgende regels gelden:

- een sprite kan maar één kleur hebben
- er kan maar één sprite per transparant geplaatst worden
- indien twee sprites geheel of gedeeltelijk achter elkaar worden geplaatst, dan wordt ook visueel de achterste sprite achter de voorste geplaatst. De voorste sprite is dus volledig te zien terwijl de daar achter liggende sprite slechts gedeeltelijk of soms helemaal niet is te zien
- er kunnen maximaal 4 sprites tegelijk op één horizontale lijn staan. Indien meer dan 4 sprites op verschillende transparanten naast elkaar worden gezet, dan verdwijnen de achterste sprites visueel geheel of gedeeltelijk. Meer dan 4 sprites ONDER elkaar is geen probleem
- een sprite is op de niet getekende plaatsen (de witte ruitjes; zie SPRITE$) doorzichtig. Zo geeft het passeren van twee sprites altijd een visueel aanvaardbaar effect.

Sprites mogen zich binnen het transparant waarop ze zich bewegen, de volgende posities innemen:



Een sprite kan zich dus geheel of gedeeltelijk buiten het beeld bevinden.

Achter PUT SPRITE dienen we als eerste gegeven het transparantnummer aan te geven van het transparant waarop de sprite moet worden geplaatst. Een eventueel in eerdere instantie op die transparant geplaatste sprite verdwijnt automatisch bij het plaatsen van de nieuwe transparant. Het transparantnummer wordt gegeven in een numerieke uitdrukking en mag niet kleiner zijn dan nul en niet groter dan 31. Een evenutele decimale fractie wordt verwaarloosd.

Als tweede gegeven achter PUT SPRITE kan de locatie van de betreffende sprite worden gegeven. Deze locatie dient aan dezelfde eisen te voldoen als de locatie zoals voorkomend in het PSET-kommando en kan dus zowel absoluut als relatief zijn. Indien de locatie wordt weggelaten, wordt de laatste gespecificeerde locatie voor dat transparant genomen. Indien nog niet eerder een locatie voor de betreffende sprite werd gespecificeerd, wordt locatie (–32,192) aangenomen.
Pas op: Indien een relatieve locatie wordt opgegeven (dus een STEP (...,...)) dan wordt deze relatieve locatie uitgevoerd *ten opzichte van de laatst gegeven locatie*. Op welk grafisch kommando deze laatste locatie van toepassing was, is dan niet van belang. De opgegeven relatieve locatie is in ieder geval NOOIT de locatie ten opzichte van de vorige locatie *voor die betreffende sprite*.

Indien de y-coördinaat van een sprite (de vertikale positionering) gelijk is aan 208, dan heeft dat tot effect dat alle sprites die op een transparant zijn geplaatst ACHTER het transparant waarop de sprite met y-coördinaat 208 is geplaatst, van het beeldscherm verdwijnen.

Als derde gegeven dient een kleur te worden opgegeven waarin de sprite dient te worden uitgevoerd. Dezelfde kleurkodering als gegeven bij het PSET-kommando geldt hier. Indien geen kleurkodering wordt opgegeven, wordt de laatste voor deze sprite gekodeerde kleur aangenomen. Indien nog niet eerder een kleur werd gespecificeerd, wordt de voorgrondkleur als geldende kleur genomen.

Als vierde en laatste gegeven kan het betreffende sprite (de betreffende spelfiguur) worden genoemd. Dit spelfiguur dient eerder met SPRITE$ te zijn gedefinieerd. Indien dit gegeven achterwege wordt gelaten, dan neemt het MSX-basic aan dat het nummer van het spelfiguur gelijk is aan het nummer van het transparant.

Hieronder volgen enkele voorbeelden. Het is noodzakelijk om veel met sprites te oefenen alvorens aan wat meer serieus programmeerwerk te beginnen.

Voorbeeld 1, definitie van een 8 bij 8 sprite. Dit voorbeeld werd bij de behandeling van SPRITE$ al uitvoerig behandeld.

```
NEW
10 SPRITE$(0)=CHR$(24)+CHR$(60)+CHR$(60)
CHR$(86)+CHR$(126)+CHR$(126)+CHR$(230)+
CHR$(124)
```

Met dit kommando is spelfiguur nummer 0 bepaald. Het spelfiguur is gelijk aan het eerder bij SPRITE$ bepaalde figuur met dit verschil dat er decimale en geen binaire konstanten zijn gebruikt.
Om geen foutmelding te krijgen, dienen we voorafgaand aan dit kommando eerst de juiste beeldscherminstelling te realiseren:

```
5 SCREEN 2,1
```

Gekozen werd (zie SCREEN) voor een grafisch beeldscherm met een sprite-grootte van 8 bij 8 beeldpunten en vergrote weergave van deze sprite.
Vervolgens gaan we de sprite op het beeldscherm plaatsen om te kijken hoe de sprite er nu werkelijk uit ziet:

```
20 PUT SPRITE 0,(100,100),15,0
30 GOTO 30
RUN
```

Op regel 20 werd de spelfiguur op transparant 0 geplaatst en wel op locatie (100,100). Als kleur werd voor wit gekozen en, in dit geval ten overvloede, werd voor sprite nummer 0 (SPRITE$(0)) gekozen. Op het beeldscherm zien we een soort 'spookje', bruikbaar in diverse reactiespelletjes. Regel 30 is een eeuwigdurende loop om het grafische beeld vast te houden. Onderbreek met CONTROL-STOP.
Vervolgens laten we een tweede spookfiguurtje in een andere kleur achter dit eerste spookje langs bewegen:

```
30 FOR I=0 TO 200
40 PUT SPRITE 1,(I,104),10,0
50 NEXT I
60 GOTO 30
RUN
```

Snel zien we een tweede spookje (donkergeel) achter het eerste spookje langs gaan, zich iets lager bewegend. Om deze beweging goed volgbaar te maken, kunnen we eventueel de volgende wijziging intoetsen:

```
30 FOR I=0 TO 200 STEP .1
```

Een derde spookje kunnen we weer achter het tweede langs laten gaan:

```
45 PUT SPRITE 2,(104,I),8,0
RUN
```

Om een grote snelheid in de beweging te realiseren, kunnen we bijvoorbeeld de volgende verandering aanbrengen:

```
30 FOR I=0 TO 200 STEP 5
```

Steeds zien we dat in de for-next-lus op de transparanten nummer 1 en 2 de figuren op een andere plaats worden afgebeeld. De eerder op deze transparant geplaatste afbeelding verdwijnt hierbij; er kan maar één sprite tegelijk op een transparant worden afgebeeld.

Vervolgens kunnen we deze figuurtjes zich voor een bepaalde achter-

grond laten bewegen:

```
11 LET A=RND(-TIME)
12 FOR I=1 TO 20
13 CIRCLE (RND(1)*255,RND(1)*191),
RND(1)*80
14 NEXT I
15 FOR I=1 TO 10
16 PAINT (RND(1)*255,RND(1)*191)
17 NEXT I
RUN
```

De figuurtjes verplaatsen zich nu voor een achtergrond van willekeurige cirkels waarvan gedeelten al of niet zijn ingekleurd. Zie de behandeling van RND en TIME voor een optimaal begrip. Elke keer dat het programma wordt opgestart, is de achtergrond weer wat anders.

Door op transparant nummer 1 de y-coördinaat 208 te gebruiken, kunnen we het spookje op transparant nummer 2 bijvoorbeeld laten knipperen:

```
40 PUT SPRITE 1,(0,208+Q)
41 LET Q=NOT(Q)
```

Regel 41 zorgt ervoor dat variabele Q afwisselend de waarde 0 en –1 krijgt. Hierdoor wordt op regel 40 afwisselend een figuurtje wel en niet op de y-coördinaat 208 van transparant 1 geplaatst. Hierdoor wordt het figuurtje op transparant 2 afwisselen wél en niet zichtbaar waardoor een knipperend effect ontstaat.

Een spelfiguur kan men laten bewegen door bijvoorbeeld afwisselend andere, iets gewijzigde sprites op dezelfde locatie op eenzelfde transparant af te beelden. Het volgende voorbeeld laat een zich schuin over het beeld verplaatsend en pulserend cirkeltje zien:

```
NEW
10 SCREEN 2,1
20 SPRITE$(0)=CHR$(126)+STRING$(6,129)+
CHR$(126)
30 SPRITE$(1)=CHR$(0)+CHR$(60)+
```

```
STRING$(4,66)+CHR$(60)+CHR$(0)
40 SPRITE$(2)=STRING$(2,0)+CHR$(24)+
STRING$(2,36)+CHR$(24)+STRING$(2,0)
50 SPRITE$(3)=STRING$(3,0)+STRING$(2,24)
+STRING$(3,0)
60 FOR I=0 TO 3.9 STEP 0.05
70 PUT SPRITE 0,STEP(1,1),15,I
80 NEXT I
90 GOTO 60
RUN
```

Om het spelfiguur zich te laten verplaatsen, werd gebruik gemaakt van de STEP-mogelijkheid. Om het spelfiguur continu te laten veranderen, werd gebruik gemaakt van vier apart gedefinieerde sprites die achter elkaar op hoegenaamd dezelfde locatie worden geplaatst. Het gebruik van de STEP op regel 60 dient om de verandering van het spelfiguur niet te snel te laten geschieden; probeer de stepwaarde maar eens te veranderen.

Het voorbeeld laat een nog niet eerder genoemd verschijnsel zien; een sprite die aan de ene kant in verband met de locatie geheel of gedeeltelijk buiten de toegestane grenzen treedt, wordt aan de tegenovergestelde kant weer (geheel of gedeeltelijk) in beeld teruggebracht. Dit gebeurt door bij/van een buiten de toegestane grenzen tredende coördinaat net zo vaak de waarde 256 op te tellen/af te trekken totdat de betreffende coördinaat binnen de toegestane grenzen valt.
De programmaregels:

```
10 PUT SPRITE 9,(100,100),15,1
10 PUT SPRITE 9,(356,356),15,1
10 PUT SPRITE 9,(-156,-156),15,1
```

hebben dus allemaal precies hetzelfde effekt.

Tot slot volgt hier een programma dat de eerder in de behandeling van SPRITE$ ontworpen 16 bij 16 sprite gebruikt:

```
NEW
10 COLOR 2,1,1:KEY OFF:LET A=RND(-TIME)
20 RESTORE 100:LET A$="":FOR I=1 TO 32
```

```
30 READ A:LET A$=A$+CHR$(A):NEXT I
40 SCREEN 1,3:SPRITE$(0)=A$
50 FOR I=1 TO 31
60 PUT SPRITE I,(RND(1)*255,RND(1)*
255),RND(1)*15,0
70 PRINT "MSX-BASIC *** ";
80 NEXT I
90 GOTO 50
100 DATA 3,15,15,31,27,25,31,31,30,59,
124,124,63,31,3,1
110 DATA 224,240,248,248,184,152,248,
248,124,220,30,60,120
120 DATA 240,192,128
RUN
```

Op beeld wordt continu de tekst MSX-BASIC *** afgedrukt. Op de voorgrond worden op willekeurige posities in willekeurige kleuren, spookfiguurtjes voor- en achter elkaar afgebeeld. Uit dit programma kunnen we concluderen dat het gebruik van sprites is toegestaan bij SCREEN 1 (alfanumeriek), SCREEN 2 (grafisch) en SCREEN 3 (grafisch). Hierdoor is het op eenvoudige wijze mogelijk om sprites zich over een tekst te laten bewegen. Ook zien we dat diverse malen een sprite slechts gedeeltelijk wordt afgedrukt; er staan op dat moment méér dan vier sprites op dezelfde regel. Ook zien we een enkele keer dat er plotseling een aantal sprites geheel verdwijnen. In dat geval heeft één van de sprites gedurende een for-next-loop een y-coördinaat 208 aangenomen.

SLEUTELWOORD # READ

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **READ is lezen**

schrijfwijze

```
READ<VARIABELE>{,<VARIABELE>}
<VARIABELE>::=<ZIE ALGEMENE SPECIFICA-
  TIES>
```

betekenis

De sleutelwoorden READ, DATA en RESTORE worden onder het sleutelwoord DATA behandeld. Zie aldaar.

SLEUTELWOORD # REM

moeilijkheidsgraad . . . . . **zeer eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **REM is afkorting van remark – opmerking**

schrijfwijze

REM<COMMENTAAR>

<COMMENTAAR>::=[<KARAKTER>]
"

<KARAKTER>::=<ZIE ALGEMENE SPECIFICATIES>

betekenis

Het REM-kommando dient slechts ter opname van kommentaar in het programma. Het programma zal bij uitvoering de REM-kommando's overslaan.

Pas op: Het REM-kommando dient altijd als LAATSTE kommando in een programmaregel te worden opgenomen. Voorbeeld:

```
NEW
10 REM DIT PROGRAMMA DOET NIETS
20 STOP
RUN
Break in 20
Ok
```

Indien kommentaar NAAST kommando's dient te worden opgenomen, kan ook het enkele aanhalingsteken (') worden gebruikt. Ook kommentaar, opgenomen achter een ('), dient als laatste in een programmaregel te zijn opgenomen. Voorbeeld:

```
NEW
10 REM VOORBEELDPROGRAMMA
20 INPUT A:REM DE WAARDE VAN A WORDT
HIER INGEGEVEN
30 PRINT A'DE WAARDE VAN A WORDT HIER
```

```
AFGEDRUKT
RUN
? 12.5
 12.5
Ok
```

SLEUTELWOORD # RENUM

moeilijkheidsgraad ..... **eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **RENUM is afkorting van renumber – opnieuw voorzien van nummers**

schrijfwijze

```
RENUM[<NIEUW REGELNUMMER>][,[<OUD RE-
  GELNUMMER>][,[<STAPGROOTTE>]]]
<NIEUW REGELNUMMER>::=<TYPE 1 INTEGERE
  KONSTANTE>
<OUD REGELNUMMER>::=<TYPE 1 INTEGERE
  KONSTANTE>
<STAPGROOTTE>::=<TYPE 1 INTEGERE KON-
  STANTE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE AL-
  GEMENE SPECIFICATIES>
```

betekenis

Met RENUM kunnen programmaregels van een nieuw regelnummer worden voorzien. Dit kan bijvoorbeeld praktisch zijn wanneer u programmaregels wilt tussenvoegen maar er tussen de twee programmaregels niet meer voldoende plaats is.

Indien u slechts RENUM intoetst, dan wordt het volledige programma hernummerd. Het eerste regelnummer is regelnummer 10 terwijl de volgende regels steeds een regelnummer krijgen dat 10 hoger ligt.
Indien u RENUM ingeeft, gevolgd door een regelnummer dan heeft dat tot gevolg dat de eerste programmaregel dit nummer krijgt en dat alle volgende programmaregels steeds een nummer krijgen dat 10 hoger ligt.
Indien u RENUM ingeeft, gevolgd door een regelnummer, een komma en weer een regelnummer, dan heeft dat tot gevolg dat het programma vanaf het tweede ingegeven regelnummer wordt hernummerd. Het eerste gedeelte blijft dus onveranderd. Het eerste te hernummeren regelnummer krijgt het eerste ingegeven regelnummer. Indien het eerste regelnummer wordt overgeslagen (dus alleen een komma en het tweede regelnummer) dan krijgt het eerste te hernummeren regelnummer het

nummer 10 mee. In beide gevallen wordt er weer met stappen van 10 opgenummerd.
Indien u RENUM intoetst, gevolgd door een regelnummer, een komma, een regelnummer, een komma en een stapgrootte dan heeft dat tot effekt dat behalve dat er hernummerd wordt zoals zojuist beschreven daarbij de ingegeven stapgrootte wordt aangenomen in plaats van de normaal gehanteerde stapgrootte van 10. Indien een RENUM wordt gegeven met daarbij alleen de stapgrootte gedefinieerd (RENUM „ stapgrootte), dan wordt het programma geheel hernummerd waarbij het eerste regelnummer gelijk is aan 10. De volgende regels liggen steeds een stapgrootte hoger.

Voorbeeld:

```
NEW
10 REM
20 REM
30 REM
RENUM 1,,1
Ok
LIST
1 REM
2 REM
3 REM
Ok
RENUM
LIST
10 REM
20 REM
30 REM
Ok
RENUM 1000,20,5
Ok
LIST
10 REM
1000 REM
1005 REM
Ok
```

RENUM werkt ALLE regelnummers naar behoren bij. Zo ook bijvoorbeeld de regelnummers die in een GOTO of een GOSUB of een ON... GOTO kommando vermeld staan.

Indien RENUM in een programmaregel is opgenomen, dan probeert MSX-basic ten onrechte ook om de regelnummers en de stapgrootte die eventueel achter het RENUM-kommando zijn vermeld, te hernummeren. Het één en ander kan leiden tot foutmeldingen en curieuze situaties. Het gebruik van een RENUM in een programmaregel is echter meestal vrij zinloos.

Kommando's die door RENUM kunnen worden herzien:

```
RENUM
GOTO
GOSUB
ON...GOTO
ON...GOSUB
AUTO
LIST
LLIST
DELETE
RUN
RESUME
```

SLEUTELWOORD # RESTORE

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **RESTORE is terug zetten**

schrijfwijze

```
RESTORE[<REGELNUMMER>]
<REGELNUMMER>::=<TYPE 1 INTEGERE KON-
  STANTE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE AL-
  GEMENE SPECIFICATIES>
```

betekenis

De sleutelwoorden READ, DATA en RESTORE worden onder het sleutelwoord DATA behandeld. Zie aldaar.

SLEUTELWOORD # RESUME

moeilijkheidsgraad . . . . . **vrij eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **RESUME is hervatten**

schrijfwijze

```
RESUME[0|NEXT|<REGELNUMMER>]
<REGELNUMMER>::=<TYPE 1 INTEGERE VARIA-
  BELE>
<TYPE 1 INTEGERE VARIABELE>::=<ZIE ALGE-
  MENE SPECIFICATIES>
```

betekenis

Dit kommando wordt behandeld onder ON ERROR GOTO; zie aldaar.

SLEUTELWOORD # RETURN

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **RETURN is keer terug**

schrijfwijze

```
RETURN[<REGELNUMMER>]
<REGELNUMMER>::=<TYPE 1 INTEGERE KONSTAN-
  TE>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE ALGE-
  MENE SPECIFICATIES>
```

betekenis

Dit kommando wordt tesamen met GOSUB behandeld.
Zie aldaar.

SLEUTELWOORD # RIGHT$

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **A-FUNKTIE**
afkomst . . . . . **RIGHT$ is afkorting van RIGHT part of string – rechter gedeelte van de string**

schrijfwijze

```
RIGHT$(<BASISSTRING>,<AANTAL TEKENS>)
<BASISSTRING>::=<A>
<AANTAL TEKENS>::=<N>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft van een alfanumerieke uitdrukking het rechtergedeelte als resultaat. Hoe groot dit rechtergedeelte is, wordt bepaald door het opgegeven aantal tekens. Bijvoorbeeld:

```
PRINT RIGHT$("MSX-BASIC",5)
BASIC
Ok
```

Het aantal tekens mag niet kleiner zijn dan nul en niet groter dan 255. Een eventuele decimale fractie wordt verwaarloosd.

Nog een voorbeeld:

```
NEW
10 LINE INPUT "HOE HEET U ?:";NAAM$
20 INPUT "HOEVEEL LETTERS HEEFT UW
ACHTERNAAM";AANTAL
30 PRINT "HALLO MENEER/MEVROUW ";RIGHT$
(NAAM$,AANTAL);"!"
RUN
HOE HEET U ?:JOHAN KARDINAAL
HOEVEEL LETTERS HEEFT UW ACHTERNAAM ? 9
HALLO MENEER/MEVROUW KARDINAAL!
Ok
```

SLEUTELWOORD RND

moeilijkheidsgraad . . . . . **vrij eenvoudig**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **RND is afkorting van random – willekeurig**

schrijfwijze

```
RND(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat een toevalsgetal. Dit getal is minimaal gelijk aan nul maar altijd kleiner dan 1.
De tussen haakjes opgenomen numerieke uitdrukking heeft de volgende betekenis:

– kleiner dan nul:	het eerste toevalsgetal uit een door de numerieke uitdrukking bepaalde reeks wordt berekend en als resultaat gegeven. Voortzetting van de reeks toevalsgetallen dient te geschieden door bij elke volgende RND een uitdrukking met positieve waarde te gebruiken
– groter dan nul:	het volgende toevalsgetal uit een reeds aangevangen reeks wordt gepresenteerd. Indien de reeks niet eerder werd aangevangen, wordt het getal 0.595219433994623 als eerste uit de reeks gegenereerd
– gelijk aan nul:	het laatste toevalsgetal wordt herhaald. Indien nog niet eerder een toevalsgetal werd bepaald, wordt het getal 0.40649651372358 gegenereerd

De term toevalsgetal is niet correct. De toevalsgetallen worden door de computer BEREKEND waarbij een zo goed mogelijke verdeling van de getallen wordt betracht. Deze berekening is echter zo complex dat een volgend getal uit een reeks niet voorspelbaar is tenzij de reeks reeds eerder werd bestudeerd.

Een voorbeeld: een speltoepassing:

```
NEW
10 REM DOBBELSTENEN
20 INPUT "GEEF EEN WAARDE IN ";A
25 OGEN=RND(-A)
30 PRINT "GEEF EEN TOETS IN"
40 K$=INKEY$:IF K$="" THEN 40
50 OGEN=INT(6*RND(1)+1)
60 PRINT "IK GOOIDE EEN";OGEN
70 GOTO 30
RUN
GEEF EEN WAARDE IN ? 11
GEEF EEN TOETS IN
IK GOOIDE EEN 5
GEEF EEN TOETS IN
IK GOOIDE EEN 5
GEEF EEN TOETS IN
IK GOOIDE EEN 3
```

(etcetera; onderbreken met CONTROL-STOP)

Op regel 25 wordt naar aanleiding van een ingegeven getal een reeks toevalsgetallen aangevangen. Op regel 50 wordt dan steeds de volgende uit de reeks bepaald. Met dit toevalsgetal wordt een berekening uitgevoerd, zodanig dat het eindresultaat een willekeurig geheel getal is, gelijk aan 1, 2, 3, 4, 5, of 6.

SLEUTELWOORD **RUN**

moeilijkheidsgraad ..... **zeer eenvoudig**
soort ..... **KOMMANDO**
afkomst ..... **RUN is rennen, lopen, werken**

schrijfwijze

RUN

```
[<REGELNUMMER>[{<KARAKTER>}\<CIJFER><...>]
          "<PROGRAMMANAAM>"[,R]]
```

```
<REGELNUMMER>::=<TYPE 1 INTEGERE KONSTANTE>
<PROGRAMMANAAM>::=<A>
<TYPE 1 INTEGERE KONSTANTE>::=<ZIE ALGE-
  MENE SPECIFICATIES>
<KARAKTER>::=<ZIE ALGEMENE SPECIFICATIES>
<CIJFER>::=<ZIE ALGEMENE SPECIFICATIES>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het RUN-kommando is het meest eenvoudige maar ook het meest belangrijke kommando dat MSX-basic kent. Met dit kommando kunnen we een in het werkgeheugen aanwezig MSX-basic programma activeren.

eenvoudig gebruik

In de meest eenvoudige vorm activeren we met RUN een programma:

```
NEW
10 PRINT "TEST 1"
20 PRINT "TEST 2"
RUN
TEST 1
TEST 2
Ok
```

Indien we achter het RUN-kommando een regelnummer opnemen, zal het programma vanaf dit regelnummer worden geactiveerd. Voorbeeld:

```
NEW
10 PRINT "TEST 1"
20 PRINT "TEST 2"
RUN 20
TEST 2
Ok
```

RUN sluit alle eventueel openstaande kanalen en maakt daarbij alle variabelen schoon. Merk op dat RUN standaard onder funktietoets 5 en 10 aanwezig is in twee verschillende uitvoeringen.

gevorderd gebruik

In verband met het gevorderde gebruik dient eerst het _SAVE-kommando en het OPEN-kommando te worden bestudeerd.
Een speciale vorm van het RUN-kommando is de RUN met daarachter een programmanaam. In deze vorm gebruikt, wordt na het RUN-kommando het programma eerst opgehaald van het aangesproken randapparaat (b.v. cassetterecorder) en pas dan geactiveerd. Met de toevoeging ,R bereiken we dat de eventueel geopende kanalen niet automatisch door het RUN-kommando worden gesloten. Bijvoorbeeld:

```
RUN "TEST"

RUN "CAS:TEST",R
```

– In de cassetterecorder dient een cassette met daarop het programma TEST te worden geplaatst en de cassetterecorder dient op afspelen te worden gezet.
– Het programma TEST wordt op de cassetteband opgezocht of de kanalen die eventueel waren geopend, worden nu niet gesloten.
– In beide gevallen wordt programma "TEST" van cassetteband gelezen en meteen opgestart.

Zie voor de toegestane programmanamen de voorwaarde voor bestandsnamen bij het OPEN-kommando.

SLEUTELWOORD # SAVE

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **SAVE is veilig stellen**

schrijfwijze

```
SAVE<BESTANDSNAAM>[,A]
<BESTANDSNAAM>::=<A>
<A>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het is noodzakelijk dat eerst de behandelingen van CLOAD en CSAVE worden doorgenomen.

Met het SAVE-kommando kunnen programma's op een extern opslagmedium worden opgeslagen. Dit vastleggen gebeurt ongekodeerd; vastleggen geschiedt tekstueel, teken voor teken, zoals het programma ook werd ingegeven (volgens de ASCII-kodering). Hierdoor is een bestand dat ontstaat door middel van een SAVE ook later weer met een OPEN-kommando (zie de betreffende behandeling) te benaderen en regel voor regel in te lezen.

Een programma, via SAVE vastgelegd, kan later via RUN direkt weer worden opgestart.

De achtervoeging *,A* is toegestaan maar heeft geen zin.

De toegestane bestandsnamen worden bij behandeling van het OPEN-kommando nader omschreven. We geven een voorbeeld op cassetterecorder.

Voorbeeld: het SAVEn van een programma op cassetteband

```
NEW
10 REM TESTPROGRAMMA
20 PRINT "DIT IS EEN TEST"
30 GOTO 20
SAVE "CAS:TEST"          (CAS: staat voor cassetterecorder;
Ok                             TEST is de programmanaam)
```

Voordat het SAVE-kommando wordt ingetoetst, dient een cassetteband in de recorder te worden geplaatst en dient deze recorder op opnemen te worden gezet. Na enige tijd verschijnt de Ok-melding; het programma is vastgelegd op band.

MSX schrijft geen lees/schrijfcontrole voor; een slechte kwaliteit apparatuur of bandmateriaal kan veroorzaken dat er tijdens het vastleggen fouten optreden.

Een in een programmaregel opgenomen SAVE-kommando heeft tot gevolg dat het programma eerst wordt vastgelegd en dat daarna de Ok-melding verschijnt; het programma gaat niet verder.

SLEUTELWOORD # SCREEN

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **KOMMANDO**
afkomst . . . . . **SCREEN is beeldscherm**

schrijfwijze

```
SCREEN[<INSTELLING>][,[<SPRITEGROOTTE>][,
  [<KLIK>][,[<CASSETTE-SCHRIJFSNELHEID>]
  [,<PRINTERTYPE>]]]]\SCREEN[<...>,{,}]
<INSTELLING>::=<N>
<SPRITEGROOTTE>::=<N>
<KLIK>::=<N>
<CASSETTE-SCHRIJFSNELHEID>::=<N>
<PRINTERTYPE>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met dit kommando kunnen diverse instellingen van de computer worden bepaald, hoofdzakelijk in verband met het beeldscherm.

Alle numerieke uitdrukkingen die binnen het SCREEN-kommando worden gebruikt, worden indien zij gebroken waarden hebben, herleid naar het grootste gehele getal kleiner dan deze gebroken waarde.

Instelling: de waarde van de numerieke uitdrukking die de instelling bepaalt, mag gelijk zijn aan 0, 1, 2 of 3. Deze waarden hebben de volgende betekenissen:

SCREEN 0 een alfanumeriek scherm van maximaal 40 karakters bij 24 regels wordt ingericht. Deze scherminrichting staat geen grafisch gebruik toe

SCREEN 1 wederom wordt een alfanumeriek scherm ingericht. Ditmaal zijn de afmetingen maximaal 32 bij 24 regels. Ook deze beeldscherminrichting staat geen grafisch gebruik toe

SCREEN 2 een grafisch scherm wordt ingericht met een hoog oplossend vermogen (er kan gedetailleerd op worden getekend). Het kleurgebruik is enigszins aan beperkin-

gen onderhevig. Deze beeldscherminrichting staat geen alfanumeriek gebruik toe

SCREEN 3 een grafisch scherm met een laag oplossend vermogen (er kunnen alleen vrij grove tekeningen op worden gemaakt) en een onbeperkte mogelijkheid tot gebruik van 15 kleuren wordt ingericht. Ook deze beeldschermstelling staat geen alfanumeriek gebruik toe.

Er zijn twee verschillende grafische inrichtingen van het beeldscherm mogelijk, namelijk een inrichting met een hoog oplossend vermogen en een inrichting met een laag oplossend vermogen. Onderstaande tabel geeft een beeld van het oplossende vermogen en de bijbehorende kleurmogelijkheden:

	beeldschermindeling	
	horizontaal	vertikaal
SCREEN 2	256 puntjes	192 puntjes
SCREEN 3	64 puntjes	48 puntjes

Sprite-grootte: de waarde van de numerieke uitdrukking die de sprite-grootte bepaalt, mag gelijk zijn aan 0, 1, 2 of 3. Deze waarden hebben de volgende betekenissen:

SCREEN ,0 de sprite-grootte is 8 bij 8 stippen
SCREEN ,1 de sprite-grootte is 8 bij 8 stippen; sprites worden vergroot weergegeven
SCREEN ,2 de sprite-grootte is 16 bij 16 stippen
SCREEN ,3 de sprite-grootte is 16 bij 16 stippen; sprites worden vergroot weergegeven.

De zin van deze inrichting van de sprite-grootte wordt pas duidelijk bij de behandeling van de SPRITE-sleutelwoorden.

Klik: de waarde van deze numerieke uitdrukking is minimaal 0 en maximaal 255. Deze waarde heeft de volgende betekenis:

SCREEN ,,0 de klik die normaal bij de toetsaanslag hoorbaar is, verdwijnt
SCREEN ,,1 bij invulling van een 1 of elke andere toegestane waarde groter dan nul is het effect dat de klik weer wordt geactiveerd.

Cassette schrijfsnelheid: de waarde van deze numerieke uitdrukking mag gelijk zijn aan 1 of 2 en heeft de volgende betekenis:

`SCREEN ,,,1` het schrijven van cassetteband gebeurt met een snelheid van 1200 baud (ongeveer 150 tekens per seconde)

`SCREEN ,,,2` het schrijven van de cassetteband gebeurt met een snelheid van 2400 baud (ongeveer 300 tekens per seconde). Deze instelling mag uitsluitend worden gebruikt indien zowel de cassetterecorder als het cassettebandje van uitmuntende kwaliteit zijn. In een ander geval is de kans op optreden van lees- en schrijffouten erg groot.

Printer type: de waarde van deze numerieke uitdrukking is minimaal gelijk aan nul en maximaal gelijk aan 255. Deze waarde heeft de volgende betekenis:

`SCREEN ,,,,0` een printer met mogelijkheid tot het afdrukken van de MSX grafische symbolen is aangesloten

`SCREEN ,,,,1` elke toegestane waarde groter dan nul geeft aan dat een printer is aangesloten zonder de mogelijkheid tot het afdrukken van de MSX grafische symbolen. In voorkomende gevallen worden in plaats van de symbolen dan spaties afgedrukt.

enkele voorbeelden:

```
SCREEN 3,0,1,2,1
```

Gekozen werd voor een grafische scherminrichting met laag oplossend vermogen en maximaal kleurgebruik. De sprite-grootte werd bepaald op 8 bij 8 stippen onvergroot, de klik bij toetsinslag werd aangezet, de cassetteschrijfsnelheid werd maximaal ingesteld en de computer werd medegedeeld dat de aangesloten printer niet in staat is om MSX grafische tekens te reproduceren.

```
SCREEN ,1,,1
```

De sprite-groote werd veranderd naar 8 bij 8 stippen vergroot terwijl de cassette-schrijfsnelheid naar 1200 baud werd teruggebracht. Voor het overige werden geen wijzigingen aan de instelling doorgevoerd.

Merk op dat wanneer een programma of een direkt kommando is afgerond en dat de MSX-computer weer de Ok-melding moet gaan afdrukken, of wanneer een INPUT vanaf het toetsenbord moet worden uitgevoerd of een foutmelding moet worden gegeven, automatisch een eventueel actieve grafische schermindeling wordt gedeactiveerd en dat de laatste geactiveerde alfanumerieke scherminstelling weer wordt geactiveerd. Deze actie is logisch; om in MSX-basic te kunnen programmeren of om een ingave te kunnen plegen is natuurlijk een alfanumerieke schermindeling noodzakelijk.

Pas bij behandeling van de diverse grafische mogelijkheden komt de zin van het SCREEN-sleutelwoord volledig tot uiting.

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **SGN is afkorting van sign – teken**

schrijfwijze

```
SGN(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie kan drie resultaten geven:

de waarde 1	wanneer de waarde van de tussen haakjes vermelde numerieke uitdrukking positief (groter dan nul) is
de waarde 0	wanneer de waarde van de tussen haakjes vermelde numerieke uitdrukking gelijk is aan nul
de waarde –1	wanneer de waarde van de tussen haakjes vermelde numerieke uitdrukking negatief (kleiner dan nul) is.

Voorbeeld:

```
NEW
10 LET A=-123.4:LET B=0:LET C=1E33
20 PRINT SGN(A);SGN(B);SGN(C)
RUN
-1  0  1
Ok
```

moeilijkheidsgraad . . **vrij moeilijk, kennis van goniometrie is noodzakelijk**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **SIN is afkorting van sine – sinus**

schrijfwijze

```
SIN(<HOEK>)
<HOEK>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de sinus van de waarde van de tussen haakjes vermelde uitdrukking. Deze waarde wordt beschouwd als de uitdrukking van een hoek in radialen. Het resultaat ligt uiteraard altijd tussen –1 en 1.

Alhoewel elke waarde voor de hoek is toegestaan, is het in verband met de precisie van berekenen raadzaam om niet te grote hoekmaten te gebruiken.

Voorbeeld:

```
NEW
10 FOR I=0 TO 90
20 LET HOEK=I/180*3.1415926535
30 PRINT I;":";SIN(HOEK)
40 NEXT I
RUN
```

(een sinustabel verschijnt op het beeldscherm. Merk op dan op regel 20 de hoek van graden naar radialen wordt geconverteerd)

SLEUTELWOORD # SOUND

moeilijkheidsgraad ..... **moeilijk, kennis van de betekenis van de register van de geluidscontrole chip is vereist**
soort ..... **KOMMANDO**
afkomst ..... **SOUND is geluid**

schrijfwijze

```
SOUND<REGISTERNUMMER>,<REGISTERINHOUD>
<REGISTERNUMMER>::=<N>
<REGISTERINHOUD>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met het SOUND-kommado kunnen de registers van de PSG (Programmable Sound Generator) worden ingevuld. Hiertoe dient achter het SOUND sleutelwoord eerst het registernummer te worden gespecificeerd, gevolgd door een komma en de waarde die het betreffende register dient aan te nemen.

Het registernummer mag niet kleiner zijn dan 0 en niet groter dan 13. Een eventuele decimale fractie wordt verwaarloosd.
De registerinhoud mag niet kleiner zijn dan 0 en niet groter dan 255. Ook hier wordt een eventuele decimale fractie verwaarloosd.

Door de registers van de PSG te besturen, kunnen diverse effecten worden bereikt. Bijvoorbeeld de ruisgenerator dient op deze wijze te worden bestuurd. Een voorbeeld:

```
NEW
10 SOUND 6,&B1101
20 SOUND 7,&B110111
30 SOUND 8,&B1111
RUN
Ok
```

Een sterke ruis is hoorbaar. Deze kan met CONROL-STOP worden afgezet.

In het bovenstaande voorbeeld werden registers 6, 7 en 8 van de PSG van bepaalden waarden voorzien, in dit geval opgegeven als binaire konstanten.

Zie voor de betekenis van de betreffende registers hoofdstuk 15.

moeilijkheidsgraad . . . . . zeer eenvoudig
soort . . . . . A-FUNKTIE
afkomst . . . . . **SPACE$ is afkorting van space string – spatie string**

schrijfwijze

```
SPACE$(<AANTAL SPATIES>)
<AANTAL SPATIES>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat een aantal spaties. Het aantal wordt bepaald door de numerieke uitdrukking, opgenomen tussen de haakjes. Indien de uitkomst van deze numerieke bewerking een gebroken getal is, wordt de grootste waarde, kleiner dan dat getal als geldige waarde genomen.

Voorbeeld:

```
NEW
10 FOR I=0 TO 9
20 PRINT SPACE$(I);"MSX"
30 NEXT I:STOP
RUN
MSX
 MSX
  MSX
   MSX
    MSX
     MSX
      MSX
       MSX
        MSX
         MSX
Break in 30
Ok
```

SLEUTELWOORD # SPRITE

moeilijkheidsgraad . . . . . **moeilijk**
soort . . . . . **KOMMANDO**
afkomst . . . . . **SPRITE is 'geest'**

schrijfwijze

```
        ON
SPRITE OFF
        STOP
```

betekenis

Voorafgaand aan de behandeling van dit kommando dienen de ON KEY GOSUB en de PUT SPRITE grondig te zijn doorgenomen.

MSX-basic biedt de mogelijkheid door een botsing van sprites het programma tijdelijk onderbreken. Met SPRITE ON geven we aan dat het programma bij een botsing van sprites tijdelijk dient te worden onderbroken. Met SPRITE OFF geven we aan dat botsingen van sprites niet meer behoeven te leiden tot een tijdelijke onderbreking van het programma. Met SPRITE STOP geven we aan dat een botsing van sprites wel door de computer dient te worden 'onthouden' maar dat het programma niet direkt mag worden onderbroken. Daadwerkelijke onderbreking vindt dan pas plaats bij een SPRITE ON kommando.

Voor een zinvol voorbeeld: zie het ON SPRITE GOSUB kommando.

moeilijkheidsgraad . . . . . **moeilijk**
soort . . . . . **SYSTEEMVARIABELE**
afkomst . . . . . **SPRITE is een afkorting van sprite string. Sprite is 'geest'**

schrijfwijze

```
SPRITE$(<SPRITE NUMMER>)
<SPRITE NUMMER>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het is noodzakelijk dat het SCREEN-kommando reeds grondig is bestudeerd.

Het MSX-basic voorziet in vele kommando's die het schrijven van spelprogramma's in BASIC bijzonder goed ondersteunen. Gebruik van de sleutelwoorden SPRITE, SPRITE$, PUT SPRITE en ON SPRITE GOSUB maken het mogelijk om:

- 32 verschillende vaste figuren (sprites) onafhankelijk van elkaar achter elkaar langs op het beeldscherm zich te laten verplaatsen;
- deze sprites of spelfiguren zelf te ontwerpen;
- een botsing van twee of meer sprites te detecteren en hierdoor (zoals bijvoorbeeld ook bij ON KEY GOSUB) het programma tijdelijk te laten onderbreken.

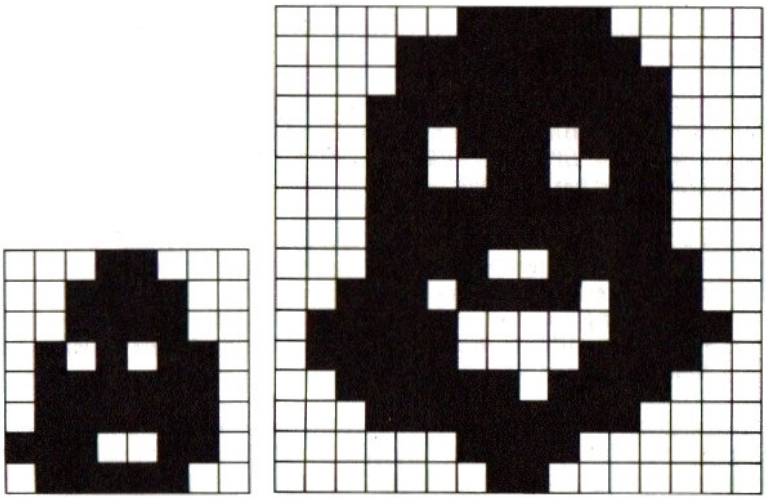
Spelprogramma's met bewegende beelden (packman-achtige spelletjes, aanvals- en schietspelletjes en dergelijke) kunnen in MSX-basic voor het eerst worden geprogrammeerd zonder dat kennis van machinetaal of assembler nodig is en met behoud van de benodigde snelheid. Daarbij zijn uiteraard ook wat serieuzere toepassingen bijzonder goed denkbaar.

Met SPRITE$ kunnen maximaal 63 (indien de sprite-grootte in het SCREEN-kommando op 2 of 3 is gezet) of 255 (indien de sprite-grootte in het SCREEN-kommando op 0 of 1 is gezet) spelfiguren onafhankelijk van elkaar worden ontworpen. Het ontwerpen van een sprite kan als volgt geschieden:

stap 1 beslis welke sprite-grootte gaat worden gebruikt. Is deze 8 bij 8 beeldpunten (sprite-grootte 0 of 1 in het SCREEN-kommando) of is deze 16 bij 16 beeldpunten (spritegrootte 2 of 3 in het SCREEN-kommando)

stap 2 maak een ontwerpblad. Dit kan het beste worden gedaan door een stuk ruitjespapier te nemen en daarop een vierkant van 8 bij 8 ruitjes of 16 bij 16 ruitjes af te bakenen, afhankelijk van de gewenste sprite-grootte.

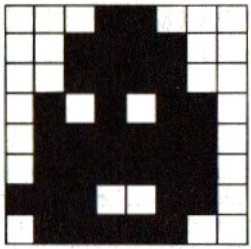
stap 3 ontwerp binnen dit vierkant het gewenste figuur door ruitjes in te kleuren of wit te laten. Een voorbeeld 8 bij 8 en 16 bij 16:



8x8 sprite

16x16 sprite

stap 4 digitaliseer de sprite. Voor een 8 bij 8 sprite is dat het gemakkelijkst. De ingekleurde ruitjes dienen voorgesteld te worden met het cijfer 1; de witte ruitjes met het cijfer 0. Zo krijgen we acht getallen van acht cijfers, allemaal nullen of enen. Voor deze acht cijfers zetten we "&B" waardoor we plotseling acht binaire konstanten hebben verkregen. Bijvoorbeeld:



8x8 sprite

```
&B 0 0 0 1 1 0 0 0
&B 0 0 1 1 1 1 0 0
&B 0 0 1 1 1 1 0 0
&B 0 1 0 1 0 1 1 0
&B 0 1 1 1 1 1 1 0
&B 0 1 1 1 1 1 1 0
&B 1 1 1 0 0 1 1 0
&B 0 1 1 1 1 1 0 0
```

digitalisatie

Voor een 16 bij 16 sprite is dat wat moeilijker. Verdeel hiertoe eerst het 16 bij 16 vierkant in vier 8 bij 8 vierkanten. Deze kunnen op de manier van de 8 bij 8 sprites weer worden gedigitaliseerd (in cijfers omgezet). Doe dit in de volgorde:

1: 8 bij 8 vierkant linksboven
2: 8 bij 8 vierkant linksonder
3: 8 bij 8 vierkant rechtsboven
4: 8 bij 8 vierkant rechtsonder

Op deze wijze worden 32 binaire kontanten verkregen.

stap 5 we kunnen 63 of 255 sprites definieren, afhankelijk van de gekozen sprite-grootte. Beslis welk sprite-nummer de door ons ontworpen sprite krijgt. De numerieke uitdrukking die als sprite-nummer wordt gebruikt, mag in waarde niet kleiner zijn dan nul en niet groter dan 63 of 255, afhankelijk van de gekozen sprite-grootte. Een eventuele decimale fractie wordt verwaarloosd. In ons voorbeeld zullen we sprite nummer 6 gaan definiëren.

stap 6 leg de sprite vast door SPRITE$(...) gelijk te stellen aan CHR$ (eerste binaire konstante)+CHR$(tweede binaire konstante)+ CHR$(derde...) etcetera totdat alle konstanten zijn opgebruikt. Bijvoorbeeld:

```
NEW
10 SPRITE$(6)=CHR$(&BOOO11000)+
CHR$(&BOO111100)+CHR$(BOO111100)+
```

```
CHR$(&B01010110)+CHR$(&B01111110)+
CHR$(&B01111110)+CHR$(&B11100110)+
CHR$(&B01111100)
```

Voor een 16 bij 16 sprite kan dit niet daar de programmaregel dan onvermijdelijk groter wordt dan 255 tekens, hetgeen verboden is in MSX-basic. Om dit te voorkomen, dienen we een methode te gebruiken die kortere programmaregels toestaat. Omdat deze methode veel minder ruimte inneemt dan de eerste methode, is deze ook in tweede instantie voor de 8 bij 8 sprites aan te raden.
Allereerst dienen we alle binaire konstanten te converteren naar decimale konstanten. Dit laten we de computer doen en wel op de volgende manier:
Voor elke binaire konstante tikken we:

```
PRINT &B...
```
(de juiste binaire konstante invullen)

De computer antwoordt onmiddellijk met de decimale waarde van deze konstante die nooit meer dan drie cijfers groot is. Bijvoorbeeld:

```
PRINT &B11111111
  255
Ok
```

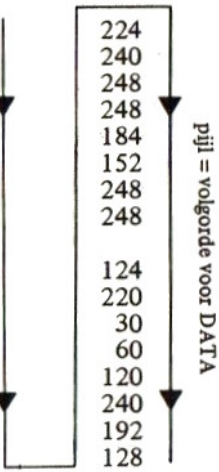
Voor elke binaire konstante laten we de computer de decimale waarde uitrekenen die we dan stuk voor stuk opschrijven. Uiteindelijk zetten we deze getallen in één of meer DATA-kommando's onder elkaar. Bijvoorbeeld (zie de eerder ontworpen 16 bij 16 sprite). (Zie vervolg tekst de volgende pagina.)

N.B.: indien met SPRITE$ een sprite wordt gedefinieerd, dient eerst een grafisch scherm te zijn geactiveerd en dient de sprite-grootte te zijn bepaald. Indien dit niet gebeurt, volgt een foutmelding of worden (bij omschakelen van alleen de sprite-grootte) de sprites weer gewist.

N.B.: Het sleutelwoord LET mag niet voor SPRITE$ worden gebruikt.

Zie het PUT SPRITE kommando voor gebruik van ontworpen sprites.

00000011	11100000	3	224
00001111	11110000	15	240
00001111	11111000	15	248
00011111	11111000	31	248
00011011	10111000	27	184
00011001	10011000	25	152
00011111	11111000	31	248
00011111	11111000	31	248
00011110	01111100	30	124
00111011	11011100	59	220
01111100	00011110	124	30
01111100	00111100	124	60
00111111	01111000	63	120
00011111	11110000	31	240
00000011	11000000	3	192
00000001	10000000	1	128



digitalisering van de 16 x 16 sprite (zie tekening)

de corresponderende decimale waarden

Uiteindelijk resultaat: de DATA-regel(s):

```
NEW
100 DATA 3,15,15,31,27,25,31,31,30,
59,124,124,63,31,3,1
110 DATA 224,240,248,248,184,152,
248,248,124,220,3,120
120 DATA 240,192,128
```

Vervolgens zetten we de sprite eerst in een gewone alfanumerieke variabele op een wijze zoals deze:

```
10 RESTORE 100:LET A$="":FOR I=1 TO 32
20 READ A:LET A$=A$+CHR$(A):NEXT I
```

Als laatste actie bepalen we de 16 bij 16 sprite definitief, bijvoorbeeld door het kommando:

```
30 SCREEN 2,2:SPRITE$(6)=A$
```

moeilijkheidsgraad . . **normaal, kennis van vierkantswortels is vereist**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **SQR is afkorting van square root – vierkantswortel**

schrijfwijze

```
SQR(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de vierkantswortel van de waarde van de uitdrukking die tussen haakjes staat. Uiteraard mag deze waarde niet negatief zijn.

Voorbeeld:

```
NEW
10 INPUT "GETAL ";A
20 PRINT "DE VIERKANTSWORTEL"
30 PRINT "IS";SQR(A):GOTO 10
RUN
GETAL ? 3
DE VIERKANTSWORTEL
IS 1.73205080705688
GETAL ? 9
DE VIERKANTSWORTEL
IS 3
GETAL ? 1024
DE VIERKANTSWORTEL
IS 32
GETAL ? -1
DE VIERKANTSWORTEL
IS
Illegal function call in 30
Ok
```

SLEUTELWOORD # STICK

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **STICK is stok**

schrijfwijze

STICK(<JOY-STICKNUMMER>)
<JOY-STICKNUMMER>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>

betekenis

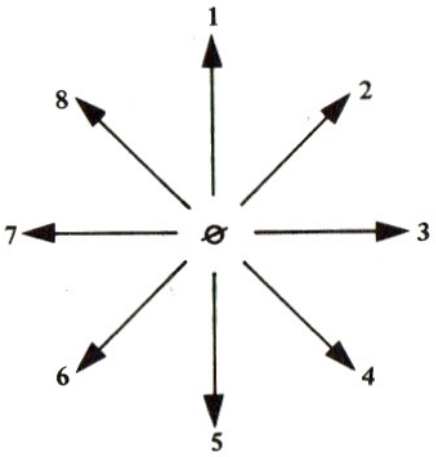
Met deze funktie kan de ingestelde richting van een joy-stick worden opgevraagd.

De numerieke uitdrukking tussen haakjes dient gelijk te zijn aan 0, 1 of 2. Indien de tussen haakjes vermelde waarde een gebroken waarde is, wordt deze herleid naar de hoogste gehele waarde, kleiner dan deze gebroken waarde.

De waarde tussen haakjes heeft de volgende betekenis:

STICK(0)	de pijltoetsen worden als joy-stick beschouwd
STICK(1)	de eerste joy-stick wordt beschouwd
STICK(2)	de tweede joy-stick wordt beschouwd

Deze funktie kan diverse waarden als resultaat hebben, die allemaal een bepaalde richting aangeven. Deze waarden worden in het volgende schema gesymboliseerd:



Voorbeeld:

```
NEW
10 LET A=STICK(0):IF A=0 THEN 10
20 RESTORE:FOR I=1 TO A
30 READ A$:NEXT I:PRINT A$
40 GOTO 10
100 DATA "NOORD","NOORD-OOST"
110 DATA "OOST","ZUID-OOST"
120 DATA "ZUID","ZUID-WEST"
130 DATA "WEST","NOORD-WEST"
RUN
```

(aan de hand van de pijltoetsen vermeldt de computer de aangegeven windrichting. Twee pijlen mogen tegelijk worden ingedrukt om een gecombineerde beweging te verkrijgen. Vul indien u een joy-stick aangesloten heeft, ook eens een andere waarde in de STICK-funktie in.)

moeilijkheidsgraad . . **normaal, in de eerste betekenis zeer eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **STOP is stoppen, stilstaan**

schrijfwijze

betekenis

eenvoudig gebruik

Met het STOP-kommando kunnen we het programma (tussentijds) beeindigen. Vooral bij het uittesten van een wat groter programma wordt vaak een stop ingelast om tussenresultaten te kunnen bekijken. Voorbeeld:

```
NEW
10 PRINT "REGEL 1"
20 STOP
30 PRINT "REGEL 2"
RUN
REGEL 1
Break in 20
Ok
```

gevorderd gebruik

Met het STOP-kommando kunnen we de afvanging van de CONTROL-STOP-toetsingave regelen.

STOP ON zet de afvanging aan.
STOP OFF zet de afvanging uit.
STOP STOP zet de afvanging stil; indien een CONTROL-STOP wordt ingedrukt, dan wordt dit door de computer slechts genoteerd. Pas wanneer een STOP ON wordt uitgevoerd, wordt de betreffende aktie

ondernomen.

Dit gebruik van het STOP-kommando is alleen zinvol in samenwerking met de ON STOP GOSUB. Zie voor een zinvol voorbeeld de behandeling van dit kommando.

De STOP ON, STOP OFF en STOP STOP hebben in het geheel geen funktie indien in het programma geen ON STOP GOSUB aanwezig is. STOP STOP heeft geen funktie wanneer niet eerst een STOP ON word uitgevoerd.

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **A-FUNKTIE**
afkomst . . . . . **STR$ is afkomstig van (convert to) string – zet over naar een stringwaarde**

schrijfwijze

```
STR$(<N>)
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft de tegenovergestelde mogelijkheid van de VAL-funktie. Vanuit een numerieke uitdrukking kan een string worden samengesteld. Bijvoorbeeld:

```
NEW
10 LET A=12.3
20 LET A$=STR$(A)
30 PRINT A$;LEN(A$)
RUN
 12.3 5
Ok
```

Uit dit voorbeeld blijkt dat A$ een voorlopende spatie heeft. STR$ geeft aan positieve waarden altijd een voorloopspatie. Bij negatieve waarden wordt deze spatie door een min-teken vervangen. Bijvoorbeeld:

```
NEW
10 LET A=-12
20 PRINT LEN(STR$(A));"FL. "+STR$(A)
30 STOP
RUN
 3 FL. -12
Break in 30
Ok
```

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **STRIG is samentrekking van shot en trigger (schot en trekker)**

schrijfwijze

```
STRIG(<JOY-STICKNUMMER>)
<JOY-STICKNUMMER>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met deze funktie kan worden getest of de vuurknop van een joy-stick al dan niet is ingedrukt.

De numerieke uitdrukking tussen haakjes dient gelijk te zijn aan 0, 1, 2, 3 of 4. Indien de tussen haakjes vermelde waarde gebroken is, wordt deze herleid naar de hoogste gehele waarde kleiner dan deze gebroken waarde.

De waarde tussen haakjes heeft de volgende betekenis:

STRIG(0)	de spatiebalk wordt als vuurknop beschouwd
STRIG(1) of STRIG(3)	de eerste of tweede vuurknop van de eerste joy-stick wordt beschouwd
STRIG(2) of STRIG(4)	de eerste of tweede vuurknop van de tweede joy-stick wordt beschouwd

Indien de funktie de waarde 0 tot gevolg heeft, betekent dat dat de vuurknop van de betreffende joy-stick niet wordt ingedrukt. Indien deze funktie echter de waarde –1 geeft, betekent dat dat de vuurknop van de betreffende joy-stick WEL wordt ingedrukt.

Een voorbeeld met de spatiebalk:

```
NEW
10 IF STRIG(0) THEN PRINT "PANG!!!":
GOTO 10 ELSE GOTO 10
```

```
RUN
PANG!!!
PANG!!!
```

Telkens wanneer de spatiebalk wordt ingetoetst, verschijnt de tekst PANG. Onderbreek dit programma met CONTROL-STOP.
Indien u joy-sticks heeft aangesloten, verander dan de waarde tussen haakjes eens en probeer de echte vuurknoppen.

moeilijkheidsgraad . . . . . **vrij moeilijk**
soort . . . . . **KOMMANDO**
afkomst . . . . . **STRIG is samentrekking van shot en trigger (schot en trekker)**

schrijfwijze

```
                                    ON
STRIG(<JOY-STICKNUMMER>)           OFF
                                    STOP
<JOY-STICKNUMMER>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het is noodzakelijk dat voorafgaand aan deze behandeling de behandeling van ON KEY GOSUB grondig is doorgenomen.

De vuurknoppen van de joy-sticks, alsmede de spatiebalk kunnen als funktietoetsen worden geactiveerd. Met behulp van deze funktietoetsen kan net zoals met de gebruikelijke funktietoetsen een programma tijdelijk worden onderbroken.

STRIG...ON, STRIG...STOP en STRIG...OFF hebben met betrekking tot de bijbehorende funktietoetsen dezelfde werking als KEY...ON, KEY...STOP en KEY...OFF. De numerieke uitdrukking bepaalt welke funktietoets wordt geactiveerd:

0 spatiebalk
1 1e vuurknop van joy-stick 1
2 1e vuurknop van joy-stick 2
3 2e vuurknop van joy-stick 1
4 2e vuurknop van joy-stick 2

De waarde van de numerieke uitdrukking tussen haakjes mag dientengevolge niet kleiner zijn dan nul en niet groter dan 4 terwijl een decimale fractie wordt verwaarloosd.

Een STRIG...STOP heeft geen zin indien deze niet werd voorafgegaan door een STRIG...ON.

moeilijkheidsgraad . . **normaal, kennis van de ASCII-tabel is een voordeel**
soort . . . . . **A-FUNKTIE**
afkomst . . . . . **STRING$ spreekt voor zich**

schrijfwijze

```
STRING$(<AANTAL TEKENS>,<TE HERHALEN KA-
  RAKTER>)
<AANTAL TEKENS>::=<N>
<TE HERHALEN KARAKTER>::=<U>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
<U>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met dit kommando is het mogelijk een lange string, opgebouwd uit allemaal dezelfde karakters, samen te stellen. Tussen haakjes dient dan als eerste het aantal te herhalen tekens te worden opgenomen. Dit aantal moet groter dan of gelijk aan nul zijn en kleiner dan 255. Indien de numerieke uitdrukking die het aantal bepaalt, een gebroken waarde bevat dan wordt de decimale fractie verwaarloosd.
Na de komma dient het te herhalen teken te worden gespecificeerd. Dit kan op twee manieren; via een numerieke uitdrukking en via een alfanumerieke uitdrukking. In het geval van een numerieke uitdrukking gelden dezelfde bepalingen als gegeven bij het aantal tekens. De waarde van de numerieke uitdrukking geeft de ASCII-kode aan van het af te drukken karakter. Zie hiervoor ondermeer de hoofdstukken 17 en 18.
In het geval van een alfanumerieke uitdrukking geldt dat het linker karakter van de waarde van deze alfanumerieke uitdrukking het teken vormt dat gaat worden herhaald. Bijvoorbeeld:

```
NEW
10 PRINT STRING$(20,"A")
20 PRINT STRING$(22,77)
30 LET A$=STRING$(10,"*")+"MSX"+
STRING$(10,"*")
40 PRINT A$
```

```
RUN
AAAAAAAAAAAAAAAAAAA
MMMMMMMMMMMMMMMMMMMM
**********MSX**********
Ok
```

(77 is de ASCII-kode voor M) _ _ _

SLEUTELWOORD # SWAP

moeilijkheidsgraad . . . . . **vrij eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **SWAP is verwisselen**

schrijfwijze

```
SWAP<ALFANUMERIEKE VARIABELE>,<ALFANUME-
  RIEKE VARIABELE>|SWAP<NUMERIEKE VARIA-
  BELE>,<NUMERIEKE VARIABELE>
<ALFANUMERIEKE VARIABELE>::=<ZIE ALGEME-
  NE SPECIFICATIES>
<NUMERIEKE VARIABELE>::=<ZIE ALGEMENE
  SPECIFICATIES>
```

betekenis

Met het kommando SWAP kunnen de waarden van twee numerieke of twee alfanumerieke variabelen worden verwisseld. Voorbeeld:

```
NEW
10 A=5:B=4:SWAP A,B:PRINT A;B
RUN
 4  5
Ok

NEW
10 DIM A(1):A(0)=5:A(1)=123
20 SWAP A(0),A(1)
30 PRINT A(0);A(1)
RUN
 123  5
Ok
```

De tweede in SWAP vermelde variabele moet in het programma reeds eerder zijn toegewezen (een waarde hebben gehad of zijn gedimensioneerd); een klein foutje in MSX-basic versie 1.0.

moeilijkheidsgraad . . vrij moeilijk, kennis van goniometrie is noodzakelijk
soort . . . . . N-FUNKTIE
afkomst . . . . . TAN is afkorting van tangent – tangens

schrijfwijze

```
TAN(<HOEK>)
<HOEK>:::=<N>
<N>:::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat de tangens van de waarde van de tussen haakjes vermelde uitdrukking. Deze waarde wordt beschouwd als de uitdrukking van een hoek in radialen.

Alhoewel elke waarde voor de hoek is toegestaan, is het in verband met de precisie van berekenen raadzaam om niet te grote hoekmaten te gebruiken.

```
NEW
10 FOR I=0 TO 90
20 LET HOEK=I/180*3.1415926535
30 PRINT I;":";TAN(HOEK)
40 NEXT I
RUN
```

Een tangenstabel verschijnt op het beeldscherm. Merk op dat in regel 20 de hoek van graden naar radialen geconverteerd wordt. Merk ook op dat, in tegenstelling tot de theorie, TAN geen foutmelding geeft bij een hoek van 90 graden (of $\frac{1}{2}\pi$ radialen).

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **SYSTEEMVARIABELE**
afkomst . . . . . **TIME is tijd**

schrijfwijze

TIME

betekenis

De MSX-computer bezit een interne klok. Deze klok wordt bij het aanzetten op 0 gezet en elke 1/50 seconde met de waarde 1 verhoogd. De waarden van de systeemklok kunnen we onder de systeemvariabele TIME opvragen en we kunnen de klok ook op 0 zetten.
Bijvoorbeeld:

```
NEW
10 TIME=0
20 FOR I=1 TO 50:LET A=SQR(I):NEXT I
30 LET T=TIME/50
40 PRINT "50 WORTELTREKKINGEN"
50 PRINT "KOSTEN";T;"SECONDEN"
RUN
50 WORTELTREKKINGEN
KOSTEN 6.08 SECONDEN
Ok
```

In bovenstaand voorbeeld werden 50 worteltrekkingen gedaan. Gemeten werd hoeveel seconden hiervoor nodig waren.

Het is niet aan te bevelen om de systeemklok voor nauwkeurige en langdurige tijdmetingen te gebruiken; daarvoor is de systeemklok te onnauwkeurig.

Indien de systeemklok (na ongeveer 22 minuten) de waarde 65535 overschrijdt, begint hij weer bij 0. De systeemklok staat stil wanneer de computer bezig is met niet onderbreekbare acties zoals het lezen van of schrijven naar cassette.

In samenwerking met de RND-funktie kan de TIME-variabele nog een heel dankbare taak vervullen.

Het vervelende van de RND-funktie is, dat een willekeurige, volkomen van de bediening onafhankelijke bepaling van een toevalsgetal niet mogelijk is. Vooral met het programmeren van spelletjes is dit een groot probleem. Door nu op een slimme wijze van TIME gebruik te maken, kan dit probleem voor goed uit de wereld geholpen worden:

```
NEW
10 REM TOEVALSGETALLEN
20 LET A=RND(-TIME)
30 PRINT RND(1)
40 GOTO 30
RUN
```

Het bovenstaande programma levert steeds een andere reeks toevalsgetallen. Het geheim zit in regel 20 waar (zie de behandeling van RND) steeds weer een andere reeks toevalsgetallen wordt geselekteerd. Doordat TIME 50 maal per seconde verandert is het welhaast onmogelijk voor de mens om via de bediening twee maal dezelfde reeks bewust te selekteren.

N.B.: Het sleutelwoord LET mag niet voor TIME worden gebruikt.

moeilijkheidsgraad . . . . . **zeer eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **TROFF is afkorting van trace off – ophouden met spoorzoeken**

schrijfwijze

TROFF

betekenis

Dit kommando wordt tesamen met TRON behandeld.
Zie aldaar.

moeilijkheidsgraad . . . . . **zeer eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **TRON is afkorting van trace on – beginnen met spoorzoeken**

schrijfwijze

TRON

betekenis

Dit kommando heeft tot gevolg dat het programma u bij uitvoering laat zien waar het binnen het programma bezig is. TRON is als zodanig een uitstekend hulpmiddel om funktionele fouten in een programma op te sporen. Voorbeeld:

```
NEW
10 TRON
20 LET A=1
30 LET B=2*A
40 LET A=2*B
50 IF A=16 THEN STOP
60 GOTO 30
RUN
[10][20][30][40][50][60][30][40][50]
Break in 50
Ok
```

De tussen de haken vermelde getallen zijn de regelnummers waar het programma mee bezig is. De loop van het programma kan op deze wijze perfekt worden bestudeerd.

TRON wordt door TROFF weer uitgeschakeld. Bijvoorbeeld:

```
10 TROFF
RUN
Break in 50
Ok
```

(TRON en TROF kunnen in een programmaregel worden gebruikt maar zijn veel bruikbaarder als direkte kommando's)

moeilijkheidsgraad . . **zeer moeilijk, kennis van machinetaal en/of ROM-inhoud is vereist**
soort . . . . . **N-FUNKTIE of A-FUNKTIE**
afkomst . . . . . **USR is afkorting van user function – door gebruiker ontworpen (niet basic) funkties**

schrijfwijze

```
USR[<CIJFER>](<U>)
<CIJFER>::=<ZIE ALGEMENE SPECIFICATIES>
<U>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie wordt onder DEF USR behandeld; zie aldaar.

moeilijkheidsgraad . . . . . **normaal**
soort . . . . . **FUNKTIE**
afkomst . . . . . **VAL is afkorting van value – waarde**

schrijfwijze

```
VAL(<A>)
<A>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met deze funktie kan een alfanumerieke uitdrukking numeriek worden uitgewaardeerd. De funktie VAL onderzoekt de tussen haakjes opgenomen alfanumerieke uidrukking van links naar rechts. Zolang geen verboden karakters voor een numerieke konstante worden tegengekomen, blijft deze funktie de alfanumerieke variabele doorwerken. Zodra echter een karakter wordt tegengekomen dat verboden is, stopt de funktie VAL het verdere onderzoek en wordt de tot dat toe opgebouwde numerieke konstante als resultaat gegeven. Bijvoorbeeld:

```
NEW
10 LET A$="123GHJ4"
20 LET A=VAL(A$)
30 PRINT A
40 STOP
RUN
 123
Ok

10 LET A$="1E22HJHKJ"
RUN
 1E+22                    (alleen regel 10 wordt vervangen)
Ok

NEW
10 LINE INPUT "WAARDE:";A$:LET A=VAL(A$)
20 PRINT "WORTEL=";SQR(A):GOTO 10
```

```
RUN
WAARDE:12
WORTEL= 3.4641016151377
WAARDE: (etcetera)
```

moeilijkheidsgraad .. **zeer moeilijk, kennis van de opbouw van het computergeheugen is noodzakelijk**
soort ..... **N-FUNKTIE**
afkomst ..... **VARPTR is afkorting van variabele pointer – adres van variabele**

schrijfwijze

```
VARPTR (<VARIABELE>)
       (#<BESTANDSNUMMER>)
<VARIABELE>::=<ZIE ALGEMENE SPECIFICA-
  TIES>
<BESTANDSNUMMER>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Deze funktie geeft als resultaat:

- bij een numerieke variabele: het geheugenadres van het eerste byte van deze variabele;
- bij een alfanumerieke variabele: het geheugenadres in het stringgebied van het eerste byte van de betreffende alfanumerieke variabele;
- bij een bestandsnummer: het geheugenadres van het eerste byte van het file control block.

Indien deze funktie een negatief resultaat geeft, dient de waarde 65536 bij dit resultaat te worden opgeteld teneinde het juiste geheugenadres te verkrijgen.

moeilijkheidsgraad . . zeer moeilijk, kennis van de betekenis van de registers van de beeldscherm controlechip is vereist
soort . . . . . SYSTEEMVARIABELE
afkomst . . . . . VDP is afkorting van video display — beeldscherm

schrijfwijze

```
VDP(<REGISTERNUMMER>)
<REGISTERNUMMER>:::=<N>
<N>:::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

De VDP-systeemvariabele representeert de inhoud van de registers van de VDP, de Video Display Processor. Achter het sleutelwoord VDP dient tussen haakjes een registernummer te worden opgegeven. Dit nummer mag niet kleiner zijn dan nul en niet groter dan 8. Een eventuele decimale fractie wordt verwaarloosd.

Door de registers van de VDP te besturen, kunnen diverse effecten worden bereikt. Onoordeelkundig gebruik van het VDP-sleutelwoord leidt meestal tot onvoorspelbare resultaten; vaak wordt de computer 'opgehangen' en kan alleen uit- en weer inschakelen de computer weer tot leven brengen.

Een voorbeeld.

```
NEW
10 VDP(2)=100
RUN
```

Het beeldscherm vertoont vreemde effecten. De meest eenvoudige oplossing is uit- en inschakelen.

Het sleutelwoord LET is voor VDP niet toegestaan. VDP(8) kan geen waarde worden toegekend; deze systeemvariabele kan slechts als waarde worden gebruikt.

Zie voor de betekenis van de betreffende VDP-registers hoofdstuk 16.

moeilijkheidsgraad . . **zeer moeilijk, kennis van de opbouw van het beeldschermgeheugen is noodzakelijk**
soort . . . . . **N-FUNKTIE**
afkomst . . . . . **VPEEK is afkorting van video memory peek – gluren in het beeldscherm geheugen**

schrijfwijze

```
VPEEK(<GEHEUGENADRES>)
<GEHEUGENADRES>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het is noodzakelijk dat eerst de behandeling van het PEEK-kommando wordt doorgenomen.

VPEEK heeft hoegenaamd dezelfde funktie als PEEK met dit verschil dat de geheugenadressering niet kleiner mag zijn dan 0 en niet groter dan 16383 en dat niet het computergeheugen maar het videogeheugen (RAM, 16k) wordt aangesproken.

Voor een goed gebruik van VPEEK is een gedetailleerde kennis van de opbouw van het video-geheugen van een MSX-computer noodzakelijk. In hoofdstuk 16 wordt hierop gedeeltelijk ingegaan. Verder dient hiertoe specialistische literatuur te worden geraadpleegd.

SLEUTELWOORD # VPOKE

moeilijkheidsgraad . . **zeer moeilijk, kennis van de opbouw van het beeldscherm geheugen is vereist**
soort . . . . . **KOMMANDO**
afkomst . . . . . **VPOKE is afkorting van video memory poke – stop weg in beeldscherm geheugen**

schrijfwijze

```
VPOKE<GEHEUGENADRES>,<GEHEUGENWAARDE>
<GEHEUGENADRES>::=<N>
<GEHEUGENWAARDE>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Het is noodzakelijk dat eerst de behandeling van POKE wordt doorgenomen.

VPOKE heeft hoegenaamd dezelfde funktie als POKE met dit verschil dat de geheugenadressering niet kleiner mag zijn dan 0 en niet groter dan 16383 en dat niet het computergeheugen maar het videogeheugen (RAM, 16k) wordt aangesproken.

Voor een goed gebruik van VPOKE is een gedetailleerde kennis van de opbouw van het video-geheugen van een MSX-computer noodzakelijk. In hoofdstuk 16 wordt hierop gedeeltelijk ingegaan. Verder dient hiertoe specialistische literatuur te worden geraadpleegd.

SLEUTELWOORD # WAIT

moeilijkheidsgraad . . **zeer moeilijk, kennis van de funkties van de poorten van het computersysteem is vereist**
soort . . . . . **KOMMANDO**
afkomst . . . . . **WAIT is wachten**

schrijfwijze

```
WAIT<POORT>,<FILTER 1>[,<FILTER 2>]
<POORT>::=<N>
<FILTER 1>::=<N>
<FILTER 2>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met dit kommando kan worden gewacht totdat de waarde van het byte op een bepaalde poort een bepaalde waarde heeft. De condities waaronder het wachten niet meer behoeft te gebeuren, zijn gekodeerd in filter 1 en filter 2.
Het poortnummer mag niet kleiner zijn dan –32768 en niet groter dan 65535. Een eventuele decimale fractie wordt verwaarloosd. Indien het poortnummer kleiner is dan 0, wordt 65536 bij dit nummer opgeteld voordat de betreffende poort wordt benaderd.
De beide filterwaarden mogen niet kleiner zijn dan 0 en niet groter dan 255 terwijl een decimale fractie wordt verwaarloosd.
Indien de tweede filterwaarde wordt weggelaten, wordt een 0 voor deze waarde aangenomen.

WAIT voert continu op binair (bit-) niveau een XOR uit tussen de waarde die op dat moment in de buffer van de betreffende poort staat en filter 2. Tussen het resultaat van deze berekening en filter 1 wordt daarna een AND uitgevoerd. Indien het uiteindelijke resultaat ongelijk is aan 0, wordt er niet meer gewacht maar wordt het programma vervolgd. Indien het uiteindelijke resultaat gelijk is aan 0, wordt de controle wederom uitgevoerd.

Filter 2 kan men beschouwen als een waarde die aangeeft, welke bits dienen te worden geinverteerd. Filter 1 is dan een waarde die aangeeft van welke bits er minimaal één op 1 moet staan voor verdere doorgang van het programma.

Voorbeeld:

```
WAIT 168,1
```

De computer wacht; alleen een CONTROL-BREAK kan dit wachten opheffen.

Voor een goed gebruik van WAIT is een gedetailleerde kennis van de hardware-opbouw van een MSX-computer noodzakelijk. Daarbij dienen de diverse componenten ook software-technisch te worden beheersd. Hiertoe dient specialistische literatuur te worden geraadpleegd; in dit handboek wordt hierop niet verder ingegaan.

SLEUTELWOORD # WIDTH

moeilijkheidsgraad . . . . . **eenvoudig**
soort . . . . . **KOMMANDO**
afkomst . . . . . **WIDTH is breedte**

schrijfwijze

```
WIDTH<TEKSTREGELBREEDTE>
<TEKSTREGELBREEDTE>::=<N>
<N>::=<ZIE ALGEMENE SPECIFICATIES>
```

betekenis

Met het WIDTH-kommando kan de gewenste regelbreedte worden ingesteld. Afhankelijk van de SCREEN-instelling (zie de behandeling van SCREEN) kan de breedte maximaal 32 of 40 karakters zijn. Het minimum is natuurlijk gelijk aan 1. Indien de numerieke uitdrukking die tussen haakjes een gebroken waarde als uitkomst heeft, dan wordt de hoogste gehele waarde kleiner dan deze gebroken waarde als geldende waarde genomen. Bij het uitvoeren van een WIDTH wordt het beeldscherm schoongemaakt tenzij de betreffende breedte reeds geactiveerd was.

Voorbeeld:

```
NEW
10 WIDTH 4
20 PRINT "HALLO ALLEMAAL"
RUN  *******        (beeldscherm wordt schoongemaakt)
                    HALL
                    O AL
                    LEMA
                    AL
                    Ok
```

De volgende bevelen zijn in het MSX-basic aanroepbaar maar geven steeds een foutmelding. Zij zijn pas funktioneel indien de computer is uitgebreid met een schijveneenheid die voldoet aan de MSX-normen.

In een apart handboek worden de MSX disk-basic sleutelwoorden behandeld. In dit hoofdstuk volgt alvast een opsomming van deze bevelen met daar achter een funktie-aanduiding.

Sleutelwoorden die reeds werden behandeld maar die een extra betekenis krijgen in MSX disk-basic:

OPEN	voor het toewijzen en openen van bestanden en voor het verbinden van een kanaalnummer aan deze bestanden.
CLOSE	voor het afsluiten van reeds geopende bestanden.
SAVE	voor het vastleggen van programma's in ASCII formaat.
LOAD	voor het weer laden van in ASCII formaat vastgelegde programma's.
BSAVE	voor het vastleggen van delen van de inhoud van het computergeheugen.
BLOAD	voor het weer ophalen van de inhoud van delen van het computergeheugen.
RUN	voor het uitvoeren van een programma.
MAXFILES	voor het vaststellen van het maximum aantal tegelijk te openen bestanden.
EOF	ter bepaling van het einde van een bestand.
INPUT#	voor het lezen van gegevens uit een bestand.
LINE INPUT#	voor het lezen van gegevens uit een bestand.
INPUT$	voor het lezen van gegevens uit een bestand.
PRINT# (USING)	voor het schrijven van gegevens naar een bestand.

Sleutelwoorden die nog niet eerder werden besproken en die betrekking hebben op het MSX disk-basic:

ATTR$	voor het opvragen van speciale gegevens van een bestand of schijf.
CVD	voor het omzetten van een stringwaarde naar een dubbele preciese waarde.
CVI	voor het omzetten van een stringwaarde naar een

	integere waarde.
CVS	voor het omzetten van stringwaarde naar een waarde met enkelvoudig precisie.
COPY	voor copieren van bestanden.
DSKF	om de vrije ruimte op schijf te vragen.
DSKI$	om een blok in zijn geheel van schijf in te lezen.
DSKO$	om een blok in zijn geheel naar schijf te schrijven.
FIELD	om de recordindeling van een random bestand te bepalen.
FILES	om een inhoudsopgave van de schijf te verkrijgen.
FPOS	om de positie van de lees/schrijfkop op te vragen.
GET	om een record uit een bestand te lezen.
KILL	om een bestand van schijf te verwijderen.
LFILES	om een inhoudsopgave van een schijf op de te verkrijgen.
LOC	om de huidige positie binnen een bestand op te vragen.
LOF	om de grootte van een bestand in bytes op te vragen.
LSET	om een record (gedeeltelijk) op te vullen.
MKD$	om van een waarde met dubbele precisie een string te maken.
MKI$	om van een integere waarde een string te maken.
MKS$	om van een waarde met enkele precisie een string te maken.
NAME	om een bestand een andere naam te geven.
PORT	om het aantal ontvangen karakters vanuit een bestand op te vragen.
PUT	om een record naar een bestand te schrijven.
RSET	om een record (gedeeltelijk) te vullen.
SET	om speciale gegevens voor een bestand of schijf te bepalen.

11 MSX-SLEUTELWOORDEN OP VOLGORDE VAN SOORT

In hoofdstuk 10 werden de sleutelwoorden op alfabetische volgorde gepresenteerd. In de volgende paragrafen zijn de sleutelwoorden per soort onderverdeeld. Sommige sleutelwoorden komen onder meer dan één soort voor.

Een enkele keer is een onderverdeling van een sleutelwoord niet helemaal verdedigbaar; de meest voor de hand liggende onderverdeling is dan gekozen.

11.1 de numerieke funkties

De volgende sleutelwoorden zijn als N-FUNKTIE (nummerieke funktie) getypeerd. Zij dienen te worden toegepast op een uitdrukking en geven altijd een numeriek resultaat.

ABS	EXP	PAD	STRIG
ASC	FIX	PDL	TAN
ATN	FN	PEEK	TIME
CDBL	FRE	POINT	USR
CINT	INP	POS	VAL
COS	INSTR	RND	VARPTR
CSNG	INT	SGN	VPEEK
EOF	LEN	SIN	
ERL	LOG	SQR	
ERR	LPOS	STICK	

11.2 de alfanumerieke funkties

De volgende sleutelwoorden zijn als A-FUNKTIE (alfanumerieke funktie) getypeerd. Zij dienen te worden toegepast op een uitdrukking en geven altijd een alfanumeriek resultaat.

BIN$	HEX$	OCT$	STRING$
CHR$	LEFT$	RIGHT$	STR$
FN	MID$	SPACE$	USR

11.3 de toekennings-kommando's

De volgende kommando's kennen een geheugendeel, een variabele, een kanaal of een funktie, een bepaalde waarde, grootte of betekenis toe of nemen deze juist af.

CLEAR	DEFINT	LET	RESTORE
CLOSE	DEFSNG	MID$	SWAP
DATA	DEFSTR	NEXT	VPOKE
DEF FN	DIM	OPEN	
DEF USR	ERASE	POKE	
DEFDBL	FOR-TO-STEP	READ	

11.4 De besturings-kommando's

De volgende kommando's hebben de mogelijkheid in zich om de volgorde van uitvoering van kommando's te beïnvloeden.

AUTO	INTERVAL	ON STOP GOSUB
CALL	KEY	ON STRIG GOSUB
CONT	NEW	RENUM
DELETE	NEXT	RESUME
END	ON ERROR GOTO	RETURN
ERROR	ON-GOSUB	RUN
FOR-TO-STEP	ON-GOTO	SPRITE
GOSUB	ON INTERVAL GOSUB	STOP
GOTO	ON KEY GOSUB	STRIG
IF-THENGOTO-ELSE	ON SPRITE GOSUB	WAIT

11.5 De invoer-kommando's

De volgende kommando's maken het mogelijk om gegevens vanuit de 'buitenwereld' (de randapparatuur) de computer binnen te halen.

BLOAD	INPUT	LOAD
CLOAD	INPUT$	MERGE
INKEY$	LINE INPUT	OPEN

11.6 De uitvoer-kommando's

De volgende kommando's maken het mogelijk om gegevens vanuit de computer de 'buitenwereld' (randapparatuur) in te sturen.

BEEP	DRAW	OPEN	PUT SPRITE
BSAVE	LINE	OUT	SAVE
CIRCLE	LIST	PAINT	SCREEN
CLOSE	LLIST	PLAY	SOUND
CLS	LOCATE	PRESET	TROFF
COLOR	LPRINT	PRINT	TRON
CSAVE	MOTOR	PSET	WIDTH

11.7 De grafische uitvoer-kommando's

De volgende uitvoerkommando's hebben betrekking op het grafische karakter van de MSX-computer.

CIRCLE	DRAW	PRESET	SCREEN
CLS	LINE	PSET	
COLOR	PAINT	PUT SPRITE	

11.8 De muzikale uitvoer-kommando's

De volgende kommando's hebben betrekking op het muzikale karakter van de MSX-computer.

BEEP	PLAY	SOUND

11.9 De systeemvariabelen

De onderstaande variabelen zijn niet zo maar variabelen maar hebben binnen het MSX-systeem een bepaalde funktie.

BASE	MAXFILES	VDP
CSRLIN	SPRITE$	

11.10 De indifferente kommando's

De volgende kommando's hebben geen enkele invloed op de door het programma uit te voeren taak.

REM	TROFF	TRON

12 MSX SLEUTELWOORDEN/AANBEVOLEN LEERVOLGORDE

In hoofdstuk 9 zijn de sleutelwoorden op alfabetische volgorde opgenomen. Hieronder volgt een opsomming in de volgorde waaronder deze sleutelwoorden het beste kunnen worden bestudeerd door de nieuweling op MSX-basic gebied.

PRINT
RUN
STOP
END
CONT
INPUT
REM
AUTO
CLS
BEEP
LIST
LLIST
DELETE
LEN
GOTO
RENUM
NEW
LET
SWAP
ABS
FIX
INT
SGN
WIDTH
FOR-to-step
NEXT
IF-then goto-else
DIM
SPACE$
LOCATE
CSRLIN
POS
GOSUB
RETURN
TRON

TROFF
ASC
CHR$
SQR
ATN
COS
SIN
TAN
EXP
LOG
BIN$
OCT$
HEX$
INKEY$
DATA
READ
RESTORE
LINE input
STRIG (funktie)
STICK
PDL
DEFSTR
DEFINT
DEFSNG
DEFDBL
CSAVE
CLOAD
MOTOR
INPUT$
CLEAR
FRE
LPRINT
LPOS
RND

SCREEN
COLOR
PSET
PRESET
POINT
LINE
CIRCLE
PAINT
DRAW
ON-goto
ON-gosub
ON error goto
ERROR
ERR
ERL
RESUME
TIME
STRING$
VAL
STR$
CINT
CSNG
CDBL
SAVE
LOAD
MERGE
MID$ (funktie)
RIGHT$
LEFT$
MID$ (komm.)
INSTR
ERASE
KEY
ON key gosub

ON stop gosub
STRIG (kommando)
ON strig gosub
INTERVAL
ON interval gosub
PAD
SPRITE$
PUT sprite
SPRITE
ON sprite gosub
DEF fn
FN
PLAY (kommando)
PLAY (funktie)
MAXFILES
OPEN
CLOSE
EOF

CALL
VARPTR
SOUND
VDP
BASE
PEEK
POKE
VPEEK
VPOKE
INP
OUT
WAIT
DEF usr
USR
BSAVE
BLOAD

De behandeling van enkele kommando's is onderverdeeld in een eenvoudig gebruik en een gevorderd gebruik. Bestudeer eerst alleen het eenvoudige gebruik. Pas nadat u de aanbevolen leervolgorde heeft afgewerkt of nadat dat nodig blijkt, kunt u de gevorderde behandeling doornemen.
De opgenomen voorbeelden zijn afgestemd op de aanbevolen leervolgorde. De omlijnde sleutelwoorden vereisen een vrij specialistisch niveau en kunnen de eerste tijd misschien beter maar worden overgeslagen.

13 MSX-FOUTMELDING OP VOLGORDE VAN NUMMER

Hieronder volgen de mogelijke foutmeldingen van MSX-basic. Zij zijn op nummer gesorteerd opgenomen terwijl de betekenis van de foutmelding daar achter is geplaatst.

01 NEXT without FOR	gepoogd werd om een NEXT uit te voeren zonder dat een bijbehorende FOR werd uitgevoerd.
02 Syntax error	er werd een fout in de schrijfwijze ontdekt.
03 RETURN without GOSUB	gepoogd werd om een RETURN uit te voeren terwijl er niet eerder een GOSUB werd uitgevoerd.
04 Out of DATA	gepoogd werd om een READ uit te voeren terwijl er geen DATA meer kon worden gevonden.
05 Illegal function call	er werd gepoogd om een kommando of funktie uit te voeren met niet toegestane waarden.
06 Overflow	het resultaat van een berekening ligt boven het toegestane maximum of onder het toegestane minimum.
07 Out of memory	er werd gepoogd om een kommando uit te voeren dat meer geheugen nodig heeft dan er beschikbaar is.
08 Undefined line number	een niet bestaand programmaregelnummer werd benaderd.
09 Subscript out of range	een array-variabele werd buiten zijn dimensies aangesproken of er werd een verkeerd aantal dimensies genoemd.
10 Redimensioned array	er werd gepoogd om een array-variabele voor een tweede keer te DIMensioneren.
11 Division by zero	gepoogd werd om een deling door nul te doen of een negatieve macht van nul te berekenen.
12 Illegal direct	er werd gepoogd om een kommando

	direkt in te tikken terwijl dit kommando slechts in een programmaregel mag voorkomen.
13 Type mismatch	een numerieke en een alfanumerieke uitdrukking werden met elkaar verwisseld.
14 Out of string space	gepoogd werd om een kommando uit te voeren waardoor het stringgeheugen te klein werd. Vergroot het string-geheugen met CLEAR.
15 String too long	er werd gepoogd om een string samen te stellen die langer werd dan 255 posities.
16 String formula too complex	de alfanumerieke uitdrukking die werd uitgewerkt is te ingewikkeld voor MSX-basic. Splits deze uitdrukking in enkele kleinere.
17 Can't continue	gepoogd werd on een programma met CONT te vervolgen terwijl dat niet (meer) gaat.
18 Undefined user function	gepoogd werd om met FN een niet gedefinieerde funktie aan te roepen.
19 Device I/O error	een fout werd ontdekt tijdens het lezen van/schrijven naar een randapparaat (cassetterecorder of printer e.d.).
20 Verify error	tijdens de controle met CLOAD? werd een verschil ontdekt tussen het programma op cassetteband en het programma in het geheugen.
21 No RESUME	de ERROR-routine werd aangeroepen en een RESUME kon niet worden gevonden.
22 RESUME without error	er werd gepoogd om een RESUME uit te voeren terwijl er geen fout via de ON ERROR GOTO werd gedetecteerd.
24 Missing operand	een noodzakelijke uitdrukking wordt in een kommando niet gevonden.
25 Line buffer overflow	een ingetikte programmaregel

	is te lang voor MSX-basic (langer dan 255 karakters).
50 FIELD overflow	deze foutmelding is wel aanwezig maar kan in het standaard MSX-basic niet voorkomen tenzij met ERROR 50 gegenereerd.
51 Internal error	deze melding wijst op een fout die niet zou mogen kunnen voorkomen. Licht uw computerleverancier of MICROSOFT in.
52 Bad file number	een niet toegestaan kanaalnummer werd gebruikt.
53 File not found	deze foutmelding is wel aanwezig maar kan in het standaard MSX-basic niet voorkomen tenzij met error 53 gegenereerd.
54 File already open	deze foutmelding is wel aanwezig maar kan in het standaard MSX-basic niet voorkomen tenzij met error 54 gegenereerd.
55 Input past end	er werd gepoogd om nog gegevens uit een bestand te lezen terwijl dit bestand reeds volledig was doorgewerkt.
56 Bad file name	een bestandsnaam werd niet volgens de voorschriften samengesteld.
57 Direct statement in file	tijdens het LOADen van een programma werd een direkt kommando (zonder regelnummer) ontdekt; LOAD werd gestopt.
58 Sequential I/O only	deze foutmelding is wel aanwezig maar kan in het standaard MSX-basic niet voorkomen tenzij met ERROR 58 gegenereerd.
59 File not open	een kanaal werd aangesproken zonder dat hierop een bestand geOPENd is.

Indien een niet bestaand foutnummer met ERROR wordt genegeerd, zal de foutmelding 'Unprintable error' verschijnen. Deze foutmeldingen kunnen door de programmeur een bepaalde betekenis worden toegedacht.

14 MSX-FOUTMELDINGEN OP ALFABETISCHE VOLGORDE

Hieronder volgen de mogelijke foutmeldingen van MSX-basic. Zij zijn op volgorde van alfabet opgenomen. De betekenissen zijn achter de foutmeldingen geplaatst.

56 Bad file name	een bestandsnaam werd niet volgens de voorschriften samengesteld.
52 Bad file number	een niet toegestaan kanaalnummer werd gebruikt.
17 Can't continue	gepoogd werd on een programma met CONT te vervolgen terwijl dat niet (meer) gaat.
19 Device I/O error	een fout werd ontdekt tijdens het lezen van/schrijven naar een randapparaat (cassetterecorder of printer e.d.).
57 Direct statement in file	tijdens het LOADen van een programma werd een direkt kommando (zonder reselnummer) ontdekt; LOAD werd gestopt.
11 Division by zero	gepoogd werd om een deling door nul te doen of een negatieve macht van nul te bepalen.
50 FIELD overflow	deze foutmelding is wel aanwezig maar kan in het standaard MSX-basic niet voorkomen tenzij met ERROR 50 gegenereerd.
54 File already open	deze foutmelding is wel aanwezig maar kan in het standaard MSX-basic niet voorkomen tenzij met error 54 gegenereerd.
53 File not found	deze foutmelding is wel aanwezig maar kan in het standaard MSX-basic niet voorkomen tenzij met error 53 gegenereerd.
59 File not open	een kanaal werd aangesproken zonder dat hierop een bestand geOPENd is.
12 Illegal direct	er werd gepoogd om een kommando direkt in te tikken terwijl dit

	kommando slechts in een programmaregel mag voorkomen.
05 Illegal function call	er werd gepoogd om een kommando of funktie uit te voeren met niet toegestane waarden.
55 Input past end	er werd gepoogd om nog gegevens uit een bestand te lezen terwijl dit bestand reeds volledig was doorgewerkt.
51 Internal error	deze melding wijst op een fout die niet zou mogen kunnen voorkomen. Licht uw computerleverancier of MICROSOFT in.
25 Line buffer overflow	een ingetikte programmaregel is te lang voor MSX-basic (langer dan 255 karakters).
24 Missing operand	een noodzakelijke uitdrukking wordt in een kommando niet gevonden.
01 NEXT without FOR	gepoogd werd om een NEXT uit te voeren zonder dat een bijbehorende FOR werd uitgevoerd.
21 No RESUME	de ERROR-routine werd aangeroepen en een RESUME kon niet worden gevonden.
04 Out of DATA	gepoogd werd om een READ uit te voeren terwijl er geen DATA meer kon worden gevonden.
07 Out of memory	er werd gepoogd om een kommando uit te voeren dat meer geheugen nodig heeft dan er beschikbaar is.
14 Out of string space	gepoogd werd om een kommando uit te voeren waardoor het stringgeheugen te klein werd. Vergroot het string-geheugen met CLEAR.
06 Overflow	het resultaat van een berekening ligt boven het toegestane maximum of onder het toegestane minimum.
22 RESUME without error	er werd gepoogd om een RESUME uit te voeren terwijl er geen

	fout via de ON ERROR GOTO werd gedetecteerd.
03 RETURN without GOSUB	gepoogd werd om een RETURN uit te voeren terwijl er niet eerder een GOSUB werd uitgevoerd.
10 Redimensioned array	er werd gepoogd om een array-variabele voor een tweede keer te DIMensioneren.
58 Sequential I/O only	deze foutmelding is wel aanwezig maar kan in het standaard MSX-basic niet voorkomen tenzij met ERROR 58 gegenereerd.
16 String formula too complex	de alfanumerieke uitdrukking die werd uitgewerkt is te ingewikkeld voor MSX-basic. Splits deze uitdrukking in enkele kleinere.
15 String too long	er werd gepoogd om een string samen te stellen die langer werd dan 255 posities.
09 Subscript out of range	een array-variabele werd buiten zijn dimensies aangesproken of er werd een verkeerd aantal dimensies genoemd.
02 Syntax error	er werd een fout in de schrijfwijze ontdekt.
13 Type mismatch	een numerieke en een alfanumerieke uitdrukking werden met elkaar verwisseld.
08 Undefined line number	een niet bestaand programmaregelnummer werd benaderd.
18 Undefined user function	gepoogd werd om met FN een niet gedefinieerde funktie aan te roepen.
20 Verify error	tijdens de controle met CLOAD? werd een verschil ontdekt tussen het programma op cassetteband en het programma in het geheugen.

Indien een niet bestaand foutnummer met ERROR wordt genegeerd, zal de foutmelding 'Unprintable error' verschijnen. Deze foutmeldingen kunnen door de programmeur een bepaalde betekenis worden toegedacht.

15 DE PROGRAMMABLE SOUND GENERATOR

De AY-3-8912 geluidsprocessor die wordt bestuurd via veertien registers, genummerd vanaf 0 tot en met 13. Deze registers dienen via het SOUND-kommando (zie de behandeling in hoofdstuk 9) te worden voorzien van bepaalde waarden.

Bij het programmeren dienen per geluidsvariatie steeds de volgende stappen te worden overwogen:

- zet de betreffende geluidskanalen uit (register 7)
* zet de toonhoogte (register 0 ... 6)
* zet het volume of het betreffende effect op de juiste waarde (register 8, 9 of 10)
* zet de tijdsduur van de effectontwikkeling op de juiste waarde (register 11 en 12)
* zet het juiste soort effect aan (register 13)
- zet de betreffende kanalen aan (register 7).

Voor de met een * gemerkte stappen geldt dat deze alleen dienen te worden ondernomen wanneer zij van toepassing zijn en nog niet eerder zijn bepaald, of veranderd dienen te worden.

Op de volgende pagina is een schema geplaatst van de PSG-registers met hun betekenis. Daarna volgt per register een korte toelichting.

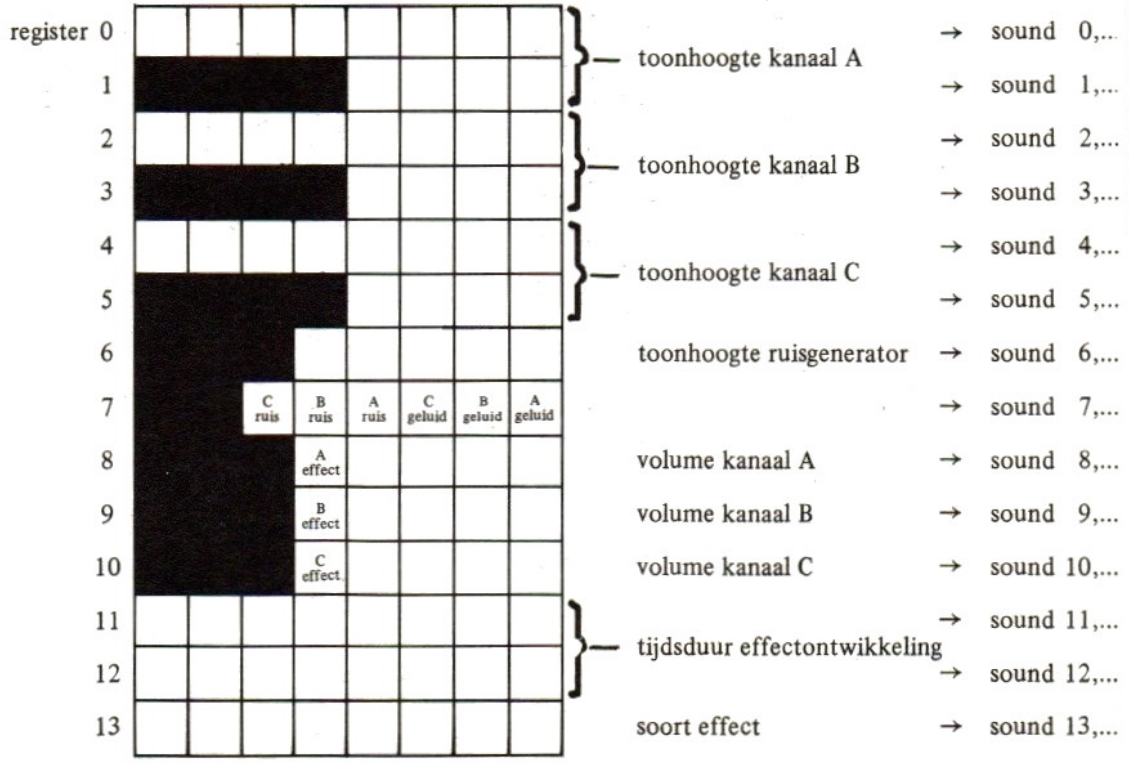
De geluidskanalen zijn de kanalen A, B en C. Per geluidskanaal kan in twee registers de toonhoogte worden gekodeerd. Voor de te programmeren toonhoogte geldt de volgende formule:

$$f = \frac{111760}{Hz} \qquad (f<4096)$$

Indien de toonhoogte in Herz bekend is, kan deze in de formule worden ingevuld; het resultaat noemen we f (frequentie).

Bijvoorbeeld: een toon van 440 Hz resulteert in een f van ongeveer 254.
Steeds dienen de eindwaarden te worden afgerond.

Nadat de waarde f is berekend, dient deze in het betreffende registerpaar te worden opgenomen. Dit gaat als volgt:


register 0
1
2
3
4
5
6
7
8
9
10
11
12
13
C ruis
B ruis
A ruis
C geluid
B geluid
A geluid
A effect
B effect
C effect
toonhoogte kanaal A
toonhoogte kanaal B
toonhoogte kanaal C
toonhoogte ruisgenerator
volume kanaal A
volume kanaal B
volume kanaal C
tijdsduur effectontwikkeling
soort effect
→ sound 0,...
→ sound 1,...
→ sound 2,...
→ sound 3,...
→ sound 4,...
→ sound 5,...
→ sound 6,...
→ sound 7,...
→ sound 8,...
→ sound 9,...
→ sound 10,...
→ sound 11,...
→ sound 12,...
→ sound 13,...

Tik in: PRINT BIN$(...) en vul op de puntjes de waarde voor f in. Bij f=254 wordt dat:

```
PRINT BIN$(254)
11111110
Ok
```

Vul vervolgens deze waarde vooraan op met nullen tot 12 posities. In ons voorbeeld met f=254 krijgen we dan:

```
000011111110
```

Kap hierna de verkregen binaire konstante van 12 posities in een deel van vier en een deel van 8 posities. Ons voorbeeld met f=254 geeft dan:

```
0000      11111110
```

Ken dan aan het betreffende registerpaar deze verkregen konstanten binair toe. Het gedeelte van 8 binaire cijfers moet in het eerste register van het betreffende kanaal worden geplaatst en het gedeelte van 4 binaire cijfers in het volgende register. Wanneer we geluidskanaal A een toon van 440 Hz willen laten produceren, programmeren we ondermeer:

```
NEW
20 SOUND 0,&B11111110
30 SOUND 1,&B0000
```

Indien u dat kunt, mag u de binaire konstanten ook omrekenen naar decimale konstanten en deze in het SOUND-kommando opnemen.

Buiten de geluidskanalen is er ook een ruiskanaal aanwezig. Via dit ruiskanaal kunnen geluidseffecten (motoren, geweerschoten e.d.) worden gesimuleerd of kan het slagwerk bij een stuk muziek worden verzorgd.

Dit ruiskanaal heeft een vrij grove frequentie-aanduiding nodig. Deze aanduiding dient in register 6 te worden opgenomen. &B00000 geeft ruis met een hoge frequentie (licht) en &B11111 geeft een ruis met een lage frequentie (zwaar). Alle tussenliggende waarden mogen van licht naar zwaar worden gebruikt.

Een vrij zware ruis programmeren wij bijvoorbeeld als volgt:

```
40 SOUND 6,&B11001
```

Om een geluidskanaal daadwerkelijk te activeren, dient het te worden aangeschakeld. Een geluidskanaal kan worden uit- of aangeschakeld via register 7. Voor elk kanaal dat uitgeschakeld moet worden of blijven, dient het binaire cijfer 1 te worden ingevuld terwijl voor een kanaal dat aangeschakeld moet worden of blijven, een binaire 0 dient te worden ingevuld. In ons voorbeeld schakelen we eerst alle kanalen uit:

```
10 SOUND 7,&B111111
```

Later, als eerste actie schakelen we het geluid over kanaal A en de ruis over in kanaal A:

```
90 SOUND 7,&B110110
```

Indien geen (volume-)effect wordt gewenst voor een betreffend kanaal, dan dient in register 8, 9 of 10 op die plaats een binaire 0 te worden ingevuld. Wordt echter wèl een effect gewenst, dan dient een binaire 1 te worden ingevuld op die plaats. Indien een 1 is ingevuld, dan heeft het verder geen zin om een volume vast te stellen; elk effect heeft een eigen volume. Wanneer een 0 werd ingevuld, dan kan echter wel het volume worden bepaald. Het volume kan vanaf helemaal dicht (&B00000) tot helemaal open (&B01111) worden geregeld. Alle tussenliggende waarden zijn toegestaan.
In ons voorbeeld kiezen we voor een effect:

```
50 SOUND 8,&B10000
```

Indien voor één of meer kanalen voor een effect werd gekozen, dan dient de tijdsduur van dit effect te worden gespecificeerd in register 11 en 12. De te specificeren waarde is dezelfde als de waarde die na het M-kommando in MML (zie PLAY) dient te worden opgenomen.

Allereerst dient de tijdsduur in seconden te worden bepaald van de effectontwikkeling. Vervolgens wordt dit getal vermenigvuldigd met de waarde 6965. Deze waarde dient te worden afgerond en mag niet groter zijn dan 65535. Tik vervolgens in: PRINT BIN$(...) met op de puntjes de verkregen waarde ingevuld. Bij een gewenste ontwikkelingstijd van 2 seconden is dit bijvoorbeeld:

```
PRINT BIN$(2*6965)
1101100l101010
Ok
```

Vul de verkregen binaire konstante vooraan aan met nullen totdat 16 cijfers zijn verkregen en splits deze konstante dan in twee delen van acht binaire cijfers. Ons voorbeeld (2 seconden):

```
00110110  01101010
```

Vul vervolgens register 11 met het tweede gedeelte van deze konstante en register 12 met het eerste gedeelte. Ons voorbeeld:

```
60 SOUND 11,&B01101010
70 SOUND 12,&B00110110
```

Indien voor één of meer kanalen voor een effect wordt gekozen, dient uiteindelijk nog het soort effect te worden gespecificeerd in register 13. Voor de mogelijke effecten dient u het betreffende onderdeel met schema in hoofdstuk 9, de behandeling van PLAY te bestuderen. Voor ons voorbeeld kiezen we voor effect nummer 1, een éénmalig wegstervend geluid. Tikt u in: PRINT BIN$ (effectnummer). In ons voorbeeld:

```
PRINT BIN$(1)
1
Ok
```

Dit uiteindelijke binaire resultaat dient aan register 13 te worden toegekend:

```
80 SOUND 13,&B1
```

Het resulterende programma in verband met het voorbeeld ziet er in totaal nu als volgt uit:

```
LIST
10 SOUND 7,&B111111
20 SOUND 0,&B11111110
30 SOUND 1,&B0000
40 SOUND 6,&B11001
50 SOUND 8,&B10000
```

```
60 SOUND 11,&B01101010
70 SOUND 12,&B00110110
80 SOUND 13,&B1
90 SOUND 7,&B110110
RUN
```

Een 'geweerschot', begeleid door een toon van 440 Hz (A), is hoorbaar. Door regel 90 te veranderen naar

```
90 SOUND 7,111110
```

of

```
90 SOUND 110111
```

kunnen we of alleen het schot of alleen de toon hoorbaar maken.

Merk op dat het uiteindelijke voorbeeldprogramma volgens de regels is opgebouwd:

- op regel 10 worden de geluidskanalen allemaal uitgezet
- op regel 20 en 30 wordt de toonhoogte van kanaal A bepaald
- op regel 40 wordt de toonhoogte van de ruisgenerator bepaald
- op regel 50 wordt het effect voor kanaal A aangezet
- op regel 60 en 70 wordt de tijdsduur van de effectontwikkeling bepaald
- op regel 80 wordt het soort effect gekozen
- uiteindelijk wordt op regel 90 de betreffende kanalen (A geluid en A ruis) aangezet; het bedoelde effect is hoorbaar.

Er is maar één manier om de geluidsgenerator onder de knie te krijgen: proberen en nog eens proberen. Zelfs indien u geen muzikale aanleg heeft, zijn door proberen bijzonder leuke geluidseffecten te realiseren. Denk er bijvoorbeeld eens aan om tesamen met effect nummer 8 de ontwikkelingstijd bijzonder kort te houden en een ruis te genereren.

Tot slot één van de mogelijke effecten: men kan zich bij het volgende geluid een vertrekkende vliegende schotel voorstellen.

```
NEW
10 SOUND 7,&B111111
20 SOUND 8,&B10000
```

```
30 SOUND 11,&B01101010
40 SOUND 12,&B11110110
50 SOUND 13,&B1
60 FOR I=4095 TO 1100 STEP -1
70 SOUND 0,I MOD 256
80 SOUND 1,I\256
90 SOUND 7,&B111110
100 NEXT I
RUN
Ok
```

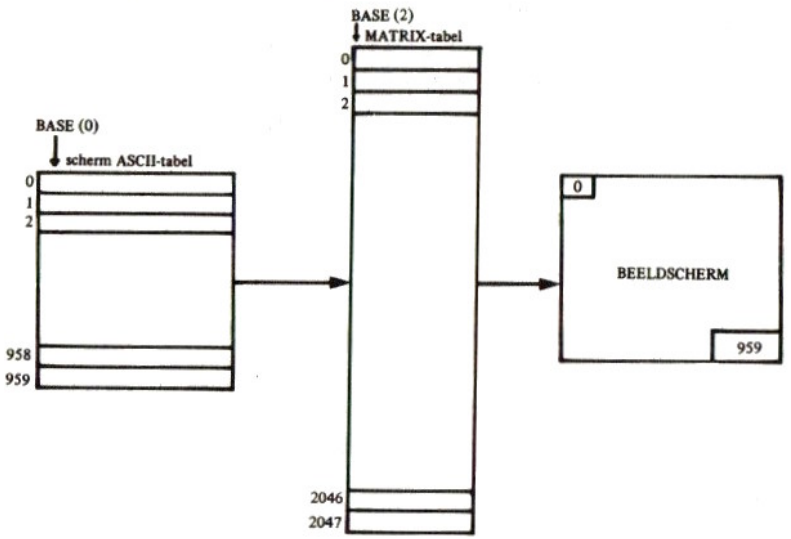
De TMS 9929A video processor wordt tesamen met het 16 kilobyte grote video-geheugen bestuurd door 9 registers. Het MSX-basic bestuurt deze processor ondermeer met behulp van de systeemvariabele BASE waarin vele tabeladressen zijn opgeslagen.

Hieronder volgt de video-geheugen-indeling en de funkties van de diverse registers. Het besturen van de VDP anders dan via de gebruikelijke MSX-basic kommando's is bijzonder moeilijk. Een vrij brede algemene computerkennis alsmede het vermogen om binair te kunnen rekenen is noodzakelijk.

Voor de beginner is het misschien verstandig om dit hoofdstuk voorlopig maar even over te slaan.

16.1 De video-geheugenindeling SCREEN 0

Indien SCREEN 0 actief is, kunnen via BASE de volgende tabellen in het video-geheugen worden gevonden:



Het beeldscherm is verdeeld in 960 posities; 24 regels van elk 40 karakters. Per karakter is in de SCHERM ASCII TABEL de bijbehorende ASCII-waarde (zie hoofdstuk 18) opgenomen. Als zodanig ligt de symbolische inhoud van het scherm vast.
Deze ASCII-waarden dienen nog te worden vertaald in zichtbare karakters. Dit gebeurt in de MATRIXTABEL waarin per karakter 8 bytes zijn opgenomen. In deze 8 bytes kan men zich een 8 bij 8 bits matrix voorstellen; met nullen zijn de vrije ruimten en met enen de in te kleuren ruimten per karakter aangegeven. Een voorbeeld: de hoofdletter A:

de hoofletter A

de hoofletter A in matrix (een gedeelte van de MATRIXTABEL)

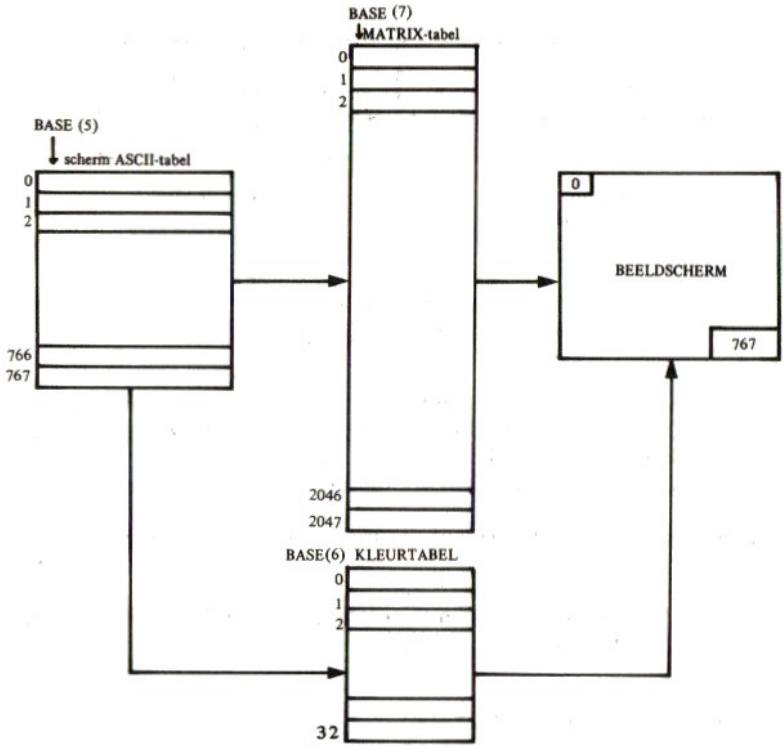
byte	bit 7	6	5	4	3	2	1	0
520	0	0	0	1	0	0	0	0
521	0	0	1	0	1	0	0	0
522	0	1	0	0	0	1	0	0
523	1	0	0	0	0	0	1	0
524	1	1	1	1	1	1	1	0
525	1	0	0	0	0	0	1	0
526	1	0	0	0	0	0	1	0
527	1	0	0	0	0	0	1	0

Via deze tabel weet de computer, met welk visueel karakter de betreffende ASCII-kode correspondeert. Via de beide besproken tabellen kan uiteindelijk het totale beeldscherm door de VDP worden samengesteld.

Merk op dat er geen kleuren kunnen worden gekodeerd; alleen de voor- en achtergrondkleur zijn actief.

16.2 De video-geheugenindeling SCREEN 1

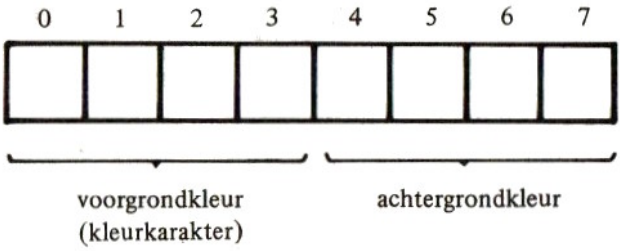
Indien SCREEN 1 actief is, kunnen via BASE de volgende tabellen in het video-geheugen worden gevonden:



Het beeldscherm is verdeeld in 768 posities; 24 regels van elk 32 karakters.

De gang van zaken is dezelfde als in de vorige paragraaf beschreven, echter met toevoeging van één nieuw aspect: de KLEURTABEL.

In de KLEURTABEL bevinden zich 32 bytes. Deze vertegenwoordigen elk een voor- en een achtergrondkleur. De voorgrondkleur bevindt zich in de eerste vier bits, de achtergrondkleur in de tweede vier bits:



vanaf &B0000(0)
t/m &B1111(15)

Het eerste byte uit de KLEURTABEL verzorgt de kleur van tekens met ASCII-kode 0 tot en met 7. Het tweede byte verzorgt de kleur van de tekens met ASCII-kode 8 ... 15 etcetera. Zo zijn er 32 kleurbytes nodig om de gehele ASCII-set van kleuren te kunnen voorzien.

Uiteindelijk stelt de VDP vanuit de drie genoemde tabellen het beeldscherm samen.

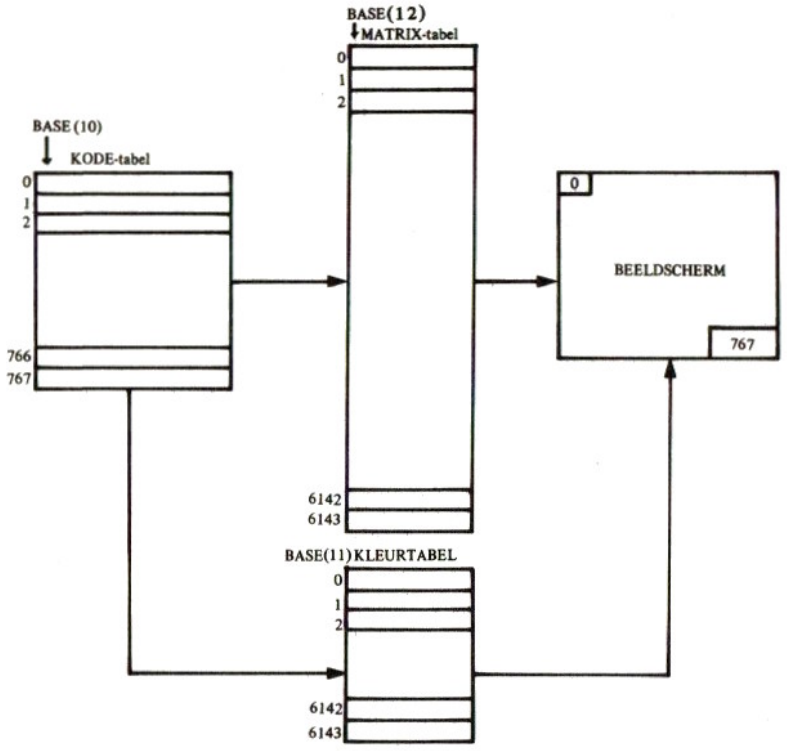
Een klein voorbeeld: het volgende programma verandert de letter A in de matrixtabel van SCREEN 0 zodanig dat deze wordt geinverteerd.

```
NEW
10 SCREEN 0:FOR I=520 TO 527
20 VPOKE BASE(2)+I,255-VPEEK(BASE(2)+I)
30 NEXT I
RUN
Ok
```

Probeer nu de letter A naast de andere letters; de A blijkt geinverteerd te zijn (de donkere plaatsen zijn nu licht en vice versa).

16.3 De video-geheugenindeling SCREEN 2

Indien SCREEN 2 actief is, kunnen via BASE de volgende tabellen in het video-geheugen worden gevonden.



Het scherm is verdeeld in 768 posities; 24 regels van elk 32 karakters. Per karakter is een kodering van 0 tot en met 255 in de KODETABEL opgenomen. Deze kodes zijn nu geen ASCII-kodes meer maar interne koderingen.

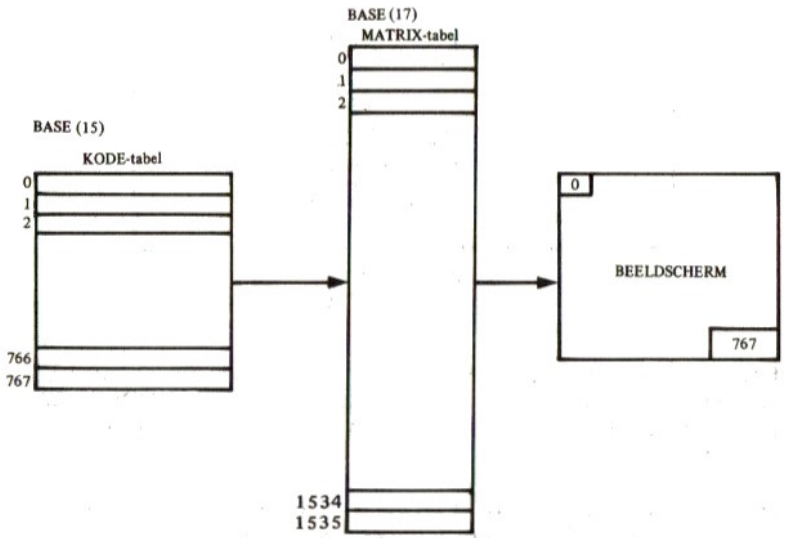
De eerste groep van 256 kodes verwijzen naar de eerste 256*8 bytes in de MATRIXTABEL, de tweede groep van 256 kodes verwijzen naar de tweede groep van 256*8 bytes in de MATRIXTABEL en de derde groep van 256 kodes verwijzen naar de derde groep van 256* bytes in de MATRIXTABEL. Bij het uitvoeren van grafische kommando's wordt de KODETABEL gevuld en worden de bijbehorende matrices van elke 8 bytes bij 8 bits gevuld. In de MATRIXTABEL ontstaat dus een zeer gefragmenteerde beeldschermopbouw.

Parallel aan de MATRIXTABEL loopt de KLEURTABEL. Per byte wordt een voorgrondkleur (bit 0, 1, 2 en 3) en een achtergrondkleur (bit 4, 5, 6 en 7) gekodeerd zoals in de vorige paragraaf besproken. Hierdoor kan per groep van acht puntjes horizontaal een voor- en een achtergrondkleur worden gespecificeerd.

Via de drie besproken tabellen stelt de VDP het uiteindelijke beeldscherm samen.

16.4 De video-geheugenindeling SCREEN 3

Indien SCREEN 3 actief is, kunnen via BASE de volgende tabellen in het video-geheugen worden gevonden:



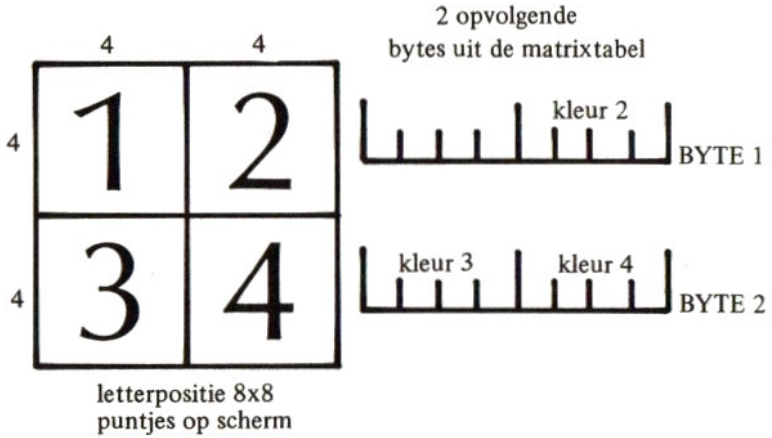
Het beeldscherm is verdeeld in 768 posities; 24 regels van elk 32 karakters. Per karakter is een kodering van 0 tot en met 255 in de kodetabel opgenomen. Deze kodes zijn geen ASCII-kodes maar interne kodering die naar de MATRIXTABEL verwijzen. De kodetabel verwijst op de volgende manier naar de MATRIXTABEL:

Stel: A = adres in KODETABEL
V = adres in MATRIXTABEL
B = inhoud van het betreffende byte in de KODETABEL

dan geldt:

```
V=8*B+2*(A\32)-8*(A\128)
```

In de matrixtabel zijn op adres V en V+1 twee bytes per beeldschermpositie opgenomen. In deze twee bytes zijn de kleuren van een 8 bij 8 beeldschermpositie als volgt opgenomen:

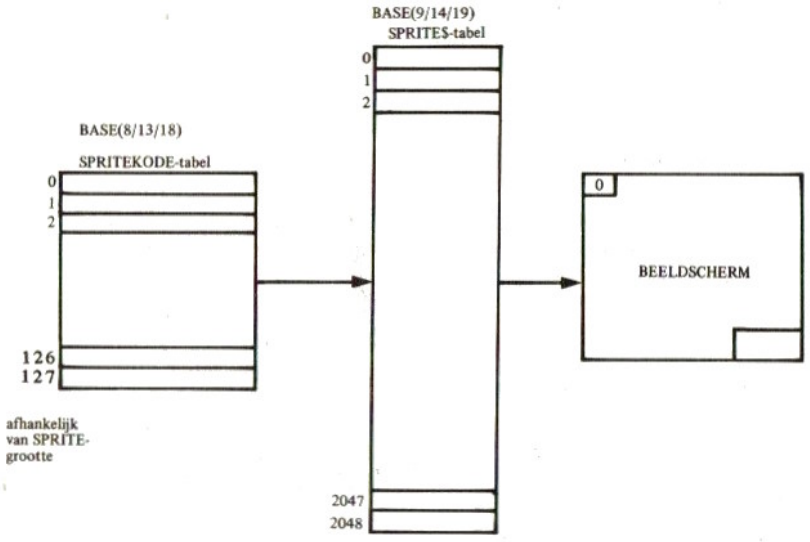


In de matrixtabel ontstaat dus een zeer gefragmenteerde beeldschermopbouw.

Via de twee besproken tabellen stelt de VDP het uiteindelijke beeldscherm samen.

16.5 De video-geheugenindeling in verband met sprites

In verband met sprites kunnen via BASE de volgende tabellen in het video-geheugen worden gevonden:

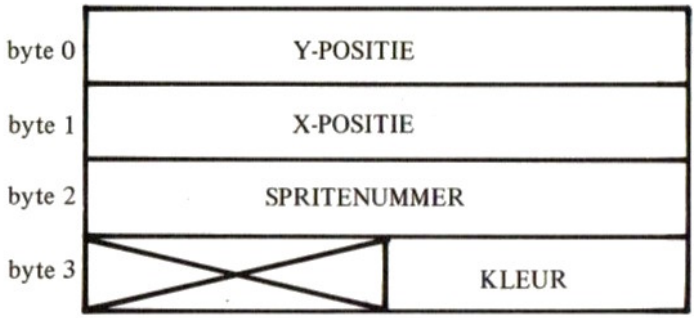


De BASE-adressen zijn afhankelijk van het geactiveerde SCREEN:

SCREEN 1 ... BASE(8) en BASE(9)
SCREEN 2 ... BASE(13) en BASE(14)
SCREEN 3 ... BASE(18) en BASE(19)

In de SPRITEKODETABEL zijn per sprite vier bytes gereserveerd die achter elkaar liggen. De kodering van sprite(0) begint dus op adres 0 in de tabel terwijl de kodering van de tweede sprite op adres 4 begint etcetera.

Per sprite zijn de volgende gegevens in de SPRITEKODETABEL opgenomen:



In de SPRITE$TABEL staan de sprites daadwerkelijk gekodeerd. Voor een 16 bij 16 sprite is per sprite een ruimte van 32 bytes gereserveerd; voor een 8 bij 8 sprite blijft de ruimte beperkt tot 8 bytes. De SPRITE$TABEL is de rechtstreekse geheugenplaats van de systeemvariabele SPRITE$; de gegevens kunnen ook op een dergelijke wijze worden teruggevonden.

Via beide tabellen stelt de VDP uiteindelijk het beeldscherm samen.

16.6 De VDP-registers

De VDP-registers kunnen als volgt worden onderverdeeld:

VDP(0)	0	0	0	0	0	0	A	D
VDP(1)	1	0	E	B	C	R	S	M
VDP(2)	0	0	0	0	kodetabel/1024			
VDP(3)				kleurtabel/64				
VDP(4)	0	0	0	0	matrixtabel/2048			
VDP(5)	0			spritekodetabel/128				
VDP(6)	0	0	0	0	0	sprite$tabel/2048		
VDP(7)		tekst kleur 1				tekst kleur 2		
VDP(8)	T	N	V			vijfde sprite		

Verklaring:

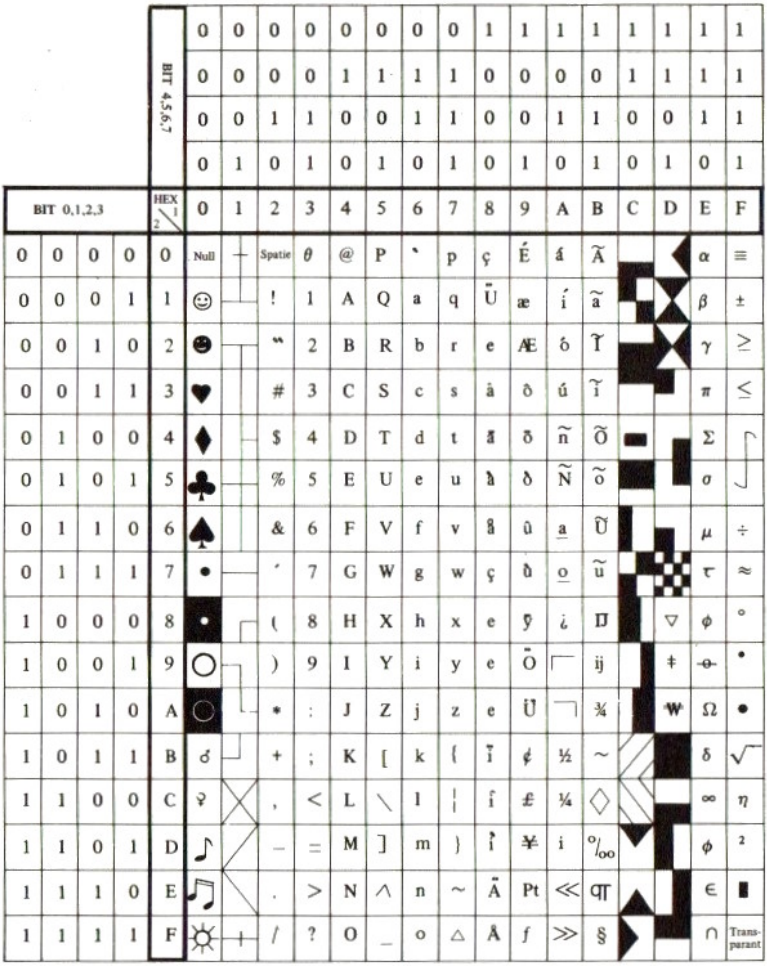
ABC	000=SCREEN 1 001=SCREEN 3 010=SCREEN 0 100=SCREEN 2
D	0=er is een externe VDP-input 1=er is geen externe VDP-input
E	0=geen VDP-interrupt 1=er is een VDP-interrupt
R	*niet vrijgegeven*
S	0=8x8 sprites actief 1=16x16 sprites actief
M	0=sprites niet vergroot 1=sprites wèl vergroot
KODETABEL	het adres van de op het moment actieve kodetabel, gedeeld door 1024 (een kodetabeladres moet dus altijd een veelvoud van 1024 zijn)
KLEURTABEL	het adres van de op het moment actieve kleurentabel, gedeeld door 64 (een kleurentabeladres moet dus altijd een veelvoud van 64 zijn)
MATRIXTABEL	het adres van de op het moment actieve matrixtabel, gedeeld door 2048 (een matrixtabeladres moet dus altijd een veelvoud van 128 zijn)
SPRITEKODETABEL	het adres van de op het moment actieve spritecodetabel, gedeeld door 128 (een spritekodetabeladres dient dus altijd een veelvoud van 128 te zijn)
SPRITE$TABEL	het adres van de op het moment actieve spritepatronentabel, gedeel door 2048 (een spritepatronentabeladres dient dus altijd een veelvoud van 2048 te zijn)

TEKSTKLEUR1	de op het moment actieve voorgrondkleur in SCREEN 0
TEKSTKLEUR2	de op het moment actieve achtergrondkleur in SCREEN 0
T	0 = VDP gereset of statusregister gelezen 1 = einde van een raster-scan
N	0 = statusregister gelezen of geen sprite-botsing 1 = sprite-botsing
V	0 = niet meer dan 4 sprites op één horizontale lijn 1 = wél meer dan vier sprites op één horizontale lijn
VIJFDE SPRITE	nummer van de vijfde sprite.

De registers 0...7 mogen gelezen en geschreven worden, ondermeer via de systeemvariabele VDP in MSX-basic. Register 8 is een statusregister en mag alleen worden gelezen.

				b_7 b_6 b_5	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1		0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	*P*		*p*
0	0	0	1	1	SOH	DCI	!	1	*A*	*Q*	*a*	*q*
0	0	1	0	2	STX	DC2	"	2	*B*	*R*	*b*	*r*
0	0	1	1	3	ETX	DC3	#	3	*C*	*S*	*c*	*s*
0	1	0	0	4	EOT	DC4	$	4	*D*	*T*	*d*	*t*
0	1	0	1	5	ENQ	NAK	%	5	*E*	*U*	*e*	*u*
0	1	1	0	6	ACK	SYN	&	6	*F*	*V*	*f*	*v*
0	1	1	1	7	BEL	ETB	'	7	*G*	*W*	*g*	*w*
1	0	0	0	8	BS	CAN	(	8	*H*	*X*	*h*	*x*
1	0	0	1	9	HT	EM	)	9	*I*	*Y*	*i*	*y*
1	0	1	0	*A*	LF	SUB	*	:	*J*	*Z*	*j*	*z*
1	0	1	1	*B*	VT	ESC	+	;	*K*	[	*k*	{
1	1	0	0	*C*	FF	FS	,	<	*L*	\	*l*	:
1	1	0	1	*D*	CR	GS	-	=	*M*	]	*m*	}
1	1	1	0	*E*	SO	RS	.	>	*N*	^	*n*	~
1	1	1	1	*F*	SI	US	/	?	*O*	—	*o*	DEL

In de bovenstaande tabel zijn de standaard ASCII karakters met hun bijbehorende binaire en hexadecimale waarden opgenomen. Door de vermelde konstanten in de CHR$-funktie op te nemen, kunnen de afgebeelde of omschreven karakters (of funkties) worden gegenereerd. De MSX-computer werkt met een afgeleide vorm van deze standaard tabel die op ondergeschikte punten afwijkt.



BIT 0,1,2,3				HEX ↓ / →	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
				BIT 4,5,6,7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
					0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
					0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
					0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	0	0	0	0	Null		Spatie	θ	@	P	`	p	ç	É	á	Ã			α	≡
0	0	0	1	1	☺		!	1	A	Q	a	q	Ü	æ	í	ã			β	±
0	0	1	0	2	☻		"	2	B	R	b	r	e	Æ	ó	Ĩ			γ	≥
0	0	1	1	3	♥		#	3	C	S	c	s	à	ð	ú	ĩ			π	≤
0	1	0	0	4	♦		$	4	D	T	d	t	ä	õ	ñ	Õ			Σ	⌠
0	1	0	1	5	♣		%	5	E	U	e	u	à	ð	Ñ	õ			σ	⌡
0	1	1	0	6	♠		&	6	F	V	f	v	å	û	a	Ũ			μ	÷
0	1	1	1	7	•		'	7	G	W	g	w	ç	ù	o	ũ			τ	≈
1	0	0	0	8	◘		(	8	H	X	h	x	e	ÿ	¿	Ð		▽	φ	°
1	0	0	1	9	○		)	9	I	Y	i	y	e	Ö	⌐	ij		ǂ	-o-	•
1	0	1	0	A	◙		*	:	J	Z	j	z	e	Ü	¬	¼		₩	Ω	●
1	0	1	1	B	♂		+	;	K	[	k	{	ĩ	¢	½	~			δ	√
1	1	0	0	C	♀		,	<	L	\	l	¦	î	£	¼	◇			∞	η
1	1	0	1	D	♪		-	=	M	]	m	}	ì	¥	i	‰			φ	²
1	1	1	0	E	♫		.	>	N	^	n	~	Ã	Pt	≪	¶			∈	∎
1	1	1	1	F	☼		/	?	O	_	o	△	Å	f	≫	§			∩	Trans-parant

In de bovenstaande tabel zijn alle MSX-karakters opgenomen met hun bijhorende binaire en hexadecimale waarden. Door de vermelde kon-

stanten in de CHR$-funktie op te nemen, kunnen de betreffende karakters worden gegenereerd. Een uitzondering vormen de karakters met een hexadecimale waarde kleiner dan &H20 (decimaal 32, binair &B00100000). Deze worden alleen gegenereerd met CHR$ indien voorafgegaan door een CHR$(1). Bij het kodenummer van het bepaalde karakter dient dan de waarde &H40 (decimaal 64, binair &B01000000) te worden opgeteld. Zo drukt PRINT CHR$(1)+CHR$(&H41) het grafische symbool onder kodenummer 1 af (eerste kolom, tweede karakter). Deze tabel is direkt afgeleid van de standaard ASCII-tabel.

19 GERESERVEERDE SLEUTELWOORDEN

Deze sleutelwoorden zijn door MSX-basic gereserveerd en mogen niet als vrije variabele-namen worden gebruikt.

ABS	DEFDBL	INP	NOT	SIN
AND	DEFSNG	INPUT	OCT$	SOUND
AS	DEFSTR	INPUT$	OFF	SPACE$
ASC	DEFUSR	INSTR	ON	SPC(
ATN	DELETE	INT	OPEN	SPRITE
ATTR$	DIM	INTERVAL	OR	SPRITE$
AUTO	DRAW	KEY	OUT	SQR
BASE	DSKF	KILL	PAD	STEP
BEEP	DSKI$	LEFT$	PAINT	STICK
BIN$	DSKO$	LEN	PDL	STOP
BLOAD	ELSE	LET	PEEK	STR$
BSAVE	END	LFILES	PLAY	STRIG
CALL	EOF	LINE	POINT	STRING$
CDBL	EQV	LIST	POKE	SWAP
CHR$	ERASE	LLIST	PORT	TAB(
CINT	ERL	LOAD	POS	TAN
CIRCLE	ERR	LOC	PRESET	THEN
CLEAR	ERROR	LOCATE	PRINT	TIME
CLOAD	EXP	LOF	PSET	TO
CLOSE	FIELD	LOG	PUT	TROFF
CLS	FILES	LPOS	READ	TRON
COLOR	FIX	LPRINT	REM	USING
CONT	FN	LSET	RENUM	USR
COPY	FOR	MAXFILES	RESTORE	VAL
COS	FPOS	MERGE	RESUME	VARPTR
CSAVE	FRE	MID$	RETURN	VDP
CSNG	GET	MKD$	RIGHT$	VPEEK
CSRLIN	GOSUB	MKI$	RND	VPOKE
CVD	GOTO	MOD	RSET	WAIT
CVI	GO TO	MOTOR	RUN	WIDTH
CVS	HEX$	MKS$	SAVE	XOR
DATA	IF	NAME	SCREEN	OUTPUT
DEF	IMP	NEW	SET	APPEND
DEFINT	INKEY$	NEXT	SGN	RANDOM

AANTEKENINGEN

AANTEKENINGEN

AANTEKENINGEN

AANTEKENINGEN

AANTEKENINGEN

Andere MSX-uitgaven

MSX DISK handboek voor iedereen, door A.C.J. Groeneveld
ISBN 90 6398 407 3

MSX Zakboekje, door Wessel Akkermans. Alle belangrijke gegevens voor zowel de BASIC- als machinetaal-programmeurs, voor zover mogelijk in de vorm van overzichten en tabellen
ISBN 90 6398 888 5

MSX BASIC leerboek deel 1, door Wessel Akkermans en Piet den Heijer. Informatie over MSX-hardware en leren programmeren
ISBN 90 6398 649 1

MSX BASIC leerboek deel 2, door Wessel Akkermans en Piet den Heijer. Doorgaan met leren programmeren en speciale aandacht voor kleur, grafieken, geluid/muziek en joy-sticks
ISBN 90 6398 769 2

MSX DOS leerboek, door Wessel Akkermans en Piet den Heijer. Het manipuleren met strings, het bewerken en opslaan van gegevens op cassette en schijf
ISBN 90 6398 519 3

SOFTWARE PLUS in MSX

INTROTAPE MSX, door A.C.J. Groeneveld. Begeleid met instrukties om de computer aan te sluiten en de tape te laden, wordt MSX op een vriendelijke en onderwijzende manier vanuit nul bij de gebruiker geïntroduceerd. Na het doorwerken van deze software is de gebruiker zelf in staat MSX-basic programma's te schrijven
ISBN 90 6398 148 1

TEKSTVERWERKER in MSX, door Ton Weijters
ISBN 90 6398 189 9

BASIC
handboek voor iedereen

Met het MSX-basic, dat door de systeemsoftware-expert MICROSOFT is ontwikkeld, is er eindelijk een einde gekomen aan het probleem van de uitwisselbaarheid van software op microcomputers. Tussen de nu meer dan DRIE HONDERD verschillende BASIC-dialecten die het levenslicht al hebben gezien, is er eindelijk een standaard opgestaan. En dat werd tijd!
De letters MSX staan voor MicroSoft eXtended basic. De MSX-taal is gebaseerd op het reeds jaren in algemeen zijnde Microsoft basic (MBASIC) maar is daarbij voorzien van vele uitbreidingen. MSX voorziet bijvoorbeeld ook in de mogelijkheid om een driestemmige toongenerator aan te sturen, om met sprites te tekenen, om met high resolution graphics te werken, etcetera. Met zijn ongeveer 150 verschillende sleutelwoorden is MSX een basic zonder weerga en wordt het reeds door vele onafhankelijke microcomputerfabrikanten als standaard taal gevoerd.
Goede documentatie loopt vooral in een land als Nederland vaak achter de feiten aan. Toen reeds diverse MSX-computers te koop waren, was een gedegen MSX-handboek zelfs bij de verantwoordelijke importeurs nog niet verkrijgbaar.
Deze handleiding omvat een volledige behandeling van het MSX-basic in het Nederlands. Het handboek geeft een antwoord op elke vraag die een programmeur, van welke scholing ook, over het MSX-basic zou kunnen stellen. De volledige syntaxisbehandeling rekent af met onzekerheden of een bepaalde schrijfwijze nu wel of niet is toegestaan. De duidelijke beschrijving geeft per sleutelwoord aan, welke de funktie hiervan is. De laatste mogelijk nog aanwezig onduidelijkheden worden vervolgens door de opgenomen, zinvolle voorbeelden weggenomen.

ISBN 90 6398 100 7