MSX

LEERBOEK BASIC

DEEL 1

WESSEL AKKERMANS/PIET DEN HEIJER

MSX LEERBOEK BASIC DEEL 1 AKKERMANS/DEN HEIJER

MSX BASIC leerboek
deel 1

MSX LEERBOEK BASIC DEEL 1

WESSEL AKKERMANS/PIET DEN HEIJER

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

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Akkermans, Wessel

MSX leerboek / Wessel Akkermans, Piet den Heijer. – Oosterend: Stark-Texel
Dl. 1
ISBN 90 6398 649 1
SISO 365.3 UDC 681.3
Trefw.: MSX (computer)

. . . . . .

1e druk 1985
ISBN 90 6398 649 1

VOORWOORD

MSX-BASIC kent ruim 150 verschillende statements en functies. Om al die statements en functies gedegen te behandelen, zou al gauw een boek van zo'n 500 of meer pagina's ontstaan. Een boek van dergelijke omvang zou, ons inziens, niet zo handelbaar zijn. Daarom hebben wij besloten het totale MSX-BASIC gebied op te splitsen in drie delen.

Bij die opsplitsing zijn wij uitgegaan van de volgende criteria. Een leerboek is voor beginnende computergebruikers. De meeste mensen die met de computer hobby beginnen, zullen beginnen met alleen een basis machine. Pas later zullen zij gaan uitbreiden en randapparatuur aan hun basis systeem toevoegen. Daarnaast zijn er verschillende toepassingsgebieden. De een zal geinteresseerd zijn in serieuze toepassingen, de ander in spelletjes.

Met deze overwegingen in het achterhoofd, kwamen wij tot de volgende indeling:

Deel 1 - Inleiding tot het programmeren, met alle BASIC-statements die nodig zijn voor het maken van serieuze toepassingen, waarbij geen gebruik van de grafische mogelijkheden van de computer wordt gemaakt.

Deel 2 - Alle BASIC-statements die nodig zijn om spelletjes te programmeren. Hierbij wordt vooral aandacht besteed aan de grafische- en geluidsmogelijkheden van de computer.

Deel 3 - Het gebruik van schijveneenheden, met alle daarbij behorende BASIC-statements. In dit deel zal ook uitgebreid worden ingegaan op bestanden, de organisatie daarvan en de manier waarop toegang tot de gegevens uit die bestanden kan

worden verkregen.

Bij het lezen van dit boek zal het u opvallen dat, steeds wanneer er een nieuw statement wordt behandeld, dit statement eenmaal vet wordt afgedrukt. Bij het doorbladeren van het boek, kan op die manier een gezocht statement gemakkelijker worden teruggevonden. Om het boek tot lang na bestudering bruikbaar te houden, hebben wij bovendien besloten een trefwoordenregister op te nemen. Alle belangrijke begrippen en alle behandelde statements zult u daarin terugvinden.

Wij hopen dat u met deze serie MSX-BASIC leerboeken uw computer op een plezierige manier leert kennen. Alle behandelde theorie wordt onmiddellijk in de praktijk gebracht. Wij raden u aan om alle gegeven voorbeelden direct op uw computer uit te proberen. Probeer tevens uw eigen voorbeelden te maken. Van de fouten die u daarbij maakt zult u veel leren. Voor degenen die graag meer oefenstof willen hebben, is een boekje met vragen en opdrachten beschikbaar. Bij elk deel van MSX-BASIC is zo'n boekje beschikbaar. Rest ons nu alleen nog u veel plezierige programmeeruurtjes te wensen.

Augustus 1985

De auteurs

INHOUDSOPGAVE

1	**Standaard Hardware en Software**	11
1.1	De Hardware van de MSX-computer	15
1.2	Beeldschermen	18
1.3	Afdrukeenheden	19
1.4	Het geheugen	21
2	**Voorbereiding tot het programmeren**	25
2.1	BASIC-statements	27
2.2	Analyse en ontwerp	29
2.3	Probleemstelling	30
2.4	Stroomdiagrammen	30
2.5	Het programma	33
2.6	Testen	34
3	**Programma's invoeren en corrigeren**	36
4	**Werken met constanten en variabelen**	46
4.1	Constanten	48
4.2	Alfanumerieke functies	54
4.3	Conversie functies	57
4.4	Declareren van variabelen	57
5	**Het toekennen van waarden aan variabelen**	60
5.1	Ingave statements	63
5.2	Ingave functies	68
6	**Afdrukken van gegevens**	71
6.1	Beeldschermmodes	71
6.2	Schoonmaken van het scherm	75
6.3	Instellen van de regellengte	76

6.4 Het werken met tabulatorstoppen 77
6.5 Het positioneren van de cursor 78
6.6 Werken met spaties 79
6.7 Reeksen met dezelfde tekens 81
6.8 Het werken met codes 82
6.9 PRINT USING 87
6.9.1 Hoe kunnen getallen worden afgedrukt 87
6.9.2 Manipuleren met strings 95

7 Sprongen 98

7.1 Het statement GOTO 98
7.2 Het IF...GOTO statement 100
7.3 Menu 101
7.4 Omrekenen temperatuurschalen 104
7.5 Subroutines 107
7.6 Programma "Menu met rekenfuncties" 109
7.7 Het statement ON...GOSUB 110

8 Lussen en opslaan 111

8.1 Relationele- en logische operators 111
8.2 IF...THEN statement 116
8.3 FOR...NEXT statement 118
8.4 Functie INT 122
8.5 Opslaan van programma's op cassette 123
8.6 Het commando CSAVE 124
8.7 Het commando CLOAD 126
8.8 Het commando SAVE 130
8.9 Het commando LOAD 131
8.10 Het commando MERGE 132
8.11 MOTOR ON/OFF 135

9 Tijd en fouten 137

9.1 De tijd 137
9.2 Controle op de ingevoerde tijd 142
9.3 Foutafhandeling 146
9.4 Foutzoeken in programma's 152

9.5 Het genereren van een piepsignaal 153
9.6 Het stoppen en vervolgen van een programma 154

10 Het programmeren van functietoetsen 156

10.1 Functietoetsen F1 t/m F10 156
10.2 Veranderen van de inhoud van functietoetsen 158
10.3 Onderbrekingen d.m.v. functietoetsen 160
10.4 KEY (functietoets) ON/OFF/STOP 164
10.5 Wat doet STOP voor ons? 165
10.6 ON STOP GOSUB 166

11 Werken met vaste gegevens 169

11.1 READ, DATA en RESTORE 170
11.2 Datum controle 175

12 Opslaan van gegevens 179

12.1 Werken met arrays 180
12.1.1 1-dimensionale arrays 181
12.1.2 2-dimensionale arrays 183
12.1.3 Het statement ERASE 186
12.2 Het werken met bestanden op cassette 188
12.2.1 Schrijven van gegevens op cassette 190
12.2.2 Lezen van gegevens van cassette 192
12.3 Hoofdprogramma "adressenbestand" 194
12.4 Werken met het adressenbestand 195
12.5 Functie INSTR 197

Appendix A - het AGENDA-programma 202

Trefwoordenlijst 225

1 Standaard Hardware en Software

Voor diegenen onder u die zich bij het lezen van dit boek voor de eerste keer met computers bemoeien, zal de titel van dit hoofdstuk al de nodige onduidelijkheid in zich bergen. Bovendien zal standaardisatie van hardware en software u niet veel zeggen. Laten we daarom beginnen met een korte verklaring van de twee begrippen, om daarna eens naar het belang van standaardisatie te kijken.

Hardware

Met het woord "hardware" wordt over het algemeen de computer-apparatuur bedoeld. Deze apparatuur kan bestaan uit de computer zelf, maar ook uit de op die computer aan te sluiten randapparaten. Enkele voorbeelden van randapparaten zijn:

Een beeldscherm. Als beeldscherm wordt meestal een normale huis tuin en keuken televisie gebruikt. Het is echter ook mogelijk om een speciale video-monitor te gebruiken. Hiermee wordt een betere beeldkwaliteit bereikt.

Een afdrukeenheid (er zijn vele soorten, in vele prijsklassen) waarmee gegevens op papier kunnen worden afgedrukt. Later komen we daar nog eens op terug.

Een cassetterecorder, waarop programma's en gegevens kunnen worden opgeslagen, om er later weer vanaf te worden gelezen.

Joy-sticks, waarmee het mogelijk wordt om actiespelletjes op de computer te spelen.

Kortom, alle tastbare delen van een computersysteem (computer + randapparaten) worden aangeduid met de naam "hardware".

Software

De programma's, waarmee de computer (de hardware) wordt verteld wat te doen, worden de "software" genoemd. Programma's kunnen in velerlei vormen voorkomen. Programma's kunnen worden opgeslagen op een audiocassette, op een flexibele schijf of in een ROM. De taal waarin die programma's zijn geschreven kan bijvoorbeeld BASIC zijn. Vrijwel alle huiscomputers kunnen in BASIC geschreven programma's uitvoeren. "Uitvoeren" wil in dit verband zeggen, de computer doet wat het programma zegt wat moet worden gedaan. Anders gezegd, de hardware doet wat de software zegt.

Standaardisatie

Tot nu toe ontwikkelde iedere computerfabrikant niet alleen zijn eigen computer, maar ook zijn eigen dialect van de programmeertaal BASIC. Hierdoor werd het mogelijk dat er in de loop van de laatste jaren ongeveer 400 verschillende dialecten van de programmeertaal BASIC ontstonden.

Nu was er een goede reden voor die fabrikanten om steeds maar nieuwe dialecten te maken. De ontwikkeling van de hardware maakte, dat de computer steeds meer mogelijkheden kreeg, voor steeds minder geld. Uit concurrentie-overwegingen was het interessant voor de fabrikant om een nieuwe computer van de nieuwste hardware te voorzien. Die nieuwe hardware moest wel door een programma kunnen worden aangestuurd, dus moest er een toevoeging aan een bestaande BASIC-versie worden gemaakt.

Hoewel deze ontwikkelingen enerzijds gunstig waren voor de gebruikers, kleefden er toch ook een aantal nadelen aan. De voordelen waren dat de huiscomputers zich snel konden ontwikkelen. Al gauw kwam de prijs van een redelijke computer

ronde de 1000 gulden te liggen. De nadelen zijn vooral dat de gebruiker programma's, die voor een bepaalde computer zijn geschreven, niet op een ander type computer kan gebruiken. Dit is natuurlijk ook vervelend voor programmeurs. Voor ieder type computer moet nu een ander programma worden geschreven. Het gevolg is dat er per programma maar een beperkt aantal copieen zullen worden verkocht. Dit houdt dan in dat de prijs van die programma's hoger komt te liggen. En natuurlijk is het de eindgebruiker, u dus, die de prijs daarvoor moet betalen.

Een aantal Japanse computerfabrikanten hebben voorgaande problemen goed bestudeerd. Daarna hebben ze onderlinge afspraken gemaakt over zowel de hardware als de software. Het doel was natuurlijk, dat programma's, geschreven voor de computer van merk A, ook zouden werken op de computer van merk B. Om dit te bereiken werden een aantal componenten binnen in de computer, de manier waarop de computer met randapparatuur wordt verbonden, en de te gebruiken taal voor het programmeren van de computer in detail vastgelegd. Deze standaard, waarvoor de firma Microsoft uit de Verenigde Staten de BASIC-taal schreef, werd MSX genoemd. Alleen computers die absoluut aan deze standaard voldoen, mogen het MSX embleem dragen.

Toen de Nederlandse elektronica gigant Philips ook besloot om huiscomputers onder de MSX-vlag te gaan produceren, nadat er ook al enkele Amerikaanse fabrikanten toe waren overgegaan, werd duidelijk dat MSX inderdaad een standaard zou worden die over heel de wereld zou worden gevoerd. We zullen dus in de toekomst kunnen rekenen op uitwisselbaarheid van zowel programma's als randapparatuur. Zowel voor de fabrikanten als voor de gebruiker kan dit grote voordelen opleveren.

Voor de fabrikanten van de programma's betekent het immers dat er een groot aantal potentiele kopers van hun producten is. Zij zullen dus graag programma's voor MSX-computers maken. Bovendien zullen deze programma's, juist door dat grote aantal kopers, laag geprijsd kunnen zijn. Het valt dan ook te verwachten, dat er een groot aantal programma's voor

redelijke prijzen te verkrijgen zal zijn.

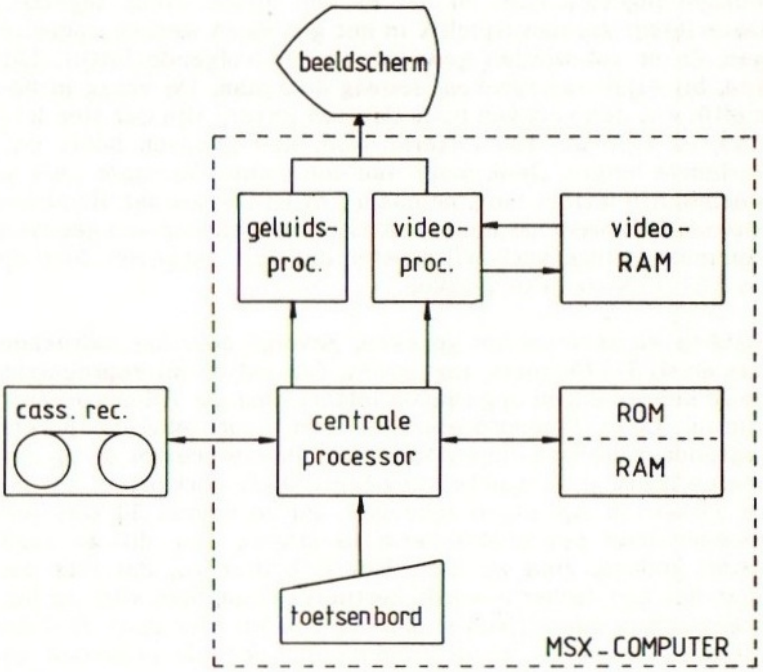
Voor de fabrikanten van randapparaten, zoals printers, flexibele schijven, joy-sticks, etc., moet het grote aantal in gebruik zijnde MSX-computers toch ook aanleiding zijn om snel met nieuwe produkten te komen. Ook hier geldt weer, dat de te verwachten grote aantallen de prijs per stuk zullen drukken.

In beide hiervoor omschreven situaties is de gebruiker degene die profijt trekt. Bent u nog niet, of nog maar net, begonnen met een MSX-computer, dan zult u nu al een redelijke keuze uit de verschillende hardware en software componenten kunnen maken. Hebt u al enige tijd geleden een aantal componenten aan uw systeem toegevoegd, dan zult u nu waarschijnlijk opmerken dat deze componenten op dit moment al goedkoper zijn dan toen u ze kocht. De vraag is natuurlijk of je moet wachten tot de prijzen zijn gedaald tot het absolute minimum. Dit is erg afhankelijk van uw wensen. Indien u graag wilt tekstverwerken, dan hebt u behalve de computer zelf ook nog een tekstverwerkingsprogramma en een afdrukeenheid nodig. Misschien dat het programma en de "printer" volgend jaar wel dertig procent goedkoper zijn dan nu. Zou u daar echter op wachten, dan kunt u tot die tijd nog steeds geen tekst verwerken. De vraag is nu of die prijsdaling wel opweegt tegen het lange wachten. Bovendien zit er een goede kans in dat de prijs van het product waarop u een jaar hebt zitten wachten, het volgende jaar weer met een aantal procenten is gedaald. Wilt u er dan nog een jaar op wachten? Of begint u toch maar liever met tekstverwerken?

U ziet, er is nauwelijks een algemeen advies te geven. Ieder heeft zijn eigen specifieke wensen. Ieder zal ook zijn eigen weg moeten volgen. Om echter met enige kennis van zaken een compleet computersysteem te kunnen opbouwen, is het wel goed om iets meer van de opbouw van een computer en van de verschillende randapparaten te weten. We zullen daarom nu eens wat nader ingaan op de hardware.

1.1 De Hardware van een MSX-computer

Hoewel het niet in de bedoeling ligt om u op te leiden tot een computer-hardware-specialist, kan het u toch goed van pas komen iets meer van de computer te kennen dan alleen de programmeertaal waarin u de computer dingen voor u laat doen. We zullen daarom de hoofddelen waaruit de MSX-computer bestaat wat nader bezien. Deze hoofddelen, en de manier waarop ze onderling samenhangen, zijn weergegeven in afbeelding 1-1.



Afb. 1-1 Blokschema van een MSX-computer.

De belangrijkste delen van de computer zijn de microprocessor en het geheugen. Het geheugen wordt gebruikt om er programma's en gegevens in op te slaan. Is een programma eenmaal in het geheugen aanwezig, dan kan de microprocessor de opdrachten uit dat programma lezen en uitvoeren. Een opdracht zou bijvoorbeeld kunnen zijn: "Druk een gegeven dat via het toetsenbord zal worden ingetypt af op het beeldscherm".

Om deze opdracht te kunnen uitvoeren zal de microprocessor moeten kijken of er gegevens op het toetsenbord worden ingetikt, zoniet, dan zal deze wachten totdat er wel gegevens worden ingetikt. Stel nu dat er een letter wordt ingetikt. Deze letter zal dan tijdelijk in het geheugen worden opgeslagen, en er zal worden gewacht op een volgende letter. Dit kan, bij wijze van spreken, eeuwig doorgaan. De vraag in namelijk wat een gegeven is. Is dat een letter, zijn dat tien letters, of zijn dat 100 letters? Nee, een gegeven heeft een variabele lengte. Denk maar aan een naam. De naam JAN is slechts drie letters lang, de naam JACOBUS bestaat uit zeven letters. Hoe weet de microprocessor nu wanneer een gegeven helemaal is ingetypt? Wij moeten dat zelf aangeven door op de RETURN-toets te drukken.

Hebben we eenmaal het gegeven, gevolgd door het indrukken van de RETURN-toets, ingegeven, dan zal de microprocessor de in het geheugen opgeslagen letters naar de Videoprocessor sturen. Deze videoprocessor is een voor MSX-computers gestandaardiseerde chip. Zodra deze videoprocessor de op het beeldscherm af te drukken gegevens heeft ontvangen, zal hij ze opslaan in zijn eigen geheugen, om ze daarna 50 keer per seconde naar het beeldscherm te sturen. Dat dit zo vaak wordt gedaan, zien we niet. Het is echter zo, dat iets dat naar het beeldscherm wordt gestuurd, maar heel kort op het beeldscherm zou blijven staan. We zouden niet eens de kans krijgen om het te lezen. Vandaar dat de videoprocessor de gegevens 50 keer per seconde naar het beeldscherm stuurt, net zo lang tot hij van de microprocessor te horen krijgt dat het niet meer hoeft.

Met het versturen van de gegevens naar de videoprocessor was de microprocessor klaar met zijn opdracht. De microprocessor zal nu dan ook in het geheugen kijken of er nog meer opdrachten in staan. Vindt hij een volgende opdracht, dan zal hij die ook uitvoeren. Stel dat er inderdaad een volgende opdracht in het geheugen staat, en dat die opdracht luidt: "Speel het liedje Vader Jacob".

De microprocessor zal de melodie, die bij Vader Jacob hoort, uit het geheugen naar de geluidsprocessor sturen. De geluidsprocessor is ook weer een standaard chip voor alle MSX-computers. Deze geluidsprocessor kan drie verschillende tonen tegelijk laten horen. Bovendien is er nog de mogelijkheid om een ruissignaal te genereren. Dit alles bij elkaar maakt het mogelijk om driestemmige muziek te maken, begeleid door tromgeroffel. Ook is het hiermee mogelijk om bijna iedere soort geluid te imiteren, van een kanonsschot tot en met de branding van de zee. Ook de geluidsprocessor kan geheel zelfstandig werken. Zodra hij de muziek heeft gekregen van de microprocessor, slaat hij deze op in zijn interne geheugen en begint hij de muziek te spelen. De microprocessor kan ondertussen weer een volgende opdracht uit het geheugen lezen.

De MSX-computer heeft zelf geen luidspreker. Het geluid wordt op het TV-signaal gemoduleerd. Op die manier wordt het beeld, samen met het geluid door de televisie ontvangen en versterkt. U kunt dan ook het volume van het geluid met behulp van de volumeregelaar van de televisie regelen.

Zou u de computer uitschakelen, dan zou alles wat in het geheugen staat verloren gaan. U zou dus het programma dat u met veel moeite in het geheugen hebt ingebracht (via het toetsenbord) met een druk op de knop kwijt kunnen raken. Er is dan ook een mogelijkheid om de inhoud van het geheugen op te slaan op (weg te schrijven naar) een cassette of een flexibele schijf. In de meeste gevallen zult u de opdracht hiertoe rechtstreeks via het toetsenbord aan de microprocessor geven. Het is ook mogelijk om deze opdracht in het programma op te nemen.

We hebben nu een aantal voorbeelden gezien waarbij er signalen van de computer naar een randapparaat gaan. We zagen het TV-beeld, het geluid en de gegevens en programma's uit het geheugen. Om er zeker van te zijn dat er bij het aansluiten van die randapparaten geen problemen ontstaan, moeten de MSX-computers en de randapparaten allemaal dezelfde soort pluggen hebben. Bovendien moeten de randapparaten allemaal dezelfde manier van aansluiten hebben. In dat geval kan van standaard kabels gebruik worden gemaakt.

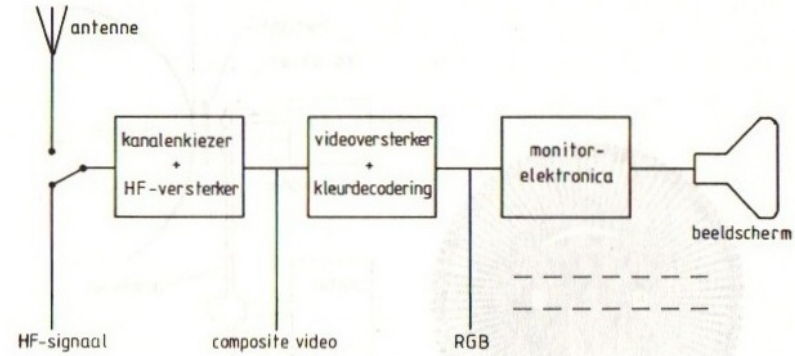
1.2 Beeldschermen

Ieder televisietoestel kan als beeldscherm voor een MSX-computer worden gebruikt. Indien u een kleurentelevisie gebruikt als beeldscherm, zult u de meest fraaie weergave verkrijgen. Standaard drukt een MSX-computer maximaal 40 tekens op een regel af. Deze beperking is aan de computer opgelegd, omdat een groter aantal tekens per regel op een kleurentelevisie niet meer te lezen zou zijn. Er zijn echter systemen denkbaar waarbij een groter aantal tekens per regel nog wel goed leesbaar zou zijn. Dat is het geval bij monitoren. Monitoren zijn televisietoestellen zonder kanalenkiezer.

Een normaal televisietoestel bestaat grofweg uit drie delen. Het eerste deel is een kanalenkiezer en een hoogfrequentversterker. In het tweede deel wordt het videosignaal versterkt en worden de kleursignalen gedecodeerd. Wat er dan nog overblijft wordt naar de monitor-elektronica gestuurd. Dit laatste deel zorgt er dan voor dat er een patroon op de beeldbuis verschijnt. In afbeelding 1-2 is een en ander grafisch weergegeven. Bovendien is daar aangegeven op welke punten een computer zou kunnen worden aangesloten.

Het eerste punt is de antenne-ingang van het televisietoestel. Het tweede punt is de composite-video-ingang. Het laatste punt is de RGB (rood/groen/blauw) ingang. De meeste MSX-computers geven u de keuze om een televisietoestel via de antenne-ingang aan te sluiten of om een monitor met een RGB-ingang aan te sluiten.

Doordat bij het aansluiten van een monitor met RGB-ingang een aantal stappen worden overgeslagen, zowel in de TV als ook in de computer, wordt het signaal minder vervormd. Dit is duidelijk te zien aan het beeld. Een monitor geeft een veel scherper beeld. Hierdoor is het mogelijk om op een monitor bijvoorbeeld 80 tekens per regel af te drukken, zonder de leesbaarheid te schaden. Er zijn reeds een aantal uitbreidingskaarten beschikbaar waarmee u die 80 tekens per regel kunt realiseren. Vooral voor bijvoorbeeld tekstverwerking is dit zeer nuttig. De meeste monitoren zijn echter niet in staat om geluid weer te geven. Voor het geluid zult u dan een aparte versterker moeten gebruiken.



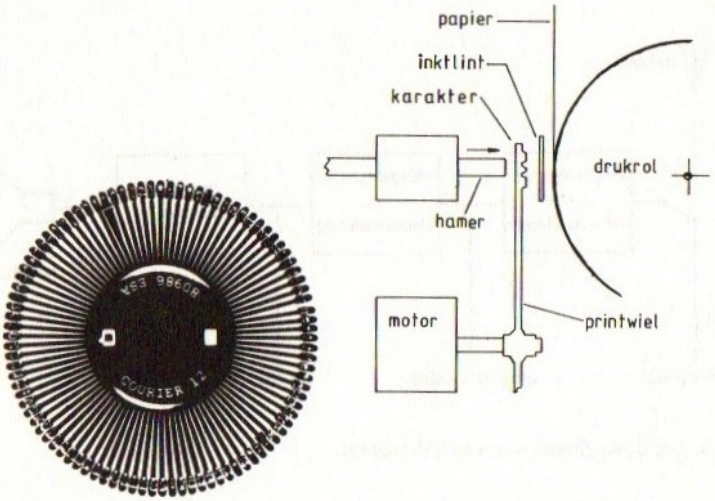
Afb. 1-2 Blokschema van een TV-toestel.

1.3 Afdrukeenheden.

Afdrukeenheden, vaak ook aangeduid met "printers", bestaan in vele soorten. De twee meest voorkomende soorten zijn "matrix" en "daisy-wheel" printers. Een ontwikkeling van de laatste jaren is de "ink-jet" printer. Voor elk van deze printers geldt echter dat, willen ze echt geschikt zijn voor aansluiten op een MSX-computer, ze hetzelfde tekenrepertoire als een MSX-computer moeten hebben. Voor matrix- en ink-

jet printers is dit wel mogelijk, doch voor een daisy-wheel printer is dit absoluut onmogelijk.

Een daisy-wheel printer maakt namelijk gebruik van een letterwiel. Op zo'n letterwiel zitten meestal 96 tekens. Een MSX-computer kent echter 256 verschillende tekens. Hieruit volgt dat een daisy-wheel printer niet alle tekens van een MSX-computer kan afdrukken. Daar echter de normale letters, cijfers en leestekens wel op een letterwiel voorkomen, zou de daisy wheel printer wel geschikt zijn voor tekstverwerking. Wilt u echter programma-listings afdrukken, dan zult u gebruik moeten maken van een matrix printer of een ink-jet printer.

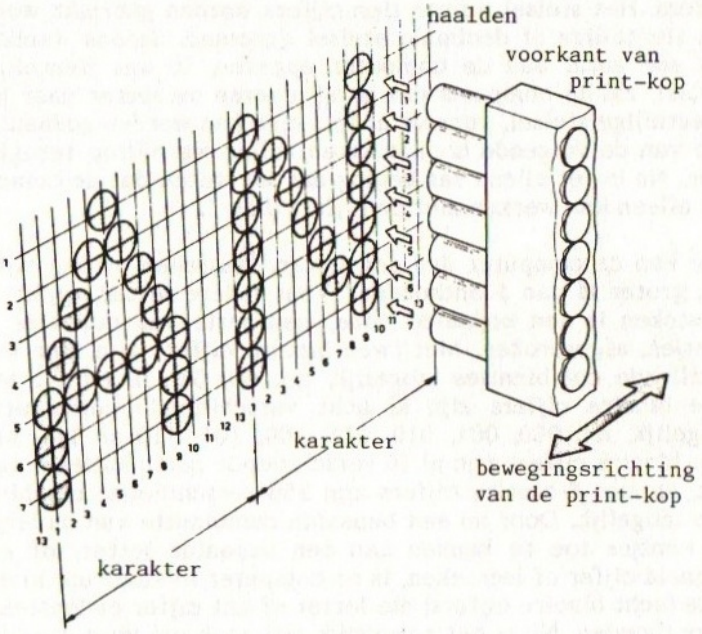


Afb. 1-3 Principe van een daisy-wheel printer.

De meeste printers hebben echter een van fabriekswege vastgestelde teken-set. U zult dan ook moeten kiezen voor een printer die de complete MSX-teken-set heeft. Ook dit is weer een voor MSX-computers gemaakte standaard. Er zijn al verschillende printers te verkrijgen met de juiste teken-set. De keuze tussen een matrix- of ink-jet printer is weer erg afhankelijk van uw persoonlijke voorkeur. Een matrix printer is

vaak goedkoop in aanschaf en in gebruik. Een ink-jet printer werkt echter geruisloos en geeft een goede (en constante) afdruk.

Welke afdrukeenheid u ook kiest, de verbinding met de computer zal altijd via een zogenaamde Centronics Interface gaan. Dit is een parallel-interface, hetgeen wil zeggen dat de gegevens van de computer naar de printer via 8 draadjes tegelijk gaat. Zowel de pluggen als de aansluiting van de pluggen is weer gestandaardiseerd.



Afb. 1-4 Principe van een matrix printer.

1.4 Het geheugen

Net als bij de meeste andere computers, kan het geheugen

van een MSX-computer verschillende groottes hebben. Geheugens worden gemeten in "bytes". Het woordje "byte" is afkomstig uit het Amerikaans en is een samenvoeging van de woorden "by eight", hetgeen wil zeggen "per acht". Een byte bestaat dan ook uit acht bitjes. Ieder bitje kan een 1 of een 0 zijn. De computer kent maar twee cijfers, de 0 en de 1. Met behulp van die twee cijfers wordt alle rekenwerk gedaan. Het talstelsel waarbij slechts twee cijfers worden gebruikt wordt het tweetallige of binaire stelsel genoemd.

Wij, mensen, gebruiken de cijfers 0 tot en met 9. Dit zijn 10 cijfers. Het stelsel waarin tien cijfers worden gebruikt wordt het tientallige of decimale stelsel genoemd. Steeds wanneer wij een getal aan de computer opgeven, in ons tientallige stelsel, zal de computer dat getal moeten omzetten naar het tweetallige stelsel, voordat er iets mee kan worden gedaan. In een van de volgende hoofdstukken zullen we hierop terugkomen. Nu is het alleen van belang dat we weten dat de computer alleen kan werken met de cijfers 0 en 1.

Hoe kan de computer dan een letter, leesteken of een cijfer dat groter is dan 1 onthouden? Voor iedere letter, cijfer en leesteken is een bepaalde code, een serie van nulletjes en eentjes, afgesproken. Met twee binaire cijfers zijn vier verschillende combinaties mogelijk, n.l. 00, 01, 10 en 11. Met drie binaire cijfers zijn al acht verschillende combinaties mogelijk, n.l. 000, 001, 010, 011, 100, 101, 110 en 111. Met vier binaire cijfers zijn al 16 verschillende combinaties mogelijk, en met 8 binaire cijfers zijn 256 verschillende combinaties mogelijk. Door nu een bepaalde combinatie van nulletjes en eentjes toe te kennen aan een bepaalde letter, of een bepaald cijfer of leesteken, is de computer in staat om in een byte (acht binaire cijfers) die letter of dat cijfer of leesteken te onthouden. Nu is het natuurlijk wel zaak om voor dezelfde letter altijd dezelfde combinatie van nulletjes en eentjes aan te houden. Per computer kan hier nog wel eens verschil in zitten. Alle MSX-computers hebben echter allemaal dezelfde combinaties van nulletjes en eentjes voor dezelfde letters, cijfers of leestekens.
Nu het duidelijk is dat er in een byte een letter, cijfer of leesteken kan worden opgeslagen, zal de maat van een geheu-

gen u ook meer aanspreken. Geheugens zijn echter zo groot, dat ze meestal worden aangegeven in kilo-byte, of zelfs Mega-byte. U kent dit wel van afstanden. De maat voor lengte is meter. Hele grote lengtes worden echter aangegeven in kilometers. Een kilometer is hetzelfde als 1000 meter. Pas echter op. In de computertechniek betekent kilo-byte niet 1000 byte.

We komen nog even terug op het binaire stelsel, om uit te leggen waarom kilo in de computer niet 1000 betekent. Met 10 bitjes zijn namelijk 1024 verschillende combinaties mogelijk. 1024 ligt qua waarde dicht bij 1000. Omdat 1024 voor de computer een rond getal is, en omdat er over de geheugengrootte van de computer zelf wordt gesproken, hebben computertechnici in het verleden afgesproken dat het woordje kilo in de computertechniek 1024 betekent. 1 kbyte betekent dus 1024 bytes, ofwel 1024 keer 8 bitjes.

De meeste MSX-computers hebben 64 kbyte gebruikersgeheugen plus nog eens 16 kbyte videogeheugen. Bovendien kan er op iedere MSX-computer een cassetterecorder worden aangesloten. Gegevens die in het geheugen van de computer staan, kunnen naar die cassetterecorder worden gestuurd. Zijn die gegevens eenmaal opgenomen op een band, dan blijven ze op die band staan en kunnen ze er later weer vanaf worden gelezen. Die band is daarom ook te beschouwen als een geheugen. De grootte van dat geheugen is afhankelijk van de lengte van de band, doch kan per band oplopen tot vele honderden kbytes.

We noemen een cassetterecorder een extern geheugen, of achtergrondgeheugen. In dit soort geheugen slaan we al die gegevens op die we niet direct nodig hebben. We kunnen er bijvoorbeeld een aantal programma's opzetten. Op het moment dat we een van die programma's willen gebruiken, lezen we dat programma vanaf de band in het computergeheugen. Staat dat programma eenmaal in het geheugen van de computer, dan kan de computer dat programma uitvoeren.

Voor de volledigheid dient hier nog te worden opgemerkt dat ook een flexibele schijf een extern geheugen is. In dit boek

willen we echter nog geen aandacht aan de flexibele schijf besteden, omdat we willen beginnen met de MSX-computer zelf. Daar iedere MSX-computer standaard wordt uitgerust met een aansluitmogelijkheid voor een cassetterecorder, zullen we ons in dit deel beperken tot het gebruik van die cassetterecorder als extern geheugen. In een volgend hoofdstuk zullen we dan ook uitgebreid terugkomen op het gebruik van die cassetterecorder.

Nu zult u ten aanzien van het computergeheugen nog een aantal vreemde uitdrukkingen tegenkomen. Vaak wordt er gesproken van ROM en RAM geheugens.

ROM is de afkorting van de woorden Read Only Memory. Dit is een soort geheugen waar alleen iets uit kan worden gelezen. De fabrikant heeft iets in dit soort geheugen geschreven, en de computer kan het er alleen maar uit lezen. Je zou kunnen zeggen dat dit soort geheugen een overeenkomst vertoont met een boek. Wat er eenmaal in staat kan niet meer worden gewijzigd. Je kunt het echter zo vaak lezen als je maar wilt. Het ROM-geheugen wordt gebruikt om er programma's, die altijd nodig zijn, in op te slaan. Zo wordt er door de fabrikant van een computer een programma in ROM gezet, dat in staat is om de door u geschreven BASIC-programma's om te zetten naar een voor de computer begrijpelijke serie van nulletjes en eentjes. Dit programma wordt aangeduid met de BASIC-interpreter.

RAM is de afkorting van de woorden Random Access Memory. Deze woorden staan voor "willekeurig toegankelijk geheugen". Willekeurig wil in dit verband zeggen, dat je er zowel iets in kunt schrijven als er iets uit kunt lezen. Dit soort geheugen is het beste te vergelijken met een cassetterecorder. Je kunt nieuwe informatie over bestaande informatie heen opnemen. De oude informatie raakt daarbij verloren.

Over geheugens zou nog veel meer te vertellen zijn. Waar nodig, voor een goed begrip van het programmeren van de computer, zullen we ook zeker terugkomen op het geheugen. In dit inleidende hoofdstuk zullen we het hier echter bij laten.

2 Voorbereiding tot het programmeren

We gaan ons nu voorbereiden op het schrijven van programma's in de programmeertaal BASIC. Er bestaan vele honderden dialecten van deze programmeertaal. MSX-BASIC is ook een dialect van die taal. Er zijn echter al zoveel fabrikanten die voor MSX hebben gekozen, dat je wel mag zeggen dat MSX-BASIC een standaard programmeertaal is geworden. MSX-BASIC is een zogenaamde probleemgerichte hogere programmeertaal.

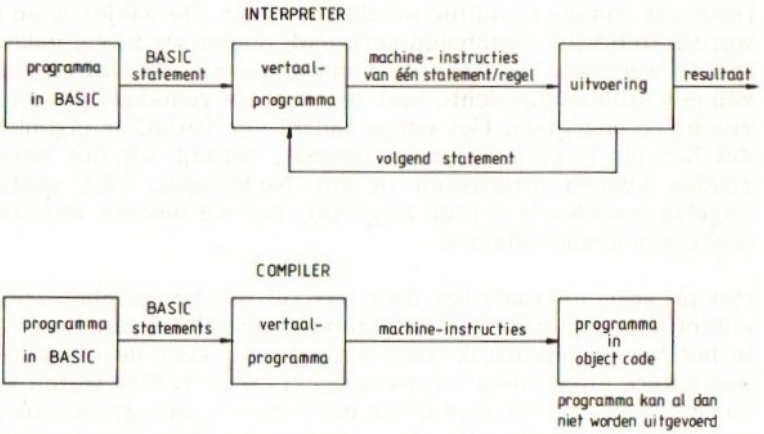
In een probleemgerichte taal zijn de instructies (in BASIC statements genoemd) gericht op het probleem dat moet worden opgelost, en niet op de manier waarop de computer die oplossing uitvoert. In BASIC kunnen we eenvoudigweg zeggen PRINT A+B, zonder ons te bekommeren over hoe de computer de getallen in de vakjes A en B bij elkaar optelt, of hoe het resultaat van die optelling wordt afgedrukt. De vakjes A en B worden in BASIC variabelen genoemd, omdat we in die vakjes steeds weer andere waarden kunnen zetten. De statements van een probleemgerichte taal zijn daarom gemakkelijk te lezen en te begrijpen. Het enige nadeel van BASIC is eigenlijk dat het uit Engelse woorden bestaat, terwijl wij ons beter zouden kunnen uitdrukken in het Nederlands. Het aantal Engelse woorden is echter zo gering, dat we ons die woorden snel eigen kunnen maken.

Het programma zoals het door ons zal worden geschreven is echter voor een computer niet zonder meer leesbaar. Zoals al in het vorige hoofdstuk werd aangetoond, kent de computer van nature alleen maar de twee cijfers 0 en 1. Ook weten we dat met een beperkt aantal van die cijfers al een groot aantal verschillende combinaties is te maken. De door ons geschreven programma-statements moeten daarom, voordat de com-

puter ze kan uitvoeren, eerst worden omgezet naar een aantal combinaties van nulletjes en eentjes. Deze combinaties worden machinetaalinstructies genoemd.
Het omzetten van BASIC-statements naar machinetaalinstructies wordt door een vertaalprogramma gedaan. Dit programma wordt meestal door de fabrikant van de computer meegeleverd met het systeem. Er zijn twee soorten vertaalprogramma's:

- Interpreters
- Compilers

Een interpreter vertaalt een regel van het programma in machinecode en voert deze dan onmiddellijk uit. Daarna wordt met het vertalen en uitvoeren van de volgende regel verder gegaan. In afbeelding 2-1 is dit weergegeven. Als er nu in het programma een aantal regels voorkomen die meerdere malen moeten worden uitgevoerd, we spreken dan van een lus in het programma, dan zullen die regels iedere keer opnieuw moeten worden vertaald naar machinecode. Dit maakt dat een interpreter relatief langzaam werkt.

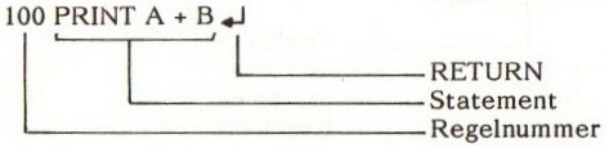


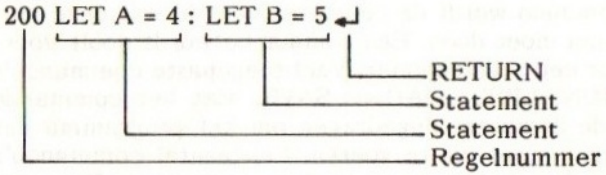
Afb. 2-1 Verwerking van programma met interpreter en compiler.

Een compiler vertaalt het hele programma in zijn geheel, en zet dat vertaalde programma op een cassette of een magnetische schijf. Vervolgens kan de programmeur het vertaalde programma, dat uit machine-instructies bestaat en ook wel object-code wordt genoemd, weer in het geheugen laden en het laten uitvoeren. Zou in dit programma ook een lus zitten, dan zullen de statements van die lus niet steeds opnieuw hoeven worden vertaald. Dit maakt dat gecompileerde BASIC-programma's (veel) sneller werken dan BASIC-programma's die met een interpreter worden uitgevoerd.

2.1 BASIC-statements

Een BASIC-programma is opgebouwd uit regels. Iedere regel begint met een regelnummer en eindigt met RETURN. Op een regel kunnen een of meer statements staan. Staan er meerdere statements op een regel, dan zullen die statements van elkaar gescheiden moeten zijn door een dubbele punt (:). Hier volgen enkele voorbeelden van BASIC-regels.





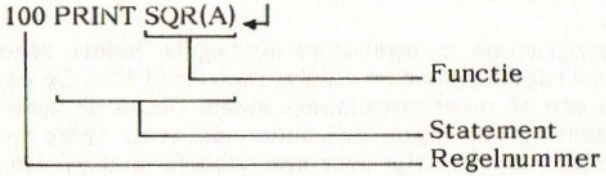
Regelnummer 100 bevat 1 statement, regelnummer 200 bevat twee statements. De maximale lengte van een regel is 255 tekens. Er zouden dus nog veel meer statements op regel 200 passen.

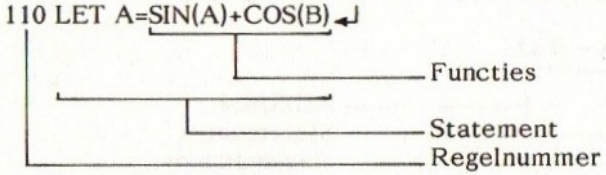
De taal BASIC is opgebouwd uit statements, functies en commando's. Alle instructies, die in een programma voorkomen,

worden statements genoemd. Ze worden altijd voorafgegaan door een regelnummer. Hierna volgen enkele voorbeelden van statements:

```
10 INPUT A
20 INPUT B
30 PRINT A + B
40 GOTO 10
```

Statements kunnen ook functies bevatten. Functies roepen reeds voorgeprogrammeerde machinetaalprogramma's op. Voorbeelden van functies zijn:





Met een commando wordt de computer verteld wat deze met een programma moet doen. Een commando wordt nooit voorafgegaan door een regelnummer. Veel toegepaste commando's zijn NEW, RUN, LIST, LOAD en SAVE. Met het commando RUN wordt de computer opgedragen om het programma dat in het geheugen staat uit te voeren. Een aantal commando's mogen ook in het programma worden opgenomen. De commando's worden dan door de computer gezien als statements. Ook het omgekeerde is mogelijk, statements kunnen zonder regelnummer worden ingegeven, en worden dan door de computer gezien als een commando. Wanneer statements en commando's door middel van een programma worden uitgevoerd, dan zeggen we dat ze indirect worden uitgevoerd. Geven we ze in zonder een regelnummer, dan zullen ze onmiddellijk

worden uitgevoerd, en spreken we van directe mode. Door bijvoorbeeld:

PRINT "Directe Mode"

in te voeren, zonder regelnummer, zal de PRINT-opdracht onmiddellijk worden uitgevoerd, zodat op het scherm de tekst:

Directe Mode

verschijnt. Het invoeren van programma's, waarvan de programmaregels worden voorafgegaan door regelnummers, wordt de indirecte mode genoemd.

2.2 Analyse en ontwerp

Het schrijven van een computerprogramma is in feite het oplossen van een probleem. Voordat een probleem kan worden opgelost moet dat probleem eerst goed duidelijk zijn. Vervolgens kan dat probleem in een aantal deelprobleempjes worden opgedeeld. Om goed in de gaten te kunnen houden hoe de verschillende deelproblemen onderling samenhangen kunnen we dan een "stroomdiagram" opstellen. Pas wanneer we er van overtuigd zijn een goed stroomdiagram te hebben, kunnen we beginnen met het schrijven van het programma. Is het programma eenmaal geschreven, dan zullen we het nog moeten testen. We zien dan of het programma foutloos is geschreven en of het werkelijk een oplossing is van het probleem.

Er zijn dus vier belangrijke punten te onderscheiden in het maken van een programma:

1. Probleemstelling
2. Schrijven van een stroomdiagram
3. Schrijven van een programma
4. Testen van het programma

Aan de hand van een voorbeeld zullen we deze punten nader

uitwerken.

2.3 Probleemstelling

Een handelaar verkoopt artikelen, die een bepaalde basisprijs hebben, en waarop afhankelijk van het aantal en de waarde van die artikelen een bepaalde korting wordt gegeven. De vraag is nu wat de prijs van een artikel is, rekening houdend met de te geven kortingen. Dit probleem moet dus zo goed mogelijk worden omschreven.

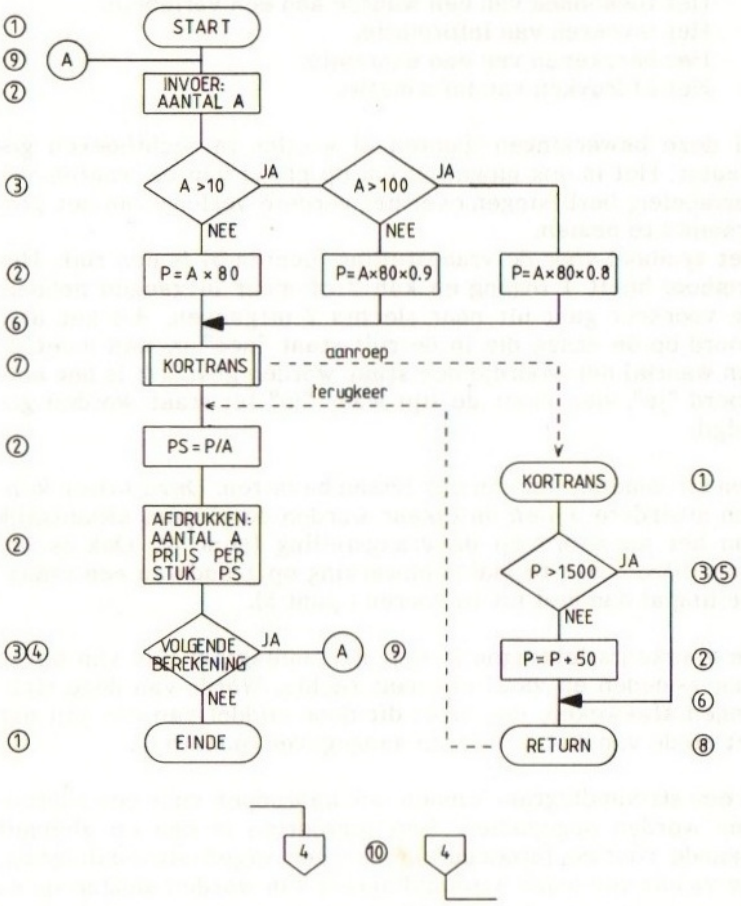
Probleem
Bereken de gemiddelde prijs van een artikel bij afname tot en met 10 stuks, van 11 tot en met 100 stuks en bij afnamen boven de 100 stuks. De prijs van het artikel bedraagt fl. 80,00 per stuk.
Bij afnamen van 11 tot en met 100 stuks wordt een korting van 10% gegeven, bij afnamen boven de 100 stuks wordt 20% korting gegeven. Bij aankopen boven de 1500 gulden wordt gratis thuisbezorgd, daarbeneden wordt 50 gulden transportkosten in rekening gebracht.

Nu het probleem goed is omschreven kan worden doorgegaan met de volgende stap, het maken van een stroomdiagram. Zo'n stroomdiagram is vooral waardevol voor onszelf. Het laat precies zien wat we willen. Het diagram bestaat uit een aantal blokken. De vorm van die blokken geeft de aard van de op dat punt te verrichten werkzaamheden weer. De blokken zijn onderling verbonden door lijnen, die de stroomrichting (Flow) door het diagram weergeven.

2.4 Stroomdiagrammen

We zullen nu het stroomdiagram (ook wel flow chart genoemd) voor het zojuist omschreven probleem gaan maken. Zie hiervoor het stroomdiagram in afbeelding 2-2. Elk stroomdiagram heeft een begin en een eind. Het begin van een stroomdiagram wordt met het START-symbool aangegeven, het einde met het EIND-symbool (punten 1). Het start-

symbool mag eventueel ook de naam of de afkorting van het probleem bevatten. Later kunnen we dan diezelfde naam ook voor het programma gebruiken.



Afb. 2-2 Stroomdiagram voor het berekenen van de gemiddelde kostprijs van een artikel.

Bij het lezen van een stroomdiagram beginnen we altijd bovenaan bij het start-symbool. Tussen het start- en eind-symbool staan de verschillende bewerkingen vermeld. Een bewerking kan onder andere zijn:

Het toekennen van een waarde aan een variabele.
Het invoeren van informatie.
Het berekenen van een expressie.
Het afdrukken van informatie.

Al dezé bewerkingen (punten 2) worden in rechthoeken geplaatst. Het is ook mogelijk, om op grond van de waarde van variabelen, beslissingen over het verdere verloop van het programma te nemen.
Het symbool voor de vraagstelling (punten 3) is een ruit. Het symbool heeft 1 ingang en kan 2 of meer uitgangen hebben. De voorkeur gaat uit naar slechts 2 uitgangen. Als het antwoord op de vraag die in de ruit staat "nee" is, dan moet de lijn waarbij het woordje nee staat worden gevolgd; is het antwoord "ja", dan moet de lijn waar "ja" bij staat worden gevolgd.

Een stroomdiagram kan ook lussen bevatten. Deze lussen kunnen meerdere malen na elkaar worden doorlopen, afhankelijk van het antwoord op de vraagstelling (punt 4). Ook is het mogelijk om een bepaalde bewerking op grond van een vraagstelling al dan niet uit te voeren (punt 5).

De voorkeursrichtingen in een stroomdiagram zijn van boven naar beneden en van links naar rechts. Wordt van deze richtingen afgeweken, dan moet dit door middel van een pijl aan het einde van de lijn worden aangegeven (punten 6).

In een stroomdiagram kunnen ook aanroepen voor een subroutine worden opgenomen. Een subroutine is een op zichzelf staande routine (procedure), met een eigen stroomdiagram, die vanuit een ander stroomdiagram kan worden aangeroepen. De aanroep van een subroutine wordt in een stroomdiagram aangegeven door een rechthoek, met aan de linker- en de rechterkant een dubbele streep (punt 7). In de rechthoek wordt dan de naam van de subroutine geschreven. Voor dit

rechthoekje in de plaats zou je het stroomdiagram van de subroutine kunnen denken. Het einde van een subroutine wordt aangegeven met het eind-symbool waarin RETURN staat, in plaats van EINDE (punt 8).

Wanneer de lijnen die lussen vormen te lang worden, kan men gebruik maken van verwijzingsconnectoren. Wordt daarmee verwezen naar een punt in het stroomdiagram op dezelfde pagina, dan wordt gebruik gemaakt van cirkelvormige connectoren (on-page connectoren, punten 9). Connectoren, waarin dezelfde benaming voorkomt, vormen een verbinding. In afbeelding 2-2 is er dus een verbinding tussen de connectoren "A". Wordt en pagina verlaten, dan wordt gebruik gemaakt van vijfhoekige connectoren (off page connectoren, punten 10). Wordt de pagina niet verlaten, dan worden letters gebruikt (A t/m Z), wordt de pagina wel verlaten, dan worden cijfers gebruikt (0 t/m 9).

2.5 Het programma

Nu moeten we het stroomdiagram omzetten naar een voor de computer uit te voeren programma. Het vertalen van het stroomdiagram levert het programma op.

Hoofdprogramma

```
100 INPUT "AANTAL";A
110 IF A>10 GOTO 130
120 LET P=A*80: GOTO 160
130 IF A>100 GOTO 150
140 LET P=A*80*.9: GOTO 160
150 LET P=A*80*.8
160 GOSUB 400
170 LET PS=P/A
180 PRINT
190 PRINT "AANTAL";A
200 PRINT "PRIJS PER STUK";PS
210 PRINT
220 INPUT "VOLGENDE BEREKENING (J/N)";J$
230 IF J$="J" OR J$="j" GOTO 100
240 END
```

subroutine

```
400 IF P>1500 GOTO 420
410 LET P=P+50
420 RETURN
```

Het is heel goed mogelijk, dat het programma nog niet wordt begrepen. In dit stadium dient het programma er alleen maar voor om te laten zien hoe een BASIC-programma wordt gemaakt, en hoe het er uitziet.

De stroomdiagrammen zijn nu vertaald in een hoofdprogramma en een subroutine. De laatste stap die nog moet worden gezet is het testen van het programma.

2.6 Testen

Als het programma eenmaal is geschreven, volgt het spannendste deel, het testen of het programma goed werkt. Meestal zal het programma de eerste keer niet goed werken. Dit kan allerlei oorzaken hebben, maar de meest voorkomende oorzaken zijn:

Syntax-fouten (verkeerd geschreven statements)
Invoerfouten
Het niet wissen van het geheugen voor het programma werd ingetikt
Logische programmafouten (fouten in stroomdiagram)

Het invoeren van fout geschreven instructies wordt door de computer ontdekt en gemeld. Deze melding komt in de vorm van een foutboodschap. Wanneer een programma zonder syntax-fouten toch een ander resultaat geeft dan men verwacht, dan kan dat worden veroorzaakt door een logische programmeerfout. Dit soort fouten kan niet door de computer worden ontdekt. In deze situaties zult u dus zelf moeten zoeken naar de oorzaak, door te onderzoeken of de probleemstelling juist was, of het stroomdiagram een goede weergave van de probleemstelling is en of het programma wel een goede

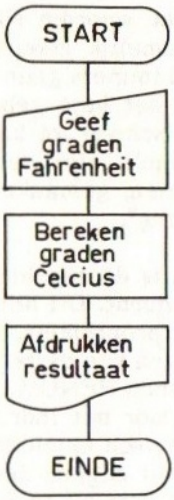
vertaling van het stroomdiagram was. Later in dit boek zullen we nog terugkomen op de in MSX-computers ingebouwde hulpmiddelen bij het zoeken naar fouten.

3 Programma's invoeren en corrigeren

We zullen nu een programma gaan schrijven. Dit programma zullen we volgens de in het vorige hoofdstuk gegeven regels opzetten. Vervolgens zullen we bij het invoeren van dat programma opzettelijk enkele fouten maken. Die fouten kunnen we dan weer corrigeren met behulp van de "edit"-functie die in onze computer is ingebouwd. Wat een "edit"-functie is zullen we straks zien. Als het programma helemaal foutloos is ingevoerd, zullen we het gaan uitvoeren. De nadruk zal in dit hoofdstuk komen te liggen op het corrigeren van het programma. U zult dan ook na dit hoofdstuk nog niet kunnen programmeren. Dat zult u in de hiernavolgende hoofdstukken leren. Daar het echter van het grootste belang is om zo snel mogelijk de computer in te schakelen bij het leren programmeren, zullen we in dit hoofdstuk leren hoe we de computer kunnen gebruiken om er een programma in te zetten en om dat programma te laten uitvoeren.

Zoals gezegd, is het programmeren het oplossen van een probleem. Om het voorbeeld eenvoudig te houden zullen we nu een eenvoudig probleem aan moeten pakken. Laten we daarom het probleem nemen van het omzetten van een gegeven temperatuur in graden Fahrenheit naar graden Celcius.

Een graad Celcius staat gelijk aan 5/9 keer het aantal graden Fahrenheit minus 32. Om van een gegeven aantal graden Fahrenheit het overeenkomstige aantal graden Celcius te kunnen bepalen zullen we eerst het aantal graden Fahrenheit moeten opgeven. Vervolgens moet het bijbehorende aantal graden Celcius worden berekend en tenslotte moet het gevonden aantal graden worden afgedrukt. Het stroomdiagram in afbeelding 3-1 laat dit zien.



Afb. 3-1 Stroomdiagram temperatuuromzetting.

Nu volgt het programma, dat is gemaakt volgens het stroomdiagram. In het programma zitten enkele fouten. Tik het programma echter precies zo in als het hier is gegeven, we zullen dan straks met behulp van de in de computer aanwezige "editor" de fouten corrigeren. Na iedere regel dient u op de RETURN-toets te drukken om de computer te laten weten dat er weer een regel is ingetypt.

```
NEW
Ok
12 INPUT "Graden Fahrenheit";F
25 LET C=5/9*(F-32)
42 PRINT "graden celcius =";
60 EIND
```

Om er zeker van te zijn dat het programma in het computergeheugen staat, kunnen we het commando LIST geven. Tik dit commando maar eens in, gevolgd door het indrukken van de RETURN-toets. U ziet nu dat het programma, precies zoals u het hebt ingetikt, nogmaals op het scherm verschijnt. Het

enige verschil is dat het woordje NEW niet opnieuw wordt afgedrukt. NEW was namelijk geen programmaregel, maar een commando (er stond immers geen regelnummer voor). Het **commando NEW** maakt het hele geheugen van de computer leeg, zodat er met een schone lei kan worden begonnen. De opnieuw afgedrukte regels zijn de programmaregels. Een afdruk van een programma, gemaakt met het LIST-commando, noemen we een "listing".

Wat in de listing opvalt is de regelnummering. De regelnummers zijn niet erg consequent. Dit heeft geen nadelige invloed op de uitvoering van het programma, maar het is gewoon geen mooi gezicht. Het wijzigen van de regelnummers is eenvoudig te doen met het commando RENUM. Tik dit commando maar eens in, weer gevolgd door het indrukken van de RETURN-toets. U ziet dat er praktisch onmiddellijk "OK" onderaan het scherm verschijnt. Dit wil zeggen dat de computer het commando heeft uitgevoerd. Om nu te kunnen zien of de regels van het programm werkelijk hernummerd zijn, moet u opnieuw het commando LIST (RETURN-toets) intypen. Weer verschijnt het programma op het scherm, doch nu met de regelnummers 10 20, 30 en 40.

Wat is er nu gebeurd? Indien het **commando RENUM** wordt ingegeven, zonder verdere aanwijzingen, dan worden alle programmaregels hernummerd. Het eerste regelnummer zal 10 zijn, de volgende regel zal steeds 10 hoger zijn dan de vorige. Het is echter mogelijk om met een lager regelnummer te beginnen en het verschil tussen de regelnummers groter of kleiner dan 10 te maken. Daartoe moeten aan het RENUM-commando een aantal zogenaamde parameters worden toegevoegd. Het formaat van het RENUM commando is als volgt:

RENUM nieuw nummer , oud nummer , verhoging

RENUM hernummert de programmaregels door alle programmaregels vanaf **oud nummer** een nieuw regelnummer te geven. Dat nieuwe regelnummer zal beginnen met **nieuw nummer**, en iedere volgende regel zal **verhoging** hoger zijn dan de vorige. Tik nu maar eens in: RENUM 1000 (gevolgd door de RETURN-toets).

Maak weer een listing van het programma en kijk wat er met de regelnummers is gebeurd. Het laagste regelnummer is nu 1000, de volgende is 1010, enz. De verhoging is dus 10. Wat hebben we namelijk gedaan? We hebben alleen de eerste parameter ingevuld. Dit houdt in, dat de computer zelf de andere twee parameters invult en wel met de waarden: hernummeren vanaf de eerste programmaregel met een verhogingsfactor van 10.

Om nu de verhogingsfactor 100 te maken moeten we intikken RENUM 1000,,100. Hiermee geven we alleen de eerste en derde parameter op. Controleer wat er is gebeurd door weer een listing te maken. Hernummer hierna alle regels vanaf regelnummer 1200 zo, dat ze beginnen met regelnummer 3333, en een verhogingsfactor van 3 hebben. Dat commando zal dan zijn: RENUM 3333,1200,3

Het is belangrijk dit commando goed onder de knie te hebben. Denk er aan dat de eerste parameter het eerste nieuwe regelnummer is, de tweede parameter de eerste te hernummeren oude regel, en dat de derde parameter de verhogingsfactor is. Speel gerust nog een tijdje met dit commando, totdat u alle mogelijkheden goed in de vingers hebt. Als u verder wilt gaan met dit hoofdstuk, geef dan het commando RENUM zonder verdere parameters in, en maak weer een listing van het programma. Die listing zal er dan als volgt uit zien:

```
10 INPUT "Graden Fahrenheit";F
20 LET C=5/9*(F-32)
30 PRINT "graden celcius =";
40 EIND
```

Het INPUT-statement in regel 10 heeft twee functies. Eerst zal het de tekst die tussen de aanhalingstekens staat op het scherm afdrukken, gevolgd door een vraagteken. Hiermee geeft de computer aan dat er een ingave op het toetsenbord van u wordt verwacht. De tweede functie is dat alle door u ingetoetste gegevens in het geheugenvakje met de naam F zullen worden opgeslagen. Tikt u bijvoorbeeld de cijfers 100 in, gevolgd door het indrukken van de RETURN-toets (om aan te geven dat u alle gegevens hebt ingetikt), dan zal het getal

100 in het geheugenvakje F worden onthouden.

In regel 20 staat een formule 5/9*(F-32). Het resultaat van die formule moet worden opgeslagen in het geheugenvakje met de naam C. Het zal u duidelijk zijn dat de formule de temperatuur in Fahrenheit omzet naar een waarde die het aantal graden Celcius aangeeft. In vakje C zal dus het aantal graden Celcius komen te staan. Dat aantal willen we met regel 30 op het scherm afdrukken. Regel 30 bevat echter een fout. Achter de puntkomma zou nog moeten staan welk geheugenvakje moet worden afgedrukt. We moeten regel 30 dus wijzigen.

De listing van het programma staat nog op het scherm. U kunt nu met de "cursor control"-toetsen, de vier toetsen met de pijltjes aan de rechterkant van het toetsenbord, het blokje, dat aangeeft waar het volgende teken dat u intikt komt te staan, naar het einde van regel 30 brengen. Zet het blokje onmiddellijk achter de puntkomma. Tik nu de kleine letter c in, en druk dan op RETURN. Stuur nu het blokje (de cursor) weer helemaal naar onder, en tik het LIST-commando in. In de nieuwe listing ziet u nu dat regel 30 eindigt met een hoofdletter C. Namen van geheugenvakjes (dit worden variabelenamen genoemd) worden altijd met hoofdletters geschreven. Ook al geven wij ze een naam met kleine letters, de computer maakt er direct hoofdletters van.

Regel 30 is nu correct. In regel 40 zit echter ook nog een fout. Daar staat het statement "EIND". Eind is echter een Nederlands woord. Er zou moeten staan END. We moeten dus de I verwijderen. Dit doen we door de cursor op de I te zetten (met behulp van de vier pijltjestoetsen). Staat de cursor op de I, dan drukken we op de DEL-toets. DEL staat voor "delete", hetgeen wil zeggen "verwijder". De letter I zal nu zijn verdwenen. Nu is de correctie wel op het beeldscherm aangebracht, maar nog niet in het programmageheugen. Op het moment dat we op de RETURN-toets drukken zal de programmaregel van het beeldscherm in het programmageheugen worden overgenomen, inclusief de aangebrachte wijzigingen.

Als alle wijzigingen goed zijn aangebracht, dan kunt u het

programma gaan uitvoeren. Hiertoe dient u de computer de opdracht **RUN** te geven. Met RUN wordt het programma dat in het geheugen staat gestart, beginnende bij de eerste regel. Op het scherm zult u nu de tekst "Graden Fahrenheit?" zien verschijnen. Met het vraagteken geeft de computer aan dat u geacht wordt antwoord te geven via het toetsenbord. Tikt u nu bijvoorbeeld 68, gevolgd door de RETURN-toets, in, dan wordt met regel 20 het bijbehorende aantal graden Celcius in variabele C gezet. Vervolgens wordt met regel 30 de tekst "graden Celcius =", gevolgd door de inhoud van variabele C, afgedrukt op het beeldscherm.

We hebben nu gezien hoe een programma wordt ingevoerd, gecorrigeerd met de "editor" en gestart. We hebben daarbij ook gezien hoe we na het invoeren de regelnummering nog kunnen wijzigen. We hebben echter nog niet gezien, hoe de computer ons bij het invoeren van programma's kan helpen, door zelf de regelnummers te genereren. Bovendien hebben we van de editor slechts een paar van de vele mogelijkheden gezien. We zullen het voorgaande programma nog eens gaan invoeren, nu met hulp van de computer. Daarna zullen we de overige mogelijkheden van de editor gaan bekijken.

Voordat we een nieuw programma kunnen invoeren, moeten we er voor zorgen dat het geheugen leeg is. Dit doen we door het commando NEW te geven, gevolgd door het indrukken van de RETURN-toets. Na ieder commando, na iedere BASIC-regel en na iedere ingave van gegevens, moeten we de computer duidelijk maken dat we klaar zijn met de ingave, door de RETURN-toets in te drukken. In het vervolg zullen we hiervan geen melding meer maken.

Nu het geheugen leeg is, zullen we het programma weer gaan ingeven. Echter, we zullen de computer nu de regelnummers zelf laten geven. Hiertoe geven we de computer de opdracht AUTO. Het commando AUTO kan nog worden gevolgd door twee parameters. Wat precies de mogelijkheden zijn wordt hierna opgesomd:

AUTO
Nummer de regels vanaf 10 met een ophoging van 10.

AUTO nn
Nummer de regels vanaf nn (hiervoor mag u zelf een getal invullen), met een ophoging van 10.

AUTO nn,
Nummer de regels vanaf nn met dezelfde ophoging als de laatst gebruikte ophoging.

AUTO nn,mm
Nummer de regels vanaf nn, met een ophoging van mm. Ook voor mm kunt u zelf een getal invullen.

Stel dat we het nieuw in te geven programma genummerd willen hebben vanaf regelnummer 1000, terwijl de ophogingsfactor 5 is. We geven dan het commando AUTO 1000,5. Hierna zien we op het scherm het regelnummer 1000 verschijnen, terwijl de cursor op de eerste positie na het regelnummer blijft staan. We kunnen nu de eerste regel van het programma intikken (zonder regelnummer). Na het indrukken van de RETURN-toets verschijnt automatisch het volgende regelnummer. We kunnen de volgende regel intikken. Dit blijft zo doorgaan. De computer weet echter niet dat wij maar vier regels willen intikken.Deze zal dan ook rustig met het vijfde regelnummer komen. Als wij nu geen nieuwe regels meer willen ingeven, dan drukken we op de CNTRL-toets, houden die ingedrukt, en drukken tegelijk op de STOP-toets. U ziet nu dat de cursor zich verplaatst van het begin van de vorige BASIC-regel naar de eerste positie onder het laatst afgedrukte regelnummer. Door nu een LIST-commando te geven, wordt het ingetikte programma op het beeldscherm afgedrukt.

Laten we nu eens zien welke functies er allemaal in de editor zitten. Er zal een korte opsomming van alle functies volgen, waarna u zelf alle functies kunt uittesten op het nog in het geheugen staande programma. De functies zijn opgedeeld in twee groepen. De eerste groep is die, waarvoor speciale toetsen op het toetsenbord aanwezig zijn. De tweede groep bestaat uit die functies, waarvoor de CTRL-toets + een lettertoets moeten worden ingedrukt.

Groep 1

BS

Door het indrukken van de Back Space toets (BS) wordt het laatst ingetikte teken gewist.

CLS

Door het indrukken van de CLS-toets (Clear Screen) wordt het beeldscherm leeggemaakt.

DEL

Door de DEL(ete)-toets in te drukken wordt de letter die onder de cursor staat gewist, en worden alle gegevens die (binnen de regel) achter de cursor volgen, een plaatsje naar links geschoven.

HOME

Het indrukken van de HOME-toets zorgt ervoor dat de cursor naar de eerste beeldschermpositie wordt verplaatst. Dit is de positie linksbovenaan het scherm.

INS

Na het indrukken van de INS-toets kunnen nieuwe letters worden tussengevoegd in een bestaande regel. De tussengevoegde letters komen op de plaats van de cursor te staan. Alle letters volgend op de cursor, zullen naar rechts schuiven. De tussenvoegfunctie blijft actief, totdat er een andere functie wordt gekozen.

Pijltjes

Met de pijltjestoetsen kan de cursor over het gehele beeldscherm worden verplaatst, zowel in horizontale als in verticale richting.

RETURN

Met de RETURN-toets worden de op het scherm aangebrachte wijzigingen in het programmageheugen aangebracht.

TAB

Het beeldscherm is ingedeeld in kolommen van 16 tekens

breed. Met de TAB-toets wordt de cursor naar het begin van een volgende kolom verplaatst.

Groep 2

CTRL + B
Hiermee wordt de cursor naar het begin van het vorige woord verplaatst.

CTRL + C
Hiermee wordt de ingave onderbroken. De reeds ingegeven wijzigingen worden niet in het programmageheugen opgenomen.

CTRL + E
Hiermee wordthet deel van de programmaregel, dat na de cursor volgt, gewist.

CTRL + F
Hiermee wordt de cursor naar het begin van het volgende woord verplaatst.

CTRL + J
Hiermee wordt de cursor naar het begin van de volgende programmaregel verplaatst.

CTRL + N
Hiermee wordt de cursor naar het einde van de programmaregel verplaatst.

CTRL + U
Hiermee wordt de hele programmaregel gewist.

Tot zover de beschrijving van de editor-functies. Probeert u de gegeven functies eens uit op het nog in het geheugen staande programma. U mag daar gerust een avondje aan spanderen. De ervaring die u daarbij nu opdoet, zult u later goed kunnen gebruiken. Om te zien of het programma in het geheugen nog hetzelfde is als dat op het beeldscherm, dient u na een of enkele wijzigingen een nieuwe listing te maken met

het LIST-commando.

Ook het LIST-commando kan worden vergezeld van een aantal parameters. Hier volgen alle mogelijke formaten van het LIST-commando.

LIST
Drukt alle programmaregels, van het begin tot het einde, af op het beeldscherm.

LIST.
Drukt de laatst behandelde regel af. Dit kan de laatste regel zijn die wij hebben ingetikt, maar het kan ook de regel zijn waarin de BASIC-interpreter tijdens het uitvoeren van het programma een fout ontdekte. Stopt de computer met een foutboodschap, waarin wordt gezegd dat er een fout in een bepaalde regel is ontdekt, dan kunnen we die regel met LIST. afdrukken.

LIST nn
Drukt programmaregel nn af op het beeldscherm.

LIST nn-
Drukt alle programmaregels, vanaf regelnummer nn tot het einde, af op het beeldscherm.

LIST nn-mm
Drukt de programmaregels nn tot en met mm af op het beeldscherm.

U hebt nu alle gereedschappen om een programma in te voeren, te corrigeren en te starten in handen. Oefen ruimschoots met alle gegeven commando's en editor-functies, voordat u verder gaat met het volgende hoofdstuk. In het volgende hoofdstuk worden een groot aantal programmavoorbeelden gegeven. Deze voorbeelden dient u in te voeren, eventueel te corrigeren en te starten. U zult dan zien wat het resultaat van die voorbeelden is. Dit zal u helpen bij het begrijpen van de geboden theorie.

4 Werken met constanten en variabelen

Voor het werken met constanten en variabelen bestaan in MSX-BASIC een aantal belangrijke regels. We zullen eens zien wat constanten en variabelen zijn en wat de daarbij behorende regels zijn.

Constanten

Constanten kunnen worden ingedeeld in twee groepen, namelijk in numerieke- en alfanumerieke constanten.

Voorbeelden van numerieke constanten zijn:

137, -6, 12.23

Voorbeelden van alfanumerieke constanten zijn:

"MSX-BASIC", "137", "FC-200"

Numerieke constanten zijn positieve- of negatieve getallen, waarmee berekeningen, zoals optellen, aftrekken, vermenigvuldigen en delen, kunnen worden uitgevoerd. Het volgende programma toont hiervan enkele voorbeelden.

```
10 PRINT 6+4
20 PRINT 12.4+8*7
30 PRINT 26.37-12.23
40 PRINT 78/4.3
```

Alfanumerieke constanten, ook wel strings genoemd, bestaan uit woorden of teksten. Deze constanten staan in tegenstelling tot numerieke constanten altijd tussen aanhalingstekens.

De maximaal toegestane lengte van een alfanumerieke constante is 255 tekens. Wanneer een alfanumerieke constante een getal vertegenwoordigt, bijvoorbeeld "137", kan daar niet zonder meer een berekening mee worden uitgevoerd. De alfanumerieke zal daarvoor eerst moeten worden omgezet in een numerieke constante. Alfanumerieke constanten kunnen alleen bij elkaar worden opgeteld, hetgeen echter een heel ander resultaat geeft dan het optellen van twee numerieke variabelen. Het volgende programma laat hiervan enkele voorbeelden zien.

```
10 PRINT "GOLDSTAR"
20 PRINT "PERSONAL"+" COMPUTER"
30 PRINT "FC-"+"200"
40 PRINT "19"+"85"
```

Variabelen

In MSX-BASIC kunnen de variabelen worden onderscheiden in numerieke- en alfanumerieke variabelen.

Voorbeelden van numerieke variabelen zijn:

A, B!, D%, E#, SOM!

Voorbeelden van alfanumerieke variabelen zijn:

A$, B$, NAAM$

Numerieke variabelen kunnen positieve of negatieve getallen bevatten. Alfanumerieke variabelen kunnen woorden of teksten bevatten. Om onderscheid tussen de twee soorten variabelen te kunnen maken, wordt aan alfanumerieke variabelen achter de naam het dollarteken ($) toegevoegd.

Een alfanumerieke variabele bezet in het geheugen drie bytes plus het aantal bytes dat nodig is voor het opbergen van de inhoud van de variabele. Voor ieder teken is een byte nodig.

Aan een numerieke variabele kan het procentteken (%), het

uitroepteken (!), het hekje (#) of niets worden toegevoegd. De betekenis van deze toevoegingen zal in paragraaf 4.1 worden toegelicht.

De naam van een variable moet altijd met een letter beginnen (A tot en met Z), terwijl de rest van de naam uit letters, cijfers of leestekens, of een combinatie daarvan mag bestaan. MSX-BASIC herkent echter alleen de eerste twee tekens van de naam. Een gevolg hiervan is, dat de variabelenamen START en STOP voor MSX-BASIC dezelfde zijn. Ze beginnen immers allebeide met ST. Het hiernavolgende programma geeft een aantal voorbeelden van het gebruik van constanten en variabelen.

```
10 A$="CONSTANTEN"
20 B$=" EN "
30 C$="VARIABELEN"
40 PRINT A$+B$+C$
50 A=12
60 B=6
70 SOM=A+B
80 PRINT SOM
```

4.1 Constanten

Numerieke constanten zijn positieve- of negatieve getallen, die in MSX-BASIC als volgt worden ingedeeld:

1. Gehele getallen (integers)
2. Constanten met vaste komma (fixed point)
3. Constanten met drijvende komma (floating point)
4. Binaire constanten
5. Hexadecimale constanten
6. Octale constanten

Groep 1 vertegenwoordigt alle gehele getallen van -32768 tot en met +32767. In gehele getallen mogen geen decimalen voorkomen, dus geen komma. In MSX-BASIC wordt de komma geschreven als punt. Gehele getallen worden aangegeven met het procentteken (%) en mogen uit maximaal vijf cijfers be-

staan. In het geheugen nemen gehele getallen twee bytes in beslag. Variabelen ten behoeve van gehele getallen worden ook met het procentteken (%) aangegeven. Enkele voorbeelden zijn:

21%
-237%
A%=57

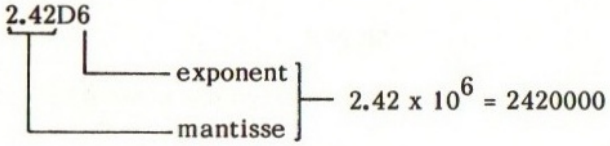
Groep 2 vertegenwoordigt de positieve en negatieve reele getallen. Deze verzameling getallen, die ook decimalen kunnen bevatten, worden constanten met vaste komma (fixed point) genoemd. Enkele voorbeelden van deze getallen zijn:

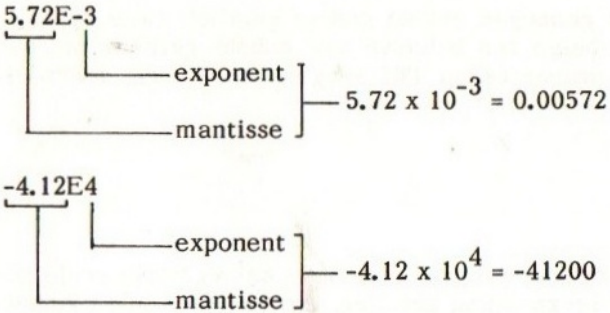
56.12
102.37
-24.6
0.0027

Groep 3 vertegenwoordigt de positieve en negatieve getallen, die in een exponentiele vorm zijn geschreven. Deze getallen worden constanten met drijvende komma (floating point) genoemd en bestaan uit een mantisse en een exponent.

De mantisse mag een geheel getal of een getal met vaste komma zijn. De waarde van de mantisse kan positief of negatief zijn.
De exponent is een positief- of negatief geheel getal.

De mantisse en de exponent worden van elkaar gescheiden door de letter E of D. De volgende voorbeelden laten een aantal drijvende komma getallen zien.





Constanten met vaste- en drijvende komma (groepen 2 en 3) kunnen in twee vormen worden gegeven:

- constanten met enkelvoudige nauwkeurigheid.
- constanten met dubbelvoudige nauwkeurigheid.

Een constante met enkelvoudige nauwkeurigheid bezet vier bytes in het geheugen en mag uit maximaal zes cijfers bestaan.

Constanten met vaste komma en enkelvoudige nauwkeurigheid worden aangegeven met een uitroepteken (!). Dit geldt ook voor variabelen waarin deze constanten moeten worden opgeborgen (bijvoorbeeld: C!).

Voor constanten met drijvende komma wordt als symbool voor de exponent de letter E gebruikt.

Een constante met dubbelvoudige nauwkeurigheid kan zijn gegeven als een constante met vaste- of met drijvende komma.

Een voorbeeld van een constante met vaste komma is:

56.25#

Een voorbeeld van een constante met drijvende komma is:

7.128D-8

Een constante met dubbelvoudige nauwkeurigheid bezet 8

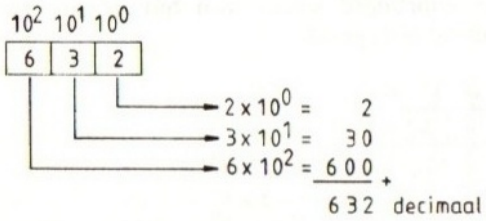
bytes in het geheugen en mag uit maximaal 14 cijfers bestaan. Constanten met dubbelvoudige nauwkeurigheid en vaste komma worden aangegeven met een hekje (#). Dit geldt ook voor de variabelen waar deze constanten in worden opgeslagen (voorbeeld: SOM#). Indien de constante met dubbelvoudige nauwkeurigheid een drijvende komma bevat, wordt als symbool voor de exponent de letter D gebruikt.

Opmerking:

Wanneer een naam van een variabele niet eindigt met %, !, # of $, dan zal MSX-BASIC deze variabele beschouwen voor dubbelvoudige nauwkeurigheid.

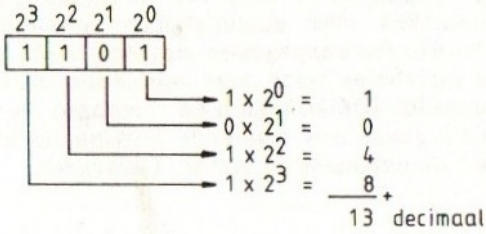
Numerieke constanten kunnen ook worden gegeven in de binaire (tweetallige), octale (achttallige) of hexadecimale (zestientallige) notatie. Voor de uitleg van het binaire-, octale- en hexadecimale talstelsel zullen we uitgaan van het reeds bekende decimale talstelsel.

In het decimale talstelsel wordt gebruik gemaakt van de cijfers 0 tot en met 9. Dit zijn 10 cijfers. We zeggen dan ook dat het grondtal 10 is. Het volgende voorbeeld laat zien hoe een getal in het decimale stelsel wordt verkregen:



Voor het binaire-, octale- en hexadecimale talstelsel kunnen we op dezelfde manier te werk gaan.

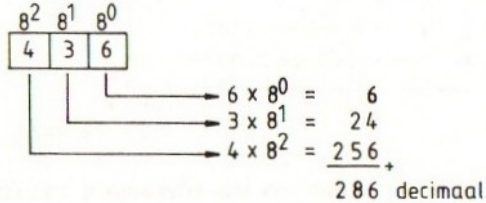
In het binaire talstelsel bestaan maar twee cijfers, de 0 en de 1. Dit betekent dat het grondtal 2 is. Het volgende voorbeeld laat zien hoe we van een binair getal de decimale waarde kunnen bepalen:



De cijfers van een binair getal worden bits genoemd. Het woordje bits is een samentrekking van de Amerikaanse woorden BInary digiTS. Binaire constanten worden in MSX-BASIC voorafgegaan door de tekens &B. Binaire constanten mogen uit maximaal 16 eentjes bestaan (dit komt overeen met de decimale waarde 65535). Het volgende programma laat zien hoe de decimale waarde van een binaire constante kan worden afgedrukt.

```
10 PRINT &B1101
20 PRINT &B10110
```

In het octale talstelsel worden de cijfers 0 tot en met 7 gebruikt. Dit zijn in totaal 8 cijfers, ofwel het grondtal is 8. In het volgende voorbeeld wordt van het octale getal 436 de decimale waarde berekend:



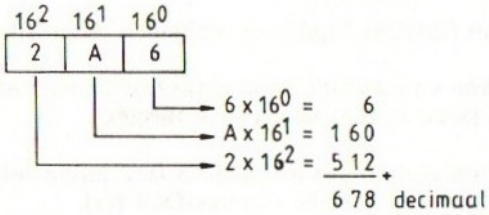
Octale constanten worden in MSX-BASIC voorafgegaan door de tekens &O (het teken & en de letter O). Een octale constante mag maximaal 177777 zijn. Dit komt overeen met de decimale waarde 65535. Het volgende programma laat zien hoe de decimale waarde van een octale constante kan worden afgedrukt.

```
10 PRINT &O124
20 PRINT &O672
```

Met binaire- en octale constanten kunnen ook berekeningen worden uitgevoerd. Enkele voorbeelden hiervan ziet u in het volgende programma.

```
10 PRINT &O27+&O142
20 PRINT &B10111+&O64
30 PRINT 12+&B101
```

In het hexadecimale talstelsel wordt gebruik gemaakt van 16 verschillende cijfers. Het grondtal is dus 16. Daar we echter maar 10 cijfers kennen, moeten we voor de overige 6 cijfers van het hexadecimale talstelsel een oplossing zoeken. Deze oplossing is gevonden door daarvoor de eerst 6 letters van het alfabet te gebruiken. De zestien cijfers zijn dus 0 tot en met 9 en A tot en met F. De hexadecimale A betekent dus 10, B betekent 11, etc. In het volgende voorbeeld wordt van het hexadecimale getal 2A6 de decimale waarde berekend:



Hexadecimale constanten worden in MSX-BASIC voorafgegaan door de tekens &H. De waarde van een hexadecimale constante mag maximaal FFFF zijn. Dit komt overeen met de decimale waarde 65535. Het volgende programma laat zien hoe men van hexadecimale constanten de decimale waarde kan laten afdrukken. In dat programma zijn ook een aantal berekeningen opgenomen. Evenals met binaire- en octale getallen kunnen ook met hexadecimale constanten berekeningen worden uitgevoerd.

```
10 PRINT &H2E6
20 PRINT &H12B+&H3E
30 PRINT &H27C-&B101011
```

Opmerking:

Wanneer de decimale waarde van een binaire-, octale- of hexadecimale constante groter is dan 32767, wordt er van het resultaat 65536 afgetrokken. Men verkrijgt dan een negatief resultaat, dat in de 2-complement notatie staat. Binair gezien is het meest significante bit (het bit met de hoogste waarde, ofwel het meest linkse bit) dan logisch 1. Dit bit wordt gezien als het teken-bit. Is het 1 dan is het getal negatief, is het 0 dan is het getal positief. Het volgende programma laat hiervan een tweetal voorbeelden zien.

```
10 PRINT &HF02A
20 PRINT &O176234
```

4.2 Alfanumerieke functies

Alfanumerieke functies bieden de volgende mogelijkheden:

Het bepalen van de binaire notatie (alfanumeriek) van een decimaal getal x, met de functie **BIN$(x).**

Het bepalen van de octale notatie (alfanumeriek) van een decimaal getal x, met de functie **OCT$(x).**

Het bepalen van de hexadecimale notatie (alfanumeriek) van een decimaal getal x, met de functie **HEX$(x).**

Het bepalen van de decimale notatie (alfanumeriek) van een numerieke waarde x, met de functie **STR$(x).**

Het bepalen van de numerieke waarde (decimaal) van een alfanumerieke constante of variabele met de functie **VAL(X$).**

Argument x kan een numerieke constante (bijvoorbeeld 37),

een numerieke variabele (bijvoorbeeld A) of een expressie (bijvoorbeeld C+D) zijn.

Voor de functies BIN$, OCT$ en HEX$ moet x een geheel getal zijn, waarvan de waarde moet liggen tussen -32769 en +65536. Wanneer x een negatief getal is wordt de 2-complementnotatie gebruikt. Voorbeelden hiervan zijn:

x = -12215 wordt x = 65536 - 12215 = 53321
x = -3 wordt x = 65536 - 3 = 65533

Indien x in de functies BIN$, OCT$ en HEX$ geen geheel getal is, worden eerst de decimalen achter de komma (decimale punt) afgekapt. Van 37.23 zal dus eerst 37 worden gemaakt. Pas daarna zal de functie worden uitgevoerd.

Argument x mag ook worden gegeven als een binaire-, octale- of hexadecimale waarde. In dat geval wordt de waarde eerst omgezet naar een decimale waarde. Daarna wordt de functie uitgevoerd.

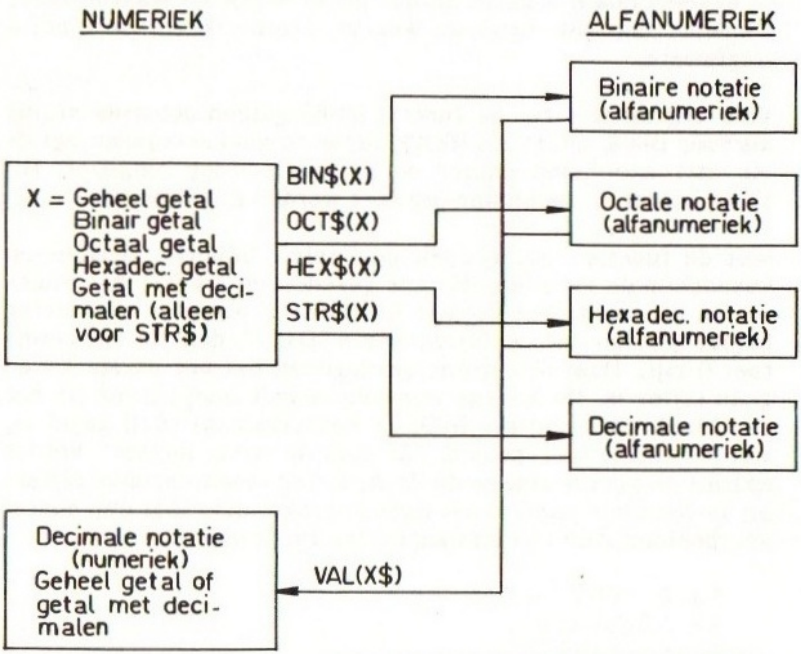
Voor argument x van de functie STR$ gelden dezelfde regels als voor BIN$, OCT$ en HEX$, maar bovendien kunnen getallen met decimalen achter de komma worden omgezet. Bij STR$ zullen de decimalen dus niet worden afgekapt.

Met de functie VAL worden decimale-, binaire-, octale- en hexadecimale getallen, die zijn geschreven als strings, omgezet in de numerieke notatie (decimaal). Wanneer een string begint met een letter, bijvoorbeeld "D127", dan zal het resultaat 0 zijn. Daarmee wordt aangegeven dat het eerste teken geen cijfer is. De tekens waarmee wordt aangegeven of het een binair (&B), octaal (&O) of hexadecimaal (&H) getal is, worden wel geaccepteerd. De functie VAL negeert verder spaties en andere tekens die in de string staan, behalve cijfers en de decimale punt. Het volgende programma laat een aantal voorbeelden zien van alfanumerieke functies:

```
10 D=106: N=-23
20 A$=BIN$(D)
30 B$=BIN$(N)
```

```
40 PRINT A$,B$
50 PRINT
60 K=10: L=&B1010: M=&O273
70 A$=STR$(K)
80 B$=STR$(L+8)
90 C$=STR$(M)
100 PRINT A$,B$,C$
110 PRINT
120 PRINT
130 A$="256.37": B$="A6 12": D$="12 STUKS"
140 A=VAL(A$)
150 B=VAL(B$)
160 C=VAL(D$)
170 PRINT A,B,C
```

Afbeelding 4-1 geeft een overzicht van de verschillende conversies.



Afb. 4-1 Overzicht conversie-mogelijkheden.

4.3 Conversie-functies

Met conversie-functies kunnen numerieke waarden (decimaal, binair, octaal of hexadecimaal) en expressies worden geconverteerd in gehele getallen, getallen met enkelvoudige nauwkeurigheid of getallen met dubbelvoudige nauwkeurigheid. Het resultaat is altijd een decimale waarde. De functies die hiervoor worden gebruikt zijn:

CINT(X)
Converteert een numerieke waarde of expressie in een geheel getal. Dat getal bestaat uit maximaal 5 cijfers en bezet 2 bytes in het geheugen.

CSNG(X)
Converteert een numerieke waarde of expressie in een getal met enkelvoudige nauwkeurigheid. Het getal bestaat uit maximaal 6 cijfers en bezet 4 bytes in het geheugen.

CDBL(X)
Converteert een numerieke waarde of expressie in een getal met dubbelvoudige nauwkeurigheid. Het getal bestaat uit maximaal 14 cijfers en bezet 8 bytes in het geheugen.

Het volgende programma laat een aantal voorbeelden van conversiefuncties zien.

```
10 L%=14: M#=6.2789876#
20 C%=CINT(4/3): PRINT "C%=";C%
30 D%=CINT(&H2A3): PRINT "D%=";D%
40 E!=CSNG(L%/3): PRINT "E!=";E!
50 F!=CSNG(M#): PRINT "F!=";F!
60 G#=CSNG(4/3): PRINT "G#=";G#
70 H#=CDBL(4/3): PRINT "H#=";H#
```

4.4 Declareren van variabelen

In MSX-BASIC is het mogelijk om in het begin van een programma de variabelen, die in dat programma zullen worden gebruikt, te definieren (declareren). Men kan de variabelen

definieren als variabele voor een geheel getal, als variabele voor een getal met enkelvoudige of dubbelvoudige nauwkeurigheid, of als string-variabele. Men hoeft dan in het programma de gedefinieerde variabelen niet meer te voorzien van de type-aanduidingen (%, !, # of $). De statements die hiervoor worden gebruikt zijn:

DEFINT lijst met beginletters van variabelen
Hiermee worden variabelen ten behoeve van gehele getallen gedefinieerd.

DEFSNG lijst met beginletters van variabelen
Hiermee worden variabelen ten behoeve van getallen met enkelvoudige nauwkeurigheid gedefinieerd.

DEFDBL lijst met beginletters van variabelen
Hiermee worden variabelen ten behoeve van getallen met dubbelvoudige nauwkeurigheid gedefinieerd.

DEFSTR lijst met beginletters van variabelen
Hiermee worden variabelen ten behoeve van alfanumerieke constanten (strings) gedefinieerd.

Als beginletter mag iedere letter van het alfabet (A t/m Z) worden gebruikt. Voor het definieren kan een keuze uit de volgende mogelijkheden worden gemaakt:

Een enkele letter, bijvoorbeeld DEFINT D.
Met dit statement worden alle variabelen die met de letter D beginnen gedefinieerd als variabelen ten behoeve van gehele getallen.

Meerdere letters, van elkaar gescheiden door een komma, bijvoorbeeld DEFSNG A,B,G.
Met dit statement worden alle variabelen die beginnen met de letters A, B en G gedefinieerd als variabelen ten behoeve van getallen met enkelvoudige nauwkeurigheid.

Een serie opeenvolgende letters, bijv. DEFSTR R-T.
Met dit statement worden alle variabelen die beginnen met de letters R tot en met T gedefinieerd als variabelen

ten behoeve van alfanumerieke constanten (strings).

Wanneer in een programma een gedeclareerde variabele wordt aangegeven met een type-aanduiding (%, !, # of $), die niet overeenkomt met het gedeclareerde type, dan zal de in het programma gegeven type aanduiding (bij de variabele zelf) de oorspronkelijk gedeclareerde type-aanduiding herroepen. Het volgende programma laat dit zien.

```
10 DEFINT A,B,C
20 A=36: B=5
30 C=A/B: PRINT C
40 C!=A/B: PRINT C!
```

In voorgaand programma herroept type-aanduiding ! in regel 40 de definitie van variabele C in regel 10. Het volgende programma geeft nog een aantal voorbeelden met definitiestatements.

```
10 DEFINT A,B
20 DEFDBL C
30 DEFSTR K-M
40 A=23.12: PRINT A
50 A=24: B=7
60 C=A/B: PRINT C
70 K="MSX": L="-": M="BASIC"
80 PRINT K;L;M
```

5 Het toekennen van waarden aan variabelen

Het geheugen kunnen we ons misschien het beste voorstellen als een grote kast met heel veel vakjes. Er zijn vakjes van verschillende grootten. Dit hebben we al in het vorige hoofdstuk gezien. We zagen immers dat gehele getallen in het geheugen twee bytes in beslag nemen. Willen we zo'n getal opbergen in het geheugen, dan zullen we een vakje van twee bytes moeten reserveren. Dat kastje geven we dan een naam, zodat we het later kunnen terugvinden. Zo'n kastje noemen we een variabele. We kunnen er immers steeds weer een andere waarde in opbergen. De naam die we er aan hebben gegeven noemen we een variabelenaam.

Hoe we in BASIC waarden in die kastjes kunnen plaatsen, of anders gezegd, hoe we aan variabelen waarden kunnen toekennen, zullen we nu gaan bekijken. Daarbij zullen we een groot aantal statements tegenkomen, die allemaal waarden in variabelen plaatsen. In de voorbeelden zullen ook een aantal statements voorkomen, die u nog niet kent. We zullen daar nu nog niet veel aandacht aan besteden. U kunt de voorbeelden in ieder geval intikken en uitvoeren, om te zien wat er gebeurt. De statements, die u nog niet helemaal begrijpt, zullen in de volgende hoofdstukken worden uitgelegd.

```
10 REM Oppervlakte berekenen
20 LET L=20
30 LET B=15
40 LET O=L*B
50 PRINT O
60 END
```

Regel 10 in voorgaand programma is een zogenaamde REM-regel. REM is de afkorting van het woord Remark. Vertaald in het Nederlands is dat "opmerking". De computer beschouwd alles wat achter het **REM-statement** volgt als commentaar, en doet daar niets mee. Dit geldt ook als er na het REM-statement nog een tweede statement in dezelfde regel staat. Die tweede statement zal niet worden uitgevoerd. De functie van het REM-statement is, om het programma beter leesbaar te maken voor ons, programmeurs.

```
10 REM commentaar: PRINT "commentaar"
```

De hier gegeven REM-regel bevat een tweede statement, die echter nooit zal worden uitgevoerd. Zodra de computer het REM-statement ziet, gaat hij er van uit dat de rest van de regel commentaar is, en dus niet hoeft worden uitgevoerd. In plaats van het woordje REM mag ook het aanhalingsteken (') worden gebruikt. De functie is precies dezelfde. De nu volgende regel 10 is functioneel exact gelijk aan de vorige regel 10:

```
10 ' commentaar: PRINT "commentaar"
```

De regels 20, 30 en 40 van het BASIC-programma kennen waarden toe aan variabelen. Het **LET-statement** betekent niet meer dan LAAT WORDEN. LET A=10 betekent dan ook LAAT A 10 WORDEN. In plaats van een constante toe te kennen aan een variabele, mogen we ook een uitdrukking specificeren. Een voorbeeld hiervan is te zien in regel 40. Daarin zeggen we: Laat O(ppervlak) gelijk zijn aan L(engte) keer B(reedte). De computer zal eerst de uitdrukking omzetten in een constante. Daarna wordt het resultaat in variabele O gezet.

Met regel 50 drukken we de inhoud van variabele O af op het beeldscherm. Anders gezegd, we schrijven de inhoud van het geheugenvakje met de naam O op het beeldscherm. Regel 60, tenslotte, beeindigd het programma op een nette manier. Na het uitvoeren van het programma zal het END-statement de tekst "Ok" op het beeldscherm te zien geven. Hiermee geeft de computer aan dat het programma is uitgevoerd.

We hebben nu twee belangrijke dingen gezien. Ten eerste dat met het LET-statement waarden aan variabelen kunnen worden toegekend. Ten tweede dat die waarde niet beslist een constante hoeft te zijn, maar dat deze ook een uitdrukking mag zijn. Het "*"-tekentje in de gegeven uitdrukking betekent "vermenigvuldig". In BASIC wordt dit teken een "rekenkundige operator" genoemd. Er zijn nog meer van die operators. In MSX-BASIC bestaan de volgende rekenkundige operators:

+	Optellen
-	aftrekken
*	vermenigvuldigen
/	delen
\	delen van gehele getallen
MOD	restbepalen
^	machtsverheffen

Optellen en aftrekken hebben hiervan de laagste prioriteit. Machtsverheffen heeft de hoogste prioriteit. Prioriteit wil hier het volgende zeggen. Stel dat er in een uitdrukking staat:

```
LET A= 10 + 5 * 2
```

Zou je deze formule gewoon van links naar rechts berekenen, dan zou de uitkomst zijn: 10 + 5 = 15, 15 * 2 = 30. Daar echter vermenigvuldigen een hogere prioriteit heeft dan optellen, zal de computer eerst uitrekenen hoeveel 5 * 2 is (10) en daar vervolgens 10 bij optellen. Het resultaat is dan dus 20. Alleen wanneer in een uitdrukking twee of meer operators van dezelfde prioriteit naast elkaar staan, zal de computer de berekeningen van links naar rechts uitvoeren.

Willen we van deze prioriteitsregels afwijken, dan moeten we gebruik maken van haakjes. Indien we dus de computer de formule inderdaad van links naar rechts willen laten uitrekenen, dan zullen we moeten programmeren:

```
LET A= (10 + 5) * 2
```

De computer zal altijd eerst de uitdrukking tussen de haakjes uitwerken, pas daarna worden de andere delen van de uitdrukking uitgewerkt. We zullen hier terdege rekening mee moeten houden, omdat het resultaat anders wel eens heel anders kan uitvallen dan we hadden verwacht. Dit is wel duidelijk gebleken uit het voorbeeldje.

Behalve rekenkundige operators zijn er ook nog functies, vergelijkende operators en logische operators. Ook tussen deze groepen van operators zijn weer prioriteitsverschillen aangebracht. Logische operators hebben de laagste prioriteit, daarna volgen de vergelijkende operators, dan de rekenkundige operators en tenslotte de functies. Een voorbeeld van een logische operator is AND. Het "is gelijk"-teken (=) is een vergelijkende operator. De functie SQR(x) is een voorbeeld van een functie. In een uitdrukking mogen alle soorten operators door elkaar worden gebruikt.

```
LET X = 3 * SQR(4)
```

In dit voorbeeld zal de computer eerst de vierkantswortel uit 4 trekken. Het resultaat daarvan wordt vermenigvuldigd met 3. We zullen nu niet verder ingaan op de verschillende operators. Waar dit te pas komt zullen we er meer over vertellen. Voor dit moment is het vooral belangrijk te weten dat ze in uitdrukkingen mogen worden gebruikt, dat de computer ze in een vastgestelde volgorde afwerkt, en dat wij die verwerkingsvolgorde kunnen beinvloeden door haakjes te gebruiken.

5.1 Ingave statements

Tot nu toe was het enige statement, waarmee we waarden aan variabelen kunnen toekennen en dat we hebben behandeld, het LET-statement. Het nadeel van dit statement is echter dat, elke keer dat het wordt uitgevoerd, dezelfde waarde aan de variabele wordt gegeven. Wanneer we steeds een andere waarde willen toekennen, zullen we een andere statement moeten gebruiken.

```
10 INPUT I
```

```
20 PRINT I
30 END
```

Als u dit programma hebt ingetikt en gestart, zal er een vraagteken op het beeldscherm verschijnen. U kunt nu een waarde intikken. Zodra u de waarde, gevolgd door de RETURN-toets, hebt ingetikt, wordt die waarde op het beeldscherm afgedrukt. De ingetoetste waarde wordt namelijk in regel 10 in variabele I opgeslagen en met regel 20 wordt de inhoud van variabele I afgedrukt. De volgende keer dat u het programma start kunt u weer een andere waarde intikken. Met regel 10 wordt die andere waarde dan aan variabele I toegekend.

Zowel met het LET-statement als met het **INPUT-statement** kunnen in plaats van waarden (numeriek), ook teksten (strings) worden toegekend aan variabelen. Een eerste voorwaarde is dan dat die variabele een string-variabele is. Een string-variabele wordt gedefinieerd door achter de naam het $-teken te plaatsen. Voorbeelden hiervan zijn:

```
10 LET A$="tekst"
20 INPUT B$
30 PRINT A$: PRINT B$
40 END
```

Zijn er dan geen verschillen tussen een LET-statement en een INPUT-statement? Toch wel. Zoals we hebben gezien mogen we in het LET-statement in plaats van een waarde ook een uitdrukking gebruiken. Dit mag in een INPUT-statement niet. Wanneer we het LET-statement gebruiken om er strings mee aan variabelen toe te kennen, dan mogen we zelfs daarbij een uitdrukking gebruiken. Een bijzondere uitdrukking is: LET A$="MSX-"+"BASIC". Hier worden twee strings (die ieder tussen aanhalingstekens staan) aan elkaar gekoppeld. Indien variabele A$ wordt afgedrukt, zal het resultaat MSX-BASIC zijn. Dit aan elkaar koppelen van strings wordt meestal aangeduid met het Engelse woord "Concatination".

Een ander verschil tussen LET en INPUT is dat het bij INPUT mogelijk is een verklarende tekst af te drukken op het beeld-

scherm.

```
10 INPUT "EERSTE GETAL";A
20 INPUT "TWEEDE GETAL";B
30 LET C=A+B
40 PRINT "RESULTAAT =";C
50 END
```

In voorgaand programma zal met het INPUT-statement op regel 10 de tussen aanhalingstekens geplaatste tekst op het beeldscherm worden afgedrukt. Daarna wordt het vraagteken afgedrukt, en blijft de computer op uw antwoord wachten. Regel 20 werkt net zo, doch drukt een andere tekst af en zet het antwoord in een andere variabele. Met regel 30 wordt de inhoud van variabele A bij die van variabele B opgeteld en wordt het resultaat van die optelling in variabele C gezet. Met regel 40 wordt de tekst RESULTAAT = afgedrukt, gevolgd door de inhoud van variabele C.

Een beperking van het INPUT-statement is, dat het niet is toegestaan om bij het ingeven van een alfanumerieke tekst komma's of aanhalingstekens te gebruiken. Indien een komma in de tekst voorkomt, neemt de computer aan dat er meerdere teksten zijn ingegeven. Was er echter slechts een variabele om een waarde aan toe te kennen, dan neemt de computer aan dat de ingave niet goed was.

```
10 INPUT "Geef 2 getallen in";A,B
20 PRINT A+B
30 END
```

Hier ziet u een voorbeeld waarbij in een INPUT-statement twee waarden worden opgevraagd. U mag nu twee waarden intikken, die van elkaar zijn gescheiden door een komma. De komma is voor de computer een aanwijzing dat de eerste waarde is ingetikt en dat het intikken van de tweede waarde begint.

Over het LET-statement dient nog iets te worden gezegd. Het LET-statement mag worden weggelaten. In plaats van LET A=10 mag je dus ook gewoon zeggen A=10. De computer weet

dan dat we daarmee bedoelen LET A=10.

Ten aanzien van het toekennen van alfanumerieke waarden (strings) aan string-variabelen nog het volgende. De aan de variabelen toegekende strings worden in een apart deel van het geheugen opgeborgen. Het deel van het geheugen dat hiervoor na het aanschakelen van de computer is gereserveerd is slechts 200 bytes groot. Voor de meeste kleine programma's zal dit wel voldoende zijn, doch wanneer er in een programma veel string-variabelen worden gebruikt, en er worden lange strings aan toegekend, dan zal 200 bytes (200 letters, cijfers of leestekens) niet voldoende zijn. Om die strings dan toch in het geheugen te kunnen opslaan, zal dit gereserveerde geheugengebied groter moeten worden gemaakt. Dit kan worden gedaan met het **CLEAR-statement.**

Geeft u maar eens het volgende programma in:

```
10 INPUT A$
20 PRINT A$
30 END
```

Als u dit programma start, en een tekst van bijvoorbeeld 10 tekens ingeeft, dan zal die tekst keurig in A$ worden opgeborgen. Geef nu echter eens de opdracht: CLEAR 5. Als u nu het programma nog eens start, en u geeft een tekst in, die langer is dan 5 tekens, dan zult u een foutboodschap krijgen waaruit blijkt dat het geheugengebied waarin de strings worden opgeborgen te klein is voor de ingegeven string (out of string space). Door nu de opdracht CLEAR 200 te geven zal er weer voldoende ruimte zijn. De volgende keer dat u het programma start werkt het weer correct.

Het CLEAR-statement doet echter nog meer. Er kunnen twee parameters aan worden meegegeven, die de volgende mogelijkheden bieden:

CLEAR

CLEAR zonder parameters geeft alle numerieke variabelen de waarde 0, maakt alle alfanumerieke (string-) variabelen leeg en sluit alle bestanden (hierop komen we in een

later hoofdstuk nog kort terug) af.

CLEAR nn

Als CLEAR, maar bovendien wordt de geheugenruimte die voor het opslaan van de inhoud van string-variabelen moet worden gereserveerd de in **nn** gegeven grootte gemaakt.

CLEAR nn,mm

Als CLEAR nn, maar bovendien wordt het hoogste geheugenadres dat door MSX-BASIC mag worden gebruikt beperkt tot het in **mm** opgegeven adres.

Dit laatste formaat van het CLEAR-statement zult u pas gaan gebruiken wanneer u machinetaalprogramma's schrijft. U kunt dan voorkomen dat uw machinetaalprogramma's door het BASIC-programma worden overschreven. Daar dit, zeker in dit boek, nog niet van toepassing is, zullen we daar niet verder op ingaan.

Een eenvoudig statement is het **SWAP-statement.** Hiermee kan de inhoud van twee variabelen onderling worden verwisseld. Dit wil zeggen dat, indien we twee variabelen hebben gevuld (bijvoorbeeld variabelen A en B), we de inhoud van variabele A in B kunnen zetten en tegelijkertijd de inhoud van variabele B in A. Het volgende voorbeeld laat dit zien.

```
10 LET A=10
20 LET B=20
30 PRINT A,B
40 SWAP A,B
50 PRINT A,B
60 END
```

Met regel 30 van het programma worden de variabelen A en B afgedrukt, en zal op het beeldscherm 10 en 20 worden afgedrukt. Met regel 40 wordt de inhoud van de variabelen A en B omgewisseld. Daarna worden met regel 50 de variabelen A en B nogmaals afgedrukt. Nu zal echter op het beeldscherm 20 en 10 worden afgedrukt. De enige regel die bij het SWAP-statement van belang is, is dat de twee variabelen allebei van

hetzelfde type dienen te zijn. Ze moeten allebei numeriek of allebei alfanumeriek zijn. SWAP A,B$ is dus niet mogelijk. SWAP A$,B$ is wel toegestaan.

Voor het toekennen van een alfanumerieke waarde aan een string-variabele bestaat nog een andere versie van het INPUT-statement, namelijk het **LINE INPUT-statement.** Het LINE INPUT-statement accepteert alleen alfanumerieke ingaven, en kent deze aan een string-variabele toe. Er bestaan twee formaten van dit statement.

LINE INPUT A$

Accepteert een alfanumerieke ingave van maximaal 254 tekens lengte, die via het toetsenbord wordt ingegeven. De belangrijkste verschillen tussen INPUT en LINE INPUT zijn, dat bij LINE INPUT geen vraagteken door de computer wordt afgedrukt, dat het bij LINE INPUT is toegestaan om in de in te geven tekst ook komma's en aanhalingstekens op te nemen, en dat er maar een variabele mag worden gebruikt.

LINE INPUT #n,A$

Hiermee kan een "record" uit een bestand worden gelezen. Later in dit boek zullen we daar nader op ingaan. Om dit statement te kunnen begrijpen moet u ook weten hoe het OPEN-statement werkt. Een bestand kan namelijk pas worden gelezen nadat het is geopend.

5.2 Ingave-functies

Er resten ons nu nog twee functies waarmee gegevens kunnen worden opgevraagd. De eerste functie is **INPUT$(n).** Met deze functie kunnen een bepaald aantal tekens van het toetsenbord worden geaccepteerd, of uit een bestand worden gelezen.

```
LET A$=INPUT$(5)
```

Hiermee wordt aan variabele A$ de alfanumerieke waarde toegekend die met de functie INPUT$ van het toetsenbord wordt gelezen. Tussen haakjes is aangegeven hoeveel tekens

er van het toetsenbord zullen worden geaccepteerd. Probeer het volgende programma maar eens uit.

```
10 PRINT INPUT$(5)
20 END
```

U ziet bij het uitvoeren van dit programma wel een vraagteken verschijnen, doch de ingetikte tekens ziet u niet. Na vijf ingetikte tekens worden die tekens echter plotseling afgedrukt. Deze functie kan erg handig zijn wanneer u in een programma een bepaald aantal tekens vanaf het toetsenbord wilt ophalen en in een variabele wilt plaatsen. Een postcode kan bijvoorbeeld nooit meer dan zes tekens bevatten. Om een postcode te laten ingeven zou in het programma de functie INPUT$(6) kunnen worden opgenomen.

De laatste in dit hoofdstuk te behandelen functie is de **INKEY$-functie.** Op het moment dat de INKEY$-functie wordt uitgevoerd, wordt het toetsenbord even gelezen. Mocht op dat moment een toets zijn ingedrukt, dan zal de code die bij die toets hoort aan het programma worden gegeven. Wordt er op dat moment geen toets ingedrukt, dan zal een lege string aan het programma worden gegeven. Het volgende voorbeeld laat dit zien. In dit voorbeeld worden een aantal statements gebruikt die pas in volgende hoofdstukken zullen worden behandeld. Mocht u die statements nog niet begrijpen, dan hoeft u zich daarover geen zorgen te maken. Ze komen later nog aan bod.

```
10 LET A$=INKEY$
20 IF A$="" GOTO 10
30 PRINT A$
40 GOTO 10
```

Zodra regel 10 wordt uitgevoerd, wordt de toets die op datzelfde moment is ingedrukt aan variabele A$ toegekend. Is er op dat moment geen toets ingedrukt dan zal variabele A$ leeg blijven. Met regel 20 wordt gevraagd of variabele A$ leeg is (er staat niets tussen de aanhalingstekens), en zo ja, dan wordt onmiddellijk teruggegaan naar regel 10. Daar wordt weer naar het toetsenbord gekeken. Dit blijft zo doorgaan

totdat er een toets is ingedrukt. In dat geval is de vergelijking in regel 20 niet meer waar, en wordt met regel 30 verder gegaan. Daar wordt de inhoud van variabele A$ (de ingedrukte toets) op het beeldscherm afgedrukt. Daarna wordt met regel 40 teruggesprongen naar regel 10. Het hele proces begint van voren af aan. Door een toets langere tijd ingedrukt te houden, kunt u zien hoe snel het programma wordt uitgevoerd. De met regel 30 af te drukken tekens (1 teken per uitvoering van het programma) volgen elkaar zeer snel op.

De INKEY$-functie is, zoals u ziet, redelijk eenvoudig. De toepassingsmogelijkheden ervan zijn echter zeer talrijk. In de verdere programma's in dit boek zult u nog meer manieren zien waarop deze functie kan worden gebruikt.

6 Afdrukken van gegevens

6.1 Beeldschermmodes

MSX-BASIC kent voor het beeldscherm de volgende modes:

a. Tekstmode.
b. Grafische mode.

Tekstmode

In de tekstmode kan het beeldscherm alleen worden gebruikt voor het afdrukken van alfanumerieke gegevens (letters, cijfers en leestekens). Grafisch gebruik is in deze mode niet of nauwelijks mogelijk. Alleen het grafische statement PUT SPRITE kan worden gebruikt.

In de tekstmode kan zowel in de directe- (commando) als de indirecte (programma) mode worden gewerkt. Binnen de tekstmode kent het beeldscherm twee instellingen. Deze instellingen worden uitgevoerd door het SCREEN-statement en staan in de volgende tabel vermeld.

statement	aantal regels	aantal tekens per regel (max.)
SCREEN 0	24	40
SCREEN 1	24	32

Zowel voor SCREEN 0 als voor SCREEN 1 wordt het beeldscherm over de volle breedte gebruikt. De tussenruimtes tus-

sen de tekens onderling bij SCREEN 0 zijn dus kleiner dan bij SCREEN 1. Wanneer MSX-BASIC wordt opgestart, zal in de tekstmode voor SCREEN 0 de lengte van de regel worden ingesteld op 37 tekens (default-waarde) en voor SCREEN 1 op 29 tekens. Dit betekent dat de maximale waarde (40 voor SCREEN 0 en 32 voor SCREEN 1) niet automatisch wordt ingesteld.

Het aantal tekens per regel kan worden gewijzigd met het statement **WIDTH** (zie ook paragraaf 6.3). Indien voor een bepaalde SCREEN-mode de regellengte opnieuw wordt ingesteld met het statement WIDTH, dan wordt dit door de computer onthouden, ook al gaat men naar een andere SCREEN-mode toe en komt men later terug in de oorspronkelijke mode. Bij het aanschakelen van de computer wordt altijd SCREEN 0 ingesteld (default).

Wanneer de regel op maximale lengte wordt ingesteld, kan het zijn dat door de afregeling van het TV-toestel aan de linker en/of de rechter kant van het scherm tekens geheel of gedeeltelijk verloren gaan.

Voor het afdrukken van gegevens op het beeldscherm in de tekstmode zijn de volgende statements erg belangrijk:

PRINT - afdrukken van gegevens.
LOCATE - bepalen van de cursorpositie.
CLS - schoonmaken van het scherm.
COLOR - bepalen kleur van scherm en tekens.

Grafische mode

In de grafische mode kan het scherm worden gebruikt voor het uitbeelden van allerlei grafische voorstellingen (tekeningen, etc.) en het afdrukken van alfanumerieke gegevens. In de grafische mode kan alleen in de indirecte (programma) mode worden gewerkt. Binnen de grafische mode zijn voor het beeldscherm twee instellingen mogelijk, die in de volgende tabel staan vermeld.

statement	beeldscherm hor.	vert.	oplossend vermogen
SCREEN 2	256 pixels	192 pixels	hoog
SCREEN 3	64 blokjes	48 blokjes	laag

Een pixel is een beeldpunt. Dit is het kleinste lichtvlekje dat de computer kan maken. De MSX-computer kan maximaal 256 * 192 van die beeldpuntjes op het scherm schrijven. Een aantal van die puntjes achter elkaar resulteren in een lijn. Dit hoeft geen rechte lijn te zijn, het kan iedere gewenste vorm aannemen. Ook de tekens (in tekstmode en in SCREEN 2 mode) worden op die manier gemaakt. In SCREEN 3 mode worden altijd blokjes, die uit 4 * 4 beeldpuntjes bestaan, geschreven.

Voor SCREEN 2 is het gebruik van kleuren aan beperkingen onderhevig. Dit geldt niet voor SCREEN 3, daarbij kunnen alle 15 kleuren worden gebruikt.

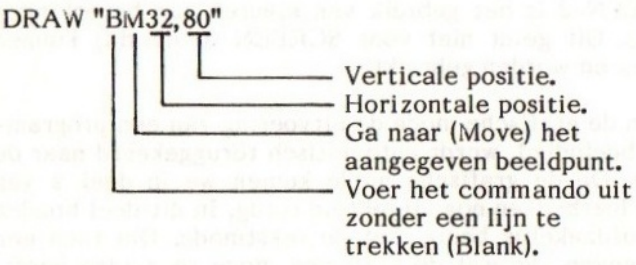
Wanneer in de grafische mode de uitvoering van een programma wordt beeindigd, wordt automatisch teruggekeerd naar de tekstmode. Op de grafische mode komen we in deel 2 van deze serie leerboeken nog uitgebreid terug. In dit deel houden we ons hoofdzakelijk bezig met de tekstmode. Om toch een indruk te geven van wat de grafische mode te bieden heeft, zullen we voor de SCREEN 3 mode een voorbeeld geven.

```
100 SCREEN 3
110 OPEN "GRP:" AS #1
120 DRAW "BM32,0"
130 PRINT #1,"BEELD-"
140 DRAW "BM32,80"
150 PRINT #1,"SCHERM"
160 DRAW "BM32,160"
170 PRINT #1,"MODE 3"
180 GOTO 180
```

De statements OPEN en DRAW zullen we niet diepgaand behandelen. Later in dit boek zullen we meer over het OPEN-statement vertellen en in het volgende deel uit deze serie boeken zullen we het DRAW-statement uitgebreid behandelen. Hier volstaan we met een korte toelichting.

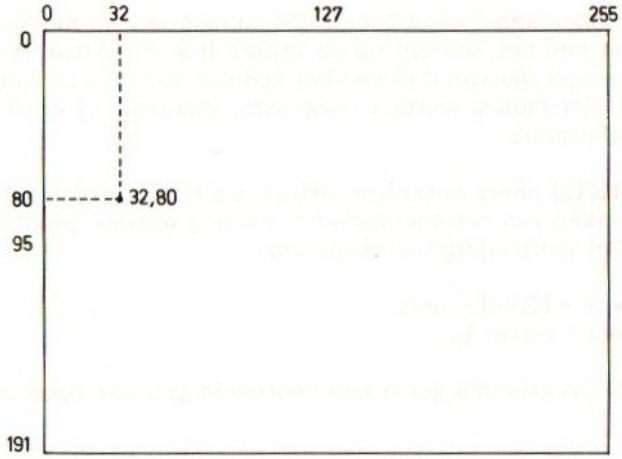
Een grafisch beeldscherm (in ons voorbeeld SCREEN 3) wordt in MSX-BASIC ook als randapparaat beschouwd, daarom dient het eerst te worden geopend, alvorens er gegevens naar toe kunnen worden gestuurd. Dit openen wordt met het OPEN-statement op regel 110 gedaan. Daarin stelt "GRP" de naam van het grafische beeldscherm voor. #1 is een referentie, die in de andere statements moet worden gebruikt, om naar het grafische scherm (GRP) te kunnen schrijven.

Het statement DRAW "BM32,80" op regelnummer 140 dient om aan te geven waar we willen beginnen met het afdrukken van de (grote) letters, van het woord dat in het PRINT-statement staat.



Het positioneren kan in SCREEN 3 op een puntje nauwkeurig worden gedaan (zie afbeelding 6-1). De kleinste eenheid die in de SCREEN 3 mode kan worden gebruikt voor het opbouwen van figuren en tekens is echter 4 bij 4 beeldpuntjes groot. In de SCREEN 2 mode is dat slechts 1 beeldpuntje. Het positioneren kan dus in beide modes op een puntje nauwkeurig geschieden. De kleinste eenheden waarmee kan worden gewerkt, verschilt dus tussen beide modes.

Met het PRINT-statement op regel 150 brengen we in grote letters (opgebouwd uit blokjes van 4 bij 4 beeldpuntjes) het



Afb. 6-1 Coordinaten van beeldpuntjes in grafische modes.

woord "SCHERM" op het beeldscherm. Zoals u ziet wordt in dit PRINT-statement gerefereerd naar het grafische scherm (met #1).

In het AGENDA-programma (in appendix A) is van deze mogelijkheid, het afdrukken van grote letters, ook gebruik gemaakt. U vindt dit op de regels 1 tot en met 13 en op de regels 5500 tot en met 5580. Hiermee wordt het programma, met weinig moeite, een leuk aanzien gegeven.

Opmerking:

Met het statement SCREEN kunnen we ook de toetsaanslag hoorbaar of onhoorbaar maken. Het formaat van het statement ziet er dan als volgt uit:

SCREEN ,,0 ⟶ onhoorbaar
SCREEN ,,1 ⟶ hoorbaar

6.2 Schoonmaken van het scherm

Met het statement **CLS** (Clear Screen) kunnen we het beeld-

scherm schoonmaken (leegmaken). De cursor wordt na het schoonmaken van het scherm op de meest linker positie van de bovenste regel (positie 0,0) van het scherm gezet. CLS kan in alle SCREEN-modes worden toegepast, dus ook bij grafische beeldschermen.

PRINT CHR$(12) heeft hetzelfde effect als CLS. Verder kan het schoonmaken van het beeldscherm ook nog worden geactiveerd door het gelijktijdig indrukken van:

a. SHIFT-toets + HOME-toets.
b. CTRL-toets + letter L.

Het volgende programma geeft een voorbeeld van het gebruik van CLS.

```
100 CLS
110 INPUT "Wat is uw naam";N$
120 PRINT "Uw naam is ";N$
130 A$=INKEY$:IF A$="" GOTO 130
140 GOTO 100
```

6.3 Instellen van de regellengte

Met het statement **WIDTH nn** kan het aantal tekens per regel worden ingesteld. Afhankelijk van de instelling van het beeldscherm (SCREEN 0 of 1), kan de lengte maximaal 40 of 32 tekens zijn. In beide gevallen is de minimale instelling 1 teken. De gewenste instelling kan in het statement worden ingevuld op de plaats van de letters "nn". Hier hoeft niet speciaal een getal te worden opgegeven, het mag ook een numerieke variabele zijn, of een uitdrukking. De ingestelde regellengte wordt altijd gecentreerd ten opzichte van het midden van het beeldscherm. Bij het uitvoeren van het statement WIDTH wordt het beeldscherm schoongemaakt, tenzij de nieuwe instelling dezelfde is als waarop het beeldscherm al was ingesteld.

Voor SCREEN 0 lopen de toegestane waarden in het WIDTH-statement van 1 tot en met 40. Voor SCREEN 1 is dat van 1

tot en met 32. Is de gespecificeerde waarde kleiner of groter dan de toegestane waarden, dan geeft de computer de boodschap "Illegal function call".

WIDTH kan zowel in de directe (commando) als in de indirecte (als programma statement) mode worden gebruikt. De volgende programmavoorbeelden laten zien hoe het WIDTH-statement kan worden gebruikt. (Schakel de computer eerst uit en dan weer aan, voordat u de programma's intikt.)

```
10 SCREEN 1
20 PRINT "De regellengte is nu 29"
30 PRINT "Druk willekeurige toets in"
40 A$=INKEY$:IF A$="" GOTO 40
50 WIDTH 18
60 PRINT "De lengte is nu 18"
70 END
```

```
10 SCREEN 0
20 X=16
30 WIDTH X
40 PRINT "De ingestelde regellengte is ";X
50 END
```

6.4 Werken met tabulatorstoppen

De **functie TAB(X)** kan worden vergeleken met de TAB-toets op een schrijfmachine, waarmee tabulatorstoppen kunnen worden ingesteld. De functie TAB wordt altijd gebruikt in combinatie met het PRINT of LPRINT-statement. Met behulp van de functie TAB kan men de computer vertellen op welke positie van de regel men wil beginnen met het afdrukken van de informatie. Hierbij is positie 0 de meest linker positie van de regel. De meest rechter positie van de regel is de maximum regellengte min 1 (WIDTH -1). Het argument X mag een numerieke constante (bijvoorbeeld: 16), een numerieke variabele (bijvoorbeeld: T), of een uitdrukking (bijvoorbeeld: Y+4) zijn. Voorbeeld:

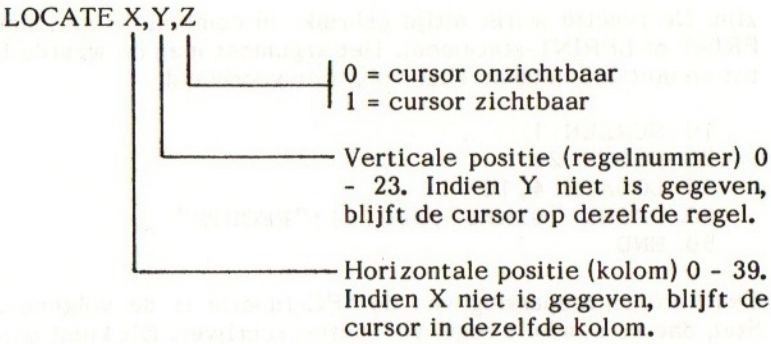
```
10 X=10
20 WIDTH X
30 PRINT "0123456789"
40 PRINT TAB(0);"L";
50 PRINT TAB(X-1);"R"
60 END
```

Wanneer de waarde tussen haakjes niet een geheel getal is, bijvoorbeeld TAB(32.75), dan wordt de functie uitgevoerd als TAB(32). Het argument mag een waarde hebben van 0 tot en met 255.
Hier volgt nog een voorbeeld:

```
10 CLS
20 PRINT TAB(4);
30 INPUT "l,b";L,B
40 PRINT
50 PRINT TAB(4);"lengte";TAB(12);"breedte";
TAB(21);"oppervlakte"
60 PRINT
70 PRINT TAB(5);L;TAB(13);B;TAB(23);L*B
80 A$=INKEY$:IF A$="" GOTO 80
90 GOTO 10
```

6.5 Hoe positioneer ik de cursor?

Met het statement **LOCATE X,Y,Z** kunnen we de cursor op een bepaalde positie op het beeldscherm plaatsen. Hierbij is 0,0 de linker bovenhoek van het scherm. LOCATE werkt alleen in de tekstmode (SCREEN 0 of 1). Meestal staat LOCATE voor een PRINT-statement. Het statement heeft het volgende formaat:



De default waarde voor de cursor is Z=1 (zichtbaar). Om de cursor alleen maar zichtbaar of onzichtbaar te maken, kan het LOCATE-statement als volgt worden gebruikt:

```
LOCATE ,,0 ——→ onzichtbaar
LOCATE ,,1 ——→ zichtbaar
```

Dan volgt nu nog een programma met een aantal voorbeelden:

```
100 KEY OFF
110 WIDTH 40
120 LOCATE 0,0
130 PRINT "Linksboven";
140 LOCATE 0,23
150 PRINT "Linksonder";
160 LOCATE 27,0
170 PRINT "Rechtsboven";
180 LOCATE 27,23
190 PRINT "Rechtsonder";
200 LOCATE 19,11,1
210 GOTO 210
```

6.6 Werken met spaties

Met de **functie SPC(X)** kunnen we, tussen twee af te drukken teksten, een aantal spaties plaatsen. Het aantal spaties wordt bepaald door het argument X. Het argument kan een numerieke constante, een numerieke variabele of een uitdrukking

zijn. De functie wordt altijd gebruikt in combinatie met het PRINT of LPRINT-statement. Het argument mag de waarde 0 tot en met 255 hebben. Hier volgt een voorbeeld:

```
10 SCREEN 1
20 WIDTH 24
30 LOCATE 4,10
40 PRINT "LINKS";SPC(8);"RECHTS"
50 END
```

Een aardige toepassing van de SPC-functie is de volgende. Stel, dat we een hele regel vol spaties schrijven. Dit komt dan in feite overeen met het wissen van een regel. Om dit duidelijk te maken volgt hier een voorbeeldprogramma, waarmee we eerst een aantal regels op het scherm schrijven, om er daarna een van te wissen.

```
10 SCREEN 0
20 WIDTH 40
30 PRINT "abcdefghijklmnopqrstuvwxyz"
40 PRINT "bcdefghijklmnopqrstuvwxyza"
50 PRINT "cdefghijklmnopqrstuvwxyzab"
60 PRINT "defghijklmnopqrstuvwxyzabc"
70 PRINT
80 INPUT "te wissen regel 0,1,2 of 3";R
90 LOCATE 0,R
100 PRINT SPC(40);
110 END
```

Met de regels 10 en 20 stellen we het beeldscherm in. Daarna schrijven we, met de regels 30 tot en met 60, op de eerste vier regels van het scherm een tekst. Nu vragen we met regel 70 welke regel moet worden gewist. Stel dat u het nummer 2 ingeeft. Dan zal de variabele R de waarde 2 bevatten. Met regel 80 wordt de cursor dan aan het begin van regel 2 gezet. Met regel 90 worden 40 spaties afgedrukt, over de bestaande tekst heen. Het resultaat zal zijn dat de tweede beeldschermregel leeg wordt.

Dit principe is ook in het AGENDA-programma (in appendix A) toegepast. In de subroutine op de regels 2400 tot en met

2470 ziet u op regel 2430 het PRINT-statement met de SPC-functie. Bij het aanroepen van deze subroutine moeten twee variabelen zijn gevuld. Variabele ER moet het regelnummer van de eerste te wissen regel bevatten, LR het regelnummer van de laatste te wissen regel. Doordat deze subroutine een lus bevat, zal het PRINT SPC statement net zo vaak worden uitgevoerd als nodig is om alle regels, van ER tot en met LR, te wissen.

Met de **functie SPACE$(X)** kunnen we een aantal spaties verkrijgen. Het aantal spaties wordt bepaald door het argument X. Argument X mag een numerieke constante, een numerieke variabele of een uitdrukking zijn. De waarde van X mag varieren van 0 tot en met 255. Wanneer de waarde van X een gebroken getal is, wordt het deel achter de komma (decimale punt) verwaarloosd en wordt met het gehele getal verder gewerkt. Hier volgen enkele voorbeelden:

```
10 CLS
20 A$=SPACE$(3)
30 PRINT "1";A$;"5";A$;"9"
40 END
```

```
10 CLS
20 I=0
30 PRINT SPACE$(I);I;"spaties"
40 I=I+1
50 A$=INKEY$:IF A$="" GOTO 50
60 GOTO 30
```

6.7 Reeksen met dezelfde tekens

Wanneer we een reeks (string) willen samenstellen, die is opgebouwd uit dezelfde tekens (bijvoorbeeld allemaal A's) kunnen we gebruik maken van de **functie STRING$**. Het formaat van deze functie is:

STRING$ (aantal tekens , te herhalen teken)

Het **aantal tekens** kan worden aangegeven als een numerieke constante, een numerieke variabele of een uitdrukking. De waarde mag varieren van 0 tot en met 255.

Het **te herhalen teken** kan worden gegeven in de ASCII-code (decimale waarde van 32 tot en met 255) van het teken. De ASCII-code voor de letter A is bijvoorbeeld 65. Het te herhalen teken mag echter ook als alfanumerieke constante (bijvoorbeeld "A"), of als alfanumerieke variabele (bijvoorbeeld C$) worden gegeven. Indien in de variabele meerdere tekens staan, zal alleen het eerste teken worden gebruikt. Hier volgen enkele voorbeelden:

```
100 CLS
110 A$="MSX-BASIC"
120 X=17
130 PRINT STRING$(19,"*")
140 PRINT "*";SPC(17);"*"
150 PRINT "*";SPC(4);A$;SPC(4);"*"
160 PRINT "*";SPACE$(X);"*"
170 PRINT STRING$(19,42)
180 END
```

6.8 Werken met codes

Er kunnen voor MSX-BASIC twee codetabellen worden onderscheiden:

a. ASCII-codetabel
b. MSX-codetabel

De ASCII-codetabel (zie tabel 6-1) kan waarden vertegenwoordigen van 0 tot en met 127 (hexadecimaal &H00 t/m &H7F). De MSX-codetabel (zie tabel 6-2) kan waarden vertegenwoordigen van 0 tot en met 255 (hexadecimaal &H00 t/m &HFF).

ASCII-code

In het eerste hoofdstuk van dit boek hebben we al gezien dat een letter, cijfer of leesteken als een serie nulletjes en eentjes in het geheugen van een computer wordt opgeslagen. Stel nu dat we afspreken, met de computer, dat de serie "01000001" de letter A voorstelt. Dit is hexadecimaal "41". We zouden dan kunnen zeggen dat de B moet worden opgeslagen als "42", enz. Stel nu vervolgens, dat we gegevens vanuit onze computer naar een andere computer willen sturen. Als twee computers met elkaar "praten", dan zullen ze dat met behulp van nulletjes en eentjes doen. Willen ze elkaar echter kunnen "verstaan", dan zullen ze wel allebei dezelfde code voor dezelfde letter moeten hebben.

binair	hex	0000 0	0001 1	0010 2	0011 3	0100 4	0101 5	0110 6	0111 7
0000	0	NUL	DLE	SP	0	@	P	`	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	'	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	,	<	L	/	l	|
1101	D	CR	GS	-	=	M	]	m	}
1110	E	SO	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O	_	o	DEL

Tabel 6-1 De ASCII tekenset.

Dit probleem deed zich al lang geleden voor. In die tijd werden computers hoofdzakelijk in Amerika gebruikt. Het waren dan ook de Amerikanen die dit probleem het eerst moesten oplossen. Verschillende fabrikanten zijn daar bij elkaar gaan zitten en hebben onderling afgesproken, welke codes ze in hun computer zouden gebruiken voor de letters, cijfers en leestekens. Deze code, die momenteel over de hele wereld wordt

gebruikt werd de ASCII-code genoemd. ASCII betekent American Standard Code for Information Interchange. Ook de MSX-computer houdt zich aan deze code. Wij dienen deze code te kennen, omdat we die in een aantal functies moeten toepassen. We zullen dat nu gaan zien.

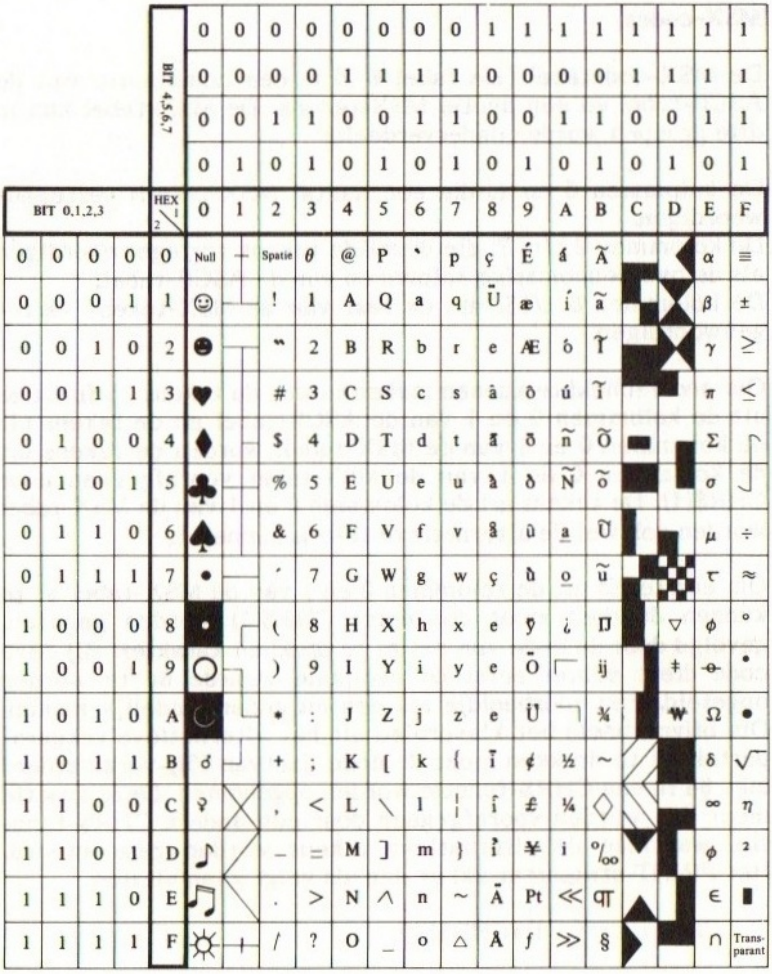
Met de **functie CHR$(X)** kan het teken of de functie worden gegenereerd, die correspondeert met de ASCII-code. De code wordt gegeven met het argument X. Dit argument kan een numerieke constante, een numerieke variabele of een uitdrukking zijn. Met PRINT CHR$(65) of PRINT CHR$(&H41) wordt de letter A afgedrukt. Met PRINT CHR$(12) of PRINT CHR$(&H0C) wordt het beeldscherm schoongemaakt. De functie CHR$ wordt meestal in combinatie met het PRINT-statement gebruikt.

In principe kunnen voor de ASCII-code alleen de decimale waarden 0 tot en met 127 worden gebruikt. De ASCII-codes voor de letters A tot en met Z zijn respectievelijk 65 tot en met 90. Met het volgende programma worden op het beeldscherm de letters A tot en met Z afgedrukt.

```
100 CLS
110 I=65
120 PRINT CHR$(I);SPC(1);
130 I=I+1
140 IF I<91 GOTO 120
150 END
```

Hoewel het statement op regel 140 nog niet is behandeld, zal het ons inziens geen moeilijkheden opleveren. Het statement controleert steeds de waarde van I, waarbij, zolang de waarde kleiner is dan 91, naar regelnummer 120 wordt teruggesprongen. In alle andere gevallen (gelijk aan of groter dan) wordt met het END-statement op de volgende regel doorgegaan.

Het is ook mogelijk om codes te combineren, door deze op te tellen. Zo zou men bijvoorbeeld achtereenvolgens het scherm kunnen schoonmaken (code 12), de letter A kunnen afdrukken (code 65), een regel kunnen opvoeren (code 10) en de letter Z kunnen afdrukken (code 90). Het statement gaat er dan als



BIT 0,1,2,3				HEX	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	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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	ã	ð	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	ã	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	Õ	⌐	ij		‡	Θ	•
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	ƒ	≫	§			∩	Trans-parant

Tabel 6-2 De MSX tekenset.

volgt uitzien:

```
PRINT CHR$(12)+CHR$(65)+CHR$(10)+CHR$(90)
```

MSX-codes

De MSX-codetabel (zie tabel 6-2) is een combinatie van de ASCII-tabel en een aantal MSX-tekens. De MSX-tabel kan in drie groepen worden onderverdeeld:

De kolommen 0 en 1, die een aantal MSX-tekens vertegenwoordigen.
De kolommen 2 t/m 7, die dezelfde tekens vertegenwoordigen als de overeenkomstige kolommen van de ASCII-tabel.
De kolommen 8 t/15, die de rest van de MSX-tekens vertegenwoordigen.

Om onderscheid te kunnen maken tussen de tekens en functies uit de kolommen 0 en 1 van de ASCII-tabel en de tekens uit de kolommen 0 en 1 van de MSX-tabel, worden de tekens uit de kolommen 0 en 1 van de MSX-tabel voorafgegaan door CHR$(1). De tekens uit de kolommen 0 en 1 van de MSX-tabel worden ook wel de alternatieve tekens genoemd.

Om een teken uit de kolommen 0 en 1 van de MSX-tabel af te kunnen drukken moet dus eerst CHR$(1) worden gegeven, gevolgd door de code van het af te drukken karakter. Bij deze code dient echter eerst de decimale waarde 64 te worden opgeteld. Een voorbeeldje zal een en ander duidelijk maken. Om bijvoorbeeld het klaverblad uit het alternatieve tekenrepertoir af te drukken moet de code daarvan (5), vermeerderd met 64 in een CHR$-functie worden opgenomen. Deze functie moet dan worden voorafgegaan door een andere CHR$-functie, waarmee de alternatieve tekens worden geselecteerd. Het PRINT-statement zal er dan als volgt gaan uitzien:

```
PRINT CHR$(1)+CHR$(69)
```

Het volgende programma drukt de MSX-tekens uit de kolommen 0 en 1 op het beeldscherm af:

```
100 CLS
110 I=0+64
120 PRINT CHR$(1)+CHR$(I);SPC(1);
130 I=I+1
140 IF I<96 GOTO 120
150 END
```

Voor het afdrukken van de tekens uit de kolommen 2 tot en met 7 behoeft de code niet te worden voorafgegaan door CHR$(1). Hetzelfde geldt voor het afdrukken van de tekens uit de kolommen 8 tot en met 15. Met het volgende programma worden alle tekens uit de kolommen 2 tot en met 15 afgedrukt. De codes van deze tekens hebben de decimale waarden 32 tot en met 255 (hexadecimaal &H20 t/m &HFF).

```
100 CLS
110 I=&H20
120 PRINT CHR$(I);SPC(1);
130 I=I+1
140 IF I<&H100 GOTO 120
150 END
```

Opmerking:

De waarden in de regels 110 en 140 zijn hexadecimale waarden. Deze worden voorafgegaan door de tekens &H.

6.9 PRINT USING

PRINT USING is het krachtigste PRINT-statement dat in MSX-BASIC bestaat. Met dit statement kunnen getallen en teksten in een bepaalde vorm en op een bepaalde positie (met behulp van een PRINT-masker) worden afgedrukt.

6.9.1 Hoe kunnen getallen worden afgedrukt.

In een PRINT USING-statement wordt met behulp van #-tekens aangegeven hoeveel cijfers er van een numerieke constante of variabele moeten worden afgedrukt. Elk #-teken vertegenwoordigt een bepaalde positie (eenheden, tientallen,

enz.) van de numerieke constante of variabele. Hieraan gekoppeld zit een vaste PRINT-positie. Dit houdt in, dat getallen, die onder elkaar worden afgedrukt, netjes rechts gericht onder elkaar komen te staan. Met het volgende programma worden een aantal getallen onder elkaar afgedrukt.

```
100 CLS
110 I=2
120 PRINT USING "###";I
130 I=I+25
140 IF I<150 GOTO 120
150 PRINT
160 END
```

Vergelijk het resultaat van dit programma eens met het resultaat, indien regelnummer 120 wordt veranderd in PRINT I.

Wanneer het af te drukken getal uit minder cijfers bestaat dan het aantal gegeven #-tekens (posities) in het PRINT USING-statement, zal MSX-BASIC de lege posities links van het getal vullen met spaties.

Indien het getal uit meer cijfers bestaat dan het aantal in het PRINT USING statement aangegeven posities, zal als waarschuwing voor het getal het percentage-teken (%) worden afgedrukt. Dit is een aanwijzing, dat het PRINT-masker eventueel moet worden veranderd. Om dit te demonstreren moet regelnummer 120 van het vorige programma als volgt worden veranderd:

```
120 PRINT USING "##";I
```

Voor het werken met gebroken getallen (bijv. 21.73) wordt met een PRINT-masker gewerkt, waarin een decimale punt is opgenomen. Het aantal #-tekens links en rechts van de decimale punt geven dan het gewenste aantal cijfers voor en achter de decimale punt aan. Er wordt dus weer, net als bij gehele getallen, voor elk cijfer een positie gereserveerd. Het volgende voorbeeld laat dit zien:

```
PRINT USING "##.##";21.126,17.3
 21.13  17.30
```

Daar er slechts twee cijfers achter de decimale punt zijn toegestaan, wordt het eerste getal afgerond op 21.13. Verder wordt bij het tweede getal de laatste positie aangevuld met een nul. Als algemene regel geldt, dat alle lege posities rechts van de decimale punt met nullen worden aangevuld.

Wanneer we een aantal gebroken en gehele getallen door middel van een PRINT USING statement onder elkaar afdrukken, komen de decimale punten onder elkaar te staan. Hetzelfde geldt voor de eenheden, tientallen, enz. Het volgende programma laat dit nog eens duidelijk zien.

```
100 CLS
110 A=2.125
120 PRINT USING "##.##";A
130 A=A*2
140 IF A<35 GOTO 120
150 PRINT
160 END
```

Bekijk, na het uitvoeren van dit programma, ook eens het resultaat indien regelnummer 120 wordt gewijzigd in PRINT A.

Wanneer we in twee kolommen willen afdrukken, waarbij de ene kolom 3 posities breed moet zijn, en de andere 5, dan zal, indien de ruimte tussen de kolommen 3 spaties moet bedragen, het PRINT-masker er als volgt uit moeten zien:

```
"###    #####"
```

Het volgende programma is een voorbeeld van het afdrukken in twee kolommen.

```
100 CLS
110 I=8
120 PRINT SPC(2);"I";SPC(5);"I^2"
130 PRINT
```

```
140 PRINT USING "###    #####";I,I^2
150 I=I*2
160 IF I<130 GOTO 140
170 PRINT
180 END
```

Wanneer met negatieve getallen wordt gewerkt, dient een extra positie te worden gereserveerd in verband met het minteken. Zou bijvoorbeeld met positieve- en negatieve getallen, die maximaal uit twee cijfers bestaan, worden gewerkt, dan zal het PRINT-masker uit 3 posities moeten bestaan, namelijk 2 voor het getal en 1 voor het minteken. Het volgende programma laat dit zien.

```
100 CLS
110 I=26
120 PRINT USING "###";I
130 I=I-6
140 IF I>-14 GOTO 120
150 PRINT
160 END
```

Het is ook mogelijk om bij positieve getallen een plusteken af te drukken en bij negatieve getallen een minteken. Het plus- en minteken kunnen hierbij of voor of achter het getal worden afgedrukt. Dit moet worden aangegeven door respectievelijk voor of achter het PRINT-masker een plusteken te plaatsen. Het volgende programma laat het effect hiervan zien.

```
100 CLS
110 A=12.3
120 PRINT USING "+##.##";A
130 A=A-6.15
140 IF A>-20 GOTO 120
150 PRINT
160 A=12.3
170 PRINT USING "##.##+";A
180 A=A-6.15
190 IF A>-20 GOTO 170
200 PRINT
210 END
```

We kunnen ook een minteken aan het PRINT-masker toevoegen in plaats van een plusteken. Het minteken mag alleen achter het PRINT-masker worden geplaatst. Het effect hiervan is, dat achter een positief getal een spatie wordt afgedrukt en achter een negatief getal een minteken. Ook hiervan weer een voorbeeldje in de vorm van een programma.

```
100 CLS
110 A=-24.15
120 PRINT USING "##.##-";A
130 A=A+8.15
140 IF A<20 GOTO 120
150 PRINT
160 END
```

Om grote getallen beter leesbaar te maken biedt MSX-BASIC de mogelijkheid om vanaf de eenheden, om de drie cijfers, een komma te plaatsen. Dit wordt bereikt door in het PRINT-masker rechts van de positie voor de eenheden een komma te plaatsen. Voor elke komma moet in het masker een positie worden gereserveerd. Het volgende programma toont het gebruik van komma's in getallen aan.

```
100 CLS
110 INPUT "Geheel getal  ";G
120 INPUT "Gebroken getal";GE
130 PRINT
140 PRINT USING "#########,";G
150 PRINT USING "#########,.##";GE
160 A$=INKEY$:IF A$="" GOTO 160
170 GOTO 100
```

Wanneer men voor het PRINT-masker twee sterren (**) plaatst, zullen lege posities links van het getal worden opgevuld met sterren. De twee sterren in het masker tellen ook mee als posities voor cijfers.

```
100 CLS
110 INPUT "Getal";G
120 PRINT USING "**###.##";G
```

```
130 PRINT
140 GOTO 110
```

Opmerking:
Het min- en plusteken is ook toegestaan wanneer gebruik wordt gemaakt van de sterren.

Wanneer men voor het PRINT-masker twee dollartekens ($$) plaatst, zal voor elk af te drukken getal een dollarteken worden geplaatst. Het tweede dollarteken in het PRINT-masker wordt gebruikt als extra cijferpositie. Het min- en plusteken is ook bij het gebruik van dollartekens toegestaan. Met het volgende programma kunnen de verschillende mogelijkheden worden uitgetest. Voer met het INPUT-statement op regelnummer 120 zowel positieve als negatieve getallen in.

```
100 WIDTH 40
110 CLS
120 INPUT "Bedrag in dollars";D
130 PRINT TAB(29);
140 PRINT USING "$$####.##";D
150 GOTO 120
```

Bij het uitvoeren van het programma is te zien, dat de decimale punten weer netjes onder elkaar worden geplaatst. Wanneer achter het PRINT-masker een plusteken wordt geplaatst, wordt bij positieve waarden een plusteken en bij negatieve waarden een minteken achter het bedrag gezet. Dit kan worden getest door regelnummer 140 van het bovenstaande programma als volgt te veranderen:

```
140 PRINT USING "$$####.##+";D
```

Wanneer achter het PRINT-masker een minteken wordt gezet, wordt voor positieve bedragen geen teken afgedrukt en voor negatieve bedragen een minteken. Het plusteken kan ook voor het PRINT-masker worden gezet, met als resultaat dat plus- en mintekens voor de bedragen worden afgedrukt.

We hebben tot nu toe kunnen zien wat er zoal mogelijk is wanneer men sterren (**) of dollartekens ($$) voor het

PRINT-masker plaatst. Deze voorzieningen zijn zeer geschikt voor administratieve doeleinden. Het is ook mogelijk om twee sterren en een dollarteken voor het PRINT-masker te plaatsen. De lege posities links van het dollarteken worden dan opgevuld met sterren. Deze mogelijk kan worden getoond door regelnummer 140 van het vorige programma als volgt te wijzigen:

```
140 PRINT USING "**$####.##";D
```

Voer bij het uittesten van het programma een aantal positieve- en negatieve getallen in. Ook in dit formaat is het toegestaan een plus- of minteken achter het PRINT-masker te plaatsen. Met het volgende programma kunnen dollarbedragen bij elkaar worden opgeteld.

```
100 WIDTH 40
110 CLS
120 I=4:E=0
130 PRINT "Druk voor het invoeren van het vo
lgende"
140 PRINT "bedrag eerst de letter V in, beha
lve"
150 PRINT "voor het eerste bedrag."
160 LOCATE 0,4
170 INPUT D
180 E=E+D
190 LOCATE 29,I
200 PRINT USING "$$######.##";D
210 I=I+1
220 A$=INKEY$:IF A$="" GOTO 220
230 IF A$="V" GOTO 170
240 LOCATE 29,I
250 PRINT STRING$(10,"-")
260 I=I+1
270 LOCATE 29,I
280 PRINT USING "$$######.##";E
290 END
```

Om van een getal de exponentiele vorm af te drukken, dient het PRINT USING statement te worden voorzien van het

PRINT-masker:

```
"#.####^^^^"
```

Met het volgende programma kunnen we het afdrukken van exponentiele vormen testen.

```
100 WIDTH 40
110 CLS
120 INPUT "Getal";G
130 PRINT "Exponentiele vorm: ";
140 PRINT USING "#.####^^^^";G
150 PRINT
160 END
```

Voor het plaatsen van plus- en mintekens voor of achter het PRINT-masker gelden dezelfde regels als voor de reeds behandelde PRINT USING statements. Ter afsluiting van het gebruik van PRINT USING voor getallen nog het volgende programma, waarmee de kapitaalaanwas over een bepaald aantal jaren wordt berekend.

```
100 WIDTH 40
110 CLS
120 INPUT "Beginkapitaal";BK
130 INPUT "Rente in %   ";RP
140 INPUT "Beginjaar    ";EJ
150 INPUT "Laatste jaar ";LJ
160 PRINT
170 PRINT TAB(1);"JAAR";SPC(3);"KAPITAAL";
SPC(2);"RENTE";SPC(3);"NIEUW KAPITAAL"
180 PRINT
190 PRINT EJ;TAB(8);
200 PRINT USING "####.##";BK;
210 PRINT TAB(17);
220 R=(BK*RP)/100
230 PRINT USING "###.##";R;
240 PRINT TAB(29);
250 NK=BK+R
260 PRINT USING "####.##";NK
270 BK=NK
```

```
280 EJ=EJ+1
290 IF EJ<=LJ GOTO 190
300 PRINT
310 END
```

6.9.2 Manipuleren met strings

Met het statement PRINT USING kan men op verschillende manieren strings bewerken. Zo kan bijvoorbeeld van een string alleen het eerste teken worden afgedrukt. Dit wordt in het PRINT USING statement aangegeven met het speciale teken "!". Het statement heeft dan het volgende formaat:

```
PRINT USING "!";"UITROEPTEKEN"
```

Van de string die achter de puntkomma staat, wordt nu alleen de eerste letter afgedrukt. Het volgende programma geeft een voorbeeld van het gebruik van het uitroepteken in een PRINT USING statement.

```
100 WIDTH 40
110 CLS
120 INPUT "Woord";W$
130 PRINT USING "!";W$;
140 PRINT " is de eerste letter van ";W$
150 PRINT
160 GOTO 120
```

Met het speciale teken "&" wordt in een PRINT USING statement aangegeven, dat een string onverkort dient te worden afgedrukt. Het vorige programma kan dan ook als volgt worden geschreven.

```
100 WIDTH 40
110 CLS
120 INPUT "Woord";W$
130 PRINT USING "! is de eerste letter van
 &";W$;W$
140 PRINT
```

```
150 GOTO 120
```

In regelnummer 130 is de eerste alfanumerieke variabele W$ gerelateerd aan ! en de tweede variabele W$ aan &.

Wanneer men slechts een aantal tekens van een string wil laten afdrukken, kan dit in een PRINT USING statement worden aangegeven door de speciale tekens "\ \". Het aantal tekens dat dient te worden afgedrukt wordt aangegeven door het aantal spaties dat tussen de twee \-tekens staat plus twee. Wil men drie tekens van een string afdrukken, dan dient er 1 spatie tussen de twee \-tekens te staan. Het volgende programma kort de namen van de maanden van het jaar af tot drie letters.

```
100 WIDTH 40
110 CLS
120 INPUT "Maand";M$
130 PRINT USING "\ \";M$
140 PRINT
150 GOTO 120
```

Het is ook mogelijk de speciale tekens te combineren in een PRINT USING statement. Hierbij mogen ook niet speciale tekens worden gebruikt. Laten we eens aannemen dat we de verschillende delen van de dag willen afdrukken als XX-Y, waarin XX de eerste twee letters van de dag zijn (bijv. wo voor woensdag) en Y de eerste letter van het deel van de dag (nacht, ochtend, middag of avond). Het masker in het PRINT USING statement gaat er dan als volgt uitzien:

```
"\\-!"
```

De dagen en delen van de dagen moeten volledig (onverkort) worden ingevoerd. Het programma ziet er dan als volgt uit.

```
100 WIDTH 40
110 CLS
120 M$="\\-!"
130 INPUT "Dag ";D$
140 INPUT "Deel";P$
```

```
150 PRINT USING M$;D$;P$
160 PRINT
170 GOTO 130
```

Opmerking:

Het PRINT-masker kan worden vastgelegd in een alfanumerieke variabele. In ons voorbeeld de variabele M$. In deze variabele zijn \\ en ! speciale PRINT USING tekens en - een niet speciaal teken, dat normaal wordt afgedrukt.

7 Sprongen

Normaal gesproken worden de regels van het BASIC-programma netjes in volgorde uitgevoerd, van de eerste tot en met de laatste. We zullen nu echter gaan zien dat het ook heel goed mogelijk is om deze volgorde te onderbreken. Dit kan op verschillende manieren worden gedaan, met verschillende statements. Ieder van die statements heeft zijn eigen voor- en nadelen. Hoe het springen binnen een programma in zijn werk gaat, en welke statements daarbij kunnen worden gebruikt, zullen we in de volgende paragrafen zien.

7.1 Het statement GOTO

Laten we aannemen, dat we een tabel willen afdrukken, waarbij in de ene kolom de temperaturen in graden Celcius staan vermeld en in de andere kolom de bijbehorende temperaturen in graden Fahrenheit. De tabel moet starten bij 0 graden Celcius en oplopen met 1 graad.

Het verband tussen het aantal graden Celcius en het aantal graden Fahrenheit is in de volgende formule vastgelegd:

X graden Celcius = (9/5 * X + 32) graden Fahrenheit

Het programma voor het oplossen van dit probleem zou er als volgt kunnen uitzien:

```
10 CLS
20 PRINT "CELCIUS     FAHRENHEIT"
30 C=0
40 F=9/5*C+32
50 PRINT USING "###";C,
```

```
60 PRINT TAB(13),
70 PRINT USING "###.##";F
80 C=C+1
90 GOTO 40
```

Het GOTO-statement op regel 90 is nieuw. Wanneer we een programma starten met het commando RUN, dan zal de computer normaal de regelnummers in oplopende volgorde uitvoeren. Door nu in het programma gebruik te maken van het statement GOTO, kan van deze vaste uitvoeringsvolgorde worden afgeweken. Tijdens het uitvoeren van het statement GOTO (Ga naar) wordt er naar het in dat statement gespecificeerde regelnummer gesprongen. Vanaf dit gegeven regelnummer wordt het programma dan weer normaal uitgevoerd.

Het in het GOTO-statement opgegeven regelnummer mag lager, gelijk aan of hoger zijn dan het regelnummer van het GOTO-statement zelf. Daar de sprong altijd wordt gemaakt, bij het uitvoeren van het GOTO-statement, wordt dit ook wel een onvoorwaardelijke sprongopdracht genoemd.

Er bestaan ook voorwaardelijke sprongopdrachten. Een voorwaardelijke sprong wordt alleen dan uitgevoerd, wanneer er aan een bepaalde voorwaarde is voldaan. Indien er niet aan de gestelde voorwaarde is voldaan, zal de sprong niet worden uitgevoerd. In dat geval zal gewoon worden verdergegaan met het statement op het eerstvolgende regelnummer.

Laten we er van uitgaan dat we ons programma willen laten uitvoeren voor de waarden van 0 tot en met 12 graden Celcius. Daar de sprong in regelnummer 90 altijd wordt gemaakt, zal het programma niet uit zichzelf stoppen bij 12 graden Celcius. Het zal eindeloos door blijven gaan. We hebben te maken met een eindeloze lus (in het Engels "loop"). Om het programma te laten stoppen, zullen we zelf moeten ingrijpen, door op het juiste moment de toetsen CTRL en STOP tegelijk in te drukken. Na het indrukken van de genoemde toetsen, krijgen we de boodschap op welk regelnummer de lus is onderbroken.

Indien een in het GOTO-statement gegeven regelnummer niet

in het programma voorkomt, zal de computer hiervan melding maken door een foutboodschap af te drukken.

7.2 IF . . . GOTO-statement

We kunnen het stoppen van het programma bij 12 graden Celcius ook overlaten aan het programma zelf. Hiervoor gebruiken we een voorwaardelijke sprongopdracht. Om een bepaalde sprong alleen op een bepaalde voorwaarde te laten uitvoeren, kan gebruik gemaakt worden van het IF . . . GOTO statement. Het formaat van dit statement is:

IF expressie GOTO regelnummer

De **expressie** (de voorwaarde waaraan moet worden voldaan) is een vergelijkende (relationele) uitdrukking, waarin de volgende vergelijkingen mogelijk zijn:

<	kleiner dan
>	groter dan
=	gelijk aan
<= of =<	kleiner dan of gelijk aan
>= of =>	groter dan of gelijk aan
<> of ><	ongelijk aan

Om het programma nu bij 12 graden Celcius te laten stoppen, zullen we het statement op regel 90 als volgt moeten wijzigen:

```
90 IF C<=12 GOTO 40
```

Het resultaat van de vergelijkende uitdrukking (C <= 12) kan **waar** of niet **waar** zijn. Alleen als het resultaat waar is, wordt de opdracht GOTO 40 uitgevoerd. Is het resultaat niet waar, dan gaat de computer verder met het statement op de volgende regel. Dit betekent in ons voorbeeld dus dat zodra C de waarde 13 heeft bereikt, er niet meer wordt gesprongen naar regelnummer 40. Er zal dan worden doorgegaan met de regel die volgt op regelnummer 90. Er volgt echter geen regel meer na regel 90. Dit betekent dat het programma beeindigd is.

Het is echter geen nette afsluiting van het programma. We doen er dan ook goed aan om nog een regel toe te voegen, en wel:

```
100 END
```

We kunnen de begin- en eindwaarde van het aantal graden Celcius ook variabel maken, door de waarden met INPUT-statements in te voeren. Het programma moet hiertoe als volgt worden aangepast:

Voeg de volgende statements toe:

```
11 INPUT "Beginwaarde";C
12 INPUT "Eindwaarde ";C1
13 PRINT
```

Laat regelnummer 30 vervallen.
Verander regelnummer 90 als volgt:

```
90 IF C<=C1 GOTO 40
```

Hernummer het programma met RENUM.
Start het programma met RUN.

7.3 Menu

In deze paragraaf zullen we aandacht besteden aan het gebruik van menu's in programma's. Een menu is een overzichtslijst van de routines die een bepaald programma kan uitvoeren. Uit die lijst kan steeds maar voor 1 routine tegelijkertijd worden gekozen.

Stel dat we temperaturen van de ene schaal naar een andere willen omrekenen, en dat we de verschillende schalen vanuit een menu willen kiezen. Het menu zou er dan als volgt uit kunnen zien:

1. Celcius naar Fahrenheit
2. Fahrenheit naar Celcius
3. Celcius naar Kelvin
4. Fahrenheit naar Kelvin

Voor elk van de vier punten bestaat in het programma een

aparte routine, die op grond van de nummers (1 tot en met 4) kan worden gekozen.

Wanneer een programma uit een aantal routines bestaat en men wil op grond van de waarde van een numerieke variabele (bijvoorbeeld K), of op grond van een numerieke expressie (bijvoorbeeld K-6) het programma naar een bepaalde routine laten springen, dan kan daarvoor het ON . . .GOTO-statement worden gebruikt. Het ON . . . GOTO-statement kent de volgende formaten:

ON variabele GOTO R1,R2, . . ,Ri
ON expressie GOTO R1,R2, . . ,Ri

Afhankelijk van de waarde van de numerieke **variabele** of het resultaat van de **expressie** wordt een sprong gemaakt naar een van de regelnummers **R1** tot en met **Ri.** Indien achter ON een uitdrukking (expressie) staat, bijvoorbeeld K-6, dan wordt eerst deze uitdrukking berekend. Indien de waarde van de variabele of van de uitdrukking 1 is, dan wordt naar het eerste regelnummer (R1), dat achter GOTO staat, gesprongen. Is de waarde 2, dan wordt naar het tweede regelnummer (R2) gesprongen en is de waarde i, dan wordt naar het regelnummer Ri gesprongen.

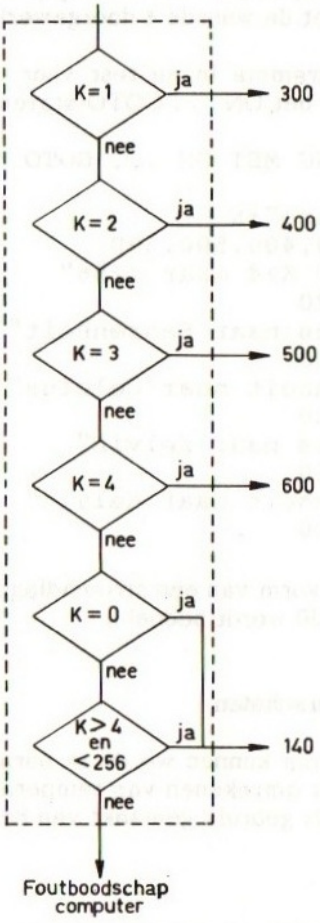
In ons voorbeeld willen we naar vier routines kunnen springen. De regel waarmee we dat kunnen doen gaat er dan als volgt uitzien (dit is de regel zoals die in het voorbeeldprogramma, dat straks volgt, is opgenomen.):

```
130 ON K GOTO 300,400,500,600
```

Uitvoering van dit statement houdt in, dat
indien K=1 er naar regelnummer 300 wordt gesprongen,
indien K=2 er naar regelnummer 400 wordt gesprongen,
indien K=3 er naar regelnummer 500 wordt gesprongen,
indien K=4 er naar regelnummer 600 wordt gesprongen.

Indien de waarde of het resultaat van de uitdrukking 0 is, of groter dan het aantal achter GOTO opgegeven regelnummers, dan gaat de computer door met het eerstvolgende statement

dat volgt op het ON . . . GOTO-statement. Indien de waarde of het resultaat van de uitdrukking kleiner is dan 0 of groter dan 255, dan volgt een foutmelding.



Afb. 7-1 Stroomdiagram van het ON...GOTO-statement.

Zou de waarde van de variabele, of de waarde van de uitdrukking, niet een geheel getal zijn, dan wordt door de computer alleen het gehele deel (integer deel) van de waarde in aanmerking genomen voor het bepalen van het te kiezen regelnummer. Zou de waarde bijvoorbeeld 4.68 zijn, dan wordt door de computer alleen met de waarde 4 doorgewerkt.

Voer nu het volgende programma in en test voor u zelf de zojuist gegeven regels voor het ON . . . GOTO-statement uit.

```
100 REM * OEFENING MET ON ... GOTO *
110 CLS
120 INPUT "Uw keuze";K
130 ON K GOTO 300,400,500,600
140 PRINT "K=0 of K>4 maar <256"
150 PRINT:GOTO 120
300 PRINT "Celcius naar Fahrenheit"
310 PRINT:GOTO 120
400 PRINT "Fahrenheit naar Celcius"
410 PRINT:GOTO 120
500 PRINT "Celcius naar Kelvin"
510 PRINT:GOTO 120
600 PRINT "Fahrenheit naar Kelvin"
610 PRINT:GOTO 120
```

Afbeelding 7-1 geeft in de vorm van een stroomdiagram weer, wat er met regelnummer 130 wordt bedoeld.

7.4 Omrekenen temperatuurschalen

Met het volgende programma kunnen we de in paragraaf 7.3 genoemde routines voor het omrekenen van temperatuurschalen uitvoeren. Daarbij wordt gebruik gemaakt van het ON . . . GOTO-statement.

```
100 CLS
110 PRINT "OMREKENINGEN THERMOMETERSCHALEN"
120 PRINT
130 PRINT "1. Celcius --> Fahrenheit"
140 PRINT "2. Fahrenheit --> Celcius"
```

```
150 PRINT "3. Celcius --> Kelvin"
160 PRINT "4. Fahrenheit --> Kelvin"
170 PRINT "5. Programma beeindiging"
180 PRINT
190 INPUT "Uw keuze";K
200 IF K<1 OR K>5 THEN PRINT "Verkeerde keu
ze!":GOTO 190
210 IF K=5 THEN END
220 ON K GOTO 300,400,500,600
230 A$=INKEY$:IF A$="" GOTO 230
240 GOTO 100
300 CLS
310 INPUT "Graden Celcius";C
320 F=9/5*C+32
330 PRINT F;"graden Fahrenheit"
340 GOTO 230
400 CLS
410 INPUT "Graden Fahrenheit";F
420 C=5/9*(F-32)
430 PRINT C;"graden Celcius"
440 GOTO 230
500 CLS
510 INPUT "Graden Celcius";C
520 KE=C+273
530 PRINT KE;"graden Kelvin"
540 GOTO 230
600 CLS
610 INPUT "Graden Fahrenheit";F
620 KE=5/9*(F-32)+273
630 PRINT KE;"graden Kelvn"
640 GOTO 230
```

Voordat het menu met de regels 110 tot en met 180 op het beeldscherm wordt afgedrukt, wordt eerst het beeldscherm met het CLS-statement van regel 100 schoongemaakt. Na het starten van het programma verschijnt als eerste het menu op het beeldscherm. Nu kan de gewenste routine vanuit het menu worden gekozen, door het intikken van het bijbehorende nummer (1 tot en met 5). Dit nummer wordt met het INPUT-statement van regel 190 ingevoerd en zal achter de tekst "Uw keuze?" worden geplaatst.

Om de routine "Fahrenheit --> Celcius" te starten, moet het cijfer 2 worden ingetikt. Voordat het programma de gewenste routine gaat uitvoeren, wordt eerst gecontroleerd of het ingetikte nummer wel in het menu voorkomt. Cijfers die lager zijn dan 1 of hoger dan 5 zijn niet toegestaan. Deze controle vindt plaats op regelnummer 200 met het IF . . . THEN-statement. Indien het ingevoerde nummer kleiner dan 1 of groter dan 5 is, dan wordt het PRINT-statement achter THEN uitgevoerd en verschijnt op het scherm de boodschap "Verkeerde Keuze". Hierna wordt de mogelijkheid gegeven een nieuwe keuze te maken. Het GOTO-statement op regel 200 wordt namelijk alleen uitgevoerd indien K kleiner is dan 1 of groter dan 5. In de andere gevallen wordt alles wat achter THEN staat niet uitgevoerd en wordt doorgegaan met het volgende regelnummer.

Indien de keuze correct is, zal het programma doorgaan met regelnummer 210, waar wordt gekeken of de keuze 5 was. In dat geval is er namelijk gekozen voor het beeindigen van het programma en wordt het END-statement uitgevoerd. Is K niet gelijk aan 5, dan wordt doorgegaan met het statement op regel 220.

Voor het selecteren van de gekozen routine wordt gebruik gemaakt van het ON . . . GOTO-statement op regelnummer 220. Indien K=1, dan wordt naar regelnummer 300 gesprongen. Bij K=2 wordt naar regel 400 gesprongen, bij K=3 naar regel 500 en bij K=4 naar regel 600. Op de regelnummers 300, 400, 500 en 600 starten de eerder genoemde routines voor het omrekenen van temperatuurschalen.

Na het uitvoeren van de gewenste routine wordt teruggesprongen naar regelnummer 230. Zolang er geen toets wordt ingedrukt, zal A$ leeg zijn, en wordt er steeds naar regelnummer 230 gesprongen. Op het moment dat er een willekeurige toets wordt ingedrukt is A$ niet meer leeg en wordt doorgegaan met het GOTO-statement op regel 240. Er wordt dan teruggesprongen naar het menu.

Opmerking:

Wanneer op een regel achter GOTO slechts 4 regelnum-

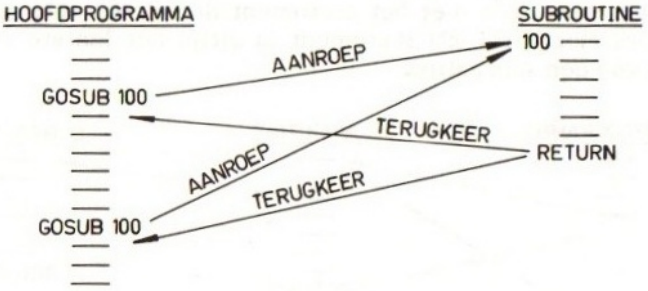
mers kunnen worden geplaatst, terwijl er bijvoorbeeld 7 nodig zijn, dan kan dit probleem als volgt worden opgelost:

```
220 ON K GOTO 300,400,500,600
230 ON K-4 GOTO 700,800,900
```

Is de waarde van K groter dan 4, dan wordt er door het statement heengevallen. Hierdoor wordt het statement van de volgende regel uitgevoerd. In het ON . . . GOTO-statement van die regel zijn de volgende drie regelnummers opgenomen. Daar regelnummer 700 overeenkomt met K=5, zou zonder verdere maatregelen ook door dit statement heengevallen worden. Dit wordt voorkomen door van de variabele K eerst 4 af te trekken. Daar K-4 (indien K=5) 1 is, zal er naar het eerste regelnummer achter GOTO worden gesprongen, dus naar regel 700.

7.5 Subroutines

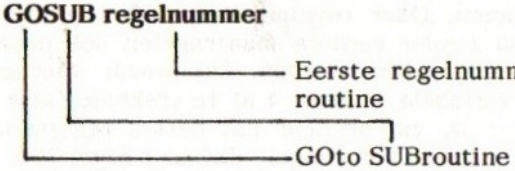
Het komt nogal eens voor, dat in een programma op verschillende plaatsen steeds dezelfde rij statements wordt gebruikt. In dat geval is het efficienter om van die rij statements een op zich zelf staand programmadeel te maken. Dat programmadeel kan dan, indien nodig, vanuit het hoofdprogramma



Afb. 7-2 Het aanroepen van subroutines.

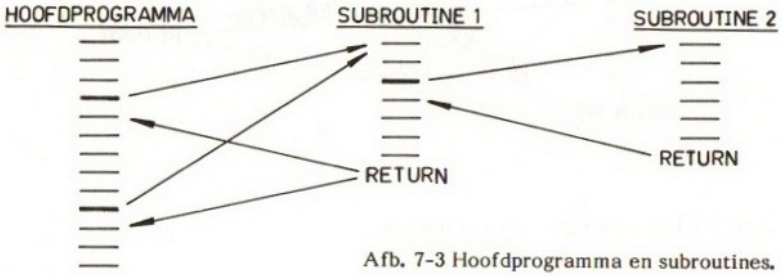
worden aangeroepen. Zo'n programmadeel wordt in computertaal een subroutine genoemd. Wanneer een subroutine wordt aangeroepen, wordt deze uitgevoerd. Na uitvoering van de subroutine wordt dan teruggesprongen naar het programma, dat de subroutine heeft aangeroepen. Het geheel aan acties is weergegeven in afbeelding 7-2.

Een subroutine wordt aangeroepen met behulp van het GOSUB-statement. Dit statement heeft het volgende formaat:



Uitvoering van een GOSUB heeft tot gevolg, dat naar het **regelnummer** wordt gesprongen, dat achter GOSUB staat. Verder onthoudt de computer vanuit welk punt in het programma er gesprongen is. Dit punt wordt het terugkeeradres genoemd.

Als een subroutine is uitgevoerd, wordt door middel van het RETURN-statement teruggekeerd (teruggesprongen) naar het punt waarvandaan gesprongen is. De computer zal nu het programma vervolgen met het statement dat direct volgt op de GOSUB. Het RETURN-statement is altijd het laatste statement van een subroutine.



Afb. 7-3 Hoofdprogramma en subroutines.

Een programma dat andere programma's kan aanroepen, maar dat zelf niet kan worden aangeroepen, wordt een hoofdprogramma genoemd. Subroutines kunnen op hun beurt weer andere subroutines aanroepen. Men spreekt dan van geneste subroutines. Een schematische voorstelling hiervan wordt gegeven in afbeelding 7-3.

7.6 Programma "Menu met rekenfuncties"

Met het volgende programma kunnen optellingen, aftrekkingen, vermenigvuldigingen en delingen worden gemaakt. Voor dit programma hebben we besloten om voor het invoeren van de variabelen A en B een subroutine te maken. Het hoofdprogramma start op regelnummer 100 en de subroutine op regelnummer 700.

```
100 CLS
110 PRINT "REKENKUNDIGE BEWERKINGEN"
120 PRINT
130 PRINT "1. Optellen          a+b"
140 PRINT "2. Aftrekken         a-b"
150 PRINT "3. Vermenigvuldigen a*b"
160 PRINT "4. Delen             a/b"
170 PRINT "5. Programma beeindiging"
180 PRINT
190 INPUT "Uw keuze";K
200 IF K<1 OR K>5 THEN PRINT "Verkeerde keu
ze!":GOTO 190
210 IF K=5 THEN END
220 ON K GOTO 300,400,500,600
300 GOSUB 700
310 PRINT A;"+";B;"=";A+B
320 GOTO 620
400 GOSUB 700
410 PRINT A;"-";B;"=";A-B
420 GOTO 620
500 GOSUB 700
510 PRINT A;"*";B;"=";A*B
520 GOTO 620
600 GOSUB 700
```

```
610 PRINT A;"/";B;"=";A/B
620 PRINT
630 INPUT "Zelfde bewerking (J/N)";J$
640 IF J$="J" GOTO 220
650 GOTO 100
700 REM * Subroutine invoeren *
710 REM * variabelen a en b   *
720 PRINT
730 PRINT "INVOEREN VARIABELEN"
740 INPUT "Variabele a";A
750 INPUT "Variabele b";B
760 PRINT:PRINT "BEREKENING"
770 RETURN
```

7.7 Het statement ON . . . GOSUB

Behalve GOSUB, kennen we ook nog het statement ON . . . GOSUB. Met dit statement kunnen we een subroutine, geselecteerd uit een aantal andere, aanroepen. Welke subroutine we aanroepen hangt af van de waarde van de variabele of de uitdrukking die achter ON staat. ON . . . GOSUB kan worden vergeleken met ON . . . GOTO. Er gelden dan ook dezelfde regels voor. Het enige verschil is, dat, indien een subroutine is uitgevoerd, er wordt teruggesprongen naar de regel die op ON . . . GOSUB volgt. Het formaat van het statement ON . . . GOSUB ziet er als volgt uit:

ON variabele GOSUB lijst met regelnummers
of
ON uitdrukking GOSUB lijst met regelnummers

Als voorbeeld kunnen we het programma in paragraaf 7.4 aanpassen voor ON . . . GOSUB. Wijzig hiervoor de volgende regels:

```
220 ON K GOSUB 300,400,500,600
340 RETURN
440 RETURN
540 RETURN
640 RETURN
```

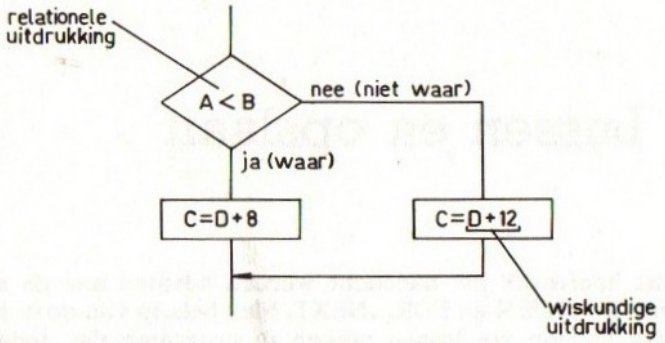
8 Lussen en opslaan

In dit hoofdstuk zal aandacht worden besteed aan de statements IF...THEN en FOR...NEXT. Met behulp van deze statements kunnen we lussen maken in programma's, zodat we bepaalde gedeelten van een programma meerdere malen kunnen laten uitvoeren. Voordat we met de uitleg van die statements beginnen, zullen we eerst iets meer vertellen over vergelijkende (relationele) en logische operators. Verder zal in het eerste deel van dit hoofdstuk ook nog de functie INT worden behandeld. In het tweede deel van dit hoofdstuk zal de nodige aandacht worden besteed aan het opslaan van programma's op cassetteband en het weer terugladen van programma's van die band naar het geheugen.

8.1 Relationele- en logische operators

Relationele operators worden gebruikt om twee numerieke waarden met elkaar te vergelijken. Op grond van het resultaat van die vergelijking kan dan een bepaald verloop aan het programma worden gegeven. Het resultaat van een relationele uitdrukking is "waar" of "niet waar". Dit is de reden dat relationele uitdrukkingen ook wel eens logische uitdrukkingen worden genoemd. Afbeelding 8-1 geeft een voorbeeld van de mogelijkheden die relationele uitdrukkingen bieden.

Indien in afbeelding 8-1 de waarde van variabele A kleiner is dan de waarde van variabele B (het resultaat van de vergelijking is dan "waar"), dan wordt aan variabele C de waarde D+8 toegekend. Is het resultaat van de uitdrukking "niet waar", dan krijgt C de waarde D+12.



Afb. 8-1 Stroomdiagram van een relationele uitdrukking.

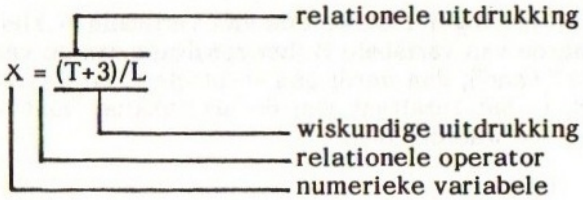
MSX-BASIC kent de volgende relationele operators:

OPERATOR	BETEKENIS	UITDRUKKING
=	GELIJK AAN	A=B
<>	NIET GELIJK AAN	A<>B
<	KLEINER DAN	A<B
>	GROTER DAN	A>B
<=	KLEINER DAN OF GELIJK AAN	A<=B
>=	GROTER DAN OF GELIJK AAN	A>=B

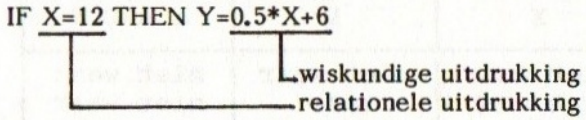
Opmerking:

Het "is gelijk teken" wordt ook gebruikt voor het toekennen van een waarde aan een variable. Bijvoorbeeld: A=3 (LET A=3).

Wanneer in een relationele uitdrukking wiskundige uitdrukkingen voorkomen, dan worden die wiskundige uitdrukkingen eerst uitgewerkt. Pas daarna wordt de vergelijking gemaakt.



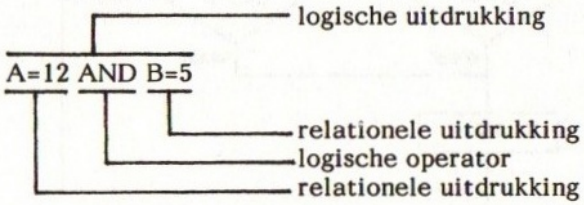
Relationele uitdrukkingen komen in MSX-BASIC voor in de IF...THEN statements. Een voorbeeld hiervan is:



Tijdens de uitvoering van het hiervoor genoemde statement wordt aan variabele Y alleen de waarde 0.5*X+6 toegekend, indien de relationele uitdrukking X=12 "waar" is.

Logische operators

Met behulp van logische operators kunnen twee of meer relationele uitdrukkingen worden getest. Op grond van het resultaat kan dan een beslissing worden genomen. Wanneer bijvoorbeeld een bepaalde wiskundige uitdrukking alleen maar mag worden uitgevoerd, indien zowel A=12 als B=5 geldt, dan moeten de twee relationele uitdrukkingen worden verbonden met de logische operator AND. De logische uitdrukking gaat er dan als volgt uitzien:



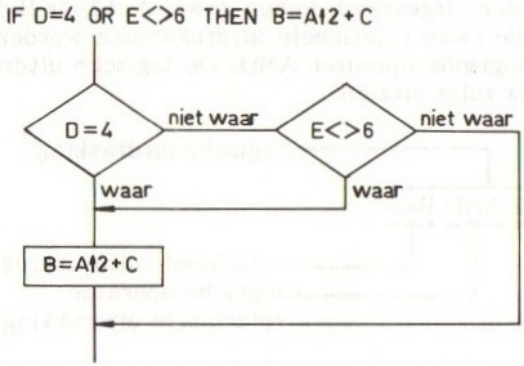
De logische operator AND heeft tot gevolg, dat het resultaat van de logische uitdrukking alleen dan "waar" is, wanneer het resultaat van beide relationele uitdrukkingen "waar" is. Indien een of beide relationele uitdrukkingen "niet waar" zijn, dan wordt het resultaat van de logische uitdrukking "niet waar".

Met twee relationele uitdrukkingen heeft men vier mogelijkheden. Door deze vier mogelijkheden in een tabel te zetten, ontstaat er een zogenaamde waarheidstabel. De relationele uitdrukking A=12 wordt in de tabel X genoemd en de relatio-

nele uitdrukking B=5 wordt Y genoemd.

X	Y	X AND Y
niet waar	niet waar	niet waar
niet waar	waar	niet waar
waar	niet waar	niet waar
waar	waar	waar

Het is ook mogelijk de logische operator OR te gebruiken. Daarbij is het resultaat van de logische uitdrukking "waar", indien het resultaat van een van beide of beide relationele uitdrukkingen "waar" is. Hieruit volgt dat het resultaat van de logische uitdrukking alleen maar "niet waar" is, indien het resultaat van beide relationele uitdrukkingen "niet waar" is.



Afb. 8-2 Stroomdiagram voor de logische operator OR.

Afbeelding 8-2 toont het stroomdiagram voor de logische operator OR. De hierna volgende tabel is daarvan de waarheidstabel. In het stroomdiagram is gebruik gemaakt van de relationele uitdrukkingen D=4 en E<>6. In de tabel komen deze uitdrukkingen respectievelijk overeen met X en Y.

X	Y	X OR Y
niet waar	niet waar	niet waar
niet waar	waar	waar
waar	niet waar	waar
waar	waar	waar

Indien men een voorwaarde wil omdraaien, dan kan men de logische operator NOT gebruiken. Laten we eens aannemen, dat we een bepaalde wiskundige uitdrukking alleen willen laten uitvoeren, indien de waarde van een variabele B niet kleiner is dan 12. Dit kan dan als volgt worden geschreven:

```
IF NOT(B<12) THEN C=D*E
```

In de volgende waarheidstabel voor de logische operator NOT komt X overeen met de uitdrukking B<12.

X	NOT X
niet waar	waar
waar	niet waar

Een logische uitdrukking kan ook uit meer dan twee relationele uitdrukkingen bestaan. Ook kunnen er binnen een logische uitdrukking verschillende relationele operators worden toegepast. De volgende voorbeelden tonen dit aan.

```
A<6 AND C=D AND E=6
A>C AND (E=8 OR F<5)
```

De logische operators werken niet alleen voor relationele uitdrukkingen. Het is ook mogelijk om de inhoud van een variabele in beschouwing te nemen. Indien de inhoud van de numerieke variabele 0 is, dan komt dat overeen met "niet waar". Is de inhoud ongelijk aan 0, dan wordt dat beschouwd als "waar". Het volgende programma laat dit zien:

```
100 A=0
110 PRINT A
```

```
120 A=NOT(A)
130 GOTO 110
```

8.2 Het IF...THEN statement

Wanneer men een bepaalde opdracht alleen maar wil laten uitvoeren op voorwaarde dat er aan een bepaalde conditie is voldaan, kan men in MSX-BASIC gebruik maken van het IF...-THEN statement. Dit statement kent de volgende formaten:

IF uitdrukking THEN statement
IF uitdrukking THEN statement ELSE statement
IF uitdrukking GOTO regelnummer

De **uitdrukking** (voorwaarde) is een relationele- of logische uitdrukking (zie paragraaf 8.1). Het resultaat van die uitdrukking kan waar of niet waar zijn. Alleen indien aan de voorwaarde achter IF is voldaan (indien het resultaat "waar" is), wordt de opdracht die achter de uitdrukking staat uitgevoerd. Wordt niet aan de voorwaarde voldaan, dan gaat de computer door met het statement op de volgende programmaregel. Alleen wanneer het een IF...THEN...ELSE statement is, zal de computer, bij het niet waar zijn van de uitdrukking, verder gaan met het statement achter ELSE. Hier volgen enkele voorbeelden:

```
IF K<2 THEN A=B+2
IF X=6 GOTO 60
IF K<1 OR K>5 THEN PRINT A$
IF X+Y<12 THEN A=B+2 ELSE A=B-6
```

Om een beter inzicht in de werking van het IF...THEN statement te krijgen, volgen hier twee programmavoorbeelden. Met het eerste programma kunnen tafels van vermenigvuldiging worden gegenereerd en met het tweede programma wordt gecontroleerd of de ingevoerde maand (01 - 12) wel juist is.

```
100 REM * TAFELS VAN VERMENIGVULDIGING *
110 CLS
120 INPUT "Welke tafel";T
130 PRINT
140 I=1
150 PRINT USING "##";I;
160 PRINT TAB(4);"x";TAB(6);T;SPC(1);"=";
170 PRINT USING "####";I*T
180 I=I+1
190 IF I<=10 GOTO 150
200 A$=INKEY$:IF A$="" GOTO 200
210 GOTO 110
```

Opmerking:

Druk voor het kiezen van de volgende tafel van vermenigvuldiging een willekeurige toets in.

Het programma "CONTROLE MAAND":

De maand moet worden ingetikt als twee cijfers (01 - 12). Het IF...THEN statement op regelnummer 130 controleert of de ingegeven lengte twee tekens is. Op regelnummer 140 wordt met de functie VAL van de alfanumerieke waarde een numerieke waarde gemaakt, opdat die waarde in regelnummer 150 op juistheid (01-12) kan worden gecontroleerd.

```
100 REM * CONTROLE MAAND *
110 CLS
120 INPUT "Maand (mm)";M$
130 IF LEN(M$)<>2 THEN PRINT "Verkeerde leng
te!":GOTO 120
140 M=VAL(M$)
150 IF M<1 OR M>12 THEN PRINT "Verkeerde waa
rde!":GOTO 120
160 PRINT "Juiste lengte en waarde."
170 A$=INKEY$:IF A$="" GOTO 170
180 GOTO 110
```

Opmerking:

De GOTO-statements op de regelnummers 130 en 150 worden alleen dan uitgevoerd, als aan de voorwaarde achter IF is voldaan. Wanneer niet aan die voorwaarde

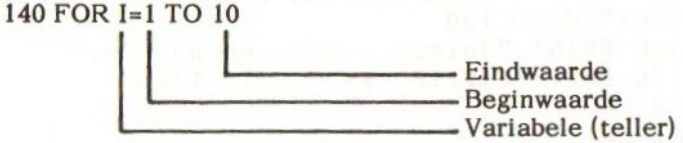
wordt voldaan, wordt direct doorgegaan met het statement op de volgende regel.

8.3 Het FOR...NEXT-statement

In programma's zijn lussen bijna onmisbaar. Bij het schrijven van ieder programma worden wel een of meer lussen toegepast. Tot nu toe hebben we lussen gevormd met behulp van het IF...THEN-statement. Het programma "TAFELS" uit paragraaf 8.2 is daarvan een voorbeeld. We kunnen het probleem echter ook op een andere manier oplossen, namelijk door het toepassen van een **FOR...NEXT-lus.** Het programma gaat er dan als volgt uitzien:

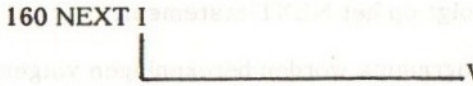
```
100 REM * TAFELS *
110 CLS
120 INPUT "Welke tafel";T
130 PRINT
140 FOR I=1 TO 10
150 PRINT USING "##";I;
160 PRINT TAB(4);"x";TAB(6);T;SPC(1);"=";
170 PRINT USING "####";I*T
180 NEXT I
190 A$=INKEY$:IF A$="" GOTO 190
200 GOTO 110
```

Voor een FOR...NEXT-lus zijn altijd twee statements nodig, het FOR-statement en het NEXT-statement. Deze twee statements zijn onverbrekelijk met elkaar verbonden. We zullen het formaat van het FOR-statement, dat in het programma staat, eens wat nader gaan bekijken.



I is een variabele, die als teller wordt gebruikt voor de FOR...NEXT-lus. Deze variabele bepaalt hoeveel keren de bewer-

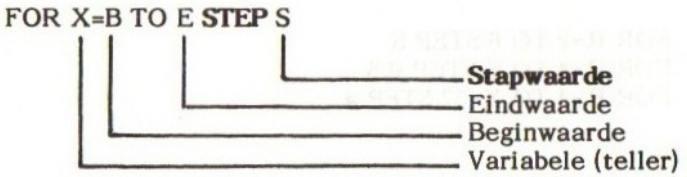
kingen binnen de lus zullen worden herhaald. Met het FOR-statement wordt aan de variabele een waarde toegekend. In ons voorbeeld is 1 de beginwaarde en 10 de eindwaarde. Indien de beginwaarde 1 is, dan is de eindwaarde tevens het maximum aantal keren dat de lus zal worden uitgevoerd. Laten we nu eens naar het bijbehorende NEXT-statement kijken.



Het **NEXT-statement** vertelt de computer waar de lus eindigt. Het is tevens een indicatie dat er moet worden teruggesprongen naar het bijbehorende FOR-statement. De waarde van I zal door het FOR-statement worden opgehoogd met 1 en vervolgens zal de lus opnieuw worden doorlopen. Wanneer I na het ophogen met de waarde 1, de waarde 10 (eindwaarde) heeft overschreden, wordt er naar het regelnummer dat volgt op het NEXT-statement gesprongen, waarmee de lus wordt verlaten.

Een FOR-statement gaat dus altijd vergezeld van een NEXT-statement. Het laatste dient voor het sluiten van de lus en het terugspringen naar het FOR-statement. Bij MSX-BASIC mag de variabele bij het NEXT-statement ook worden weggelaten.

We hebben gezien dat de waarde van de variabele van het FOR-statement elke keer automatisch met 1 wordt verhoogd. Het is echter ook mogelijk om de waarde van die variabele met een andere waarde dan 1 op te hogen. In dit geval gaat het FOR-statement er als volgt uitzien:



De variabele X zal steeds met de stapwaarde S worden opge-

hoogd. Indien S een negatieve waarde heeft, dan zal de variabele X met de in S staande negatieve waarde worden verlaagd. De stapwaarde in S hoeft niet een geheel getal te zijn. Het mag ook een breuk zijn (bijvoorbeeld 2.5, 0.5 of 1.25). Indien de waarde in S negatief is, zal de waarde in B (beginwaarde) groter moeten zijn dan de waarde in E (eindwaarde). Is dit niet het geval, dan zal de computer direct doorgaan met het statement dat volgt op het NEXT-statement.

Met het volgende programma worden berekeningen volgens de wet van Ohm gemaakt. De spanning U wordt hierbij ingevoerd met een INPUT-statement.

```
100 REM * WET VAN OHM *
110 CLS
120 INPUT "Spanning U";U
130 PRINT
140 FOR R=2 TO 6 STEP .5
150 PRINT "I = U : R =";U;":";R;TAB(21);"=";
160 PRINT USING "####.##";U/R
170 PRINT
180 NEXT R
190 A$=INKEY$:IF A$="" GOTO 190
200 GOTO 110
```

De beginwaarde B, de eindwaarde E en de stapwaarde S kunnen zijn uitgedrukt in:

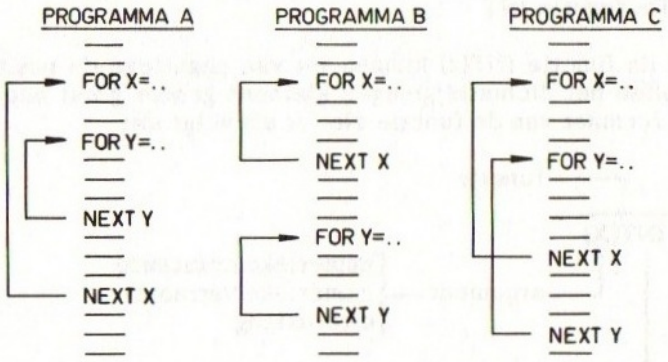
- een numerieke constante (bijv. 12)
- een numerieke variabele (bijv. A)
- een uitdrukking (bijv. X*2)

Voorbeelden hiervan zijn:

```
FOR R=2 TO 6 STEP S
FOR R=4 TO E STEP 0.5
FOR R=4 TO X+32 STEP 4
```

Opdracht:
Pas het laatste programma zodanig aan, dat de stapwaarde S ook via een INPUT-statement wordt ingevoerd.

Het is ook mogelijk om FOR...NEXT-lussen te "nesten". "Nesten" wil zeggen, dat er een FOR...NEXT-lus binnen een andere FOR...NEXT-lus zit (zie afbeelding 8-3). De variabelen van de FOR...NEXT-lussen mogen dan niet dezelfde namen hebben. Ook is het niet toegestaan dat de lussen elkaar kruisen, zoals in programma C uit afbeelding 8-3 is te zien. De programma's A en B uit die afbeelding zijn wel toegestaan. Later zullen we nog een aantal voorbeelden van geneste lussen tegenkomen.



Afb. 8-3 Schematische voorstelling van FOR...NEXT-lussen.

FOR...NEXT-lussen zijn zeer geschikt voor het maken van tabellen. Laten we er eens van uitgaan, dat we een tabel willen produceren, waarin het volgende komt te staan:

n, n^2, n^3.

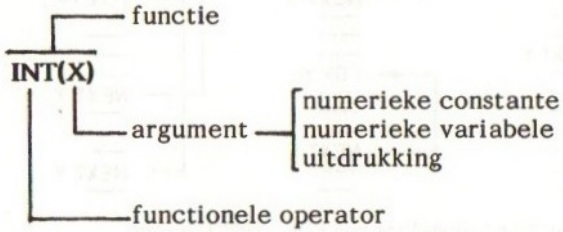
Verder zouden we de begin- en eindwaarde van n zelf willen bepalen door deze tijdens de uitvoering van het programma via het toetsenbord in te voeren.

```
100 REM * MAKEN VAN TABELLEN *
110 CLS
120 INPUT "Beginwaarde";B
130 INPUT "Eindwaarde ";E
140 PRINT
150 PRINT TAB(2);"n";TAB(14);"n^2";TAB(28);
"n^3"
```

```
160 PRINT
170 FOR N=B TO E
180 PRINT USING "###";N;
190 PRINT TAB(12):PRINT USING "#####";N^2;
200 PRINT TAB(24):PRINT USING "#######";N^3
210 NEXT N
220 A$=INKEY$:IF A$="" GOTO 220
230 GOTO 110
```

8.4 De functie INT

Met de functie INT(x) kunnen we van negatieve en positieve getallen het dichtstbijgelegen kleinere gehele getal bepalen. Het formaat van de functie ziet er als volgt uit:



Voorbeelden hiervan zijn:

```
100 A=12.23:B=-137.47:C=26.68
110 PRINT INT(A),INT(B+C),INT(-12.6)
120 END
```

In het volgende programma wordt van de functie INT gebruik gemaakt om te bepalen door welke getallen een, via het toetsenbord, ingevoerd getal deelbaar is. In het programma wordt in de FOR...NEXT-lus (regelnummers 170 tot en met 210) het getal G door de getallen 2 tot en met INT(G/2) gedeeld. In regelnummer 180 wordt bepaald of het getal deelbaar is door D. Indien G/D=INT(G/D), dan is het getal deelbaar door D en wordt het afgedrukt. Stel bijvoorbeeld dat G=72 en D=12, dan zal G/D=6 en INT(G/D)=6 zijn. dit wil dus zeggen dat G deelbaar is door D. Met I=I+1 (regelnummer 200) wordt gedetec-

teerd of het getal een priemgetal is. Indien alle waarden van D aan bod zijn geweest en I is nog steeds nul, dan is het getal G door geen enkel getal deelbaar. Het getal G is dan een priemgetal.

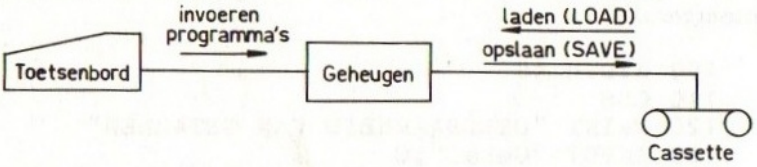
```
100 WIDTH 40
110 CLS
120 PRINT "DEELBAARHEID VAN GETALLEN"
130 INPUT "Getal";G
140 I=0
150 PRINT
160 PRINT "Het getal";G;"is deelbaar door:"
:PRINT
170 FOR D=2 TO INT(G/2)
180 IF G/D<>INT(G/D) GOTO 210
190 PRINT D;
200 I=I+1
210 NEXT D
220 IF I>0 GOTO 250
230 PRINT "Het getal is ondeelbaar,"
240 PRINT "dus een priemgetal."
250 PRINT:PRINT:PRINT
260 INPUT "Volgend getal (J/N)";J$
270 IF J$="J" GOTO 110
280 END
```

8.5 Opslaan van programma's op cassette

Tot nu toe zijn alle programma's die we hebben ingetikt verloren gegaan. Dit had een van de volgende oorzaken:
- Het geven van het commando NEW.
- Het uitschakelen van de computer.

Er is echter een mogelijkheid aanwezig, om een programma vanuit het geheugen van de computer naar een audio-cassette te schrijven, waarna we naar believen commando's NEW kunnen geven of de computer kunnen uitzetten, zonder dat hierdoor de programma's werkelijk verloren gaan. Ze staan dan nog steeds op de cassette. Uiteraard is er ook een mogelijkheid om programma's die op cassette staan weer in te lezen

in het geheugen van de computer. Een en ander is weergegeven in afbeelding 8-4.



Afb. 8-4 Opslaan en laden van programma's.

De commando's die worden gebruikt voor het opslaan van programma's op cassette en het weer laden van programma's van cassette in het geheugen, staan vermeld in de volgende tabel:

COMMANDO	RICHTING	CODE
CSAVE	Geheugen naar cassette	Binair
CLOAD	Cassette naar geheugen	Binair
SAVE	Geheugen naar cassette	ASCII
LOAD	Cassette naar geheugen	ASCII
MERGE	Cassette naar geheugen	ASCII

8.6 Het commando CSAVE

Het commando CSAVE wordt gebruikt voor het opslaan (schrijven) van programma's vanuit het geheugen op cassette. De programma's worden in dezelfde code als waarin de programma's in het geheugen staan op cassette geschreven, namelijk in binaire vorm. Dit in tegenstelling tot het commando SAVE, waarmee de programma's in ASCII-formaat (tekstformaat) op cassette worden geschreven.

Programma's die met het commando CSAVE op cassette zijn geschreven, kunnen weer in het geheugen worden geladen met het commando CLOAD. Het formaat van het CSAVE-commando is:

CSAVE "programmanaam",X

De **programmanaam**, die men zelf mag kiezen, moet tussen aanhalingstekens staan en mag uit niet meer dan 6 tekens bestaan. **X** is niet verplicht. De waarde van X, als die wordt gespecificeerd, mag 1 of 2 zijn. X heeft de volgende functie:

X=1 - de schrijfsnelheid is 1200 baud. Dit komt overeen met 1200 bits (ongeveer 150 tekens) per seconde.

X=2 - de schrijfsnelheid is 2400 baud. Dit komt overeen met 2400 bits (ongeveer 300 tekens) per seconde.

Wanneer geen snelheid is gespecificeerd, wordt de laatst gebruikte snelheid genomen. Na het aanschakelen van de computer wordt automatisch de laagste snelheid (1200 baud) genomen.

De schrijfsnelheid naar de cassette kan ook worden gespecificeerd met het **statement SCREEN.** Hierbij is:

SCREEN , , ,1 - 1200 baud.
SCREEN , , ,2 - 2400 baud.

Het kiezen van de snelheid is grotendeels afhankelijk van de kwaliteit van de cassetterecorder en de cassetteband. Indien de kwaliteit van de cassetterecorder of -band niet hoog is, kan de hoge schrijfsnelheid beter niet worden gebruikt. Alleen bij gebruik van uitstekende cassetterecorder en een goede kwaliteit cassetteband, kunt u de snelheid van 2400 baud proberen.

We zijn nu op het punt aangeland, dat we met het commando CSAVE een programma op cassette kunnen schrijven. We gaan hiervoor eerst het volgende programma invoeren in het geheugen:

```
100 REM * TEST COMMANDO SAVE *
110 CLS
120 PRINT "Dit programma heet CASTES"
```

```
130 PRINT "en wordt met het commando"
140 PRINT "CSAVE op cassette geschreven."
150 END
```

Wanneer we het programma via het toetsenbord hebben ingevoerd, geven we het de naam "CASTES" (cassette test). Nu gaan we dit programma op cassette schrijven. Hiervoor moet de volgende procedure (ingeval van een recorder zonder REMOTE-voorziening) worden gevolgd:

1. Laad een cassetteband in de recorder.
2. Zet de volumeregelaar op ongeveer 2/3.
3. Tik het commando CSAVE "CASTES" in.
4. Zet de recorder in de opname stand.
5. Druk op de RETURN-toets.
   Het programma wordt nu naar cassette geschreven.
6. Wanneer het programma volledig op cassette is geschreven, verschijnt op het scherm de melding Ok.
7. Druk de STOP-toets op de recorder in.
8. Druk de REWIND-toets op de recorder in, waarmee de band wordt teruggespoeld.

Opmerking:

Voor recorders met een REMOTE-voorziening zal punt 4 direct na punt 2 moeten worden uitgevoerd. Punt 7 kan vervallen, omdat de recorder automatisch zal stoppen.

Programma's die met CSAVE op cassette zijn geschreven, kunnen niet met RUN "programmanaam" worden geladen en automatisch gestart. Dit kan wel met programma's die met het SAVE-commando naar cassette zijn geschreven.

8.7 Het commando CLOAD

Programma's die met CSAVE in binaire vorm op cassette zijn geschreven, moeten met het commando CLOAD in het geheugen worden geladen. Bij CLOAD behoeft niet de snelheid te worden opgegeven, de computer bepaalt die snelheid zelf. CLOAD kan, evenals CSAVE, als een statement in het programma worden opgenomen, of als commando direct via het

toetsenbord worden gegeven. Er zijn twee formaten voor het CLOAD-commando mogelijk:

CLOAD "programmanaam"
CLOAD? "programmanaam"

De **programmanaam** mag worden weggelaten. In dat geval zal het eerste programma, dat de computer van de band leest, in het geheugen worden geladen. Wordt de programmanaam wel opgegeven, dan mag die naam uit maximaal 6 tekens bestaan. De computer zal nu de band afzoeken naar het programma met de opgegeven naam, en dat programma zal in het geheugen worden geladen.

Zodra de computer een programma op de band ziet staan, wordt op het scherm een melding gegeven. Deze melding kan zijn:

Found: programmanaam

Found (gevonden) geeft aan dat de computer het gevraagde programma in het geheugen schrijft. Wanneer het programma helemaal van de cassette naar het geheugen is geschreven, verschijnt op het scherm de melding "Ok".

Wanneer de naam van een op de band gevonden programma echter niet dezelfde is als de gevraagde programmanaam, dan zal op het scherm de melding:

Skip: programmanaam

verschijnen. Skip (overslaan) betekent dat de computer wel een programma heeft gevonden, maar dat het niet wordt ingelezen. MSX-BASIC gaat dan weer verder met zoeken naar het opgegeven programma. Voer nu, voor het testen van het commando CLOAD, het volgende programma in via het toetsenbord.

```
100 REM * TEST COMMANDO CLOAD *
110 CLS
120 PRINT "Dit programma heet LOTES"
```

```
130 PRINT "en wordt met het commando"
140 PRINT "CLOAD van cassette gelezen."
150 END
```

Schrijf het programma (na het intikken) onder de naam "LOTES" (LOAD TEST) naar cassette. Volg hiervoor de in paragraaf 8.6 gegeven procedure. Het programma moet achter het programma "CASTES" op de band worden gezet. Spoel eventueel de band een eind op. Wanneer we het programma hebben weggeschreven, gaan we het programma "LOTES" weer in het geheugen laden met het commando CLOAD. Maak echter eerst het geheugen schoon met het commando NEW. Volg voor het laden van een programma van cassette de volgende procedure (voor cassetterecorders zonder REMOTE-voorziening):

1. Laad de band in de cassetterecorder (indien nodig).
2. Spoel de band terug (indien nodig).
3. Zet de volumeregelaar op 2/3.
4. Tik het commando CLOAD "LOTES" in.
5. Druk op RETURN.
6. Zet de recorder op afspelen.
7. Het programma "LOTES" wordt nu gezocht.
8. Zodra "LOTES" is gevonden verschijnt de melding Ok.
9. Stop de cassetterecorder.
10. Spoel de cassette terug.

Opmerking:

Voor recorders met REMOTE-voorziening zal punt 6 voor punt 4 moeten worden uitgevoerd. Verder zal de recorder automatisch stoppen na het verschijnen van de melding Ok, zodat punt 9 kan vervallen.

Mocht er helemaal niets gebeuren (noch Found:, noch Ok.) dan staat de volumeregelaar waarschijnlijk niet in de goede stand. Experimenteer hier een beetje mee, en noteer de stand van de volumeregelaar waarbij programma's goed worden geladen. Bij een schaalverdeling van 0 tot 10 wordt meestal een goed resultaat bereikt bij de stand 7.

Voor ons voorbeeld moet de volgende informatie op het beeldscherm verschijnen:

```
CLOAD "LOTES"
Skip: CASTES
Found: LOTES
Ok
```

Om te controleren of een programma, dat we met CSAVE naar cassette hebben geschreven, goed op die cassette staat, gebruiken we het CLOAD?-commando. De cassette is nu eenmaal niet het meest betrouwbare geheugenmedium. Het is dan ook zaak om elke keer, nadat een programma naar cassette is geschreven, te controleren of dat ook goed is gegaan. Tik voor een "juistheidstest" het volgende programma in:

```
100 REM * TEST MET COMMANDO CLOAD? *
110 CLS
120 FOR I=1 TO 5
130 PRINT "TEST";I
140 NEXT I
150 END
```

Plaats het programma na het intikken onder de naam "TEST" achter het programma "LOTES" op cassette. Spoel hierna de cassette terug en geef het commando CLOAD? "TEST" in. Pas voor CLOAD? dezelfde procedure toe als voor CLOAD. Op het scherm zal dan de volgende informatie verschijnen:

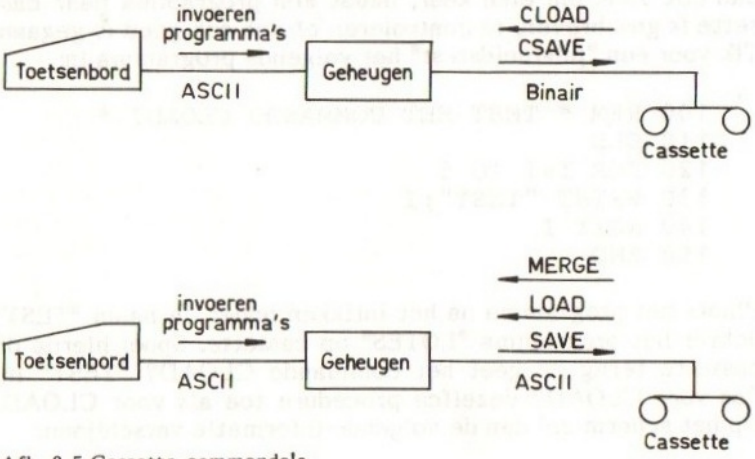
```
CSAVE "TEST"
OK
CLOAD? "TEST"
Skip : CASTES
Skip : LOTES
Found: TEST
Ok
```

Wanneer het programma dat nog in het geheugen staat, en het programma dat op de band werd gevonden, gelijk zijn, zal MSX-BASIC de boodschap Ok geven. Zijn de programma's niet gelijk, dan zal op het scherm de boodschap "Verify error"

worden gegeven.

8.8 Het commando SAVE

Het **commando SAVE** werkt zowel voor cassettes als voor schijveneenheden. De werking van SAVE voor schijveneenheden is echter niet precies gelijk aan de werking voor cassettes. Daar we het in dit deel nog niet over het gebruik van schijveneenheden hebben, zullen we de werking van SAVE nu alleen voor cassette behandelen.



Afb. 8-5 Cassette-commando's.

Programma's die met het commando SAVE op cassette zijn geschreven, kunnen met het **commando LOAD of MERGE** weer in het geheugen worden geladen (zie afbeelding 8-5). Het commando SAVE kan ook in een programma als een statement worden opgenomen. Het formaat van het SAVE-commando voor cassette ziet er als volgt uit:

SAVE "CAS:programmanaam"

Het schrijven op cassette gebeurt ongecodeerd. Dit wil zeg-

gen, dat teken voor teken in ASCII-code wordt weggeschreven. Met andere woorden, op de cassette komen exact dezelfde tekens te staan die we bij het invoeren van het programma via het toetsenbord hebben ingetikt.

Een programma, dat op cassette is opgeslagen met behulp van het commando SAVE, kan met het commando **RUN "programmanaam"** worden geladen van cassette, waarna het automatisch wordt gestart. Dit is niet mogelijk met programma's die met het CSAVE-commando zijn opgeslagen.

Voor het schrijven van een programma op cassette met het commando SAVE geldt dezelfde procedure als voor CSAVE (zie paragraaf 8.6).

8.9 Het commando LOAD

Ook voor LOAD wordt hier alleen ingegaan op de werking bij gebruik van de cassetterecorder. Programma's die met het commando SAVE op cassette zijn opgeslagen, kunnen met het commando LOAD weer in het geheugen worden geladen. Het formaat van het LOAD-commando is:

LOAD "CAS:programmanaam",R

De letter **R** (met de komma ervoor) hoeft niet te worden gebruikt. Wordt de letter R wel ingetikt, dan wordt het programma, direct nadat het is geladen, gestart. Voer nu het volgende programma in:

```
100 WIDTH 40
110 CLS
120 PRINT "Dit is een test met het"
130 PRINT "commando SAVE en LOAD"
140 PRINT "voor cassette."
150 END
```

Schrijf het programma onder de naam "SALO" (SAVE en LOAD) aan het begin van de cassetteband. Spoel hierna de band terug en maak het geheugen schoon met het commando NEW. Laad vervolgens het programma "SALO" met het com-

mando LOAD in het geheugen. Het programma moet direct na het laden automatisch worden gestart. Het bovenstaande proces zal als volgt verlopen:

```
SAVE "CAS:SALO"
Ok
Stoppen en terugspoelen van de band.
NEW
LOAD "CAS:SALO",R
Found: SALO
Het programma wordt automatisch gestart.
```

Opmerking:

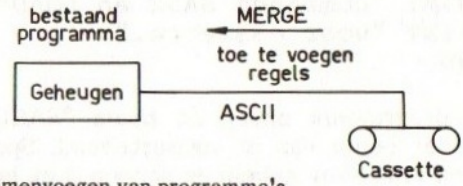
Wanneer de R achter het LOAD-commando wordt weggelaten, wordt na "Found: SALO" de melding Ok op het scherm gezet. Het programma kan dan met RUN worden gestart.

Wanneer LOAD in een programma is opgenomen, blijven boodschappen als "Found" en "Skip" achterwege.

8.10 Het commando MERGE

Met het commando MERGE kunnen programmaregels, die met het commando SAVE op cassette zijn gezet, van die cassette worden gelezen en aan het programma, dat op dat moment in het geheugen staat, worden toegevoegd (zie afbeelding 8-6).

Wanneer een of meerdere regels hetzelfde regelnummer hebben als die welke reeds in het geheugen staan, zullen de in het geheugen staande regels worden overschreven door de regels die van cassette worden gelezen.



Afb. 8-6 Het samenvoegen van programma's.

Het commando MERGE komt in zoverre overeen met het commando LOAD, dat:

- MERGE het geheugen niet eerst schoonmaakt en LOAD wel.
- MERGE de toevoeging "R" niet toestaat.

Het niet schoonmaken van de geheugeninhoud is juist het sterke punt van MERGE. Het is zodoende mogelijk programma's met elkaar te combineren. MERGE is ook zeer nuttig bij het in gedeelten (modulair) ontwikkelen van programma's. De verschillende gedeelten kunnen dan nadat ze zijn geschreven en getest, direct naar cassette worden geschreven als losse modules. Later kunnen die modules dan met behulp van MERGE worden samengevoegd in het geheugen. Ook komt het nog al eens voor dat een bepaalde module in meerdere programma's wordt gebruikt. Deze module (subroutine) kan met MERGE aan ieder in het geheugen staand programma worden toegevoegd, vooropgesteld dat de regelnummers elkaar niet overlappen.

Het formaat van het commando MERGE voor cassette ziet er als volgt uit:

MERGE "CAS:programmanaam"

Voer, voor het testen van het commando MERGE, het volgende programma in:

```
200 REM * PROGRAMMA EINDE? *
210 PRINT:PRINT
220 INPUT "Programma einde (J/N)";N$
230 IF N$="N" GOTO 110
240 END
```

Schrijf het programma, na het invoeren in het geheugen, met het commando SAVE op cassette. Noem het programma "PREIND". Geef vervolgens het commando NEW en voer het volgende programma in:

```
100 REM * TAFELS VAN VERMENIGVULDIGING *
110 CLS
120 INPUT "Welke tafel";T
130 PRINT
140 FOR I=1 TO 10
150 PRINT USING "##";I;
160 PRINT TAB(4);"x";TAB(6);T;SPC(1);"=";
170 PRINT USING "####";I*T
180 NEXT I
200 A$=INKEY$:IF A$="" GOTO 200
210 GOTO 110
```

Schrijf dit programma na het invoeren met het commando SAVE op cassette. Noem het programma "TAFELS". MERGE vervolgens het programma "PREIND" met het programma "TAFELS", dat in het geheugen staat. Het MERGE-commando dient er dan als volgt uit te zien:

```
MERGE "CAS:PREIND"
```

Voor het commando MERGE kan dezelfde cassettebehandelings-procedure worden gebruikt als voor het commando LOAD. Geef na het beeindigen van het commando MERGE het commando LIST en start vervolgens het programma met het commando RUN.

Kort samengevat verloopt het proces als volgt:

1. Invoeren van programma PREIND.
2. SAVE "CAS:PREIND"
   Ok
3. NEW
   Ok
4. Invoeren programma TAFELS via toetsenbord.
5. SAVE "CAS:TAFELS"
   Ok
6. Spoel cassette terug.
7. MERGE "CAS:PREIND"
   Found: PREIND
   Ok

8. LIST
9. RUN

8.11 MOTOR ON/OFF

Wanneer een cassetterecorder is voorzien van een speciale REMOTE-aansluiting (afstandsbediening van de motor), kan de motor van de recorder door middel van het statement MOTOR worden aangestuurd. Het statement kent drie formaten:

MOTOR ON - De recorder motor wordt aangezet.
MOTOR OFF - De recorder motor wordt uitgezet.
MOTOR - De recordermotor wordt uitgezet, indien deze aanstond, of de motor wordt aangezet, indien deze uitstond.

Bij een aantal commando's, zoals CSAVE en CLOAD, wordt de motor van de recorder automatisch door de computer aan- en uitgeschakeld. Daarom mogen bij deze commando's (bij cassetterecorders met remote aansluiting) de toetsen PLAY (LOAD) of PLAY+RECORD (SAVE) worden ingedrukt, voor het commando CLOAD of CSAVE is gegeven. Op het moment dat de RETURN-toets wordt ingedrukt (activeren van de commando's), wordt de motor automatisch door de computer gestart.

Wanneer er geen gebruik wordt gemaakt van statements die de cassetterecorder automatisch starten, kan men, om de motor te starten of te stoppen, gebruik maken van het statement MOTOR. Laten we eens uitgaan van de situatie, dat we de cassetteband willen terugspoelen en dat de recorder is voorzien van afstandsbediening. We kunnen dan de volgende procedure volgen:

a) MOTOR ON (directe mode)
b) terugspoelen (REWIND-toets)
c) MOTOR OFF (directe mode)

Met het volgende programma kunnen we de verschillende formaten van MOTOR testen.

```
100 REM * TESTEN VAN MOTORFUNCTIES *
110 WIDTH 40
120 PRINT "Stel recorder in op terugspoelen
 en"
130 PRINT "druk daarna een willekeurige toe
ts in."
140 A$=INKEY$:IF A$="" GOTO 140
150 MOTOR ON
160 PRINT
170 PRINT "Druk een willekeurige toets in,"
180 PRINT "wanneer cassetteband is terugges
poeld."
190 A$=INKEY$:IF A$="" GOTO 190
200 MOTOR OFF
210 PRINT "Stel recorder in op opnemen en"
220 PRINT "druk daarna een willekeurige toe
ts in."
230 A$=INKEY$:IF A$="" GOTO 230
240 PRINT
250 PRINT "Cassetteband wordt getransportee
rd"
260 PRINT "naar begin band."
270 MOTOR ON
280 FOR I=1 TO 5000:NEXT I
290 MOTOR OFF
300 PRINT
310 PRINT "Programma wordt op band geschrev
en."
320 CSAVE "TESMOT"
330 PRINT "Programma is geschreven,"
340 PRINT "zet recorder af."
350 END
```

9 Tijd en fouten

In dit hoofdstuk zullen we ons gaan bezig houden met twee interessante onderwerpen, namelijk met de "tijd" en met het "afhandelen van fouten".

Verder wordt er met behulp van de functies LEFT$, MID$, RIGHT$ en LEN ook nog gemanipuleerd met strings (alfanumerieke constanten en variabelen).

De statements die betrekking hebben op de "tijd" zijn ON INTERVAL GOSUB en INTERVAL ON/OFF/STOP. Daarnaast zullen we gebruik maken van de systeemvariabele TIME.

Voor het afhandelen van fouten hebben we te maken met de statements ON ERROR GOTO, RESUME en ERROR en de functies ERL en ERR.

Voor het testen van programma's en het foutzoeken in programma's zal aandacht worden besteed aan TRON en TROFF. Deze twee commando's kunnen tevens als statements in een programma worden gebruikt. Daarnaast zullen we, in hetzelfde verband, het statement STOP en het commando CONT behandelen.

Verder zal ook nog het statement BEEP, dat een kort piepsignaal produceert, aan de orde komen.

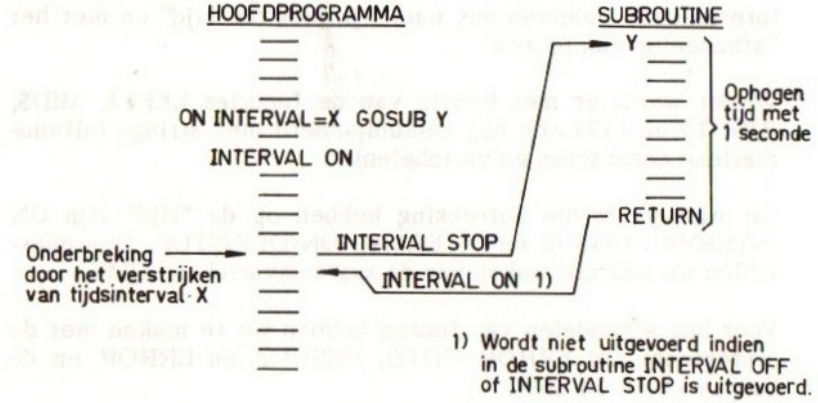
9.1 De tijd

Wat is mooier dan te beginnen met een programma voor een elektronische (digitale) klok, die de uren, minuten en seconden aangeeft. In dit programma komen we de belangrijke sta-

tements ON INTERVAL GOSUB en INTERVAL ON/STOP/OFF tegen. Met behulp van het statement:

ON INTERVAL = tijdsinterval GOSUB regelnummer

wordt de mogelijkheid geschapen om op regelmatige tussenpozen het programma te onderbreken en naar een speciale subroutine te springen (zie afbeelding 9-1).



Afb. 9-1 Programma-onderbreking door het verstrijken van een INTERVAL.

Een onderbreking wordt pas in bewerking genomen, als het statement dat in uitvoering is op het moment dat het verzoek tot onderbreking komt, volledig is afgewerkt. Een onderbreking wordt ook wel gezien als een niet geprogrammeerde sprong.

Het tijdsinterval wordt aangegeven in het aantal eenvijftigste seconden. Om het programma iedere seconde te onderbreken moet voor het tijdsinterval de waarde 50 worden ingevuld. 50 keer 1/50 seconde is 1 seconde. Het regelnummer achter GOSUB geeft de eerste regel aan van de subroutine, waar steeds na het verstrijken van het tijdsinterval naar toe wordt gesprongen. In ons voorbeeld zal de subroutine de tijd steeds met 1 seconde ophogen.

Het ON INTERVAL GOSUB statement kunnen we ergens aan het begin van het programma zetten. Zodra dit statement is uitgevoerd, weet de computer, als er een onderbreking plaats vindt door het verstrijken van het tijdsinterval, naar welke subroutine moet worden gesprongen.

Voordat de subroutine kan worden aangeroepen, moet het onderbreken van het programma door het verstrijken van het tijdsinterval worden geactiveerd door **INTERVAL ON.** Na uitvoering van INTERVAL ON zal MSX-BASIC na uitvoering van elk volgend statement controleren of de tijdsinterval is verstreken. Indien dit zo is, zal het programma worden onderbroken en zal de subroutine worden aangeroepen. Samengevat kan men zeggen, dat MSX-BASIC de gespecificeerde subroutine in het statement ON INTERVAL GOSUB op regelmatige tussenpozen van X/50 seconden zal uitvoeren. X is het gespecificeerde tijdsinterval in ON INTERVAL GOSUB.

Wanneer een onderbreking optreedt en de subroutine wordt aangeroepen, wordt automatisch het statement **INTERVAL STOP** uitgevoerd (zie afbeelding 9-1). Dit betekent dat onderbreking door het verstrijken van het tijdsinterval tijdens het uitvoeren van de subroutine geen effect zal hebben, zolang er geen INTERVAL ON wordt gegeven.

Wanneer het, in een bepaald gedeelte van een programma, niet wenselijk is dat het programma wordt onderbroken, maar dat het verstrijken van het tijdsinterval wel moet worden onthouden, kan dit worden geregeld met het statement INTERVAL STOP. Is het tijdsinterval verstreken, dan gebeurt er ogenschijnlijk niets, echter, zodra het onderbreken door het tijdsinterval weer wordt geactiveerd met het statement INTERVAL ON, wordt het programma alsnog onderbroken en wordt de bijbehorende subroutine uitgevoerd.

Aan het einde van de subroutine keert het programma door middel van het statement RETURN terug naar het punt waar het eerder, door het verstrijken van het tijdsinterval, werd onderbroken. Tijdens uitvoering van RETURN wordt automatisch INTERVAL ON gegeven. Dit gebeurt echter niet, wanneer in de subroutine het statement INTERVAL OFF of

INTERVAL STOP werd uitgevoerd.

Wanneer de mogelijkheid tot onderbreken van het programma door het verstrijken van het tijdsinterval dient te worden opgeheven, kan het statement **INTERVAL OFF** worden gegeven. Bij het beeindigen van een programma wordt automatisch een INTERVAL OFF uitgevoerd.

Elektronische klok

We zijn nu op het punt aangekomen, dat we het programma voor de klok kunnen invoeren.

```
100 REM * ELEKTRONISCHE KLOK *
110 CLS
120 ON INTERVAL=50 GOSUB 500
130 INPUT "Momentele tijd (U,M,S)";U,M,S
140 INTERVAL ON
150 CLS
160 LOCATE 7,2
170 PRINT "ELEKTRONISCHE KLOK"
180 GOTO 180
500 S=S+1
510 IF S>59 THEN S=0:M=M+1
520 IF M>59 THEN M=0:U=U+1
530 IF U>23 THEN U=0
540 LOCATE 12,10
550 PRINT USING "##";U;:PRINT ":";
560 PRINT USING "##";M;:PRINT ":";
570 PRINT USING "##";S
580 RETURN
```

Met het ON INTERVAL GOSUB statement op regelnummer 120 wordt aangegeven, dat het programma om de 50/50 = 1 seconde zal worden onderbroken en naar de subroutine op regelnummer 500 zal worden gesprongen. In de subroutine wordt de tijd met 1 seconde verhoogd, waarna de nieuwe tijd op het scherm wordt afgedrukt. Het INPUT-statement op regelnummer 130 dient om de momentele tijd in uren, minuten en seconden in te voeren. Wanneer dit is gebeurd wordt het statement INTERVAL ON uitgevoerd. Vanaf dit moment

wordt het programma door het verstrijken van het tijdsinterval onderbroken en wordt de subroutine op regel 500 uitgevoerd.

Door middel van de statements 150 tot en met 170 wordt de tekst "Elektronische klok" afgedrukt. Het hoofdprogramma eindigt met het GOTO-statement op regelnummer 180. Als de subroutine niet in uitvoering is, dan wordt het GOTO-statement uitgevoerd. Het statement springt namelijk steeds naar zichzelf toe. Dit betekent dat elke onderbreking op regelnummer 180 zal plaatsvinden. De subroutine begint op regelnummer 500 en spreekt voor zichzelf.

Stopwatch

We hebben nu gezien, hoe we, met behulp van speciaal voor dat doel ontworpen BASIC-statements, het verstrijken van de tijd kunnen gebruiken om er een klok mee te maken. Een in de computer ingebouwde klok zorgt er voor dat het normale verloop van het programma 50 keer per seconde wordt onderbroken. Tijdens die onderbreking wordt een tellertje bijgehouden. Met behulp van deze teller kan de computer nagaan of het in het BASIC-programma opgegeven tijdsinterval al is verstreken. Deze teller wordt echter bijgehouden in een systeemvariabele, waarvan wij, ook zonder het statement ON INTERVAL, gebruik kunnen maken.

De bedoelde systeemvariabele heeft de naam TIME. Na het aanschakelen van de computer zal deze variabele de inhoud "0" krijgen, doch al spoedig staat hierin een groter getal. Er wordt immers 50 keer per seconde een 1 bijgeteld. Net als bij alle andere variabelen, kunnen wij een waarde aan TIME toekennen. En net als bij alle andere variabelen, kunnen we de inhoud van TIME uitlezen, afdrukken, enz. Hiervan wordt in het volgende programma gebruik gemaakt.

```
10 TIME=0
20 LOCATE 15,10
30 PRINT USING "###.#";TIME
40 GOTO 20
```

Met regel 10 zorgen we ervoor dat de inhoud van TIME nul wordt. Vervolgens drukken we met de regels 20 tot en met 30 de inhoud van de systeemvariabele TIME af. Het effect is, dat er een soort stop-watch ontstaat. Om er een echte stop-watch van te maken, zullen we nog een mogelijkheid voor het starten en stoppen van de stop-watch moeten aanbrengen. We krijgen dan het volgende programma.

```
10 SCREEN 0: WIDTH 40: CLS
20 PRINT "Druk op RETURN om te starten"
30 PRINT "Druk nog eens op RETURN om te stop
pen"
40 I$=INKEY$: IF I$<>CHR$(13)GOTO 40
50 TIME=0
60 LOCATE 18,10: PRINT USING "###.#";TIME/50
70 IF INKEY$<>CHR$(13) GOTO 60
80 LOCATE 0,10: PRINT "Verstreken tijd:";
90 LOCATE 24,10: PRINT "seconden"
100 END
```

9.2 Controle op de ingevoerde tijd

We kunnen de tijd in het programma uit paragraaf 9.1 ook invoeren als een alfanumerieke constante van 6 tekens en deze dan T$ noemen. Het INPUT-statement op regelnummer 130 moeten we dan laten vervallen en vervangen door:

```
125 INPUT "Momentele tijd (UUMMSS)";T$
```

In het vorige programma zijn er geen controles uitgevoerd op de ingevoerde tijd. Indien voor een van de tijdsvariabelen bijvoorbeeld in plaats van een cijfer een letter zou zijn ingegeven, zou het programma zijn beeindigd met een foutboodschap. Om een verkeerde invoer te voorkomen, zullen we nu de inhoud van T$ gaan controleren op het volgende:

1. Is het aantal ingegeven tekens 6?
2. Zijn alleen cijfers ingegeven?
3. Is de waarde van UU kleiner dan 24?

4. Is de waarde van MM en SS kleiner dan 60?

Met de functie **LEN(T$)** kunnen we de lengte van T$ (het aantal tekens dat in T$ staat) bepalen. Spaties en niet-afdrukbare tekens, zoals het teken voor "regelopvoer", tellen ook als een teken. Het volgende voorbeeld laat het gebruik en de werking van LEN() zien.

```
100 A$="MSX-BASIC"
110 B$="MSX-BASIC"+CHR$(13)
120 T$="123643"
130 PRINT LEN(A$)
140 PRINT LEN(B$)
150 PRINT LEN(T$)
160 END
```

Opmerking:
CHR$(13) vertegenwoordigt het teken voor "Carriage Return".

We kunnen nu met behulp van de functie LEN de lengte van de string T$ (de tijd UUMMSS) als volgt controleren:

```
126 IF LEN(T$)<>6 THEN PRINT "Lengte is geen
 6 tekens!":GOTO 125
```

We kunnen, na de controle van de juiste lengte, controleren of de ingevoerde tekens wel cijfers zijn. Deze controle wordt uitgevoerd met de **functie MID$**. De functie MID$ heeft het volgende formaat:

MID$(X$,X,Y)

Met deze functie kunnen we een bepaald gedeelte van een string afscheiden. In de functie geeft **Y** het aantal opeenvolgende tekens aan uit string **X$**, dat begint bij het **X**-de teken van de string. X mag niet kleiner zijn dan 1 en niet groter zijn dan de lengte van X$. De werking blijkt uit het volgende voorbeeld:

```
PRINT MID$("MSX-STATEMENT",5,4)
STAT
```

Het controleren of elk teken van variabele T$ wel een cijfer is, gaat als volgt:

```
127 FOR I=1 TO 6
128 C$=MID$(T$,I,1)
129 IF C$<"0" OR C$>"9" THEN PRINT "Alleen c
ijfers invoeren!":GOTO 125
130 NEXT I
```

In C$ komt steeds 1 teken van variabele T$ te staan, te beginnen bij het eerste teken. Variabele I geeft aan welk teken aan C$ wordt toegekend.

Voor het controleren van de waarden van UU, MM en SS van variabele T$ maken we onder andere gebruik van de functies LEFT$, RIGHT$ en MID$. Met de functie **LEFT$(X$,X)** kunnen we het linker gedeelte van variabele X$ afsplitsen. De grootte van het afgesplitste gedeelte wordt bepaald door de waarde van X, die niet kleiner mag zijn dan 0 en niet groter dan 255 mag zijn. Wanneer X groter is dan de lengte van X$, dan zal X$ volledig worden gegeven. Ook hiervan weer een voorbeeld:

```
PRINT LEFT$("TIJDSCONTROLE",4)
TIJD
```

We kunnen in ons programma de ingevoerde uren (UU) als volgt controleren:

```
131 U$=LEFT$(T$,2):U=VAL(U$)
132 IF U>23 THEN PRINT "Uren fout!":GOTO 125
```

Opmerking:
De twee statements op regelnummer 130 kunnen ook worden gecombineerd tot: U=VAL(LEFT$(T$,2)).

Met de functie **RIGHT$(X$,X)** kunnen we het rechter gedeelte van variabele X$ afsplitsen. De grootte van het afgesplitste

gedeelte wordt bepaald door de waarde van X, die niet kleiner dan 0 en niet groter dan 255 mag zijn. Wanneer X groter is dan de lengte van X$, dan zal X$ volledig worden gegeven. Voorbeeld:

```
PRINT RIGHT$("TIJDSCONTROLE",8)
CONTROLE
```

In ons programma kunnen we de ingevoerde seconden (SS) als volgt controleren:

```
133 S=VAL(RIGHT$(T$,2))
134 IF S>59 THEN PRINT "Seconden fout!":GOTO
 125
```

Voor het controleren van de minuten maken we gebruik van de functie MID$. De programmaregels gaan er als volgt uitzien:

```
135 M=VAL(MID$(T$,3,2))
136 IF M>59 THEN PRINT "Minuten fout!":GOTO
125
```

Wanneer het programma uit paragraaf 9.1 is aangepast met de verschillende controles, is het belangrijk deze controles goed uit te testen. Schrijf na het uittesten het programma met het commando SAVE op cassette. Noem het programma "KLOK".

Om wat meer inzicht te krijgen in de hiervoor behandelde functies, kan worden geoefend met het volgende programma. Let in dat programma ook op de OR-functies (op de regelnummers 140,170,180,190 en 220), waarmee het wordt toegestaan het antwoord in zowel hoofdletters als kleine letters te geven.

```
100 REM * LEFT$, MID$ EN RIGHT$ *
110 CLS
120 INPUT "Woord";W$
130 INPUT "Begin (L,M of R)";B$
140 IF B$="M" OR B$="m" GOTO 160
150 INPUT "Lengte";L:GOTO 170
```

```
160 INPUT "Begin en lengte (B,L)";B,L
170 IF B$="L" OR B$="l" THEN PRINT "LEFT$(";
W$;",";L;") = ";LEFT$(W$,L)
180 IF B$="R" OR B$="r"THEN PRINT "RIGHT$(";
W$;",";L;") = ";RIGHT$(W$,L)
190 IF B$="M" OR B$="m"THEN PRINT "MID$(";W$
;",";B;",";L;") = ";MID$(W$,B,L)
200 PRINT
210 INPUT "Volgend woord (J/N)";J$
220 IF J$="J" OR J$="j" GOTO 110
230 END
```

Op de functies LEN, LEFT$, RIGHT$ en MID$ wordt nog uitgebreid teruggekomen in het derde leerboek uit deze serie.

9.3 Foutafhandeling

Wanneer het statement **ON ERROR GOTO X** in een programma is gegeven, zal bij elke fout, die in een programma optreedt, naar een foutafhandelingsroutine worden gesprongen. Dit betekent, dat bij het optreden van een fout, de uitvoering van het programma niet wordt gestopt, maar dat er naar de foutafhandelingsroutine wordt gesprongen. X is de eerste regel van die routine.

Wanneer de computer het ON ERROR GOTO statement tegen komt, weet de computer vanaf dat moment waar naar toe moet worden gesprongen in het geval dat er zich een fout voordoet. Wanneer de computer een fout ontdekt, bijvoorbeeld "delen door 0", of het "toekennen van een letter aan een numerieke variabele", dan zal de computer het regelnummer waarin de fout werd gemaakt, in systeemvariabele **ERL** plaatsen en het nummer van de fout (de foutcode) in systeemvariabele **ERR**.

Elke fout heeft binnen MSX-BASIC een vast nummer. De gedefinieerde foutcodes lopen vanaf 1 (NEXT without FOR) tot en met 59 (File not OPEN). De foutcodes 26 tot en met 49 en 60 tot en met 255 zijn niet gedefinieerd en kunnen worden

gebruikt voor het definieren van eigen codes. Hierbij dient nog wel te worden opgemerkt, dat er voor het gebruik van schijven ook nog een aantal foutcodes zijn gedefinieerd. Deze codes worden pas actief op het moment dat een schijveneenheid wordt aangesloten. De codes voor schijven gaan van 60 tot en met 71. Het is daarom verstandig om alleen voor de codes boven 71 eigen definities te maken.

We zullen aan de hand van het volgende programma demonstreren wat er gebeurt, wanneer de computer een fout ontdekt (delen door nul) en er in het programma geen ON ERROR GOTO X is gegeven.

```
110 INPUT "Getal";G
120 PRINT 1/G
130 GOTO 110
RUN
Getal? 0
Division by zero in 120
Ok
```

We zien dat het programma door de computer wordt gestopt en de aard van de fout op het scherm wordt afgedrukt.

Laten we nu eens kijken wat er met hetzelfde programma gebeurt, wanneer we het uitbreiden met het statement ON ERROR GOTO. In de foutafhandelingsroutine laten we de inhouden van de systeemvariabelen ERR en ERL afdrukken.

```
100 ON ERROR GOTO 200
110 INPUT "Getal";G
120 PRINT 1/G
130 GOTO 110
200 PRINT "Foutcode";ERR;"op regelnummer";
ERL
210 GOTO 110
RUN
Getal? 0
Foutcode 11 op regelnummer 120
Getal?
```

We zien dat het programma bij het optreden van een fout nu niet wordt gestopt, maar dat er naar de foutafhandelingsroutine wordt gesprongen, welke de foutcode en het regelnummer waarin de fout is opgetreden afdrukt op het scherm.

De foutcodes met de bijbehorende foutboodschappen kunnen met het statement **ERROR X** worden gegenereerd. Wanneer X een bestaande foutcode is, zal het ERROR-statement de bij X behorende fout simuleren en de bijbehorende foutboodschap produceren. Wanneer X een foutcode is, waarvoor geen foutboodschap is gedefinieerd, zal op het scherm de boodschap "Unprintable error" verschijnen. Met het volgende programma is dit te testen.

```
100 INPUT "Welke foutcode";F
110 ERROR F
120 END
RUN
Welke foutcode? 11
Division by zero in 110
Ok
Welke foutcode? 21
No RESUME in 110
Ok
```

Opmerking:

Het programma moet voor de volgende foutcode steeds weer met het commando RUN worden gestart.

Het programma, waarin de fout werd ontdekt, kan vanuit de foutafhandelingsroutine eventueel weer verder worden gestart met het statement **RESUME.** Het statement RESUME werkt alleen, indien het wordt voorafgegaan door het statement ON ERROR GOTO. Het statement kent drie formaten:

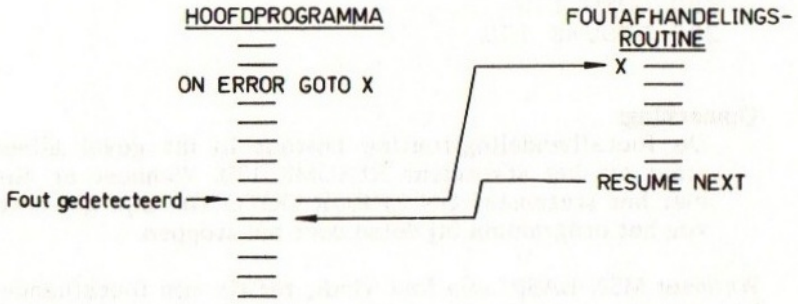
RESUME of RESUME 0

De uitvoering wordt hervat met het statement, waarbij de fout werd gedetecteerd.

RESUME NEXT

De uitvoering wordt hervat met het statement dat volgt op

het statement waarbij de fout werd gedetecteerd (zie afbeelding 9-2).



Afb. 9-2 Programma-onderbreking door het optreden van een ERROR.

RESUME X
De uitvoering wordt hervat met regelnummer X.

Voor ons demonstratieprogramma, waarbij een fout optreedt wanneer er door nul wordt gedeeld, kunnen we de foutafhandelingsroutine op twee manieren door RESUME laten eindigen, namelijk:

```
210 RESUME NEXT of
210 RESUME 110
```

Met het volgende programma wordt de numerieke constante G gedeeld door alle gehele getallen uit het interval +4 tot en met -4. De computer zal voor het getal 0 uit het interval een fout detecteren. Deze fout dient te worden opgevangen, hetgeen als volgt geschiedt:

```
100 ON ERROR GOTO 200
110 CLS
120 INPUT "Getal";G
130 PRINT
140 FOR I=4 TO -4 STEP -1
150 S=G/I
```

```
160 PRINT G;":";I;"=";S
170 NEXT I
180 A$=INKEY$:IF A$="" GOTO 180
190 GOTO 110
200 RESUME 170
```

Opmerking:

De foutafhandelingsroutine bestaat in dit geval alleen maar uit het statement RESUME 170. Wanneer er niet met het statement ON ERROR GOTO zou zijn gewerkt, zou het programma bij delen door nul stoppen.

Wanneer MSX-BASIC een fout vindt, terwijl een foutafhandelingsroutine wordt uitgevoerd, wordt de normale foutboodschap gegeven. Wanneer het statement ON ERROR GOTO 0 in een programma of foutafhandelingsroutine is gegeven, zal MSX-BASIC de laatst gedetecteerde fout alsnog op het scherm afdrukken en vervolgens het programma stoppen.

In het volgende programma zal naar de foutafhandelingsroutine worden gesprongen, indien er met het INPUT-statement een groter getal dan 24 wordt ingevoerd. Op regel 120 zal in dat geval met het statement ERROR X de foutcode 102 worden gegenereerd (ERR=102). Indien met het statement op regel 130 het cijfer 1 door nul wordt gedeeld zal de foutcode 11 worden gegenereerd (ERR=11). Indien ERR=11, dan wordt de foutboodschap alsnog afgedrukt en het programma gestopt. Indien ERR=102 (zelf gedefinieerde foutcode), dan wordt de boodschap "Getal kleiner dan 25!" afgedrukt en wordt het programma hervat op regelnummer 110.

```
100 ON ERROR GOTO 500
110 INPUT "Getal";G
120 IF G>24 THEN ERROR 102
130 PRINT 1/G
140 PRINT
150 GOTO 110
500 IF ERR=11 THEN ON ERROR GOTO 0
510 IF ERR=102 THEN PRINT "Getal kleiner dan
```

```
 25!"
520 RESUME 110
```

We zullen het onderwerp foutafhandeling afsluiten met het aanpassen van ons programma "KLOK", dat we in paragraaf 9.2 op cassette hebben gezet. We zullen voor de fouten, die in het programma kunnen optreden, de volgende foutcodes uit het niet gedefinieerde gebied reserveren:

ERROR 106 - Lengte is geen 6 tekens!
ERROR 107 - Alleen cijfers invoeren!
ERROR 108 - Uren fout!
ERROR 109 - Seconden fout!
ERROR 110 - Minuten fout!

Regelnummer 126 zal dan als volgt moeten worden veranderd:

```
126 IF LEN(T$)<>6 THEN ERROR 106
```

Wanneer de lengte van T$ niet gelijk is aan 6, zal naar de foutafhandelingsroutine worden gesprongen. Het eerste regelnummer van de routine staat vermeld in het ON ERROR GOTO statement. Wanneer ERROR 106 wordt uitgevoerd, krijgen de systeemvariabelen de volgende waarden toegekend: ERR=106 en ERL=126.

Laten we eens aannemen, dat we de routine op regelnummer 200 laten beginnen en het ON ERROR GOTO statement op regelnummer 115 plaatsen. Het statement ziet er dan als volgt uit:

```
115 ON ERROR GOTO 200
```

Verder zullen in het programma nog de volgende regels moeten worden veranderd:

```
129 IF C$<"0" OR C$>"9" THEN ERROR 107
132 IF U>23 THEN ERROR 108
134 IF S>59 THEN ERROR 109
136 IF M>59 THEN ERROR 110
```

De foutafhandelingsroutine gaat er nu als volgt uitzien:

```
200 IF ERR=106 THEN PRINT "Lengte is geen 6
tekens!":GOTO 250
210 IF ERR=107 THEN PRINT "Alleen cijfers in
voeren!":GOTO 250
220 IF ERR=108 THEN PRINT "Uren fout!":GOTO
250
230 IF ERR=109 THEN PRINT "Seconden fout!":G
OTO 250
240 IF ERR=110 THEN PRINT "Minuten fout!"
250 RESUME 125
```

9.4 Foutzoeken in programma's

Als een programma op papier staat, is er nog geen zekerheid omtrent de goede werking van het programma. Nadat het programma is ingevoerd, begint het spannendste gedeelte: het testen of het programma werkt. In geval van programmafouten zul je het programma stap voor stap moeten nakijken.

Het commando **TRON** (Trace On) is een zeer nuttig instrument om functionele fouten in een programma op te zoeken. Wanneer het commando TRON is gegeven, is het mogelijk om tijdens de uitvoering van een programma de stappen binnen het programma op de voet te volgen. Men is zodoende in de gelegenheid het programma op fouten te bekijken. Nadat TRON is ingegeven, zal MSX-BASIC tijdens de uitvoering van een programma alle regelnummers in volgorde van uitvoering tussen vierkante haken op het scherm afdrukken.

Voer, voor het testen van het commando TRON, het volgende programma in:

```
100 CLS
110 INPUT "Deeltal";D
120 FOR I=2 TO 4
130 PRINT D;":";I;"=";D/I
140 NEXT I
150 PRINT
160 INPUT "Volgend deeltal (J/N)";J$
```

```
170 IF J$="J" GOTO 100 ELSE END
RUN
[110]Deeltal? 6
[120][130] 6 : 2 = 3
[140][130] 6 : 3 = 2
[140][130] 6 : 4 = 1.5
[140][150]
[160]Volgend deeltal (J/N)?
```

Nadat het programma is ingevoerd, geven we het commando TRON en vervolgens starten we het programma met het commando RUN. Om TRON weer uit te schakelen, maken we gebruik van het commando **TROFF** (Trace Off). Het commando NEW schakelt "Trace" ook uit, doch NEW maakt tevens het geheugen schoon.

Opmerking:

TRON en TROFF kunnen het beste worden gebruikt in de directe (commando) mode.

9.5 Het genereren van een piepsignaal.

Het is soms, om de aandacht te trekken, nodig een attentiesignaal te geven. Een goed voorbeeld hiervan is, het geven van een kort piepsignaal, wanneer iets via het toetsenbord moet worden ingevoerd (INPUT-statement). Het statement, dat voor het opwekken van het piepsignaal zorgt, is **BEEP.** Hetzelfde effect als door BEEP verkregen, verkrijgt men ook door PRINT CHR$(7) en het tegelijkertijd indrukken van de toetsen CTRL en G.

Om het effect van het statement BEEP te demonstreren, kunnen we aan het programma uit paragraaf 9.4 de volgende regel toevoegen:

```
105 BEEP
```

Wanneer we het signaal te kort vinden, kunnen we het eventueel als volgt verlengen:

```
105 FOR I=1 TO 5
106 BEEP
107 NEXT I
```

Opmerking:
Wanneer een TV-toestel als beeldscherm wordt gebruikt, moet er wel voor worden gezorgd dat het volume niet helemaal dichtgedraaid is. Het piepsignaal komt namelijk via de luidspreker van het TV-toestel.

9.6 Het stoppen en vervolgen van een programma.

Om een programma tijdens de uitvoering ervan te stoppen, kunnen we in het programma het statement **STOP** opnemen. Wanneer STOP door de computer wordt uitgevoerd, wordt de volgende boodschap op het beeldscherm afgedrukt:

Break in **regelnummer**

Het statement STOP mag overal in een programma worden geplaatst om de uitvoering te beeindigen. Het statement STOP kan worden gebruikt voor het in gedeelten testen van een programma. De uitvoering kan namelijk weer worden hervat door het geven van het commando **CONT.** Het commando CONT zit ook onder functietoets F8.

We kunnen het programma uit paragraaf 9.4 weer als voorbeeld nemen, door de volgende regel aan het programma toe te voegen:

```
155 STOP
```

Na het toevoegen van de regel kunnen we het programma starten met RUN. We zullen dan zien dat het programma gedeeltelijk wordt uitgevoerd en op het scherm de boodschap:

```
Break in 155
Ok
```

wordt afgedrukt. Door vervolgens het commando CONT te geven, zal de uitvoering van het programma weer worden hervat.

De volgende tabel geeft het verschil aan tussen wat er gebeurt na het geven van het commando CONT, wanneer het programma werd gestopt door het statement STOP en door het indrukken van de toetsen CTRL en STOP.

Stoppen	Hervatten	Uitwerking
STOP	CONT	Uitvoering wordt weer voortgezet met de volgende programmaregel.
CTRL+STOP	CONT	Uitvoering wordt weer voortgezet met het statement, waar het programma werd onderbroken.

10 Het programmeren van functietoetsen

10.1 Functietoetsen F1 t/m F10.

Bij het inschakelen van de MSX-computer krijgen de 10 functietoetsen F1 t/m F10 van te voren vastgestelde inhouden toegekend. Deze inhouden staan afgedrukt op de laatste regel van het beeldscherm (regel 24). Daar er op deze regel maar vijf inhouden tegelijk kunnen worden afgedrukt, worden alleen de inhouden van de eerste vijf functietoetsen afgedrukt. Om de inhouden van de tweede vijf functietoetsen te zien, moeten we de SHIFT-toets indrukken. Zolang er op de laatste regel van het beeldscherm inhouden van functietoetsen worden afgedrukt, kan die laatste regel niet voor andere doeleinden worden gebruikt.

Tikt u, voor een verdere uitleg van de functietoetsen, eerst het volgende programma in:

```
10 PRINT "Met functietoets F5"
20 PRINT "kunnen we een programma"
30 PRINT "starten."
```

Wanneer we nu functietoets F5 (run) indrukken, wordt op het scherm "run" afgedrukt en wordt vervolgens het voorgaande programma uigevoerd.

Wordt functietoets F4 ingedrukt, dan wordt op het scherm "list" afgedrukt. Door nu de RETURN-toets in te drukken, wordt op het scherm de listing van het in het geheugen aanwezige programma afgedrukt.

Met **KEY LIST** kunnen we een volledige lijst met inhouden van de 10 functietoetsen op het scherm afdrukken:

```
KEY LIST
color
auto
goto
list
run
color 15,4,4
cload"
cont
list.
 run
```

Bovenstaande lijst kan ook met behulp van het volgende programma worden gemaakt:

```
10 CLS
20 PRINT "OVERZICHT FUNCTIETOETSEN"
30 PRINT
40 KEY LIST
50 END
```

Wanneer functietoets F10 (SHIFT + F5/F10) wordt ingedrukt, wordt eerst het scherm schoongemaakt, daarna wordt het programma gestart met "run". In de lijst van functietoetsen ziet u dat er voor "run" een spatie staat. In werkelijkheid staat hier een niet afdrukbaar teken. Dat teken komt overeen met de code CHR$(12). CHR$(12) is de code voor het commando CLS (CLear Screen). Achter het woordje "run" staat nog een code, die niet zichtbaar is. Dit is de code CHR$(13), die overeenkomt met RETURN. Bij het uitvoeren van functietoets 10 zal dus achtereenvolgens het scherm schoongemaakt worden, het in het geheugen staande programma worden geselecteerd voor uitvoering, en door de RETURN-code zal dat programma ook direct worden gestart. Omdat eventuele controle-codes niet worden afgedrukt, volgt hier een nieuwe lijst van functietoetsen, echter nu compleet met controle-codes.

```
F1  color[spatie]
F2  auto[spatie]
F3  goto[spatie]
```

```
F4  list[spatie]
F5  run[RETURN]
F6  color 15,4,4[RETURN]
F7  cload"
F8  cont[RETURN]
F9  list.[RETURN][u][u] *
F10 [cls]run[RETURN]
```

* u = cursor 1 positie omhoog.

Met functietoets F9 wordt het **LIST.**-commando gegeven. De punt achter LIST geeft aan dat alleen de laatst geschreven of gewijzigde BASIC-regel moet worden ge-LIST. Doordat er na het LIST-commando twee keer een code wordt gegeven waarmee de cursor een regel omhoog gaat, zal, na het uitvoeren van de functie (F9), de cursor op de laatst ingegeven of gewijzigde regel staan. Met behulp van de "full screen editor" kunt u die regel dan nogmaals wijzigen.

Het is ook mogelijk om de inhouden van een aantal functietoetsen aan te vullen. Drukt u bijvoorbeeld functietoets F4 in, dan verschijnt op het scherm "list" met een spatie. Tikt u nu de cijfers 2 en 0 in, gevolgd door de RETURN-toets, dan wordt alleen regelnummer 20 van het programma afgedrukt.

Met **KEY OFF** kunnen we de afgedrukte definities van de functietoetsen, op regel 24, weghalen. Hierdoor komt deze regel vrij voor andere toepassingen. Willen we de inhouden van de functietoetsen weer laten verschijnen, dan geven we de opdracht **KEY ON.** KEY OFF en KEY ON kunnen ook als statements in een programma worden gebruikt. Probeer dit maar eens, zowel in de directe mode, als in de programma mode.

10.2 Veranderen van de inhoud van functietoetsen.

Nadat we hebben gezien welke functies er standaard aan de functietoetsen zijn toegekend, zullen we nu eens nagaan, hoe we daar zelf nieuwe functies aan kunnen toekennen. We zullen eens kijken hoe we met functietoets F3 het beeldscherm

kunnen laten schoonmaken.

We zullen dan eerst de inhoud van functietoets F3 (goto) moeten veranderen. De inhoud van een functietoets kan met het volgende statement worden veranderd:

KEY functietoetsnummer , string-uitdrukking

Het nummer van de functietoets moet een positief geheel getal van 1 tot en met 10 zijn. Is het geen geheel getal, dan zal de computer de cijfers achter de komma verwaarlozen.

De string-uitdrukking mag uit maximaal 15 tekens bestaan en dient voor het definieren van de nieuwe inhoud van een functietoets. Om in functietoets F3 de functie van het schoonmaken van het beeldscherm te zetten, moeten we het volgende statement gebruiken:

```
KEY 3,"CLS"+CHR$(13)
```

Hierin stelt CHR$(13) de functie RETURN voor. We kunnen voor CLS ook de controle-code CHR$(12) gebruiken. Dat heeft echter tot nadeel dat er geen enkele tekst op de laatste regel van het beeldscherm verschijnt, en dus, dat we niet weten wat de functie van die toets is. Willen we met 1 functietoets achtereenvolgens het beeldscherm schoonmaken en een "listing" maken, dan kunnen we die toets de volgende inhoud geven:

```
KEY 3,CHR$(12)+CHR$(13)+"list"+CHR$(13)
```

CHR$(12) is hierin de controle-code voor het schoonmaken van het scherm (CLS). In de string-uitdrukking vertegenwoordigen CHR$(12) en CHR$(13) ieder slechts 1 teken.

De inhoud van de functietoets zou ook door middel van een programma kunnen worden veranderd. Het programma kan er dan als volgt uit zien:

```
10 A$="LIST"
20 KEY 3,CHR$(12)+CHR$(13)+A$+CHR$(13)
```

30 END

10.3 Onderbrekingen d.m.v. functietoetsen

Functietoetsen kunnen ook worden gebruikt voor het onderbreken van programma's. Het is hierbij de bedoeling, dat bij het indrukken van een functietoets het programma dat in uitvoering is, tijdelijk wordt onderbroken en dat er een subroutine, die betrekking heeft op de ingedrukte functietoets, wordt aangeroepen. Een onderbreking wordt pas in bewerking genomen, indien het statement dat in uitvoering is op het moment dat de onderbreking plaatsvindt, volledig is afgewerkt. Een onderbreking wordt ook wel gezien als een niet geprogrammeerde sprong.

Het statement RETURN aan het einde van de aangeroepen subroutine zorgt ervoor, dat het programma de "draad" weer oppakt waar het eerder, tijdens het indrukken van de functietoets, werd onderbroken. Een onderbreking vindt niet plaats wanneer de computer niet bezig is met het uitvoeren van een programma.

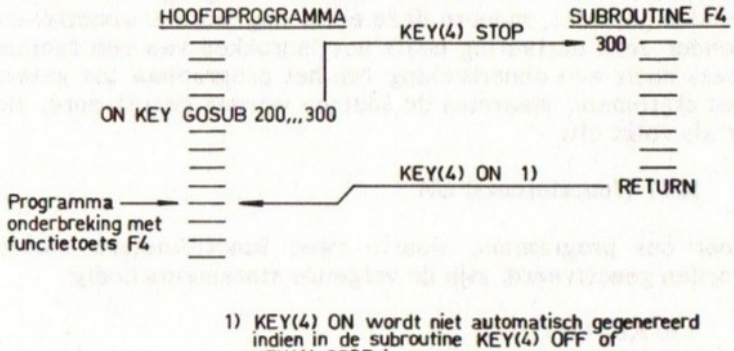
Met behulp van het statement

ON KEY GOSUB lijst met regelnummers

wordt de mogelijkheid geschapen om bij het indrukken van een functietoets (KEY) naar een subroutine op een bepaald regelnummer (uit de lijst) te springen. Dit statement kunnen we ergens aan het begin van het programma zetten. Zodra dit statement is uitgevoerd weet de computer dat, indien er op een functietoets wordt gedrukt, er mogelijkerwijs naar een subroutine moet worden gesprongen. Het indrukken van de functietoets kan echter op iedere andere plaats in het programma gebeuren. Afbeelding 10-1 geeft een grafische weergave van de gebeurtenissen.

Voor een verdere uitleg van onderbrekingen door middel van functietoetsen gaan we met functietoets F1 een subroutine (die op regel 200 begint) aanroepen, waarmee het renteper-

centage op een persoonlijke lening wordt berekend. Met functietoets F4 roepen we een subroutine (die op regel 350 begint) aan, waarmee de annuiteit wordt berekend voor een hypotheek, waarvan het kapitaal, het rentepercentage en de looptijd moeten worden ingegeven.



Afb. 10-1 Programma-onderbreking door functietoetsen.

De inhoud van de functietoetsen F1 en F4 wordt als volgt veranderd:

```
110 KEY 1,"RENTE"
120 KEY 4,"LENING"
```

Het ON KEY GOSUB statement gaat er voor ons programma nu als volgt uitzien:

```
150 ON KEY GOSUB 200,,,350
```

Het aantal regelnummers achter GOSUB mag maximaal 10 zijn. De in ons programma opgenomen regelnummers corresponderen met de functietoetsen F1 en F4. Daar de functietoetsen F2 en F3, die tussen F1 en F4 liggen, niet voor onderbreking worden gebruikt, zijn die plaatsen leeg. De twee extra komma's tussen 200 en 350 mogen niet worden vergeten, daar anders de subroutine op regelnummer 350 wordt aangeroepen bij het indrukken van functietoets F2.

Wanneer voor een functietoets in het ON KEY GOSUB statement geen regelnummer is gespecificeerd, dan maakt het niet uit of deze functietoets al dan niet wordt ingedrukt; een onderbreking zal in dat geval nooit plaatsvinden.

Voordat de functietoetsen F1 en F4 in ons programma kunnen worden gebruikt, moeten deze eerst nog worden geactiveerd. Zonder deze activering heeft het indrukken van een functietoets nooit een onderbreking van het programma tot gevolg. Het statement, waarmee de toetsen worden geactiveerd, ziet er als volgt uit:

KEY (functietoets) ON

Voor ons programma, waarin twee functietoetsen moeten worden geactiveerd, zijn de volgende statements nodig:

```
130 KEY(1) ON
140 KEY(4) ON
```

Het hoofdprogramma gaat er nu als volgt uit zien:

```
100 REM * VOORBEELD ONDERBREKINGEN *
110 KEY 1,"RENTE"
120 KEY 4,"LENING"
130 KEY(1) ON
140 KEY(4) ON
150 ON KEY GOSUB 200,,,350
160 CLS
170 PRINT "FUNCTIETOETSEN F1 EN F4"
180 PRINT "KUNNEN WORDEN GEACTIVEERD."
190 GOTO 190
```

De subroutine, die bij functietoets F1 hoort ziet er als volgt uit:

```
200 CLS
210 INPUT "TE LENEN BEDRAG";B
220 INPUT "AANTAL AFLOSSINGEN";A
230 INPUT "AFLOSSINGSBEDRAG";C
240 S=A*C
```

```
250 D=(1+A)/2
260 R=S-B
270 K=R/D*12
280 P=K/(B/100)
290 PRINT
300 PRINT "BETAALD BEDRAG  =";S
310 PRINT "RENTE + KOSTEN  =";R
320 PRINT "RENTEPERCENTAGE =";P;"%"
330 RETURN
```

De subroutine, die bij functietoets F4 hoort, ziet er als volgt uit:

```
350 CLS
360 INPUT "KAPITAAL";K
370 INPUT "RENTEPERCENTAGE";R
380 INPUT "LOOPTIJD IN JAREN";L
390 X=(1+R/100)^L
400 S=(K*R/100)/(1-1/X)
410 PRINT
420 PRINT "ANNUITEIT/JAAR  =";S
430 RETURN
```

Als u dit programma hebt ingetikt en uitgevoerd, zullen we nog even kijken wat er nu precies gebeurt. Wanneer functietoets F4 wordt ingedrukt (zie ook afbeelding 10-1), heeft dit automatisch het aanroepen van de bijbehorende subroutine (die op regel 350 begint) tot gevolg. Verder zal door het indrukken van de functietoets F4 ook automatisch KEY(4) STOP worden gegeven. Dit betekent, dat het opnieuw indrukken van functietoets F4, tijdens het uitvoeren van de subroutine, geen effect zal hebben, zolang er geen KEY(4) ON wordt gegeven.

Aan het einde van de subroutine keert het programma door middel van het statement RETURN weer terug naar het punt waar het eerder, door het indrukken van F4, werd onderbroken. Tijdens de uitvoering van het RETURN-statement wordt automatisch KEY(4) ON gegeven. Dit gebeurt echter niet, wanneer in de subroutine het statement KEY(4) OFF of

KEY(4) STOP werd uitgevoerd.

Bij het uitvoeren van het programma zal het u zijn opgevallen, dat behalve de inhoud van de functietoetsen F1 en F4, ook de inhouden van de andere functietoetsen nog op het beeldscherm komen te staan. Wilt u die teksten niet meer op het scherm hebben, dan kunt u voor ieder van die teksten het volgende statement gebruiken:

KEY functietoetsnummer,""

Tussen de aanhalingstekens staat niets. Dit houdt in, dat er ook niets op het beeldscherm zal worden afgedrukt. In ons voorbeeld is het op die manier niet erg duidelijk dat de tweede tekst bij functietoets F4 hoort. We zouden dan ook, in plaats van niets, een cijfer (het nummer van de functietoets) tussen de aanhalingstekens kunnen zetten. Dat cijfer wordt dan wel op het beeldscherm afgedrukt.

10.4 KEY(functietoets) ON/OFF/STOP

Na uitvoering van het statement **KEY(functietoets) ON** zal MSX-BASIC na uitvoering van elk statement controleren of de gegeven functietoets is ingedrukt. Indien dit zo is, zal het programma worden onderbroken en zal de bij de functietoets behorende subroutine worden aangeroepen. Deze subroutine is gegeven in het ON KEY GOSUB statement.

Wanneer de mogelijkheid tot onderbreken van het programma voor een bepaalde functietoets dient te worden opgeheven, kan dit worden bewerkstelligd met het statement **KEY(functietoets) OFF.** MSX-BASIC zal dan niet meer controleren of de betreffende functietoets is ingedrukt. Om alle functietoetsen te de-activeren kan een FOR . . . NEXT lus worden gebruikt:

FOR N=1 TO 10: KEY(N) OFF: NEXT N

Hetzelfde effect kan worden verkregen, door het statement ON KEY GOSUB ,,,,,,,,, (negen komma's) te geven. Dit is

echter een oneigenlijk gebruik van dit statement.

Na het beeindigen van het programma wordt automatisch voor elke functietoets een KEY(n) OFF gegeven.

Wanneer het in een bepaald gedeelte van het programma niet wenselijk is dat dit wordt onderbroken, maar dat het indrukken van een functietoets wel moet worden onthouden, dan kan dit worden geregeld met het statement **KEY(functietoets) STOP.** Wordt de betreffende functietoets ingedrukt, dan gebeurt er ogenschijnlijk niets. Zodra echter de betreffende functietoets weer wordt geactiveerd (met het KEY(n) ON statement), wordt het programma alsnog onderbroken en wordt de bij de functietoets behorende subroutine uitgevoerd.

Verander het functietoetsenprogramma (hoofdprogramma) nu eens als volgt:

```
190 KEY(4) STOP
192 FOR I=1 TO 8000
194 NEXT I
196 KEY(4) ON
198 GOTO 198
```

Start het programma en druk functietoets F4 even in. Wacht nu rustig af wat er gebeurt. Op regel 192 en 194 is een vertraging ingebouwd. Ook al ziet de computer wel dat de functietoets F4 is ingedrukt, hij kan dat pas laten merken nadat regel 196 is uitgevoerd. En dat duurt enige tijd.

10.5 Wat doet STOP voor ons?

Wanneer de computer een programma voor ons uitvoert, zal door het indrukken van de **STOP-toets** het programma tijdelijk worden onderbroken. Wordt de STOP-toets hierna weer ingedrukt, dan zal de uitvoering van het programma worden voortgezet. Wanneer de **CTRL-toets samen met de STOP-toets** wordt ingedrukt, dan zal de computer de uitvoering van het programma beeindigen en terugkeren naar de directe mode. De computer zal dan de volgende boodschap geven:

Break in regelnummer

Het **regelnummer** geeft de regel aan waarop de uitvoering is gestopt. Wanneer STOP wordt gebruikt als statement in een programma, dan zal de uitvoering van het programma worden beeindigd en zal MSX-BASIC terugkeren naar de directe mode. De computer zal dan dezelfde boodschap geven als na CTRL+STOP. Het programma kan echter worden hervat met de opdracht CONT. Het volgende voorbeeld illustreert het gebruik van het statement STOP en de opdracht CONT.

```
100 PRINT "Dit is een STOP-test"
110 PRINT "STOP op regelnummer 120"
120 STOP
130 PRINT "Dit is een END-test"
140 PRINT "END op regelnummer 150"
150 END
```

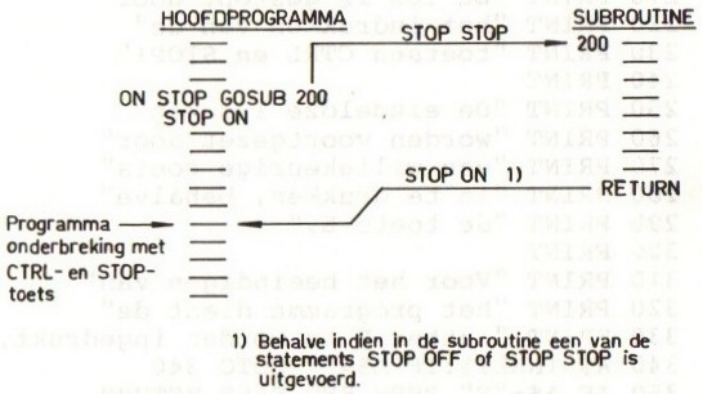
Voer dit programma in en start het. Nadat het is gestopt (door de uitvoering van regel 120) kunt u het programma herstarten met het CONT-commando.

10.6 ON STOP GOSUB

Met het statement **ON STOP GOSUB regelnummer** is het mogelijk om een programma te onderbreken en vervolgens een subroutine aan te roepen. De onderbreking vindt plaats door tegelijkertijd de CTRL-toets en de STOP-toets in te drukken. Het regelnummer waar de subroutine start staat aangegeven achter GOSUB. Voordat een programma door middel van de CTRL- en STOP-toets kan worden onderbroken, moet de STOP-toets eerst worden geactiveerd. Dit doen we met het statement **STOP ON.**

Nadat het STOP ON statement is uitgevoerd, zal MSX-BASIC na uitvoering van elk statement controleren of de CTRL- en STOP-toets tegelijkertijd zijn ingedrukt. Indien dit zo is, zal het programma tijdelijk worden gestopt en zal de subroutine worden aangeroepen. Deze subroutine staat achter het ON STOP GOSUB gespecificeerd. Verder zal ook automatisch

STOP STOP worden geactiveerd. Hierdoor kan tijdens het uitvoeren van de subroutine, deze door het indrukken van de CTRL- en STOP-toets niet nogmaals worden aangeroepen. Een en ander is weergegeven in afbeelding 10-2.



Afb. 10-2 Programma-onderbreking door CTRL- en STOP-toets.

Na het uitvoeren van STOP STOP zal het indrukken van de CTRL- en STOP-toets niet direct worden gehonoreerd. Het indrukken daarvan zal door de computer worden onthouden, zonder dat er direct actie op volgt. Pas nadat de subroutine is uitgevoerd, wordt het ingedrukt geweest zijn van de CTRL- en STOP-toets gehonoreerd, tenzij in de subroutine een van de statements STOP OFF of STOP STOP is uitgevoerd.

Na het statement **STOP OFF** zal MSX-BASIC niet langer meer controleren of de toetsen CTRL en STOP tegelijkertijd zijn ingedrukt. Wanneer een programma is beeindigd, wordt automatisch een STOP OFF gegeven. Het volgende programma illustreert het gebruik van het ON STOP GOSUB statement.

```
100 REM * ONDERBREKING D.M.V. DE *
110 REM *  TOETSEN CTRL EN STOP  *
120 KEY OFF
```

```
130 ON STOP GOSUB 200
140 STOP ON
150 PRINT "Test met ON STOP GOSUB X"
160 GOTO 150
200 PRINT
210 PRINT "De lus is gestopt door"
220 PRINT "het indrukken van de"
230 PRINT "toetsen CTRL en STOP!"
240 PRINT
250 PRINT "De eindeloze lus kan"
260 PRINT "worden voortgezet door"
270 PRINT "een willekeurige toets"
280 PRINT "in te drukken, behalve"
290 PRINT "de toets E."
300 PRINT
310 PRINT "Voor het beeindigen van"
320 PRINT "het programma dient de"
330 PRINT "letter E te worden ingedrukt."
340 A$=INKEY$:IF A$="" GOTO 340
350 IF A$="E" THEN END ELSE RETURN
```

Voer het programma in, start het en druk op de CTRL- en STOP-toets. U ziet dan wat het resultaat van het programma is.

11 Werken met vaste gegevens

Alvorens we beginnen met de uitleg van de statements DATA, READ en RESTORE zullen we eerst een voorbeeld geven.

Wanneer we in een programma vaste gegevens (bijvoorbeeld de dagen van de week) gebruiken, kunnen we deze vastleggen in een DATA-statement. De inhoud van het DATA-statement lezen we met een READ-statement. Laten we aannemen, dat we de gemiddelde neerslag willen berekenen over een week en voor het invoeren van de neerslag per dag een voor een de dagen van de week op het scherm willen afdrukken. Hiertoe kunnen we de dagen van de week in twee DATA-statements plaatsen, die dan door een READ-statement een voor een kunnen worden gelezen. DATA-statements worden ook wel DATA-regels genoemd.

Wanneer het programma wordt gestart, wordt bij het uitvoeren van het eerste READ-statement het eerste element uit de eerste DATA-regel gelezen. Tijdens het uitvoeren van een READ-statement wordt de pointer (aanwijzer), die het te lezen element aanwijst, automatisch naar het volgende element in de DATA-regel verplaatst. Wanneer alle elementen in een DATA-regel aan bod zijn geweest, wordt de pointer automatisch verplaatst naar het eerste element van de volgende DATA-regel. De DATA-regels mogen overal in het programma staan.

Wanneer men de pointer weer wil terugzetten naar het eerste element van de eerste DATA-regel, kan dit door het uitvoeren van een RESTORE-statement. In het volgende programma, dat dient voor het berekenen van de gemiddelde neerslag over een week, is gebruik gemaakt van de READ-, DATA- en RESTORE-statements.

```
100 WIDTH 40
110 CLS
120 R=0
130 PRINT "Invoeren neerslag in mm."
140 PRINT
150 FOR I=1 TO 7
160 READ D$
170 PRINT "Neerslag ";D$;TAB(18);
180 INPUT N
190 R=R+N
200 NEXT I
210 GT=R/7
220 PRINT
230 PRINT "Gemiddelde neerslag is";
240 PRINT USING "##.##";GT;
250 PRINT " mm"
260 PRINT
270 INPUT "Volgende berekening (J/N)";J$
280 IF J$="J" THEN RESTORE:GOTO 110
290 END
300 DATA zondag,maandag,dinsdag,woensdag
310 DATA donderdag,vrijdag,zaterdag
```

11.1 READ, DATA en RESTORE

Met het statement READ kunnen we alfanumerieke- en numerieke constanten lezen van een DATA-regel. De gelezen informatie wordt toegekend aan alfanumerieke- en numerieke variabelen. Wanneer een READ-statement wordt uitgevoerd, wordt er informatie gelezen van een DATA-regel. Een READ-statement wordt altijd in combinatie met een DATA-statement gebruikt. Het formaat van een READ-statement is:

READ lijst met variabelen

In de **lijst met variabelen** mogen zowel alfanumerieke- als numerieke variabelen worden geplaatst. De variabelen moeten van elkaar worden gescheiden door komma's. Enkele voorbeelden van READ-statements zijn:

```
READ A$
READ A$,B$,C$
READ X
READ X,Y
READ X,A$,B$
```

Het DATA-statement heeft het volgende formaat:

DATA lijst met constanten

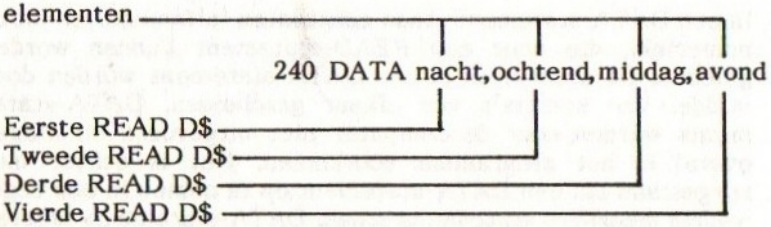
In een DATA-statement staan **constanten** (alfanumeriek en/of numeriek), die door een READ-statement kunnen worden gelezen. De elementen in een DATA-statement worden door middel van komma's van elkaar gescheiden. DATA-statements worden door de computer niet uitgevoerd en mogen overal in het programma voorkomen. Het is echter niet toegestaan om een DATA-statement op te nemen in een regel waarin meerdere statements staan. DATA-statements moeten altijd alleen in een regel staan. De gegevens, die uit een DATA-regel worden gelezen, moeten overeenkomen met de gespecificeerde variabelen in het READ-statement. Wanneer in een READ-statement een numerieke variabele staat, dan moet het eerstvolgende te lezen element uit de DATA-regel een numerieke constante zijn. Is dit niet het geval, dan wordt de foutmelding "Syntax error" gegeven.

Met het volgende programma worden door het READ-statement een aantal alfanumerieke constanten (strings) van een DATA-regel gelezen.

```
100 CLS
110 R=0
120 PRINT "Neerslag per dag."
130 PRINT
140 FOR I=1 TO 4
150 READ D$
160 PRINT "Neerslag ";D$;TAB(17);
170 INPUT N
180 R=R+N
190 NEXT I
200 PRINT
```

```
210 PRINT "Neerslag is ";
220 PRINT USING "##.##";R;
230 PRINT " mm"
240 DATA nacht,ochtend,middag,avond
250 END
```

Elke keer dat het READ-statement op regelnummer 150 wordt uitgevoerd, zal de computer de pointer (aanwijzer) verplaatsen naar het volgende element in de DATA-regel.



Wanneer het laatste element is bereikt, moet de pointer worden teruggezet naar het eerste element in de DATA-regel. Hiertoe dient het statement RESTORE. Met het statement RESTORE wordt de pointer naar het eerste element in de DATA-regel teruggezet. Dit houdt in, dat DATA-regels meerdere malen kunnen worden gebruikt.

Wanneer we na het lezen van het laatste element uit een DATA-regel weer een READ-statement uitvoeren, zonder een RESTORE te hebben gegeven, krijgen we de foutmelding "Out of DATA". We kunnen dit demonstreren door aan het vorige programma de volgende twee regels toe te voegen en het programma tweemaal uit te voeren.

```
235 A$=INKEY$: IF A$="" GOTO 235
237 GOTO 100
```

Wanneer het programma de eerste keer is uitgevoerd en we daarna een willekeurige toets hebben ingedrukt, krijgen we de boodschap "Out of DATA in 150". We kunnen dit probleem oplossen door aan het programma de volgende regel toe te voegen:

236 RESTORE

Hierna volgt nog een programma waarin steeds twee elementen tegelijkertijd van een DATA-regel worden gelezen.

```
100 CLS
110 FOR I=1 TO 2
120 READ X,Y
130 PRINT X;"x";Y;"=";X*Y
140 NEXT I
150 DATA 123,6,-12,29
160 PRINT
170 END
```

In een programma mogen meerdere DATA-regels worden geplaatst, terwijl er slechts een READ-statement in het programma voorkomt. Wanneer in dit geval alle data-elementen van een DATA-regel zijn gelezen, gaat de computer verder met de data-elementen van de eerstvolgende DATA-regel. De DATA-regels mogen op verschillende plaatsen in het programma stan. De computer handelt de DATA-regels af in volgorde van regelnummer, te beginnen met de DATA-regel met het laagste regelnummer. De voorkeur gaat uit naar het plaatsen van DATA-regels aan het einde van het programma.

Dan volgt nu een voorbeeld, waarin meerdere DATA-regels voorkomen en slechts een READ-statement.

```
100 CLS
110 PRINT "PRIORITEIT BEWERKINGEN"
120 PRINT
130 FOR I=1 TO 5
140 READ N,D$
150 PRINT D$;TAB(17);N
160 PRINT
170 NEXT I
180 PRINT
190 DATA 1,machtsverheffen,2,vermenigvuld
igen
200 DATA 2,delen,3,optellen,3,aftrekken
210 END
```

Ook het tegenovergestelde is mogelijk, namelijk meerdere READ-statements en slechts een DATA-regel:

```
100 CLS
110 FOR I=1 TO 2
120 READ X,Y
130 PRINT X*Y;
140 READ Z
150 PRINT X*Y+Z
160 NEXT I
170 DATA 6,-9,8,23,-7,23
180 END
```

Wanneer meerdere DATA-regels in een programma staan, zal bij het uitvoeren van het RESTORE-statement de pointer naar het eerste data-element van de eerste DATA-regel worden verplaatst. Bij MSX-BASIC is het mogelijk de pointer naar het begin van een bepaalde DATA-regel terug te zetten. Hiertoe wordt dan aan het RESTORE-statement een regelnummer toegevoegd (bijvoorbeeld RESTORE 150). Tijdens het uitvoeren van RESTORE 150 wordt de pointer teruggezet naar het eerste data-element op regel 150.

Strings kunnen in DATA-regels al dan niet tussen aanhalingstekens worden gezet. Wanneer de string (alfanumerieke constante) in een DATA-regel een komma of dubbele punt bevat, moet de string tussen aanhalingstekens worden gezet. Bovendien moeten strings tussen aanhalingstekens worden gezet, indien men bewust spaties in de string wil opnemen. Wanneer men in een DATA-regel spaties plaatst voor een string en men plaatst het geheel niet tussen aanhalingstekens, dan worden de spaties door de computer genegeerd (zie regelnummer 190 van het hiernavolgende programma). In regelnummer 200 van het programma hebben we wel aanhalingstekens gebruikt. De spaties van die regel worden dan wel door de computer in acht genomen. Het volgende voorbeeldprogramma laat een en ander zien.

```
100 CLS
110 PRINT "DIT IS EEN SPATIE TEST"
120 PRINT
```

```
130 FOR I=1 TO 3
140 READ A$,B$,C$
150 PRINT A$;B$;C$
160 PRINT
170 NEXT I
180 DATA Spaties,tussen,woorden
190 DATA Spaties, tussen ,woorden
200 DATA Spaties," tussen ",woorden
210 END
```

Opmerking

Spaties die achter de string staan en dus voor de komma, worden door de computer wel in acht genomen.

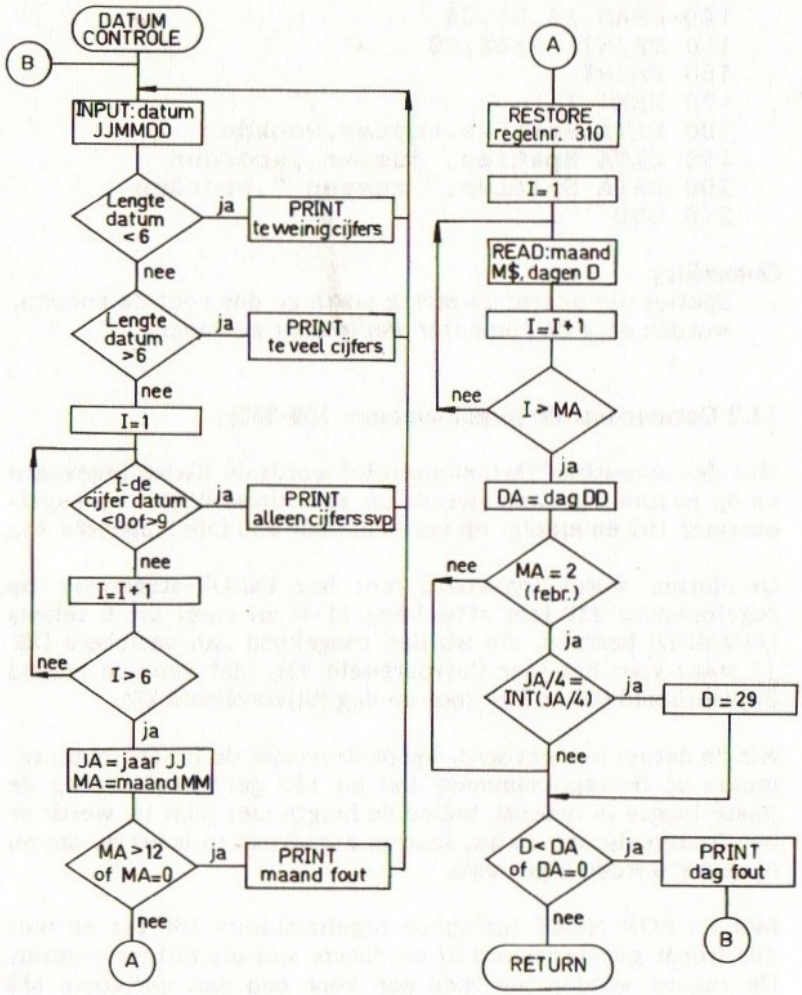
11.2 Datumcontrole (regelnummers 100-330)

Met de subroutine "Datumcontrole" wordt de datum ingevoerd en op juistheid gecontroleerd. De subroutine begint op regelnummer 100 en eindigt op regelnummer 330 (zie Appendix A).

De datum wordt ingevoerd met het INPUT-statement op regelnummer 110 (zie afbeelding 11-1) en moet uit 6 tekens (JJMMDD) bestaan, die worden toegekend aan variabele D$. JJ staat voor het jaar (bijvoorbeeld 85), MM voor de maand (bijvoorbeeld 03) en DD voor de dag (bijvoorbeeld 25).

Als de datum is ingevoerd, wordt deze met de IF-THEN statements op de regelnummers 140 en 150 gecontroleerd op de juiste lengte (6 tekens). Indien de lengte niet juist is, wordt er een foutboodschap op het scherm afgedrukt en moet de datum opnieuw worden ingevoerd.

Met de FOR-NEXT lus op de regelnummers 160 tot en met 190 wordt gecontroleerd of de datum wel uit cijfers bestaat. De tekens worden hiervoor een voor een aan variabele M$ toegekend (regelnummer 170) en met het IF-THEN statement op regelnummer 180 gecontroleerd. Indien een van de tekens geen cijfer is, wordt er een foutboodschap op het scherm afgedrukt en moet de datum opnieuw worden ingevoerd.



Afb. 11-1 Stroomdiagram van de datumcontrole-routine.

Vervolgens worden op de regelnummers 200 en 210 met de functie VAL de numerieke waarden van het jaar (JJ) en de maand (MM) bepaald. Deze waarden worden respectievelijk aan de numerieke variabelen JA en MA toegekend.

Met het IF-THEN statement op regelnummer 220 wordt gecontroleerd of de maand de juiste waarde heeft (1 tot en met 12). Indien deze waarde niet juist is, wordt er een foutboodschap op het scherm afgedrukt en moet de datum opnieuw worden ingevoerd.

Vervolgens wordt met het RESTORE-statement op regelnummer 230 de pointer op het eerste element van de eerste DATA-regel (regelnummer 310) geplaatst. In de drie DATA-regels (regelnummers 310 tot en met 330) staan de afkortingen van de 12 maanden en hoeveel dagen elke maand telt. Deze gegevens dienen om te kunnen controleren of de ingevoerde datum (DD) wel juist is, opdat bijvoorbeeld niet 850631 (31 juni 85) wordt geaccepteerd.

Met het READ-statement op regelnummer 250 wordt uit de DATA-regels de afkorting van de maand en het aantal dagen waaruit de maand bestaat, gelezen. Deze gegevens worden respectievelijk toegekend aan de variabelen M$ en D. Het lezen geschiedt in een FOR-NEXT lus. Het aantal keren dat de lus wordt doorlopen, wordt bepaald door het nummer van de maand (MA), zodat bij de laatste doorloop de variabelen M$ en D de juiste waarden hebben. Er wordt steeds begonnen met M$="JAN" en D=31.

Na het lezen van de afkorting van de maand en het aantal dagen van de maand, wordt met de functie VAL de numerieke waarde van de dag (DD) bepaald en toegekend aan de numerieke variabele DA.

Daar in de DATA-regels staat vermeld, dat februari 28 dagen telt, moet voor een schrikkeljaar nog een correctie worden uitgevoerd. Dit wordt gedaan met het IF-THEN statement op regelnummer 280. Er wordt in de eerste plaats gecontroleerd of de ingevoerde maand wel februari is (MA=2) en in de tweede plaats of het een schrikkeljaar is. Indien dit zo is, dan

is het jaar deelbaar door vier (JA/4). Er is dan geen rest. Door nu het resultaat van JA/4 te vergelijken met het gehele gedeelte (INT) van het resultaat van JA/4, kunnen we bepalen of het een schrikkeljaar is.

Voorbeeld:

Jaar 86 - JA/4 = 21.5
INT(JA/4) = 21
JA/4 ongelijk INT(JA/4) dus
86 is geen schrikkeljaar.

Jaar 84 - JA/4 = 21
INT(JA/4) = 21
JA/4 = INT(JA/4) dus
84 is schrikkeljaar.

Indien er sprake is van een schrikkeljaar, wordt D = 29.

Als laatste wordt op regelnummer 290 met een IF-THEN statement gecontroleerd of de ingevoerde dag (DA) niet groter is dan het aantal dagen dat de maand telt (D). De ingevoerde dag mag ook niet 0 zijn. Indien de ingevoerde dag niet correct is, wordt er een foutboodschap op het scherm afgedrukt en moet de datum opnieuw worden ingevoerd.

12 Opslaan van gegevens

Een van de belangrijkste facetten in het computergebeuren is het opslaan van gegevens op massamedia, zoals cassette of schijf en het later weer raadplegen, sorteren en veranderen van deze gegevens.

In dit hoofdstuk zullen we het opslaan van gegevens in een geheugen-array en het opslaan van gegevens op cassette behandelen. Ongetwijfeld is de cassetterecorder niet bepaald het geschiktste massageheugen, echter het is wel het goedkoopste en praktisch alle bezitters van MSX-computers zullen een cassetterecorder ter beschikking hebben. In deel 3 van deze serie leerboeken komen we nog uitgebreid terug op het gebruik van schijveneenheden.

Gegevens die op cassette worden opgeslagen en die van dezelfde aard zijn, bijvoorbeeld een lijst met adressen, vormen een bestand. In de computertaal wordt vaak van een "file" gesproken. Elk bestand heeft zijn eigen naam.

Voor het creeren van een array (gebied) in het geheugen hebben we te maken met de statements DIM en ERASE.

Voor het schrijven van gegevens (bestand) op cassette en het weer lezen van gegevens van cassette dienen onder andere de statements OPEN, CLOSE en MAXFILES en de functie EOF.

Verder zal in dit hoofdstuk ook nog de functie INSTR worden behandeld, waarmee de positie van een gespecificeerd teken of deel van een string binnen een totale string (alfanumerieke variabele) kan worden bepaald.

In dit hoofdstuk zal de rode draad het werken met een adres-

senbestand zijn. Het bestand zal eerst in een geheugen-array worden ingevoerd en vervolgens vanuit het geheugen, onder de bestandsnaam "ADRES" op cassette worden geschreven. Wanneer een adres moet worden geraadpleegd, moet het bestand eerst van cassette worden gelezen en in een array worden geplaatst. Vanuit de array kunnen dan de adressen worden opgevraagd. Wanneer het bestand in het geheugen staat, kunnen nieuwe adressen aan het bestand worden toegevoegd en bestaande adressen worden veranderd.

Met de opgedane kennis uit dit hoofdstuk zult u geen moeite hebben met het begrijpen van de modules "ARRAY A$ NAAR CASSETTE" (regelnummers 4600-4720) en "LEES AFSPRAKENBESTAND" (regelnummers 1000 - 1150) uit het AGENDA-programma in appendix A.

12.1 Het werken met arrays

Het is mogelijk om meerdere numerieke waarden of strings (alfanumerieke constanten) onder een variabele-naam in het geheugen te plaatsen. Om de verschillende numerieke waarden of strings uit elkaar te kunnen houden, worden aan de variabele-naam indexnummers of indices toegevoegd. Men spreekt dan van geindiceerde variabelen. Het geheel aan geindiceerde variabelen, dat bij elkaar behoort, wordt array of matrix genoemd. De geindiceerde waarden worden elementen genoemd.

Laten we aannemen, dat we een array met de variabele-naam X hebben, die uit zes elementen bestaat. Deze array bestaat dan uit de volgende geindiceerde variabelen:

X(0), X(1), X(2), X(3), X(4) en X(5).

De indices moeten hierbij altijd tussen haakjes staan. Het hier gegeven voorbeeld is een 1-dimensionale array. Daarnaast bestaan ook nog meer-dimensionale arrays. In dit hoofdstuk zullen we de volgende arrays behandelen:

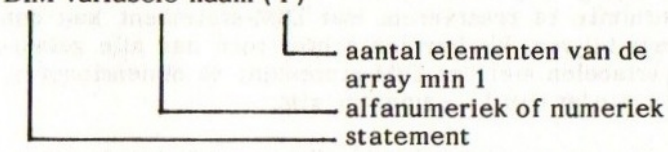
1-dimensionale arrays
2-dimensionale arrays

12.1.1 1-dimensionale arrays

Het eenvoudigste voorbeeld van een array is een rij geindiceerde variabelen, bijvoorbeeld A(0), A(1), . . . , A(5). De rij bestaat uit zes elementen, met de indices 0 tot en met 5. Om een array als geheel te kunnen aangeven wordt aan de array een variabele-naam toegekend, in ons voorbeeld de naam A. Het gewenste element wordt achter de variabele-naam, door middel van een index die tussen haakjes staat, aangegeven. Indices mogen alleen gehele positieve getallen zijn. De maximale waarde van indices is 65535. Een index mag worden uitgedrukt als:

een numerieke constante	X(5), N$(4)
een numerieke variabele	X(I), T(U)
een uitdrukking	X(I*3)

Voordat men met een array kan gaan werken, moet men dit de computer vertellen, zodat deze hiervoor geheugenruimte kan reserveren. Dit wordt gedaan door middel van het **DIM-statement.** DIM is de afkorting van DIMension (=grootte). Het formaat van het DIM-statement voor 1-dimensionale arrays is als volgt:

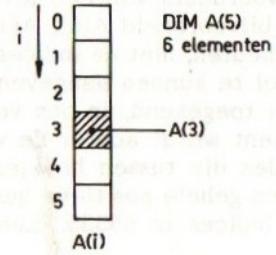


Bij het uitvoeren van een DIM-statement krijgen alle elementen van de gespecificeerde array de waarde nul. DIM-statements worden aan het begin van een programma geplaatst. Voor ons voorbeeld zal in het programma het volgende statement moeten worden opgenomen:

```
110 DIM A(5)
```

Met dit statement zal in het geheugen plaats worden gereserveerd voor een array, genaamd A, die bestaat uit de numerie-

ke variabelen A(0) tot en met A(5). Zie afbeelding 12-1.



Afb. 12-1 Een 1-dimensionale array.

Als er meerdere arrays in een programma worden gebruikt, zal elke array moeten worden gedimensioneerd. Voor het reserveren van geheugenruimte voor bijvoorbeeld array X, bestaande uit 15 elementen en voor array A$, bestaande uit 50 elementen, ziet het DIM-statement er als volgt uit:

```
110 DIM X(14),A$(49)
```

Wanneer een array uit minder dan 12 elementen bestaat, is het niet nodig om, door middel van het DIM-statement, geheugenruimte te reserveren. Het DIM-statement kan dan achterwege blijven. We bevelen echter toch aan alle geindiceerde variabelen met het DIM-statement te dimensioneren, ook als er minder dan 12 elementen zijn.

Wanneer een index wordt gebruikt, die groter is dan de gespecificeerde waarde in het DIM-statement, wordt de volgende foutboodschap op het scherm afgedrukt:

Subscript out of range in **regelnummer**

We zullen nu verder gaan met een eenvoudig testprogramma, waarin we met een READ-statement numerieke waarden vanuit een DATA-regel lezen, die we toekennen aan de elementen van een array.

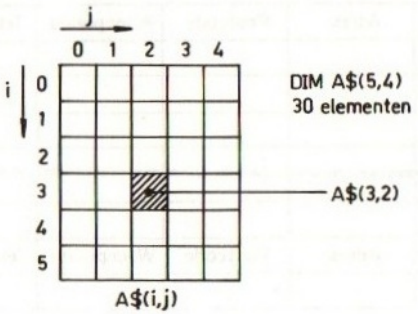
```
100 REM * 1-DIMENSIONALE ARRAY *
110 DIM A(5)
120 FOR I=0 TO 5
130 READ A(I)
140 NEXT I
150 INPUT "Welke element (0-5)";I
160 PRINT A(I)
170 INPUT "Volgend element (J/N)";J$
180 IF J$="J" GOTO 150
190 DATA 8,-23,78,149,-97,5
200 END
```

12.1.2 2-dimensionale arrays

Bij 2-dimensionale arrays hebben we te maken met twee afmetingen. Deze afmetingen worden uitgedrukt in rijen en kolommen. Wanneer een array uit i rijen en j kolommen bestaat, dan bezit die array i maal j elementen (zie afbeelding 12-2). Voor het reserveren van geheugenruimte voor 2-dimensionale arrays ziet het DIM-statement er als volgt uit:

DIM variabele-naam (i , j)

- aantal kolommen min 1
- aantal rijen min 1
- alfanumeriek of numeriek



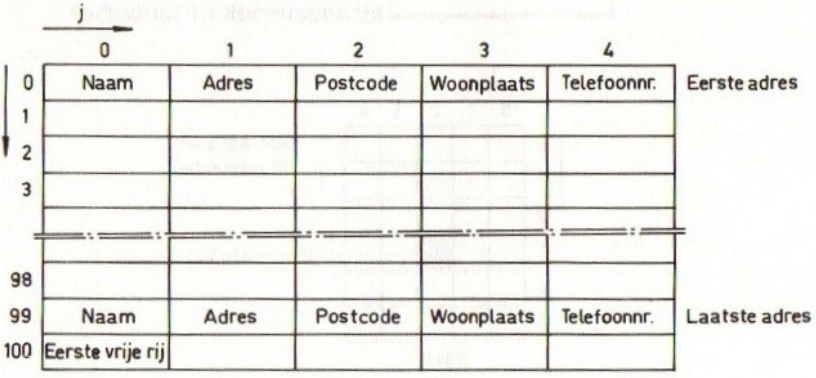
Afb. 12-2 Een 2-dimensionale array.

Voor het specificeren van een element in een 2-dimensionale array zijn twee indices nodig, namelijk een voor het aangeven van de rij en een voor het aangeven van de kolom. De twee indices moeten door een komma van elkaar worden gescheiden. Het element A$(3,2) bijvoorbeeld, staat in rij 3 en kolom 2. De nummering van rijen en kolommen begint bij 0. Het eerste element is dus A$(0,0). Met dit systeem kunnen we alle elementen in de array selecteren.

Het volgende programma is een van de subroutines van het programma ADRESSENBESTAND, waarover we in het begin van dit hoofdstuk hebben gesproken. De subroutine dient voor het invoeren van een adressenbestand via het toetsenbord in een 2-dimensionale array. Elk adres bestaat uit de volgende vijf elementen (velden):

1. Naam (kolom 0)
2. Adres (kolom 1)
3. Postcode (kolom 2)
4. Woonplaats (kolom 3)
5. Telefoonnummer (kolom 4)

De array is zodanig gedimensioneerd, dat er 100 adressen in kunnen worden geplaatst. Voor elk adres zijn vijf elementen gereserveerd, die in een en dezelfde rij staan (zie afbeelding 12-3). De array heeft de variabele-naam A$ en is als volgt



Afb. 12-3 Array A$.

gedimensioneerd: DIM A$(100,4).

De array bestaat uit 101 rijen, waarvan de rijen 0 tot en met 99 dienen voor het opslaan van adressen en rij 100 kolom 0 voor het noteren van de eerste vrije rij, zodat wanneer er een nieuw adres aan het bestand moet worden toegevoegd, het programma weet in welke rij het moet worden geplaatst.

Subroutine invoeren adressenbestand:

```
200 REM * INVOEREN ADRESSENBESTAND *
205 REM *         IN ARRAY A$      *
206 CLEAR 8000
207 WIDTH 40:KEY OFF
210 DIM A$(100,4)
211 S=0:B=0
215 CLS
220 PRINT "INVOEREN ADRESSEN IN ARRAY A$"
225 PRINT
230 FOR I=0 TO 99
235 LINE INPUT "Naam       : ";A$(I,0)
240 LINE INPUT "Adres      : ";A$(I,1)
245 LINE INPUT "Postcode   : ";A$(I,2)
250 LINE INPUT "Woonplaats : ";A$(I,3)
255 LINE INPUT "Telefoonnr.: ";A$(I,4)
260 PRINT
265 INPUT "Gegevens juist (J/N)";J$
270 IF J$="N" THEN PRINT "Voer gegevens op
nieuw in!":GOTO 235
271 IF B=1 GOTO 790
275 INPUT "Laatste adres  (J/N)";J$
280 IF J$="J" GOTO 295
285 PRINT
286 IF S=1 THEN I=I+1:GOTO 235
290 NEXT I
295 A$(100,0)=STR$(I+1)
296 IF S=1 GOTO 925
300 RETURN
```

Normaal wordt door MSX-BASIC een maximale geheugenruimte van 200 bytes toegewezen voor alfanumerieke varia-

belen (strings). Deze ruimte is natuurlijk veel te klein voor ons adressenbestand. Het zou namelijk al na ongeveer 4 adressen vol zijn. Wanneer we uitgaan van gemiddeld 80 tekens (bytes) per adres, dan hebben we een geheugenruimte nodig van 80 keer 100 is 8000 bytes. Met het statement CLEAR X kunnen we de geheugenruimte uitbreiden tot X bytes. Voor ons programma gaat het statement er dan als volgt uitzien: CLEAR 8000.

Mocht de ruimte na enige tijd toch nog te klein zijn, dan kunnen we de ruimte, door het veranderen van de waarde achter CLEAR, alsnog aanpassen.

De gegevens van elk adres worden ingevoerd met het LINE INPUT statement en achtereenvolgens in de kolommen 0 tot en met 4 van rij i geplaatst. Op regelnummer 265 wordt afgevraagd of de gegevens juist zijn. Wanneer het antwoord n(ee) is, dan kunnen de gegevens opnieuw worden ingevoerd.

Op regelnummer 275 wordt afgevraagd of het laatste adres is ingevoerd. Wanneer het antwoord j(a) is, dan wordt er naar regelnummer 295 gesprongen, waar het nummer van de eerste vrije rij in A$(100,0) wordt genoteerd. Vervolgens wordt naar het hoofdprogramma teruggekeerd. Wanneer het antwoord n(ee) is, kunnen de gegevens van het volgende adres worden ingevoerd.

Op de betekenis van de statements op de regelnummers 211, 271, 286 en 296 wordt later teruggekomen.

Schrijf de subroutine met het commando SAVE onder de naam "MOD1" op cassette, zodat we de routine later, met de andere nog te behandelen subroutines, met het commando MERGE kunnen samenvoegen tot een programma.

12.1.3 Het statement ERASE

Wanneer een array in een programma niet meer wordt gebruikt, kan deze worden opgeheven (gewist) met het **statement ERASE.** Arrays kunnen namelijk grote delen van het

geheugen innemen. Een voorwaarde is echter dat het betreffende DIM-statement werd uitgevoerd. Na het uitvoeren van het statement ERASE is de ruimte weer vrij voor andere doeleinden. Voorbeelden van het statement zijn:

```
ERASE A
ERASE A,B
ERASE N$
```

Achter ERASE staan de variabele-namen van de arrays, die moeten worden gewist.

Met de **functie FRE(0)** kunnen we de nog vrije geheugenruimte (in bytes) opvragen. Dit betekent dat we met deze functie erachter kunnen komen, wat een bepaalde array aan geheugenruimte opeist. Met het volgende programma kunnen we dit aantonen.

```
100 CLS
110 PRINT "GEHEUGENRUIMTE ARRAYS"
120 PRINT
130 PRINT "Vrije geheugenruimte voor"
140 PRINT "dimensionering is";FRE(0);"byte
s."
150 PRINT:INPUT "I,J";I,J
160 DIM A(I,J)
170 PRINT
180 PRINT "Vrije geheugenruimte na"
190 PRINT "dimensionering is";FRE(0);"byte
s."
200 ERASE A
210 A$=INKEY$:IF A$="" GOTO 210
220 GOTO 100
```

Met de **functie FRE("")** verkrijgt men het aantal nog beschikbare bytes in het geheugen voor alfanumerieke variabelen (strings). Het volgende korte programma toont dit aan.

```
100 CLS
110 PRINT "Aantal beschikbare bytes";FRE("
")
```

```
120 PRINT
130 INPUT "Geheugenruimte";X
140 CLEAR X
150 PRINT:GOTO 110
```

Wanneer een variabele weer wordt gedimensioneerd in een DIM-statement, zonder dat eerst het statement ERASE voor deze variabele is uitgevoerd, zal de boodschap "Redimensioned array" op het scherm worden afgedrukt. Het volgende programma geeft hier een voorbeeld van.

```
100 DIM X(50)
110 ERASE X
120 DIM X(100)
130 END
```

Opmerking:
Voer het programma ook eens uit zonder regelnummer 110.

Wanneer variabele-namen, die reeds eerder in een programma zijn toegepast, in een DIM-statement worden gebruikt, moeten deze eerst met het statement ERASE worden gewist. In het volgende programma wordt dit getoond.

```
100 CLS
110 FOR I=0 TO 5
120 A(I)=I*3:PRINT "A(";I;") =";A(I)
130 NEXT I
140 ERASE A
150 DIM A(15)
```

Opmerking:
Voer het programma ook eens uit zonder regelnummer 140.

12.2 Het werken met bestanden op cassette

Onder een bestand (file) verstaan we een verzameling gegevens, die bij elkaar behoren, bijvoorbeeld een adressenlijst,

voorraadadministratie, personeelsgegevens enz. Om toegang te kunnen verkrijgen tot de gegevens van een bestand, moet het bestand eerst worden geopend met het **statement OPEN.** Dit geldt ook wanneer een nieuw bestand moet worden gecreeerd. Bestanden staan meestal op massamedia, zoals cassette of schijf.

In dit hoofdstuk wordt het statement OPEN alleen behandeld voor bestanden op cassette. In deel 2 en 3 van deze serie leerboeken komen we nog uitgebreid terug op het OPEN-statement, ten behoeve van het grafische- en tekstscherm, de printer en de schijveneenheid.

Voor cassette kunnen we voor het statement OPEN twee formaten onderscheiden:

OPEN "CAS:bestandsnaam" FOR OUTPUT AS #X
en
OPEN "CAS:bestandsnaam" FOR INPUT AS #X

a b c d

In het statement OPEN betekent:

a)

Naam van het randapparaat, waarop het bestand staat of komt te staan. In het geval van de cassetterecorder is dit altijd CAS.

b)

Naam van het bestand. De naam mag uit maximaal 6 tekens bestaan. Wanneer er toch meer dan 6 tekens worden gebruikt, worden de meerdere tekens door de computer genegeerd.

c)

Geeft aan hoe het bestand gaat worden gebruikt.

FOR INPUT
gegevens worden van cassette gelezen en in het geheugen

geladen (bestand naar computer).

FOR OUTPUT
gegevens worden vanuit het geheugen op cassette geschreven (computer naar bestand).

d)

Een eenmaal geopend bestand is verder toegankelijk onder het nummer X. Dit betekent dat bij het lezen of schrijven niet meer de bestandsnaam behoeft te worden gebruikt. Het nummer staat bekend onder de naam bestandsnummer of kanaalnummer. Normaal wordt bij het inschakelen van de computer het aantal bestanden dat tegelijkertijd kan worden geopend op 1 gesteld. Wanneer men meer bestanden tegelijkertijd wil openen, dan kan dit door de waarde van de **systeemvariabele MAXFILES** te veranderen (de default-waarde is 1). De waarde mag niet kleiner zijn dan 0 en niet groter dan 15. Het nummer geeft het aantal bestanden aan, dat tegelijkertijd geopend kan zijn. Een bestandsnummer kan dan minimaal 1 zijn en maximaal de waarde, die aan MAXFILES is toegekend. De waarde nul betekent, dat er geen bestanden kunnen worden geopend.

12.2.1 Schrijven van gegevens op cassette

Met de commando's SAVE en CSAVE kunnen programma's vanuit het geheugen op cassette worden geschreven (zie hoofdstuk 8). Wanneer men andere gegevens dan programma's op cassette wil schrijven, dan zal eerst een bestand moeten worden geopend. Laten we aannemen, dat we het adressenbestand in array A$ (zie paragraaf 12.1.2) onder de bestandsnaam "ADRES" op cassette willen schrijven. Het formaat van het OPEN-statement gaat er dan als volgt uit zien:

```
OPEN "CAS:ADRES" FOR OUTPUT AS #1
```

Bij uitvoering van dit statement wordt op cassette een bestand geopend onder de naam ADRES.

De gegevens uit array A$ zullen met de volgende subroutine naar het bestand ADRES worden geschreven.

```
350 REM * ARRAY A$ NAAR CASSETTE *
355 CLS
360 PRINT "LAAD LEGE CASSETTE IN RECORDER."
365 PRINT "ZET RECORDER IN STAND OPNEMEN."
370 PRINT "DRUK OP RETURN-TOETS."
375 IF INKEY$<>CHR$(13) GOTO 375
380 OPEN "CAS:ADRES" FOR OUTPUT AS #1
385 FOR I=0 TO 100
390 FOR J=0 TO 4
395 PRINT #1,A$(I,J)
400 NEXT J
405 NEXT I
410 PRINT
415 PRINT "BESTAND STAAT OP CASSETTE."
420 PRINT "DRUK OP RETURN-TOETS."
425 IF INKEY$<>CHR$(13) GOTO 425
430 CLOSE #1
435 RETURN
```

De gegevens uit array A$ worden in twee geneste FOR-NEXT lussen (regelnummers 385 tot en met 405) op cassette geschreven. Het schrijven gebeurt met het PRINT-statement op regelnummer 395. In het statement behoeft alleen het bestandsnummer #1 (referentienummer) te worden genoemd. De computer weet dan precies voor welk apparaat en bestand de gegevens bestemd zijn. Dit is reeds bekend gemaakt in het OPEN-statement.

Voordat de subroutine wordt beeindigd, wordt met het statement CLOSE eerst nog het bestand afgesloten en de betrokken geheugenbuffer vrijgemaakt. Achter CLOSE (regelnummer 430) staat het bestandsnummer vermeld, van het bestand, dat afgesloten moet worden. Wanneer achter CLOSE niets staat, dan zullen alle bestanden, die zijn geopend op het moment van uitvoering van het CLOSE-statement, worden afgesloten. Dit betekent, dat vanaf dat moment bestanden niet meer toegankelijk zijn. Hiervoor zullen ze dan eerst weer moeten worden geopend met het statement OPEN.

Wanneer bijvoorbeeld de bestanden met de nummers 1 en 3 moeten worden afgesloten, dan gaat het statement er als volgt uit zien:

```
CLOSE #1,#3
```

Schrijf de zojuist behandelde subroutine met het commando SAVE onder de naam MOD2 op cassette, zodat deze later met MERGE, samen met de andere routines, tot een programma kan worden samengevoegd.

12.2.2 Lezen van gegevens van cassette

Met de commando's LOAD en CLOAD kunnen programma's van cassette worden gelezen en in het geheugen worden geladen (zie hoofdstuk 8). Wanneer men andere gegevens dan programma's van cassette wil lezen, dan zal eerst een bestand moeten worden geopend. Het geopenden bestand dient dan op cassette te staan.

Laten we aannemen, dat we het bestand ADRES van cassette willen lezen en in zijn geheel in array A$ in het geheugen willen laden. Het formaat van het OPEN-statement gaat er dan als volgt uitzien:

```
OPEN "CAS:ADRES" FOR INPUT AS #1
```

Voordat met het lezen van de gegevens uit het bestand kan worden begonnen, zal eerst de cassetteband met het bestand ADRES moeten worden geladen en worden teruggespoeld naar het begin. Vervolgens zal de cassetterecorder op de stand afspelen (PLAY) moeten worden ingesteld.

De volgende subroutine zorgt voor het lezen van de gegevens van het bestand ADRES en het laden van deze gegevens in array A$ in het geheugen.

```
450 REM * LEZEN VAN BESTAND *
455 CLS
460 PRINT "PLAATS CASSETTE MET BESTAND"
```

```
465 PRINT "ADRES IN RECORDER. SPOEL VERVOL
GENS"
470 PRINT "CASSETTE TERUG NAAR BEGIN EN ST
EL"
475 PRINT "RECORDER IN OP AFSPELEN (PLAY)."
480 PRINT "DRUK OP RETURN-TOETS."
485 IF INKEY$<>CHR$(13) GOTO 485
490 OPEN "CAS:ADRES" FOR INPUT AS #1
495 FOR I=0 TO 100
500 FOR J=0 TO 4
505 IF EOF(1)=-1 GOTO 525
510 LINE INPUT #1,A$(I,J)
515 NEXT J
520 NEXT I
525 PRINT
530 PRINT "BESTAND IS GELEZEN."
535 PRINT "DRUK OP RETURN-TOETS."
540 IF INKEY$<>CHR$(13) GOTO 540
545 CLOSE #1
550 RETURN
```

Op regelnummer 510 worden met het statement LINE INPUT de gegevens van cassette gelezen en in array A$ in het geheugen geladen. In het statement behoeft alleen maar het bestandsnummer te worden genoemd.

Op regelnummer 505 staat de **functie EOF** (End Of File). De functie EOF controleert of het einde van een sequentieel (volgordelijk) bestand is bereikt. De functie geeft als resultaat -1, indien het einde van het bestand is bereikt. Indien het einde nog niet is bereikt zal het resultaat van de functie 0 zijn.

In EOF(X) is X het bestandsnummer, waaronder het bestand is geopend. Het bestand moet zijn geopend voor de INPUT-mode. De functie EOF wordt meestal toegepast, wanneer de lengte van een bestand niet bekend is.

Schrijf de routine "LEZEN VAN EEN BESTAND" met het commando SAVE onder de naam MOD3 (module 3) op cassette.

12.3 Hoofdprogramma Adressenbestand

Het hoofdprogramma voor het adressenbestand bestaat uit een menu (regelnummers 110 tot en met 155), het maken van een keuze uit het menu (regelnummer 160) en het vervolgens aanroepen van de gekozen subroutine (regelnummer 175). Daar het programma voor zichzelf spreekt, zal er geen verdere uitleg volgen.

Listing van het hoofdprogramma:

```
100 REM * ADRESSENBESTAND *
101 CLEAR 8000
102 WIDTH 40:KEY OFF
103 DIM A$(100,4):S=0:B=0
105 CLS
110 PRINT "MENU ADRESSENBESTAND"
115 PRINT:PRINT
120 PRINT "1. Invoeren bestand in geheugen
":PRINT
125 PRINT "2. Schrijven van bestand op cas
sette":PRINT
130 PRINT "3. Lezen van bestand van casset
te":PRINT
135 PRINT "4. Opvragen van een adres":PRIN
T
140 PRINT "5. Wijzigen van een bestaand ad
res":PRINT
145 PRINT "6. Toevoegen van een nieuw adre
s":PRINT
150 PRINT "7. Programma beeindiging"
155 PRINT:PRINT
160 INPUT "Welke functie";F
165 IF F<1 OR F>7 THEN PRINT "Verkeerde ke
uze!":GOTO 160
170 IF F=7 THEN END
175 ON F GOSUB 200,350,450,600,750,900
180 GOTO 105
```

Schrijf het hoofdprogramma met het commando SAVE onder de naam "HOPRO" op cassette.

12.4 Het werken met het adressenbestand

Wanneer het adressenbestand in geheugen-array A$ staat, kunnen de volgende acties worden ondernomen:

1. Opvragen van een adres.
2. Toevoegen van een nieuw adres.
3. Wijzigen van een bestaand adres.

We beginnen met de listing van de subroutine voor het "Toevoegen nieuw adres". In deze subroutine maken we gebruik van een gedeelte van de subroutine "Invoeren bestand in geheugen".

```
900 REM * TOEVOEGEN NIEUW ADRES *
905 CLS
910 S=1
915 I=VAL(A$(100,0))
920 GOTO 235
925 S=0
930 RETURN
```

Op regelnummer 910 wordt S=1 gemaakt. De variabele S wordt gebruikt als "switch" in de subroutine "Invoeren bestand in geheugen". Na het bepalen van de eerste vrije rij in array A$ op regelnummer 915, wordt naar regelnummer 235 gesprongen, waar het invoeren van het nieuwe adres start. Wanneer het adres is ingevoerd, wordt er teruggesprongen naar regelnummer 925 en vervolgens teruggekeerd naar het hoofdprogramma.

De volgende listing is van de subroutine "Opvragen van een adres", welke begint op regelnummer 600.

```
600 REM * OPVRAGEN VAN EEN ADRES *
605 CLS
606 T=0
610 LINE INPUT "Welke naam: ";N$
615 L=LEN(N$)
620 FOR I=0 TO 99
625 IF N$=LEFT$(A$(I,0),L) GOTO 635
```

```
630 GOTO 680
635 T=T+1:CLS
640 PRINT A$(I,0):PRINT
645 PRINT A$(I,1):PRINT
650 PRINT A$(I,2);SPC(3);
655 PRINT A$(I,3):PRINT
660 PRINT A$(I,4)
665 PRINT:PRINT
670 INPUT "Juiste adres (j/n)";J$
675 IF J$="j" GOTO 690
680 NEXT I
685 IF T=0 THEN PRINT "Adres niet gevonden
!"
690 PRINT "Druk RETURN-toets in."
695 IF INKEY$<>CHR$(13) GOTO 695
700 IF B=1 GOTO 770
705 RETURN
```

Op regelnummer 625 wordt de ingevoerde naam vergeleken met de namen uit array A$. Als de naam gevonden is, worden alle gegevens, die bij de naam behoren, op het scherm afgedrukt. De rest van het programma spreekt voor zichzelf.

De laatste listing is die van de subroutine "Wijzigen van een bestaand adres". De subroutine begint op regelnummer 750.

```
750 REM * WIJZIGEN BESTAAND ADRES *
755 CLS
760 B=1
765 GOTO 606
770 PRINT
775 PRINT "Adres kan gewijzigd worden."
780 PRINT
785 GOTO 235
790 B=0
795 RETURN
```

Ook in deze subroutine wordt gebruik gemaakt van gedeelten van andere subroutines. Op regelnummer 765 wordt naar de subroutine gesprongen, waarmee het adres wordt opgezocht. In deze routine wordt ook de rij (i), waar het adres staat,

bepaald. Als dit is gebeurd, wordt teruggesprongen naar regelnummer 770. Op regelnummer 785 wordt naar de subroutine gesprongen, waarmee het adres wordt veranderd. Wanneer dit is gebeurd, wordt teruggesprongen en teruggekeerd naar het hoofdprogramma (menu).

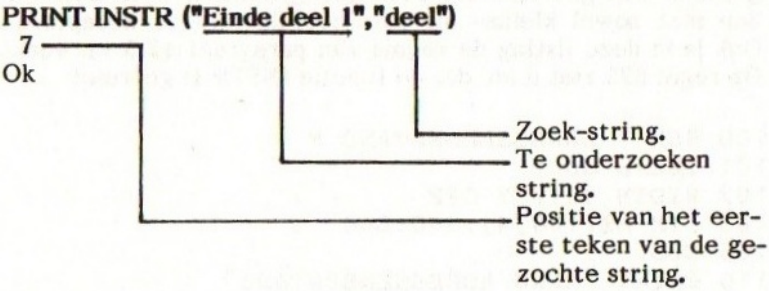
Opmerking:

Voordat met het totale programma kan worden gewerkt, moet eerst regel 210 worden verwijderd. Het DIM-statement is namelijk opgenomen in het hoofdprogramma. Wordt het statement niet verwijderd, dan wordt de volgende boodschap op het scherm afgedrukt: "Redimensioned array in 210". Verder kunnen ook de regelnummers 206 en 207 worden verwijderd.

Voeg nu het hoofdprogramma en de subroutines samen. Gebruik hierbij het commando MERGE.

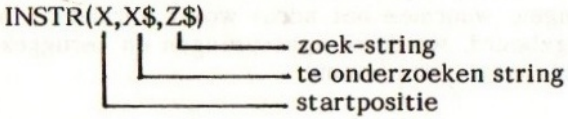
12.5 De functie INSTR

Wanneer men binnen een string de beginpositie van een bepaald woord of tekencombinatie uit die string wil bepalen, kan men gebruik maken van de functie INSTR. Bijvoorbeeld:



Wanneer de te onderzoeken string niet de zoek-string bevat, levert de functie de waarde 0 op. Wanneer slechts een gedeelte van een string dient te worden onderzocht, zal de positie moeten worden aangegeven waar moet worden begonnen. De beginpositie mag niet kleiner zijn dan 1 en niet groter dan

255. Het algemene formaat van INSTR ziet er als volgt uit:



Het volgende programma is een voorbeeld met de functie INSTR.

```
100 CLS
110 LINE INPUT "Zin: ";X$
120 LINE INPUT "Te onderzoeken string: ";Z
$
130 INPUT "Startpositie";X
140 PRINT
150 PRINT INSTR(X,X$,Z$)
160 A$=INKEY$:IF A$="" GOTO 160
170 GOTO 100
```

Tot slot van dit hoofdstuk volgt nog een complete listing van het adressenbestandsprogramma. Ten opzichte van de hiervoor gegeven listings van de onderdelen van het programma, zijn er enkele kleine wijzigingen aangebracht. Zo is het programma wat gebruikersvriendelijker gemaakt, door antwoorden met zowel kleine- als met hoofdletters te accepteren. Ook is in deze listing de kennis van paragraaf 12.5 verwerkt. Op regel 625 ziet u nu, dat de functie INSTR is gebruikt.

```
100 REM * ADRESSENBESTAND *
101 CLEAR 8000
102 WIDTH 40:KEY OFF
103 DIM A$(100,4):S=0:B=0
105 CLS
110 PRINT "MENU ADRESSENBESTAND"
115 PRINT:PRINT
120 PRINT "1. Invoeren bestand in geheugen":
PRINT
125 PRINT "2. Schrijven van bestand op casse
tte":PRINT
```

```
130 PRINT "3. Lezen van bestand van cassette
":PRINT
135 PRINT "4. Opvragen van een adres":PRINT
140 PRINT "5. Wijzigen van een bestaand adre
s":PRINT
145 PRINT "6. Toevoegen van een nieuw adres"
:PRINT
150 PRINT "7. Programma beeindiging"
155 PRINT:PRINT
160 INPUT "Welke functie";F
165 IF F<1 OR F>7 THEN PRINT "Verkeerde keuz
e!":GOTO 160
170 IF F=7 THEN END
175 ON F GOSUB 200,350,450,600,750,900
180 GOTO 105
200 REM * INVOEREN ADRESSENBESTAND *
205 REM *         IN ARRAY A$        *
211 S=0:B=0
215 CLS
220 PRINT "INVOEREN ADRESSEN IN ARRAY A$"
225 PRINT
230 FOR I=0 TO 99
235 LINE INPUT "Naam       : ";A$(I,0)
240 LINE INPUT "Adres      : ";A$(I,1)
245 LINE INPUT "Postcode   : ";A$(I,2)
250 LINE INPUT "Woonplaats : ";A$(I,3)
255 LINE INPUT "Telefoonnr.: ";A$(I,4)
260 PRINT
265 INPUT "Gegevens juist (J/N)";J$
270 IF J$="N" OR J$="n" THEN PRINT "Voer geg
evens opnieuw in!":GOTO 235
271 IF B=1 GOTO 790
275 INPUT "Laatste adres  (J/N)";J$
280 IF J$="J" OR J$="j" GOTO 295
285 PRINT
286 IF S=1 THEN I=I+1:GOTO 235
290 NEXT I
295 A$(100,0)=STR$(I+1)
296 IF S=1 GOTO 925
300 RETURN
350 REM * ARRAY A$ NAAR CASSETTE *
```

```
355 CLS
360 PRINT "LAAD LEGE CASSETTE IN RECORDER."
365 PRINT "ZET RECORDER IN STAND OPNEMEN."
370 PRINT "DRUK OP RETURN-TOETS."
375 IF INKEY$<>CHR$(13) GOTO 375
380 OPEN "CAS:ADRES" FOR OUTPUT AS #1
385 FOR I=0 TO 100
390 FOR J=0 TO 4
395 PRINT #1,A$(I,J)
400 NEXT J
405 NEXT I
410 PRINT
415 PRINT "BESTAND STAAT OP CASSETTE."
420 PRINT "DRUK OP RETURN-TOETS."
425 IF INKEY$<>CHR$(13) GOTO 425
430 CLOSE #1
435 RETURN
450 REM * LEZEN VAN BESTAND *
455 CLS
460 PRINT "PLAATS CASSETTE MET BESTAND"
465 PRINT "ADRES IN RECORDER. SPOEL VERVOLGE
NS"
470 PRINT "CASSETTE TERUG NAAR BEGIN EN STEL"
475 PRINT "RECORDER IN OP AFSPELEN (PLAY)."
480 PRINT "DRUK OP RETURN-TOETS."
485 IF INKEY$<>CHR$(13) GOTO 485
490 OPEN "CAS:ADRES" FOR INPUT AS #1
495 FOR I=0 TO 100
500 FOR J=0 TO 4
505 IF EOF(1)=-1 GOTO 525
510 LINE INPUT #1,A$(I,J)
515 NEXT J
520 NEXT I
525 PRINT
530 PRINT "BESTAND IS GELEZEN."
535 PRINT "DRUK OP RETURN-TOETS."
540 IF INKEY$<>CHR$(13) GOTO 540
545 CLOSE #1
550 RETURN
600 REM * OPVRAGEN VAN EEN ADRES *
605 CLS
```

```
606 T=0
610 LINE INPUT "Welke naam: ";N$
620 FOR I=0 TO 99
625 IF INSTR(A$(I,0),N$)<>0 GOTO 635
630 GOTO 680
635 T=T+1:CLS
640 PRINT A$(I,0):PRINT
645 PRINT A$(I,1):PRINT
650 PRINT A$(I,2);SPC(3);
655 PRINT A$(I,3):PRINT
660 PRINT A$(I,4)
665 PRINT:PRINT
670 INPUT "Juiste adres (j/n)";J$
675 IF J$="j" OR J$="J" GOTO 690
680 NEXT I
685 IF T=0 THEN PRINT "Adres niet gevonden!"
690 PRINT "Druk RETURN-toets in."
695 IF INKEY$<>CHR$(13) GOTO 695
700 IF B=1 GOTO 770
705 RETURN
750 REM * WIJZIGEN BESTAAND ADRES *
755 CLS
760 B=1
765 GOTO 606
770 PRINT
775 PRINT "Adres kan gewijzigd worden."
780 PRINT
785 GOTO 235
790 B=0
795 RETURN
900 REM * TOEVOEGEN NIEUW ADRES *
905 CLS
910 S=1
915 I=VAL(A$(100,0))
920 GOTO 235
925 S=0
930 RETURN
```

Appendix A
Het AGENDA-programma

Het AGENDA-programma is bedoeld als voorbeeld van wat u met de in dit boek opgedane kennis kunt doen. Alle in dit programma gebruikte statements zijn in dit boek behandeld. Er komen in dit programma geen statements voor, die nog niet zijn behandeld. Door zoveel mogelijk verschillende statements in het programma toe te passen en door de opgelegde beperkingen, is niet altijd voor de meest logische oplossing gekozen. Als voorbeeld hiervan is te noemen, dat een aantal functies met behulp van de functietoetsen zijn aan te roepen en dat een aantal functies uit een menu moeten worden gekozen. U begrijpt dat de reden hiervoor was, dat we zowel het bebruik van de functietoetsen als het gebruik van een menu in het programma wilden laten zien.

Uw speciale aandacht vragen we voor de foutafhandelingsroutine in dit programma. Het gebruik van de verschillende foutdetectie- en foutafhandelingsstatements is niet ingewikkeld, maar zoals u in het programma kunt zien, loopt het aantal te gebruiken statements al gauw op. Dit is de reden dat de meeste programmeurs deze mogelijkheden niet gebruiken. Het is te veel type-werk. Toch zouden wij u willen aanraden de mogelijkheden die worden geboden wel te gebruiken. Voor de eindgebruiker van het programma is het plezierig om goede en duidelijke foutmeldingen te krijgen, en voor uzelf, als programmeur, kan het tijdens het ontwikkelen en testen van het programma ook zijn voordelen opleveren.

Bij de opzet van het AGENDA-programma zijn we ervan uitgegaan, dat het programma moest kunnen werken op iedere MSX-computer met 64K geheugen en een cassetterecorder. Het afsprakenbestand moet kunnen worden bewaard. Dit kan niet in het geheugen, omdat de gegevens dan verloren zouden

gaan zodra de computer wordt uitgeschakeld. Gegevens op cassette kunnen echter alleen volgordelijk worden verwerkt, terwijl we kriskras door de agenda willen kunnen bladeren. We willen dus willekeurig toegang tot de gegevens hebben. Om dit te bereiken, moeten we de gegevens in volgorde van cassette in het geheugen lezen. Staan alle gegevens eenmaal in het geheugen, dan hebben we binnen dat geheugen willekeurig toegang tot de gegevens. Dit betekent echter, dat niet alleen het programma, maar ook alle gegevens in het geheugen moeten staan. We zullen dus spaarzaam met de beschikbare geheugenruimte om moeten gaan, ook al heeft de MSX-computer een flink geheugen.

Dit "gebrek" aan geheugen heeft er toe geleid, dat er soms wat moeilijker routines zijn ontstaan dan op het eerste gezicht nodig lijkt. Het is echter al gauw voordeliger om een paar extra programmaregels op te nemen, dan om ruimte voor bijvoorbeeld 100 afspraken te reserveren.

Voordat met het programmeren kon worden begonnen, moest eerst goed worden vastgelegd hoe het bestand, waarin de gegevens zouden worden opgeslagen, er precies uit zou zien. Voor een goed begrip van het programma dient u dit ook te weten.

In array A$ worden de afspraken opgeslagen. Deze array wordt geinitialiseerd als A$(30,10). Dit is voor 31 dagen (0 t/m 30), terwijl er voor iedere dag 11 (0 t/m 10) velden beschikbaar zijn. Het eerste veld, A$(nn,0), bevat (indien er op die dag al een afspraak is gemaakt) de datum. Het tweede veld, A$(nn,1), bevat de gegevens voor de afsprakenbalk. In deze balk is te zien op welke tijden er nog vrije tijd beschikbaar is voor nieuwe afspraken. De afspraken zelf staan in de velden 2 tot en met 10. Er kunnen dus maximaal 9 afspraken per dag worden gemaakt.

Het datumveld in de array bevat de datum in het formaat JJMMDD. Twee cijfers voor het jaartal, twee cijfers voor de maand (ook als de maand slechts 1 cijfer zou zijn, er wordt dan een 0 voor gezet) en twee cijfers voor de dag (ook hier wordt indien nodig een 0 toegevoegd).

In het veld voor de afsprakenbalk staan 40 cijfers. Deze cijfers kunnen 0 of 1 zijn. Ieder cijfer heeft betrekking op 1 kwartier. Is een cijfer 0, dan is het bijbehorende kwartier nog vrij. Is het cijfer 1 dan is het bijbehorende kwartier bezet. 40 kwartieren zijn samen 10 uren. In ons programma hebben we de kwartieren gelegd tussen 08.00 en 18.00 uur.

De velden voor de afspraken zelf bevatten de volgende informatie. De eerste vier posities zijn de begintijd van de afspraak. De daaropvolgende vier posities bevatten de eindtijd van de afspraak. De daaropvolgende posities bevatten de afspraak zelf, die geen vaste lengte heeft. Deze kan varieren van 1 teken tot maximaal 60 tekens.

Met deze gegevens in het achterhoofd zal het gemakkelijker zijn om de programma-listing te lezen. Een aantal van de subroutines uit het programma werd reeds in de loop van het boek behandeld. We zullen nu dan ook geen verdere uitleg over de werking van het programma geven. In plaats daarvan volgt nu een gebruiksaanwijzing van het programma.

GEBRUIKSAANWIJZING

Na het laden en opstarten van AGENDA wordt u gevraagd de huidige datum en tijd in te geven. Zowel de datum als de tijd zullen bovenaan het scherm worden weergegeven. De tijd wordt keurig netjes bijgehouden. Na iedere minuut die is verstreken wordt dat ook zichtbaar op het scherm. Zodra het aantal minuten 60 wordt, wordt het aantal uren met 1 verhoogd en wordt het aantal minuten weer 0. Mocht het aantal uren de 24 bereiken, dan is er een dag verstreken en zal de datum worden verhoogd.

Wanneer u AGENDA voor het eerst gaat gebruiken, hebt u nog geen afsprakenbestand op cassette staan. Vandaar dat u nu de vraag krijgt of u een afsprakenbestand van cassette wilt laden ja of nee. Omdat u dat (in de toekomst) meestal wel zult willen, is iedere toetsaanslag, behalve de letter n, een teken voor de computer om een bestand van cassette te gaan lezen. De eerste keer dat u het programma draait, zult u dus

specifiek n moeten intikken. In dat geval wordt er geen cassette-bestand ingelezen, maar wordt alleen een lege array gegenereerd.

Had u al een afsprakenbestand op cassette staan, dan zal dat bestand nu worden ingelezen. Echter, om ruimte te besparen, worden uit dat bestand alleen de gegevens over de dagen die gelden vanaf de aan het begin ingegeven "huidige datum", in het geheugen opgenomen. In uw agenda zult u dus nooit gegevens van de vorige dag of dagen vinden. Wilt u dat toch, dan moet u bij het opstarten van het programma niet de datum van vandaag ingeven, maar een datum van een of meer dagen geleden.

Het maximum aantal dagen waarover gegevens in het geheugen kunnen worden opgenomen is echter 31. Dit heeft weer te maken met het "beperkte" geheugen. Per dag kunnen maximaal 9 afspraken worden bewaard. Iedere afspraak bestaat uit een begin- en eindtijd en de afspraak zelf. Deze afspraak is beperkt tot maximaal 60 tekens. Een klein rekensommetje leert ons dus, dat indien alle afspraken de maximale lengte zouden innemen, er 30 keer 9 keer 60 is 16200 tekens zouden moeten worden opgeborgen in het geheugen. Dit is meer dan er in het geheugen passen. Het programma zelf is ruim 12K groot en bovendien moeten er nog meer gegevens (begin- en eindtijd, datum en gegevens voor de afsprakenbalk) worden onthouden. Daar er in werkelijkheid nooit 9 afspraken per dag, 7 dagen per week, gemaakt zullen worden, zal de beperking van 30 dagen meestal voldoende zijn. Is uw ervaring dat u slechts 1 of 2 afspraken per dag maakt, dan kunt u het aantal dagen vergroten. U moet dan de afspraken-array vergroten, en in een aantal subroutines zult u een kleine aanpassing moeten maken.

Is het bestand eenmaal geladen, dan zal de computer het beeldscherm opbouwen. Op de bovenste regel verschijnt de huidige datum en tijd. Op de derde regel verschijnt de datum waarop de agenda "open" ligt, met daaronder een balk waarin kan worden gezien op welke tijden afspraken zijn gemaakt, en welke tijden nog vrij zijn. Mochten er al afspraken op die dag zijn gemaakt, dan zal afspraak nummer 1 onder de afspraken-

balk worden weergegeven, inclusief de begin- en eindtijd. Op de onderste regel, tenslotte, verschijnen de teksten, die bij de vijf functietoetsen horen. Vanaf dit moment gebeurt er niets meer. Indien u echter goed oplet, ziet u dat de tijd op de bovenste regel wel wordt bijgehouden.

Alle functietoetsen hebben op dit moment "blader"-functies. Dit wil zeggen dat de subroutines, die met de functietoetsen worden aangeroepen, het afsprakenbestand alleen lezen, en er geen wijzigingen in aanbrengen. Na het indrukken van F5 krijgen de functietoetsen nieuwe functies toegewezen. Met deze nieuwe functies kunnen wijzigingen in het afsprakenbestand worden gemaakt. Zodra een wijzigingsfunctie is uitgevoerd, zullen de functietoetsen automatisch weer een bladerfunctie krijgen. We zullen nu alle functietoetsen een voor een behandelen.

BLADERFUNCTIES

F1 (terugbladeren)
Hiermee wordt naar de voorgaaande pagina in de agenda gebladerd. Mocht er geen voorgaande dag meer zijn, dan zal automatisch naar de laatste dag van de agenda worden gegaan. Daarna kan weer naar de voorgaande dag worden gebladerd.

F2 (vooruitbladeren)
Hiermee wordt naar de volgende pagina in de agenda gebladerd. Indien het einde van de agenda is bereikt, zal naar de eerste dag worden teruggekeerd. Daarna kan weer naar de volgende dag worden gebladerd.

F3 (afspraak)
Wanneer de agenda open ligt, is alleen de eerste afspraak van die dag zichtbaar. Met F3 wordt steeds de volgende afspraak op het beeldscherm zichtbaar gemaakt. Is de laatste afspraak van de dag afgedrukt, en wordt weer op F3 gedrukt, dan zal de eerste afspraak weer worden afgedrukt. Daarna weer de volgende, etc.

F4 (datum)
Indien u wilt zien welke afspraken op een bepaalde datum zijn gemaakt, dan kunt u F4 indrukken. U wordt dan gevraagd naar de datum die u wilt zien, waarna de afspraken van die dag op het beeldscherm verschijnen. Geeft u een datum in die later is dan in de agenda voor komt, dan zal de laatste dag van de agenda worden getoond.

F5 (functies)
Door het indrukken van deze functietoets worden nieuwe functies aan de functietoetsen F1 tot en met F5 toegekend. Deze functies worden beschreven onder de kop "schrijffuncties".

SCHRIJFFUNCTIES

F1 (afspraak toevoegen)
Na het indrukken van deze functietoets kunt u een nieuwe afspraak aan het bestand toevoegen. Er zal echter eerst worden gecontroleerd of er op het moment waarop u de afspraak wilt maken wel tijd vrij is.

F2 (afspraak wijzigen)
Met deze functietoets geeft u te kennen dat u in een bestaande afspraak een wijziging wilt aanbrengen. U zult worden gevraagd of het een wijziging van de tijd dan wel van de inhoud van de afspraak betreft. Wijzigt u de tijd, dan zal er weer worden gecontroleerd of de nieuwe tijd wel vrij is.

F3 (afspraak verwijderen)
Hiermee kunt u een bestaande afspraak uit uw agenda verwijderen.

F4 (keuze-menu)
Na het indrukken van deze functietoets wordt er een menu op het beeldscherm afgedrukt. U kunt nu, vanuit het menu, de volgende functies kiezen:

- 1 Schrijf de afspraken naar cassette
- 2 Schrijf programma en afspraken naar cassette

- 3 Maak een back-up van het programma
- 4 Zoek een bepaalde afspraak
- 5 Beeindig het programma

De eerste drie keuzemogelijkheden spreken voor zichzelf. Kiest u mogelijkheid 4, dan zult u worden gevraagd naar een tekst. Deze tekst zal dan in de agenda worden opgezocht. U hebt vervolgens de keuze of die tekst moet worden gezocht vanaf de dag waarop u de agenda hebt open liggen, of dat de hele agenda moet worden onderzocht. Deze functie kan handig zijn als u weet dat u een afspraak heeft met Jansen, maar u weet niet meer wanneer. U tikt dan de naam Jansen in. Het programma zoekt dan in de agenda naar die naam en laat de afspraak waarin die naam voorkomt zien.

Keuzemogelijkheid 5 uit het menu beeindigt het programma. Deze mogelijkheid beeindigt het programma op een fatsoenlijke manier. Denkt u er echter wel om, dat eventuele wijzigingen in het afsprakenbestand **niet** naar cassette worden geschreven. Voordat u het programma beeindigt, dient u eerst uw afsprakenbestand naar cassette te schrijven met keuzemogelijkheid 1 uit het menu.

F5 (terug naar bladerfuncties)
Hiermee kunt u aan de functietoetsen weer bladerfuncties toekennen.

Hebt u geen 64K geheugen in uw computer, dan zult u dit programma alleen kunnen gebruiken als u het programma intikt zonder REM-statements. Bovendien zult u dan op regel 18, achter het CLEAR-statement een lager getal moeten zetten. Laadt het programma eens met in regel 18 CLEAR 1000. Kijk daarna hoeveel geheugen er nog vrij is met PRINT FRE(0). Vervolgens kunt u het getal achter CLEAR zoveel verhogen als er nog vrije geheugenruimte is.

Wij wensen u tenslotte met dit programma vele leerzame, nuttige en plezierige uurtjes toe.

```
1 '*************************
2 '* HOOFDPROGRAMMA: AGENDA *
3 '*************************
4 SCREEN 3
5 OPEN "grp:" AS #1
6 DRAW "BM32,0"
7 PRINT #1,"welkom"
8 DRAW "BM96,80"
9 PRINT #1,"in"
10 DRAW "BM32,160"
11 PRINT #1,"agenda"
12 FOR I=1 TO 2000:NEXTI
13 SCREEN 0
14 '
15 '*************************
16 '*     INITIALISEREN      *
17 '*************************
18 CLEAR 8000
19 WIDTH 40: CLS
20 DIM A$(30,10):'bladen in agenda
21 DIM MY(12):'dagen per maand
22 DIM DW$(7):'dagen van de week
23 KEY OFF
24 ON INTERVAL=3000 GOSUB 1600:'AFDRUK DATUM
/TIJD
25 INTERVAL OFF
26 ON ERROR GOTO 600:'FOUTROUTINE
27 '
28 '*************************
29 '*      HUIDIGE DATUM     *
30 '*************************
31 X=0:Y=0
32 GOSUB 100:'DATUM INGAVE/CONTROLE
33 X=0:Y=21
34 DS$=D$
35 '
36 '*************************
37 '*    MOMENTELE TIJD     *
38 '*************************
39 X=0:Y=1
40 GOSUB 400:'TIJD INGAVE/CONTROLE
```

```
41 X=0:Y=21
42 J1=JA:M1=MA:D1=DA
43 U2=UU:M2=MM
44 CLS
45 '
46 '*************************
47 '* AFDRUKKEN DATUM/TIJD  *
48 '*************************
49 GOSUB 1600:'AFDRUK DATUM/TIJD
50 INTERVAL ON
51 '
52 '*************************
53 '*   ZET FUNCTIETOETSEN  *
54 '*************************
55 GOSUB 1200:'BLADERFUNCTIES
56 '
57 '*************************
58 '*  LEES AGENDA-BESTAND  *
59 '*************************
60 ER=3:LR=6:GOSUB 2400:'WIS REGELS
61 LOCATE 0,3
62 INPUT "AFSPRAKEN VAN CASSETTE LEZEN";I$
63 I$=LEFT$(I$,1)
64 IF I$="N" OR I$="n" GOTO 70
65 GOSUB 1000:'LEES AFSPRAKEN-FILE
66 '
67 '*************************
68 '*  PRINT AFSPRAKENBALK  *
69 '*************************
70 A=0
71 GOSUB 1900:'PRINT DATUM/BALK
72 GOSUB 3000:'PRINT AFSPRAAK
73 '
74 '*************************
75 '* EINDE HOOFDPROGRAMMA: *
76 '*     EINDELOZE LUS     *
77 '*************************
78 GOTO 78
79 '
97 '*************************
98 '*    DATUM CONTROLE     *
```

```
99 '************************
100 LOCATE X,Y
110 INPUT "DATUM (JJMMDD)";D$:D$=LEFT$(D$,6)
120 ER=18:LR=20:GOSUB 2400:'WIS REGELS
130 LOCATE X,Y
140 IF LEN(D$)<6 THEN ERROR 106
150 IF LEN(D$)>6 THEN ERROR 107
160 FOR I=1 TO 6
170 M$=MID$(D$,I,1)
180 IF M$<"0" OR M$>"9" THEN ERROR 101
190 NEXT I
200 JA=VAL(LEFT$(D$,2))
210 MA=VAL(MID$(D$,3,2))
220 IF MA>12 OR MA=0 THEN ERROR 102
230 RESTORE 310
240 FOR I=1 TO MA
250 READ M$,D
260 NEXT I
270 DA=VAL(RIGHT$(D$,2))
280 IF MA=2 AND JA/4=INT(JA/4) THEN D=29
290 IF D<DA OR DA=0 THEN ERROR 103
300 RETURN
310 DATA JAN,31,FEB,28,MRT,31,APR,30
320 DATA MEI,31,JUN,30,JUL,31,AUG,31
330 DATA SEP,30,OKT,31,NOV,30,DEC,31
340 '
397 '************************
398 '*     TIJD CONTROLE     *
399 '************************
400 LOCATE X,Y
410 INPUT "TIJD (UUMM)";T$
420 ER=18:LR=20:GOSUB 2400:'WIS REGELS
430 IF LEN(T$)<4 THEN ERROR 106
440 IF LEN(T$)>4 THEN ERROR 107
450 FOR I=1 TO 4
460 M$=MID$(T$,I,1)
470 IF M$<"0" OR M$>"9" THEN ERROR 101
480 NEXT I
490 UU=VAL(LEFT$(T$,2))
500 IF UU>23 THEN ERROR 104
510 MM=VAL(RIGHT$(T$,2))
```

```
520 IF MM>59 THEN ERROR 105
530 RETURN
540 '
597 '*************************
598 '*    FOUTAFHANDELING    *
599 '*************************
600 LOCATE 0,20
610 IF ERR=101 THEN GOTO 690
620 IF ERR=102 THEN GOTO 730
630 IF ERR=103 THEN GOTO 760
640 IF ERR=104 THEN GOTO 790
650 IF ERR=105 THEN GOTO 820
660 IF ERR=106 THEN GOTO 850
670 IF ERR=107 THEN GOTO 890
680 RESUME NEXT
690 PRINT "ALLEEN CIJFERS SVP.";
700 IF ERL=180 THEN RESUME 100
710 IF ERL=470 THEN RESUME 400
720 RESUME NEXT
730 PRINT "MAAND FOUT (1 - 12)";
740 IF ERL=220 THEN RESUME 100
750 RESUME NEXT
760 PRINT "DAG FOUT (1 -";D;")";
770 IF ERL=290 THEN RESUME 100
780 RESUME NEXT
790 PRINT "UREN FOUT (0 - 23)";
800 IF ERL=500 THEN RESUME 400
810 RESUME NEXT
820 PRINT "MINUTEN FOUT (0 - 59)";
830 IF ERL=520 THEN RESUME 400
840 RESUME NEXT
850 PRINT "TE WEINIG CIJFERS";
860 IF ERL=140 THEN RESUME 100
870 IF ERL=430 THEN RESUME 400
880 RESUME NEXT
890 PRINT "TE VEEL CIJFERS";
900 IF ERL=150 THEN RESUME 100
910 IF ERL=440 THEN RESUME 400
920 RESUME NEXT
930 '
997 '*************************
```

```
998 '* LEES AFSPRAKENBESTAND *
999 '************************
1000 OPEN "CAS:AFSPR" FOR INPUT AS #1
1010 FOR I=0 TO 30
1020 IF EOF(1)=-1 GOTO 1140
1030 LINE INPUT #1,I$:'DATUM
1040 IF I$<>"" AND I$=>D$ THEN A$(I,0)=I$:GO
TO 1100
1050 FOR J=1 TO 10:'DUMMY LEZEN
1060 LINE INPUT #1,I$
1070 NEXT J
1080 NEXT I
1090 GOTO 1140
1100 FOR J=1 TO 10:'LEES AFSPRAKEN
1110 LINE INPUT #1,A$(I,J)
1120 NEXT J
1130 NEXT I
1140 CLOSE #1
1150 RETURN
1160 '
1196 '************************
1197 '*  ZET FUNCTIETOETSEN:  *
1198 '*    BLADER-FUNCTIES    *
1199 '************************
1200 KEY OFF
1210 KEY 1," <---"
1220 KEY 2," --->"
1230 KEY 3," AFSPR"
1240 KEY 4," DATUM"
1250 KEY 5,"FUNCTIE"
1260 KEY ON
1270 FOR I=1 TO 5
1280 KEY (I) ON
1290 NEXT I
1300 ON KEY GOSUB 2600,2500,3000,2700,1400
1310 RETURN
1320 '
1396 '************************
1397 '*  ZET FUNCTIETOETSEN:  *
1398 '*   SCHRIJF-FUNCTIES    *
1399 '************************
```

```
1400 KEY OFF
1410 KEY 1," BIJ"
1420 KEY 2,"WIJZIG"
1430 KEY 3,"  AF"
1440 KEY 4," MENU"
1450 KEY 5,"FUNCTIE"
1460 KEY ON
1470 FOR I=1 TO 5
1480 KEY (I) ON
1490 NEXT I
1500 ON KEY GOSUB 3500,3200,4000,4300,1200
1510 RETURN
1520 '
1597 '************************
1598 '*  HUIDIGE DATUM/TIJD   *
1599 '************************
1600 IF M2>59 THEN M2=0:U2=U2+1
1610 IF U2>23 THEN U2=0:D1=D1+1
1620 RESTORE 310
1630 FOR I=1 TO M1
1640 READ M$,D
1650 NEXT I
1660 IF M1=2 AND J1/4=INT(J1/4) THEN D=29
1670 IF D1>D THEN D1=1:M1=M1+1:READ M$,D
1680 IF M1>12 THEN M1=1:RESTORE 310:READ M$,
D:J1=J1+1
1690 IF J1>99 THEN J1=0
1700 LOCATE 0,0
1710 PRINT "DATUM: ";
1720 PRINT USING "##";D1;
1730 PRINT "-";
1740 PRINT M$;"-";
1750 PRINT USING "##";J1;
1760 LOCATE 28,0
1770 PRINT "TIJD: ";
1780 PRINT USING "##";U2;
1790 PRINT ":";
1800 PRINT USING "##";M2;
1810 M2=M2+1
1820 RETURN
1830 '
```

```
1897 '************************
1898 '* DATUM + AFSPRAKENBALK *
1899 '************************
1900 ER=1:LR=20
1910 GOSUB 2400:'WIS REGELS
1920 IF A$(A,0)="" THEN GOSUB 4800:A$(A,0)=D
D$
1930 '******* dag van de week *******
1940 JX=VAL(LEFT$(A$(A,0),2))
1950 MX=VAL(MID$(A$(A,0),3,2))
1960 DX=VAL(RIGHT$(A$(A,0),2))
1970 DY=DX
1980 IF (JX MOD 4)=0 AND MX>2 THEN DY=DY+1
1990 RESTORE 2290
2000 FOR I=1 TO 12:READ MY(I):NEXT I
2010 FOR I=1 TO 7:READ DW$(I):NEXT I
2020 FOR I=1 TO M:DY=DY+MY(I):NEXT I
2030 DT=DY+JX+INT((JX+3)/4)
2040 DW=((DT-1)MOD 7)+1
2050 RESTORE 310
2060 FOR I=1 TO VAL(MID$(A$(A,0),3,2))
2070 READ M$,D
2080 NEXT I
2090 INTERVAL STOP
2100 LOCATE 0,3
2110 PRINT "datum: ";
2120 PRINT DW$(DW);", ";
2130 PRINT USING "##";DX;
2140 PRINT "-";
2150 PRINT M$;"-";
2160 PRINT USING "##";J;
2170 LOCATE 0,4
2180 PRINT STRING$(40,CHR$(&H5F))
2190 LOCATE 0,5
2200 FOR I=1 TO 40
2210 I$=MID$(A$(A,1),I,1)
2220 IF I$="1" THEN PRINT CHR$(&HDB); ELSE P
RINT "_";
2230 NEXT I
2240 FOR I=0 TO 9
2250 LOCATE I*4,6: PRINT USING "##";I+8
```

```
2260 NEXT I
2270 BEEP
2280 INTERVAL ON
2290 DATA 0,31,28,31,30,31,30,31,31,30,31,30
2300 DATA Zondag,Maandag,Dinsdag,Woensdag
2310 DATA Donderdag,Vrijdag,zaterdag
2320 RETURN
2330 '
2397 '************************
2398 '*      WIS REGELS      *
2399 '************************
2400 INTERVAL STOP
2410 FOR I=ER TO LR
2420 LOCATE 0,I
2430 PRINT SPC(40);
2440 NEXT I
2450 LOCATE 0,ER
2460 INTERVAL ON
2470 RETURN
2480 '
2497 '************************
2498 '*   VOLGENDE PAGINA    *
2499 '************************
2500 A=A+1
2510 IF A>30 THEN A=0
2520 GOSUB 1900:'AFSPRAKENBALK
2530 AF=1
2540 GOSUB 3000:'AFSPR. KIJKEN
2550 RETURN
2560 '
2597 '************************
2598 '*    VORIGE PAGINA     *
2599 '************************
2600 A=A-1
2610 IF A<0 THEN A=30
2620 GOSUB 1900:'AFSPRAKENBALK
2630 AF=1
2640 GOSUB 3000:'AFSPR. KIJKEN
2650 RETURN
2660 '
2697 '************************
```

```
2698 '*  NAAR PAGINA "DATUM" *
2699 '************************
2700 X=0:Y=18:GOSUB 100
2710 IF D$="" GOTO 2910
2720 DD$=D$
2730 DH=0
2740 RESTORE 310
2750 FOR I=1 TO VAL(MID$(DS$,3,2))-1
2760 READ M$,D
2770 DH=DH+D
2780 NEXT I
2790 DH=DH+VAL(RIGHT$(DS$,2))
2800 DG=0
2810 RESTORE 310
2820 FOR I=1 TO VAL(MID$(DD$,3,2))-1
2830 READ M$,D
2840 DG=DG+D
2850 NEXT I
2860 DG=DG+VAL(RIGHT$(DD$,2))
2870 A=DG-DH
2880 IF A<0 THEN A=0
2890 IF A>30 THEN A=30
2900 GOSUB 1900:'AFSPRAKENBALK
2910 RETURN
2920 '
2997 '************************
2998 '*  AFSPRAKEN BEKIJKEN  *
2999 '************************
3000 AF=AF+1
3010 IF AF<2 OR AF>10 THEN AF=2
3020 ER=10:LR=19:GOSUB 2400:'WIS REGELS
3030 IF A$(A,AF)="" AND AF=>2 THEN AF=2
3040 LOCATE 0,10:PRINT "AFSPRAAK";AF-1
3050 LOCATE 0,12:PRINT "VAN: ";LEFT$(A$(A,AF
),4)
3060 PRINT "TOT: ";MID$(A$(A,AF),5,4)
3070 LOCATE 0,15:PRINT "MET/OVER:"
3080 PRINT RIGHT$(A$(A,AF),(LEN(A$(A,AF))-8)
)
3090 RETURN
3100 '
```

```
3197 '************************
3198 '*  WIJZIGEN AFSPRAAK   *
3199 '************************
3200 GOSUB 3000:'AFSPRAAK BEKIJKEN
3210 ER=18:LR=19:GOSUB 2400:'WIS REGELS
3220 LOCATE 0,18:INPUT "WIJZIGEN";I$
3230 ER=18:LR=19:GOSUB 2400:'WIS REGELS
3240 IF I$<>"J" AND I$<>"j" THEN GOTO 1200
3250 IF AF>1 AND A$(A,AF)="" GOTO 1200
3260 ER=18:LR=19:GOSUB 2400
3270 LOCATE 0,18:INPUT "TIJD OF INHOUD (T/I)
";I$
3280 IF I$="T" OR I$="t" GOTO 3380
3290 IF I$<>"I" AND I$<>"i" GOTO 3260
3300 ER=18:LR=19:GOSUB 2400
3310 LOCATE 0,18:INPUT "MET WIE/WAAROVER";I$
3320 IF LEN(I$)>60 THEN I$=LEFT$(I$,60)
3330 A$(A,AF)=LEFT$(A$(A,AF),8)+I$
3340 GOSUB 3010:'AFSPRAAK BEKIJKEN
3350 GOSUB 1200:'BLADERFUNCTIES
3360 RETURN
3370 '**** VERWIJDEREN OUDE TIJD
3380 T1=INT((60*VAL(LEFT$(A$(A,AF),2))+VAL(M
ID$(A$(A,AF),3,2)))/15)-32
3390 T2=INT((60*VAL(MID$(A$(A,AF),5,2))+VAL(
MID$(A$(A,AF),7,2)))/15)-32
3400 X$=""
3410 FOR I=T1 TO T2
3420 X$=X$+"0"
3430 NEXT I
3440 A$(A,1)=LEFT$(A$(A,1),T1)+X$+RIGHT$(A$(
A,1),40-T2)
3450 L=LEN(A$(A,AF))-8
3460 A$(A,AF)=RIGHT$(A$(A,AF),L)
3470 GOTO 3590
3480 '
3497 '************************
3498 '*  AFSPRAAK TOEVOEGEN  *
3499 '************************
3500 GOSUB 2700:'NAAR PAGINA
3510 AF=0
```

```
3520 FOR I=2 TO 10
3530 IF A$(A,I)="" THEN AF=I:GOTO 3550
3540 NEXT I
3550 IF AF=0 THEN RETURN
3560 ER=17:LR=19:GOSUB 2400:'WIS REGELS
3570 LOCATE 0,18:INPUT "MET WIE/WAAROVER";A$
(A,AF)
3580 IF LEN(A$(A,AF))>60 THEN A$(A,AF)=LEFT$
(A$(A,AF),60)
3590 ER=17:LR=19:GOSUB 2400:'WIS REGELS
3600 X=0:Y=18
3610 LOCATE X,Y-1:PRINT "BEGIN";
3620 GOSUB 400:INV+CONTR. TIJD
3630 T1$=T$
3640 IF T1$>"1745" GOTO 3600
3650 IF T1$<"0800" GOTO 3600
3660 ER=17:LR=19:GOSUB 2400:'WIS REGELS
3670 X=0:Y=18
3680 LOCATE X,Y-1:PRINT "EIND-";
3690 GOSUB 400:INV.+CONTR. TIJD
3700 T2$=T$
3710 IF T2$<T1$ GOTO 3670
3720 IF T2$>"1800" GOTO 3670
3730 '*** BEPAAL PLAATS EERST BLOK
3740 T1=INT((60*VAL(LEFT$(T1$,2))+VAL(RIGHT$
(T1$,2)))/15)-32
3750 '*** BEPAAL PLAATS LAATSTE BLOK
3760 T2=INT((60*VAL(LEFT$(T2$,2))+VAL(RIGHT$
(T2$,2)))/15)-32
3770 IF A$(A,1)="" THEN A$(A,1)=STRING$(40,4
8)
3780 '**** CONTROLE OP VRIJ ZIJN *****
3790 FOR I=T1 TO T2
3800 IF MID$(A$(A,1),I+1,1)="1" GOTO 3600
3810 NEXT I
3820 '**** TOEVOEGEN AFSPRAAKTYD *****
3830 A$(A,AF)=T1$+T2$+A$(A,AF)
3840 X$=LEFT$(A$(A,1),T1)+STRING$(T2-T1,49)+
RIGHT$(A$(A,1),40-T2)
3850 A$(A,1)=X$
3860 GOSUB 1900:'AFSPRAKENBALK
```

```
3870 ER=10:LR=19:GOSUB 2400:'WIS REGELS
3880 LOCATE 0,10:PRINT "AFSPRAAK";AF-1;":";
3890 LOCATE 0,12:PRINT "VAN: ";LEFT$(A$(A,AF
),4)
3900 PRINT "TOT: ";MID$(A$(A,AF),5,4)
3910 LOCATE 0,15: PRINT "MET/OVER:"
3920 PRINT RIGHT$(A$(A,AF),(LEN(A$(A,AF))-8)
);
3930 GOSUB 1200:'BLADERFUNCTIES
3940 RETURN
3950 '
3997 '************************
3998 '* AFSPRAAK VERWIJDEREN *
3999 '************************
4000 ER=18:LR=19:GOSUB 2400:'WIS REGELS
4010 LOCATE 0,18:PRINT "VERWIJDEREN?";
4020 I$=INKEY$:IF I$="" GOTO 4020
4030 IF I$<>"J" AND I$<>"j" THEN RETURN
4040 '**** TIJD UIT BALK VERWIJDEREN
4050 T1=INT((60*VAL(LEFT$(A$(A,AF),2))+VAL(M
ID$(A$(A,AF),3,2)))/15)-32
4060 T2=INT((60*VAL(MID$(A$(A,AF),5,2))+VAL(
MID$(A$(A,AF),7,2)))/15)-32
4070 X$=""
4080 FOR I=T1 TO T2
4090 X$=X$+"0"
4100 NEXT I
4110 A$(A,1)=LEFT$(A$(A,1),T1)+X$+RIGHT$(A$(
A,1),40-T2)
4120 IF A$(A,AF+1)="" GOTO 4160
4130 A$(A,AF)=A$(A,AF+1)
4140 AF=AF+1
4150 GOTO 4120
4160 A$(A,AF)=""
4170 GOSUB 1900:AF=AF-1:GOSUB 3000
4180 GOSUB 1200
4190 RETURN
4200 '
4297 '****************************
4298 '* MENU MET DIVERSE FUNCTIES *
4299 '****************************
```

```
4300 ER=2:LR=19:GOSUB 2400:'WIS REGELS
4310 KEY OFF
4320 FOR I=1 TO 5
4330 KEY (I) OFF
4340 NEXT I
4350 LOCATE 5,5:PRINT "1 = SAVE AFSPRAKEN"
4360 LOCATE 5,7:PRINT "2 = SAVE PROGRAMMA+AF
SPRAKEN"
4370 LOCATE 5,9:PRINT "3 = SAVE PROGRAMMA"
4380 LOCATE 5,11:PRINT "4 = ZOEK AFSPRAAK"
4390 LOCATE 5,13:PRINT "5 = BEEINDIG PROGRAM
MA"
4400 LOCATE 0,17:PRINT "WELKE FUNCTIE WENST
U?       (1 T/M 5)"
4410 I$=INKEY$:IF I$="" GOTO 4410
4420 IF I$<"1" OR I$>"5" GOTO 4410
4430 F=VAL(I$)
4440 ON F GOSUB 4600,5000,5100,5200,5500
4450 CLS
4460 GOSUB 1600:'PRINT DATUM/TIJD
4470 GOSUB 1900:'PRINT AFSPRAKENBALK
4480 GOSUB 3000:'PRINT AFSPRAAK
4490 GOSUB 1200:'ZET BLADERFUNCTIES
4500 RETURN
4510 '
4597 '************************
4598 '* ARRAY A$ NAAR CASS.  *
4599 '************************
4600 PRINT
4610 PRINT "LEG LEGE CASSETTE IN RECORDER"
4620 PRINT "ZET RECORDER IN STAND OPNEMEN"
4630 PRINT "DRUK OP RETURN OM OP TE NEMEN"
4640 IF INKEY$<>CHR$(13) GOTO 4640
4650 OPEN "CAS:AFSPR" FOR OUTPUT AS #1
4660 FOR I=0 TO 30
4670 FOR J=0 TO 10
4680 PRINT #1,A$(I,J)
4690 NEXT J
4700 NEXT I
4710 CLOSE #1
4720 RETURN
```

```
4730 '
4797 '************************
4798 '*      DATUM = D$+A      *
4799 '************************
4800 M=VAL(MID$(DS$,3,2))
4810 J=VAL(LEFT$(DS$,2))
4820 RESTORE 310
4830 FOR I=1 TO M
4840 READ M$,D
4850 NEXT I
4860 IF M=2 AND (INT(J/4)=J/4) THEN D=D+1
4870 DD=VAL(RIGHT$(DS$,2))+A
4880 IF DD>D THEN DD=DD-D:M=M+1
4890 IF M>12 THEN M=1:J=J+1
4900 IF J>99 THEN J=0
4910 DD$=RIGHT$(STR$(J),2)+RIGHT$(STR$(M),2)
+RIGHT$(STR$(DD),2)
4920 FOR I=1 TO 6
4930 IF MID$(DD$,I,1)=" " THEN MID$(DD$,I,1)
="0"
4940 NEXT I
4950 RETURN
4960 '
4997 '*****************************
4998 '* SAVE PROGRAMMA+AFSPRAKEN *
4999 '*****************************
5000 PRINT
5010 PRINT "LEG LEGE CASSETTE IN RECORDER"
5020 PRINT "ZET RECORDER IN STAND OPNEMEN"
5030 PRINT "DRUK OP RETURN OM OP TE NEMEN"
5040 IF INKEY$<>CHR$(13) GOTO 5040
5050 CSAVE "AGENDA"
5060 GOTO 4650
5070 '
5097 '****************************
5098 '*      SAVE PROGRAMMA      *
5099 '****************************
5100 PRINT
5110 PRINT "LEG LEGE CASSETTE IN RECORDER"
5120 PRINT "ZET RECORDER IN STAND OPNEMEN"
5130 PRINT "DRUK OP RETURN OM OP TE NEMEN"
```

```
5140 IF INKEY$<>CHR$(13) GOTO 5140
5150 CSAVE "AGENDA"
5160 RETURN
5170 '
5197 '****************************
5198 '*      ZOEK AFSPRAAK       *
5199 '****************************
5200 CLS
5210 PRINT "INDIEN U EEN TE ZOEKEN TEKST OP
GEEFT,  ZAL IK U DE AFSPRAAK WAARIN DIE TEKS
T   VOOR KOMT LATEN ZIEN."
5220 LOCATE 0,4
5230 PRINT "U MAG KIEZEN OF IK VANAF HET BEG
IN MOET ZOEKEN, OF DAT IK MOET ZOEKEN VANAF
DE  LAATST BEKEKEN DAG."
5240 LOCATE 0,12
5250 INPUT "WAT IS DE TE ZOEKEN TEKST";Z$
5260 LOCATE 0,12
5270 INPUT "VANAF HET BEGIN OF VANAF LAATST
BEKENEN DAG (B/L)";I$
5280 I$=LEFT$(I$,1)
5290 IF I$<>"B" AND I$<>"L" AND I$<>"b" AND
I$<>"l" GOTO 5270
5300 IF I$="B" OR I$="b" THEN A=0
5310 FOR I=A TO 30
5320 FOR J=2 TO 10
5330 IF INSTR(A$(I,J),Z$)<>0 GOTO 5420
5340 NEXT J
5350 NEXT I
5360 ER=12:LR=13:GOSUB 2400:'WIS REGELS
5370 LOCATE 0,12
5380 PRINT Z$
5390 PRINT " KOMT NIET VOOR IN DE AGENDA"
5400 FOR I=1 TO 2000:NEXT I
5410 RETURN
5420 A=I:AF=J-1:RETURN
5430 '
5497 '****************************
5498 '*     EINDE PROGRAMMA      *
5499 '****************************
5500 SCREEN 3
```

```
5510 OPEN "GRP:"AS #1
5520 DRAW "BM48,64"
5530 PRINT #1,"EINDE"
5540 DRAW "BM32,128"
5550 PRINT #1,"AGENDA"
5560 FOR I=1 TO 3000:NEXT I
5570 SCREEN 0:WIDTH 40
5580 END
```

Alfabetische trefwoordenlijst

! 50
50, 88
$ 47
% 48, 88
&B 52
&H 53
&O 52
* 62
+ 62
- 62
/ 62

1-dimensionale array 181
2-complement 54
2-dimensionale array 183

A
aftrekken 62
alfanumerieke constanten 46
alfanumerieke functies 54
alfanumerieke variabelen 47
alternatieve tekens 86
analyse 29
AND (operator) 113
argument 122
array 179, 180
ASCII-code 82
ASCII-codetabel 83
ASCII-formaat 124
AUTO (commando) 41

B
Back Space 43
BASIC-statements 27
baud 125
beeldpuntje 73
beeldscherm 18
beeldschermmodes 71
BEEP (statement) 153
beginwaarde 119
bestand 179, 188
bestandsnaam 189
bestandsnummer 190
BIN$(x) (functie) 54
binair talstelsel 22, 51
binaire cijfers 22
binaire notatie 54
bit 52
break 166
BS (toets) 43
byte 22

C
CAS: 133
cassette 123
cassette-commando's 130
cassetterecorder 23
CDBL(x) (functie) 57
Centronics interface 21
CHR$(12) 76
CHR$(7) 153
CHR$(x) (functie) 84
CINT(x) (functie) 57
CLEAR (statement) 66, 186
Clear Screen 43, 75
CLOAD (commando) 126
CLOAD (statement) 135
CLOAD? (commando) 129
CLOSE (statement) 192
CLS (statement) 43, 75
codes 82
commando 28
compiler 26
composite video 18
concatenation 64
constanten 46, 48
CONT (commando) 154
conversie-functies 57
conversie-mogelijkheden 56
CSAVE (commando) 124
CSAVE (statement) 135
CSNG(x) (functie) 57

CTRL+B 44
CTRL+C 44
CTRL+E 44
CTRL+F 44
CTRL+G 153
CTRL+J 44
CTRL+N 44
CTRL+STOP (toetsen) 155, 165
CTRL+U 44
cursor 40
cursor control toetsen 40

D
daisy-wheel printer 19
DATA (statement) 169, 171
decimaal talstelsel 51
decimale punt 89
declareren (variabelen) 57
DEFDBL (functie) 58
DEFINT (functie) 58
DEFSNG (functie) 58
DEFSTR (functie) 58
DEL (toets) 43
delen 62
DIM (statement) 181, 183, 188
directe mode 29
DRAW (statement) 74
drijvende komma 49
dubbelvoudige nauwkeurigheid 50

E
editor 41
editor-functies 42
eigen foutcodes 151
eindeloze lus 99
eindwaarde 119
elementen 180
end of file 193
enkelvoudige nauwkeurigheid 50
EOF(x) (functie) 193
ERASE (statement) 186, 188
ERL (systeemvariabele) 146
ERR (systeemvariabele) 146
ERROR x (statement) 148
exponent 49
exponentiele vorm 49, 93
extern geheugen 23

F
file 179, 188
fixed point 49
flexibele schijf 23
floating point 49
flow chart 30
FOR (statement) 118
FOR...NEXT-lus 118
Found: 127
foutafhandeling 146
foutcodes 146
foutzoeken 152
FRE("") (functie) 187
FRE(0) (functie) 187
functies 28, 63
functietoetsen 156

G
gegevens afdrukken 72
geheel deel 104
gehele getallen 48
geheugen 17, 23, 60
geheugenvakjes 40
geindiceerde variabelen 180
geluidsprocessor 17
GOSUB (statement) 108
GOTO (statement) 98
grafische mode 72
GRP: 74

H
hardware 11
hernummeren 39
HEX$(x) (functie) 54
hexadecimaal talstelsel 53
hexadecimale notatie 54
HOME (toets) 43
hoofdprogramma 33, 107

I
IF...GOTO (statement) 100, 116
IF...THEN (statement) 116
IF...THEN...ELSE 116
indexnummers 180
indices 180
indirecte mode 28
ingave statements 63
ingave-functies 68
ink-jet printer 19
INKEY$ (functie) 69
INPUT (statement) 64
INPUT$(n) (functie) 68
INS (toets) 43
INSTR (functie) 197
INT(x) (functie) 122
integer deel 104
integers 48
interpreter 26

INTERVAL OFF (statement) 140
INTERVAL ON (statement) 139
INTERVAL STOP (statement) 139
invoer-controle 142

J
juistheidstest 129

K
kanaalnummer 190
KEY () OFF (statement) 164
KEY () ON (statement) 162, 164
KEY () STOP (statement) 165
KEY LIST (comm./stat.) 156
KEY OFF (statement) 158
KEY ON (statement) 158
KEY x,string (statement) 159
kolom 184
komma in print-masker 91

L
LEFT$ (functie) 144
LEN(X$) (functie) 143
LET (statement) 61, 65
LINE INPUT (statement) 68
LIST (commando) 37, 45
LOAD (commando) 131
LOCATE (statement) 78
logische operators 63, 113
loop (Engels) 99
lussen 32, 111

M
machinecode 26
machinetaalinstructies 26
machtsverheffen 62
mantisse 49
matrix 180
matrix printer 19
MAXFILES (systeemvariabele) 190
menu 101, 105
MERGE (commando) 132
microprocessor 16
MID$ (functie) 143
min-teken 90
MOD 62
monitor 18
MOTOR (statement) 135
MSX-BASIC 25
MSX-codes 86
MSX-codetabel 85
MSX-computer 15

N
negatieve stapwaarde 120
nesten 121
NEW (commando) 38, 153
NEXT (statement) 118
NOT (operator) 115
numerieke constanten 46
numerieke variabelen 47
numerieke waardebepaling 54

O
OCT$(x) (functie) 54
octaal talstelsel 52
octale notatie 54
off-page connectoren 33
ON ERROR GOTO (statement) 146
ON INTERVAL GOSUB (stat.) 138
ON KEY GOSUB (statement) 160
ON STOP GOSUB (statement) 166
on-page connectoren 33
ON...GOSUB (statement) 110
ON...GOTO (statement) 102
OPEN (statement) 74, 189
operators 62, 111
opslaan 111
optellen 62
OR (operator) 114, 145

P
piepsignaal 153
pijltjestoetsen 43
pixel 73
plus-teken 90
PRINT (statement) 72, 74
PRINT USING (statement) 87
PRINT-masker 87
print-masker met "!" 95
print-masker met "#" 88
print-masker met "$$" 92
print-masker met "&" 95
print-masker met "**" 91
print-masker met "**$" 93
print-masker met "\\" 96
print-masker met "^^^^" 94
printer 19
prioriteit 62
probleemgerichte taal 25
probleemstelling 30
programma 33
programma's laden 124
programma's opslaan 124
programmanaam 127

R
R (LOAD-parameter) 131
RAM 24
Random Access Memory 24
READ (statement) 169, 170
Read Only Memory 24
reele getallen 49
regellengte 76
regelnummering 38, 41
rekenkundige operators 62
relationele operators 111
relationele uitdrukking 100
REM (statement) 61
remote-voorziening 126
RENUM (commando) 38
restbepalen 62
RESTORE (statement) 169, 172
RESUME (statement) 148
RESUME NEXT (statement) 148
RESUME x (statement) 149
RETURN (statement) 108
RETURN (toets) 37, 43
RGB 18
RIGHT$ (functie) 144
rij 184
ROM 24
RUN (commando) 131

S
samenvoegen (programma's) 132
SAVE (commando) 130
schrijfsnelheid 125
SCREEN (statement) 71, 75, 125
SCREEN 0 71
SCREEN 1 71
SCREEN 2 73
SCREEN 3 73
Skip: 127
software 12
sound-processor 17
SPACE$(x) (functie) 81
spaties 79
SPC(x) (functie) 79
springen 98
sprongopdracht 99
standaardisatie 12
stapwaarde 119
statement 28
STOP (statement) 154
STOP (toets) 165
STOP OFF (statement) 167
STOP ON (statement) 166
STOP STOP (statement) 167
STR$(x) (functie) 54
STRING$ (functie) 81
string-bewerkingen 95
string-koppeling 64
string-variabele 64
strings 46
strings in DATA-regels 174
stroomdiagram 30
subroutine 32, 34, 107
subscript out of range 182
SWAP (statement) 67
syntax-fouten 34

T
TAB (toets) 43
TAB(x) (functie) 77
tabulatorstoppen 77
televisietoestel 18
tijdsinterval 138
TIME (systeemvariabele) 142
toekennen van waarden 60
toetsaanslag 75
Trace off 153
Trace on 152
TROFF (commando) 153
TRON (commando) 152
TV-toestel 18, 154
tweetallig stelsel 22

V
VAL(X$) (functie) 54
variabelen 47, 60
vaste komma 49
vergelijkende operators 63
vergelijkende uitdrukking 100
vergelijkingen 100
Verify (error) 129
vermenigvuldigen 62
vertaalprogramma 26
verwijzingsconnectoren 33
video (composite-) 18
videoprocessor 16
volumeregelaar 128
voorkeursrichting 32

W
waarden toekennen 60
WIDTH (statement) 72, 76
wissen van een regel 80

Z
zoeken (naar string) 197

Nederlandstalige MSX handboeken

MSX BASIC handboek voor iedereen, door A.C.J. Groeneveld
Een kompleet nederlandstalig handboek voor iedere MSX computergebruiker. Dit handboek 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 1007

MSX ZAKBOEKJE door Wessel Akkermans
Een vlot geschreven naslagwerk na of naast het handboek. U vindt er o.a. in: niet computergerichte tabellen; de MSX-BASIC instructieset; diverse tabellen die het BASIC-programmeren kunnen versnellen; de Z80 instructieset; hardware-gegevens (connectoren) en een aantal programmaatjes
ISBN 90 6398 888 5

MSX DISK handboek voor iedereen, door A.C.J. Groeneveld
Handboek voor diskdrivebezitters om naast het grote handboek te gebruiken. Een zeer volledige behandeling van het disk-gebeuren zelf en de specifieke disk kommando's, uitgebreid met voorbeelden, tabellen en overzichten. Het handboek is aangevuld met interessante programma's, waaronder een tekentafelprogramma en een basisprogramma voor basisonderhoud
ISBN 90 6398 407 3

MSX PRAKTIJKPROGRAMMA'S door Wessel Akkermans
Praktische programma's met waar nodig eerst een stukje theorie. Erg handig bij het maken van uw programma's. Een greep uit de onderwerpen: priemgetallen; zoeken en sorteren; trefwoordenlijsten; converteren van getallen; enz.
ISBN 90 6398 437 5

MSX QUICK DISK handboek voor iedereen, door A.C.J. Groeneveld
Hèt handboek voor iedere QUICK DISK gebruiker. Uitvoerige behandeling van de sleutelwoorden aangevuld met duidelijke voorbeelden met listing
ISBN 90 6398 254 2

MSX DOS handboek voor iedereen, door A.C.J. Groeneveld
Dit handboek geeft u op een heldere wijze een totaalbeeld van de mogelijkheden van het MSX-DOS. Ook is dit handboek voorzien van een inleiding op het begrip 'operating system' en dus echt een handboek voor iedereen
ISBN 90 6398 674 2

MSX TRUUKS EN TIPS
door A.C.J. Groeneveld
Hoe laat ik de computer een cirkel arceren, hoe tover ik mijn computer om in een elektronisch orgeltje, hoe maak ik een mooie intro voor een spelletje. Allemaal vraagstukken die zich lastig laten programmeren maar die iedere MSX-er toch graag opgelost wil zien.
Dit boekje staat boordevol truuks en tips, allemaal in gewoon MSX basic geschreven. Bladerend door dit boek komt u tot de ontdekking dat er voornamelijk korte maar uiterst krachtige en bijzonder goed bruikbare routines zijn opgenomen. Dit boekje geeft kort maar krachtig een antwoord op al uw programmeervragen.
deel 1 ISBN 90 6398 900 8
deel 2 ISBN 90 6398 340 9

SOFTWARE PLUS IN MSX

INTROTAPE MSX door A.C.J. Groeneveld
Heeft u nog maar net een MSX computer gekocht en wilt u graag weten wat de computer kan en hoe u hem kunt leren programmeren? Deze cassette introduceert MSX op een uiterst vriendelijke en onderwijzende manier. U krijgt instructies hoe u de computer aan moet sluiten en de tape laden. Daarna volgt een demonstratie van de mogelijkheden in MSX, zoals het tekenen van sprites en het werken met de driestemmige toongenerator. Het geheel wordt afgesloten met twee 'les' gedeeltes. In anderhalf à drie uur weet u wat de MSX computer is, wat hij kan, en heeft u haast ongemerkt al wat regels geprogrammeerd.
ISBN 90 6398 148 1

MSX SCRIPT door Ton Weijters
Een menugestuurde nederlandstalige tekstverwerker. Het programma is geschikt om efficiënt grotere of kleinere teksten te bewerken. Paginaindeling (regellengte, paginalengte, marge, inspringen, centreren, enz.) wordt door het programma verzorgd. Dit geldt ook voor de paginatelling, toptitel en het eventueel uitvullen van de regels. Ook corrigeren, zoeken, string-substitutie, blokken tekst verplaatsen, copiëren of verwijderen, onderstrepen en vet zetten, is mogelijk met dit programma.
ISBN 90 6398 189 9

MSX DRAWS door A.C.J. Groeneveld
Een tekenprogramma in MSX basic, waarmee u al binnen 10 minuten uw eerste tekening kunt maken. Draws werkt erg vriendelijk en maakt gebruik van alle grafische mogelijkheden van de MSX computer. U kunt met Draws zowel technisch als creatieve tekeningen maken. Het programma heeft een effectief bereik van ruim 30.000 bij 30.000 puntjes met mogelijkheden als lijnen, cirkels, krommen, inkleuren, vergroten, verkleinen, verschuiven, verdraaien en andere tekeningen invoegen
ISBN 90 6398 754 4

LEERBOEK BASIC deel 1

De serie MSX Leerboeken geeft een komplete kursus MSX-BASIC programmeren, in drie delen.

Deze leerboeken zijn gericht op de beginnende programmeur. De moeilijkheidsgraad van de leerstof wordt dan ook slechts geleidelijk hoger. De gebruikte voorbeelden zijn zo praktisch mogelijk gekozen. Hierdoor kunnen al in een vroeg stadium bruikbare programma's worden gemaakt. Dit zal de lezer/leerling er toe aansporen om verder te gaan. Aan het eind van ieder deel is een groot voorbeeldprogramma opgenomen. Dit programma laat zien waartoe de lezer/leerling na bestuderen van het betreffende leerboek in staat zal zijn.

Bij ieder leerboek is een afzonderlijk „Opdrachten en uitwerkingen" boekje te verkrijgen. In deze boekjes staan, in volgorde van de hoofdstukken uit het leerboek, vragen en opdrachten met antwoorden en uitwerkingen.
Zowel voor gebruik op school als voor individueel gebruik zullen deze boekjes erg nuttig zijn.

ISBN 90 6398 649 1